第 1 章　数据库的启动和关闭

通常所说的 Oracle Server 主要由两个部分组成：Instance 和 Database。Instance 是指一组后台进程（在 Windows 上是一组线程）和一块共享内存区域；Database 是指存储在磁盘上的一组物理文件。通过 Instance 与 Database 协同，Oracle 数据库才能形成一个动态的可访问关系型数据库系统。

本章将由数据库如何启动与关闭入手，开始和大家一起进入 Oracle 数据库的国度。

1. 数据库的启动

从表象来看，数据库的启动极其简单，只需要以 SYSDBA/SYSOPER 身份登陆，敲一条 startup 命令既可启动数据库。然而在这条命令之后，Oracle 需要执行一系列复杂的操作，深入理解这些操作不仅有助于了解 Oracle 数据库的运行机制，还可以在故障发生时帮助大家快速的定位问题的根源所在，所以接下来让我们一起分析一下数据库的启动过程。

Oracle 数据库的启动主要包含三个步骤：

- 启动数据库到 Nomount 状态
- 启动数据库到 Mount 状态
- 启动数据库到 Open 状态

完成这三个过程，数据库才能进入就绪状态，准备提供数据访问。图 1-1 描述了一个数据库从关闭状态（shutdown）到 OPEN 状态经历的这些步骤（关闭步骤与此相反，是一个逆向过程）：

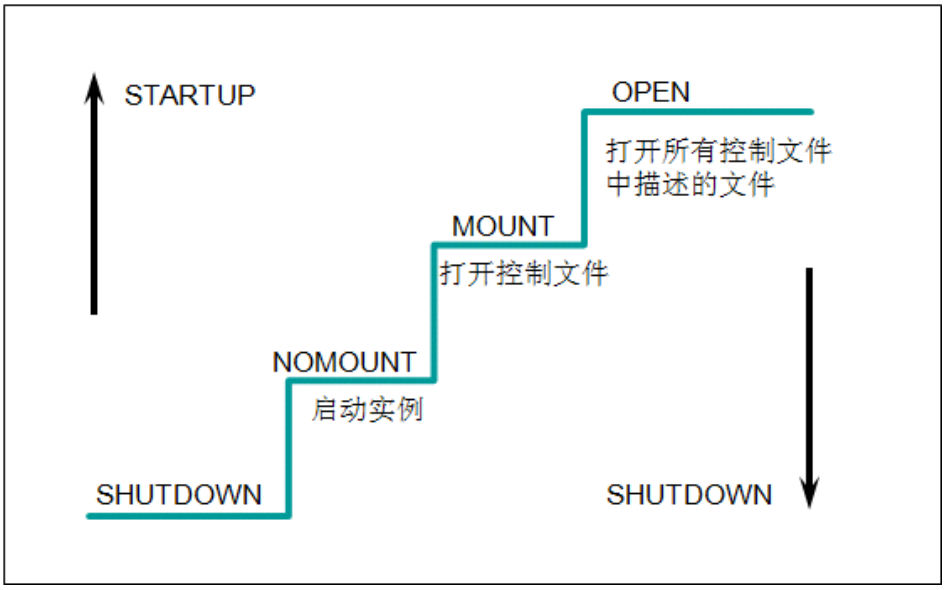


图 0-1-1 数据库的起停步骤

下面逐个来看看以上各个步骤的具体过程以及含义。

1.1 启动数据库到 Nomount 状态

在启动的第一步骤，Oracle 首先寻找参数文件(pfile / spfile)，然后根据参数文件中的设置（如内存分配等设置），创建实例（INSTANCE），分配内存，启动后台进程。**Nomount 的过程也就是启动数据库实例的过程**。这个过程在后台是启动 Oracle 可执行程序的过程，Windows 上是执行 oracle.exe 文件进行初始化，在 Unix/Linux 上是执行 oracle 可执行文件进行初始化。

1.1.1. Oracle 的可执行文件

Windows 上 Oracle 11g 的执行文件大小约为 86M，而 Linux 上 Oracle 11g 的执行文件达到 145M 左右，在 Oracle 12c 版本中，Oracle 可执行文件的大小达到了约 280M，由此可见不同版本源码复杂度的增加：

```
D:\oracle\product\11.1.0\BIN>dir oracle.exe
2007-10-03  17:42        89,702,400 oracle.exe
[oracle@localhost bin]$ ls -al $ORACLE_HOME/bin/oracle
-rwsr-s--x  1 oracle dba 151901909 Jul  4 15:13 /oracle/product/11.1.0/bin/oracle
eygle-/home/oracle$ ls -l $ORACLE_HOME/bin/oracle
-rwsr-s--x. 1 oracle oinstall 281911888 Mar 16 23:40 /oracle/product/12.1.0/bin/oracle
```

在 Unix/Linux 上可以通过 file 命令查看 oracle 执行文件来判断 Oracle 是 64 位或是 32 位的，以下是 Linux 平台的一个示范输出，输出显示 Oracle 为 32 位：

```
[oracle@localhost bin]$ file $ORACLE_HOME/bin/oracle
/opt/oracle/product/11.1.0/bin/oracle: setuid setgid ELF 32-bit LSB executable, Intel
80386, version 1 (SYSV), for GNU/Linux 2.2.5, dynamically linked (uses shared libs), not
stripped
```

在 Windows 上，也有可选的命令增强工具可以提供类似的功能。由于 32 位的 Oracle 数据库，最多只能使用约 1.7G 的内存，会极大约束 Oracle 的性能，在生产环境中，应当使用 64 位系统，部署 64 位的 Oracle 数据库。

了解 Oracle 可执行文件还有另外一个用途，在 Unix/Linux 上通过 strings 命令可以将 oracle 可执行文件中的字符文本转储出来，在转储的文本中可以找到很多有意思的信息，比如一些 Oracle 未公开的 Hints 信息、数据库字典基表创建信息等，类似如下一条命令可以完成这样的工作：

```
strings $ORACLE_HOME/bin/oracle > oracle.txt
```

比如如下命令显示了源码中和 Oracle Database 12c 新特性 Pluggable 数据库相关的一些信息：

```
eygle-/home/oracle$ grep -i pluggable oracle.txt|more
Pluggable Database Open/Close
Pluggable Database
```

```
Enqueue held by foreground or DBWR to synchronize database mount/open or pluggable database
open with other operations
Synchronize pluggable database open/close
_enable_pluggable_database  Enable Pluggable Database
_pluggable_database_debug   Enable checking of pluggable database ID in redo
Debug flag for pluggable database related operations
pluggable database open
pluggable database close
Pluggable database resetlogs
Coordinate pluggable database operations
Pluggable Database File Copy
PLUGGABLE_DATABASE
Pluggable database
Unsupported Rollback Segment/Undo Tablespace operation issued (and ignored) in a Pluggable
Database n
```

在 Oracle 10g 的版本中，某位程序员甚至引入了一段英国 Radiohead 乐队 Creep 这首歌的歌词，他说他不属于这里：

```
[eygle@hpserver2 ~]$ strings $ORACLE_HOME/bin/oracle |grep radiohead
I'm a creep, I'm a winner, what the hell am I doing here.I don't belong here - radiohead
```

当然在后续版本中，这段表白被去除了。

1.1.2.　实例以及进程的创建

在 Nomount 初始化的过程中，只要拥有了一个参数文件，就可以凭之启动实例(INSTANCE)，这一步骤并不需要任何控制文件或数据文件等的参与。以下是在 Linux 平台上正常启动实例到 nomount 状态的过程：

```
eygle-/home/oracle$ export ORACLE_SID=eygle
eygle-/home/oracle$ echo "db_name=eygle" > $ORACLE_HOME/dbs/initeygle.ora
eygle-/home/oracle$ sqlplus "/ as sysdba"
SQL*Plus: Release 12.1.0.0.1 Beta on Thu Jun 14 16:59:39 2012
Copyright (c) 1982, 2012, Oracle.  All rights reserved.

Connected to an idle instance.

SQL> startup nomount;
ORACLE instance started.

Total System Global Area  238034944 bytes
```

```
Fixed Size                  2272840 bytes
Variable Size             180355512 bytes
Database Buffers           50331648 bytes
Redo Buffers                5074944 bytes
```

注意观察，Oracle 根据参数文件的内容，创建了 instance,分配了相应的内存区域，启动了相应的后台进程。SGA 的分配信息从以上输出中可以看到。

观察告警日志文件（alert_<ORACLE_SID>.log），可以看到这一阶段的启动过程：**读取参数文件，应用参数启动实例**。所有在参数文件中定义的非缺省参数都会记录在告警日志文件中，以下是这一过程的日志摘要示例：

```
Thu Jun 14 16:55:14 2012
WARNING: unknown state for DB spfile location resource
Starting ORACLE instance (normal)
CLI notifier numLatches:3 maxDescs:222
****************** Large Pages Information *****************

Total System Global Area in large pages = 0 KB (0%)

Large pages used by this instance: 0 (0 KB)
Large pages unused system wide = 0 (0 KB) (alloc incr 4096 KB)
Large pages configured system wide = 0 (0 KB)
Large page size = 2048 KB

RECOMMENDATION:
  Total System Global Area size is 230 MB. For optimal performance,
  prior to the next instance restart increase the number of unused
  large pages by at least 115 (page size 2048 KB, total size 230 MB)
  system wide to get 100% of the System
  Global Area allocated with large pages
************************************************************
LICENSE_MAX_SESSION = 0
LICENSE_SESSIONS_WARNING = 0
Picked latch-free SCN scheme 3
Using LOG_ARCHIVE_DEST_1 parameter default value as /u01/oracle/product/12.1.0/dbs/arch
Autotune of undo retention is turned on.
IMODE=BR
ILAT =22
LICENSE_MAX_USERS = 0
SYS auditing is disabled
```

```
kfvLoadLib can not be called in this instance type
Starting up:
Oracle Database 12c Enterprise Edition Release 12.1.0.0.1 - 64bit Beta
With the Partitioning, OLAP, Data Mining and Real Application Testing options.
ORACLE_HOME = /u01/oracle/product/12.1.0
System name:  Linux
Node name:    eygle
Release:  2.6.32-100.28.5.el6.x86_64
Version:  #1 SMP Wed Feb 2 18:40:23 EST 2011
Machine:  x86_64
VM name:  VMWare Version: 6
Using parameter settings in server-side pfile
/u01/oracle/product/12.1.0/dbs/initeygle.ora
System parameters with non-default values:
  db_name                  = "eygle"
```

应用参数创建实例之后，后台进程依次启动。

注意在以下日志输出中包含了 PID 信息以及 OS ID 两个信息，PID 代表该进程在数据库内部的标识符编号，而 OS ID 则代表该进程在操作系统上的进程编号：

```
Thu Jun 14 16:59:43 2012
PMON started with pid=2, OS id=2609
Thu Jun 14 16:59:43 2012
PSP0 started with pid=3, OS id=2611
Thu Jun 14 16:59:44 2012
VKTM started with pid=4, OS id=2613
VKTM running at (100ms) precision
Thu Jun 14 16:59:44 2012
GEN0 started with pid=5, OS id=2617
Thu Jun 14 16:59:44 2012
DIAG started with pid=6, OS id=2619
Thu Jun 14 16:59:44 2012
OFSD started with pid=7, OS id=2621
Thu Jun 14 16:59:44 2012
DBRM started with pid=8, OS id=2623
Thu Jun 14 16:59:44 2012
DIA0 started with pid=9, OS id=2625
Thu Jun 14 16:59:44 2012
MMAN started with pid=10, OS id=2627
Thu Jun 14 16:59:44 2012
```

```
DBW0 started with pid=11, OS id=2629
Thu Jun 14 16:59:44 2012
LGWR started with pid=12, OS id=2631
Thu Jun 14 16:59:44 2012
CKPT started with pid=13, OS id=2633
Thu Jun 14 16:59:44 2012
SMON started with pid=14, OS id=2635
Thu Jun 14 16:59:44 2012
RECO started with pid=15, OS id=2637
Thu Jun 14 16:59:44 2012
LREG started with pid=16, OS id=2639
Thu Jun 14 16:59:44 2012
MMON started with pid=17, OS id=2641
Thu Jun 14 16:59:44 2012
MMNL started with pid=18, OS id=2643
ORACLE_BASE from environment = /u01/oracle
Using default pga_aggregate_limit of 1024 MB
```

在这里提醒大家注意一下Oracle不同版本告警日志信息的变化，在Oracle 9i早期版本中，后台进程启动的日志信息里并不包含OS ID，以下是Oracle 9.2.0.4的日志信息（在Oracle 9.2.0.8中已经包含了 OS ID 信息）：

```
PMON started with pid=2
DBW0 started with pid=3
LGWR started with pid=4
CKPT started with pid=5
SMON started with pid=6
RECO started with pid=7
```

自 Oracle Database 11g 开始，这部分信息有了进一步的增强，输出中不仅包含了 OS ID，而且每个后台进程的启动都有单独的时间标记（时间标记可以帮助判断每个后台进程启动时所消耗的时间，从而辅助进行问题诊断）：

```
Sat Jul 05 09:53:55 2008
PMON started with pid=2, OS id=13898
Sat Jul 05 09:53:55 2008
VKTM started with pid=3, OS id=13900 at elevated priority
VKTM running at (20)ms precision
Sat Jul 05 09:53:55 2008
DIAG started with pid=4, OS id=13904
Sat Jul 05 09:53:55 2008
DBRM started with pid=5, OS id=13906
```

```
Sat Jul 05 09:53:55 2008
PSP0 started with pid=6, OS id=13908
```

提示：从 Oracle 不同版本中的变化来体会 Oracle 的技术进步、甚至借鉴这些变化是学习 Oracle 的方法之一。任何细微的变化都值得注意，**认真、细致、严谨是对 DBA 的基本素质要求**。

1.1.3. Oracle 的后台进程

在本章开篇已经提到：Oracle 的实例（Instance)是指一组后台进程（在 Windows 上是一组线程）和一块共享内存区域。图 1-2 即为 Oracle 数据库的实例概要图，可以看到实例的内存区域 SGA 和一系列的后台进程示意（详细介绍将在后面章节逐步展开）：

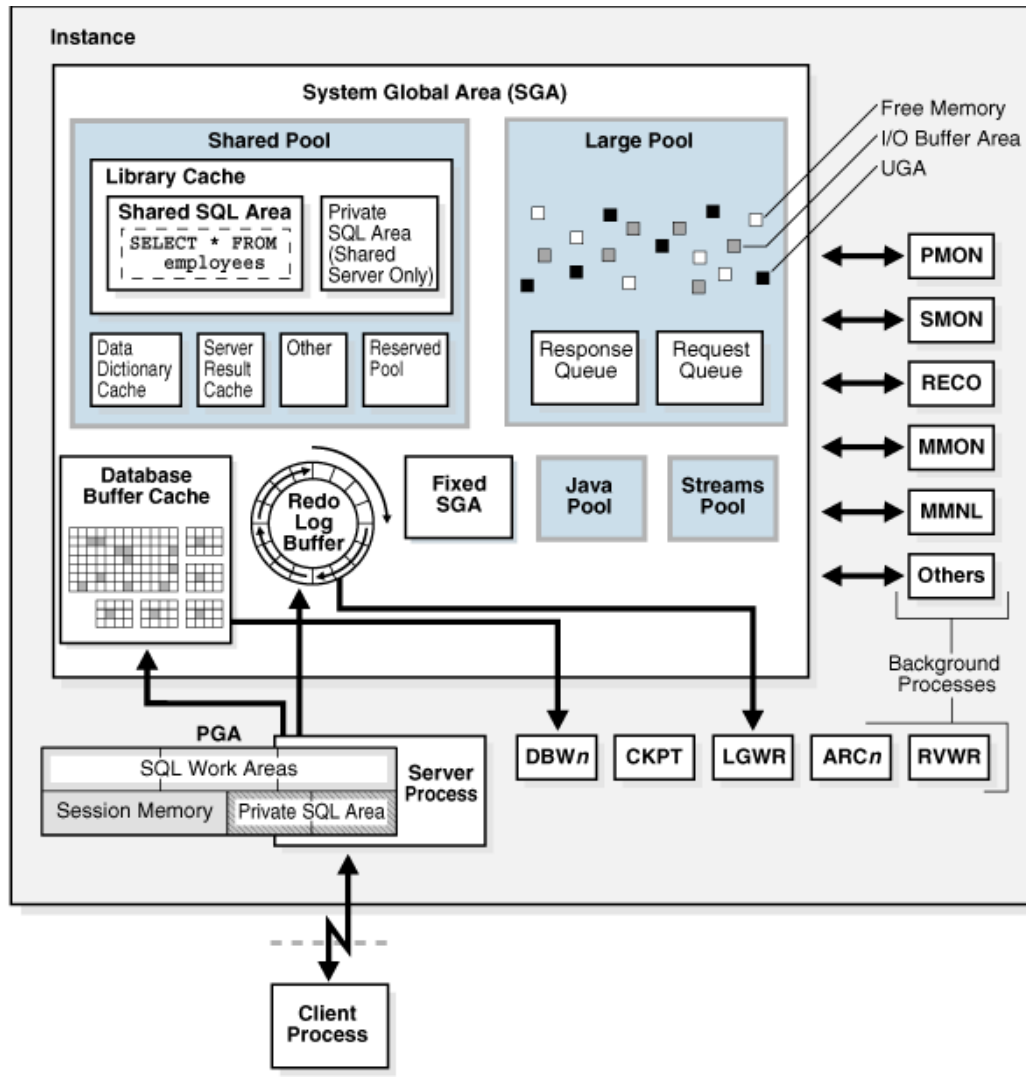


图 1-0-2 Oracle 实例示意图

通过上一节提到的日志信息，可以清晰的看到后台进程启动的过程，这里对于数据库运行最为至关重要的进程主要有：PMON、DBWn、LGWR、CKPT、SMON、VKTM 等。如果这些进程出现异常终止，则数据库将会崩溃。

下面简要介绍一些这些进程的作用。

PMON – 其全名为 Process Monitor – 进程监控进程，用于监控其他用户进程，当出现用户进程失败时，执行进程恢复工作。PMON 的主要恢复职责在于释放失败进程未能正常释放的内存锁和其他资源，如清理事务表状态等。

作为实例第一个启动的后台进程，PMON 还肩负监控其他 Oracle 后台进程的职责，在必要时重启这些后台进程；此外，PMON 还负责向 TNS 监听器注册实例。

DBWn – 其全名为 DataBase Writer Process – 数据库写进程，有时也被写作 DBWR 进程，该进程负责将内存中变更的数据写入磁盘数据文件。由于可以存在多个进程，所以增加 n 作为后缀，虽然大多数情况下一个数据库写进程（DBW0）就足**够**了，但是 Oracle 允许配置最多 20 个写进程，名称为：DBW0~DBW9，DBWa~DBWj。对于多处理器的系统，多个 DBWR 进程可以提升写性能。

DBWn 进程在满足以下条件时执行脏数据（Dirty Buffer）写出:

- 当一个 Server 进程扫描一定量的 Buffer 后仍然不能发现可用的 Buffer 时，通知 DBWn 执行写出；
- DBWn 会自动阶段性的执行写出，增进检查点。

由于内存中数据块的变更是零散的，所以 DBWn 执行批量写出才能有利于性能提升，LGWR 进程通过日志写操作，延迟 DBWn 的写操作，从而实现了协同与性能促进。

LGWR – 其全名为 Log Writer Process – 日志写进程，LGWR 进程管理 SGA 中的 Redo Log Buffer，将其中的内容连续写出到在线的日志文件中。Oracle 的 Redo 中记录的是可以重演事务变更的必要信息，在事务提交时，日志必须写出到日志文件中。通过 LGWR 对于日志文件的连续写出，就可以推延了 DBWR 对于脏数据的写出，从而保证 DBWR 可以实现批量写出的性能提升。

在以下条件下，LGWR 将执行写出操作：

- 用户提交事务
- 在线日志发生切换（Log Switch）
- 距上次写操作 3 秒
- Redo Log Buffer 1/3 满或者具备 1 MB 数据
- 在 DBWn 执行写出前，需要确保相关 Dirty Buffer 的对应日志都已写出，此时会通知 LGWR 写出此前未完成写出的相关日志。

CKPT – 其全名为 Checkpoint Process – 检查点进程，它负责更新控制文件和数据文件头的检查点信息，CKPT 进程还会通知 DBWn 进程去执行写操作。检查点信息包括检查点位置、SCN、Redo 的恢复位置等。

SMON – 其全名为 System Monitor Process – 系统监控进程，负责执行一系列的系统级别的清理和其他工作，包括：

- 在数据库启动时，执行必要的实例恢复。在 RAC 数据库中，SMON 进程负责为失败实例执行实例恢复。

- 执行在实例恢复时跳过的中断事务，如由于读文件或表空间离线等错误，在实例恢复时无法执行，SMON 则可以在表空间或文件在线时执行事务恢复。
- 回滚死事务，当一个进程异常中断，它的未提交事务就成为死事务，由 SMON 进程负责回滚。
- 清理不再使用的临时段，例如当索引创建失败时产生的遗留临时段。
- 在字典管理表空间中，执行连续自由区间的合并。
- 维护 SMON_SCN_TIME 系统表。SMON_SCN_TIME 表记录了 SCN 与时间的对应关系，由 SMON 定期进行定新，并将一些较老的数据定期删除。
- 维护 col_usage$数据字典表、维护 mod_mods$系统表。
- 定期 OFFLINE 多余的回滚段。
- 清理 obj$数据字典表中的垃圾数据。

VKTM – 其全名为Virtual Keeper of TiMe – 虚拟时钟进程，这个进程是在Oracle Database 11g 中引入的进程，用于提供一个数据库内部的虚拟时钟，以计算各种时间间隔量度等，VKTM 的引入降低和操作系统之间的交互，从而提高了性能。

VKTM 有两种运行模式：

- 基础模式，秒级间隔更新；
- 或者作为参考时间计数器，这种方式每 20 毫秒更新一次，仅在高优先级时可用。

对于不同的版本，VKTM 的设定模式可能不同，具体可以从告警日志文件中查知。

在 Oracle 11g 启动的过程中，信息提示：

```
VKTM running at (100ms) precision
```

在 Oracle 12c 中，信息提示：

```
VKTM started with pid=3, OS id=13900 at elevated priority
VKTM running at (20)ms precision
```

这个进程作为一个必选进程，对于数据库运行变得必不可少。

1.1.4.　V$PROCESS 视图

细心的读者朋友通过前面的阅读或许已经注意到，在日志的进程启动信息里，并没有 pid=1 的进程，那么这个进程是否存在呢？

通过数据库中的 v$process 视图，可以找到对应于操作系统的每个进程信息：

```
SQL> select addr,pid,spid,username,program from v$process;
ADDR             PID SPID       USERNAME PROGRAM
-------- ---------- ---------- -------- ----------------------------------------
5FE162AC           1                    PSEUDO
5FE16860           2 6290        oracle   oracle@eygle (PMON)
5FE16E14           3 6292        oracle   oracle@eygle (PSP0)
5FE173C8           4 6294       oracle   oracle@eygle (MMAN)
5FE1797C           5 6296       oracle   oracle@eygle (DBW0)
5FE17F30           6 6298       oracle   oracle@eygle (LGWR)
```

```
5FE184E4          7 6300       oracle    oracle@eygle (CKPT)
.........
```

注意以上输出，pid=1 的进程是一个 PSEUDO 进程，这个进程被认为是初始化数据库的进程，启动其他进程之前即被占用，并在数据库中一直存在。通过 Oracle 的 oradebug 工具可以稍微浏览一下这个进程的信息，以下是简要步骤，转储 SYSTEMSTATE：

```
SQL> connect / as sysdba
Connected.
SQL> oradebug setmypid
Statement processed.
SQL> oradebug dump systemstate 10
Statement processed.
SQL> show parameter user_dump

NAME                                 TYPE  VALUE
------------------------------------ ----------- ------------------------------
user_dump_dest                       string    /t1/orat1/diag/rdbms/t111g/T111g/trace
```

在 user_dump_dest 目录中可以找到进程转储文件,其中进程 1 的信息如下，供读者参考：

```
PROCESS 1:
  ----------------------------------------
  SO: 0x91485a00, type: 2, owner: (nil), flag: INIT/-/-/0x00 if: 0x3 c: 0x3
   proc=0x91485a00, name=process, file=ksu.h LINE:12616, pg=0
  (process) Oracle pid:1, ser:0, calls cur/top: (nil)/(nil)
            flags : (0x20) PSEUDO
            flags2: (0x0),  flags3: (0x10)
            intr error: 0, call error: 0, sess error: 0, txn error 0
            intr queue: empty
    ksudlp FALSE at location: 0
  (post info) last post received: 0 0 0
              last post received-location: No post
              last process to post me: none
              last post sent: 0 0 0
              last post sent-location: No post
              last process posted by me: none
    (latch info) wait_event=0 bits=0
    O/S info: user: , term: , ospid:  (DEAD)
    OSD pid info: Unix process pid: 0, image: PSEUDO
    ----------------------------------------
    SO: 0x6000c838, type: 5, owner: 0x91485a00, flag: INIT/-/-/0x00 if: 0x3 c: 0x3
```

```
    proc=(nil), name=kss parent, file=kss2.h LINE:138, pg=0
  PSO child state object changes :
Dump of memory from 0x0000000091471AA8 to 0x0000000091471CB0
091471AA0                    00000000 00000000          [........]
091471AB0 00000000 00000000 00000000 00000000  [................]
  Repeat 31 times
```

在 v$process 的查询输出中，SPID 列代表的就是操作系统上的进程号，通过 SPID 可以将进程从操作系统到数据库关联起来：

```
[oracle@eygle bdump]$ ps -ef|grep ora_
oracle    6290     1  0 12:42 ?        00:00:00 ora_pmon_eygle
oracle    6292     1  0 12:42 ?        00:00:00 ora_psp0_eygle
oracle    6294     1  0 12:42 ?        00:00:00 ora_mman_eygle
oracle    6296     1  0 12:42 ?        00:00:00 ora_dbw0_eygle
oracle    6298     1  0 12:42 ?        00:00:00 ora_lgwr_eygle
oracle    6300     1  0 12:42 ?        00:00:00 ora_ckpt_eygle
.........
```

如果在操作系统上发现某个进程表现异常（如占用很高的 CPU 资源），那么**通过操作系统上的 PID 和 V$PROCESS 视图中的 SPID 关联**，就可以找到这个 OS 上的进程在数据库内部的化身，从而可以进行进一步的跟踪诊断。

V$PROCESS 视图包含当前数据库中活动进程的相关信息，这些进程在操作系统上都存在与之对应的 OS 进程。其中 LATCHWAIT 列代表进程当前正在等待的 LATCH 信息，LATCHSPIN 则记录进程正在通过 SPIN 进行 LATCH 的竞争。Latch 通常被称为闩，是数据库内部的串行锁机制，主要用来控制内存上的并发，在多处理器系统上，Oracle 进程通过自旋（spin）来进行 Latch 争夺。

这个视图结构如下所示（Oracle 10gR2 版本）：

```
SQL> desc v$process
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------
 ADDR                                               RAW(4)
 PID                                                NUMBER
 SPID                                               VARCHAR2(12)
 USERNAME                                           VARCHAR2(15)
 SERIAL#                                            NUMBER
 TERMINAL                                           VARCHAR2(30)
 PROGRAM                                            VARCHAR2(48)
 TRACEID                                            VARCHAR2(255)
 BACKGROUND                                         VARCHAR2(1)
```

LATCHWAIT	VARCHAR2(8)
LATCHSPIN	VARCHAR2(8)
PGA_USED_MEM	NUMBER
PGA_ALLOC_MEM	NUMBER
PGA_FREEABLE_MEM	NUMBER
PGA_MAX_MEM	NUMBER

注意这里的 ADDR 字段代表的是进程的地址，进程的状态等信息在内存中记录，这个 ADDR 记录的正是这样的内存地址信息。ADDR 在数据库中（甚至是所有软件中）是非常重要的，虽然通常并不会用到，但是深入理解这些知识有助于大家更好的了解 Oracle 数据库。

进程的地址（Address of process）进一步的被缩写为 PADDR，在 **V$SESSION 视图中记录的 PADDR 就是 V$PROCESS.ADDR 的进一步延伸，通过两者关联，可以向数据库进一步深入**。

如果向操作系统端延伸，则 SPID 代表的正是操作系统进程标识符（Operating system process identifier），通过 SPID 和 OS 中看到的进程 PID 关联，就可以建立从操作系统到数据库的关联。

所以 **V$PROCESS 被认为是从操作系统到数据库的入口**。此外，和 PGA 相关的几个字段则记录了进程的 PGA 使用情况。

1.1.5. 从操作系统到数据库

在上一节我们提到，**V$PROCESS 被认为是从操作系统到数据库的入口，而进入数据库内部，进程需要创建会话（Session）执行数据库操作和访问，通常一个进程创建一个会话，特殊情况下，一个进程也可以创建多个数据库会话。会话信息通过视图 V$SESSION 进行体现和管理。**

V$SESSION 视图在不同版本中被不断增强，尤其是自 Oracle 10g 开始，整合了 V$SESSION_WAIT(会话正在发生的等待信息)视图。

我们来看一下，如何完成从系统到数据库内部的导引。以下是一个简单的诊断分析案例，在故障时刻，通过 TOP 工具观察到，有两个操作系统进程占用了极高的 CPU 资源：

```
$ top
load averages:  1.61,  1.28,  1.25          HSWAPJSDB             10:50:44
172 processes: 160 sleeping, 1 running, 3 zombie, 6 stopped, 2 on cpu
CPU states:     % idle,     % user,     % kernel,     % iowait,     % swap
Memory: 4.0G real, 1.4G free, 1.9G swap in use, 8.9G swap free

   PID USERNAME THR PR NCE  SIZE   RES STATE   TIME FLTS    CPU COMMAND
 20521 oracle     1 40   0  1.8G  1.7G run     6:37    0 47.77% oracle
 20845 oracle     1 40   0  1.8G  1.7G cpu02   0:41    0 40.98% oracle
```

```
 20847 oracle     1 58   0  1.8G  1.7G sleep   0:00    0  0.84% oracle
 20780 oracle     1 48   0  1.8G  1.7G sleep   0:02    0  0.83% oracle
 15828 oracle     1 58   0  1.8G  1.7G sleep   0:58    0  0.53% oracle
```

这两个进程分别是 20521 和 20845.通过 PS 命令可以找到这两个进程更为详细的信息：

```
$ ps -ef|grep 20521
  oracle 20909 20875  0 10:50:53 pts/10   0:00 grep 20521
  oracle 20521     1 47 10:43:59 ?        6:45 oraclejshs (LOCAL=NO)
$ ps -ef|grep 20845
  oracle 20845     1 44 10:50:00 ?        0:55 oraclejshs (LOCAL=NO)
  oracle 20918 20875  0 10:50:59 pts/10   0:00 grep 20845
```

这两个 LOCAL=NO 的进程显然是来自应用的远程连接，通过如下脚本，关联 V$PROCESS 视图和 V$SESSION 视图、V$SQLTEXT 视图，可以找出这个进程正在执行的 SQL 语句，这里只需要一个“发动”条件，就是进程号（PID）：

```
SELECT   /*+ ORDERED */
         sql_text
    FROM v$sqltext a
   WHERE (a.hash_value, a.address) IN (
            SELECT DECODE (sql_hash_value,
                           0, prev_hash_value,
                           sql_hash_value
                          ),
                   DECODE (sql_hash_value, 0, prev_sql_addr, sql_address)
              FROM v$session b
             WHERE b.paddr = (SELECT addr
                                FROM v$process c
                               WHERE c.spid = '&pid'))
ORDER BY piece ASC
/
```

注意这里我们涉及了 3 个视图,并应用其关联进行数据获取，核心逻辑分解如下：

1.首先输入一个 pid,这个 pid 即 Process id,也就是在 Top 或 ps 中看到的 PID。

2.通过 pid 和 v$process.spid 相关联我们可以获得 Process 的相关信息。

3.通过 v$process.addr 和 v$session.paddr 相关联,可以获得和 session 相关的所有信息。

4.再结合 v$sqltext,即可获得当前 session 正在执行的 SQL 语句。

可见通过 v$process 视图，我们得以把操作系统和数据库关联了起来。

以 SYSDBA 身份连接到数据库，执行这个 SQL，提供 PID 号，就找到了该进程正在执行的 SQL 代码：

```
SQL> @getsql
Enter value for spid: 20521
```

```
old  10: where c.spid = '&pid'
new  10: where c.spid = '20521'

SQL_TEXT
------------------------------------------------------------------
select * from (select VC2URL,VC2PVDID,VC2MOBILE,VC2ENCRYPTFLAG,S
ERVICEID,VC2SUB_TYPE,CISORDER,NUMGUID,VC2KEY1, VC2NEEDDISORDER,V
C2PACKFLAG,datopertime from hsv_2cpsync where datopertime<=sysda
te and numguid>70000000000308 order by NUMGUid) where rownum<=20
```

那么这段代码就可能是当前正在疯狂消耗 CPU 的罪魁祸首。接下来需要进行的工作就是找出这段代码的问题，看是否可以通过优化提高其效率，减少资源消耗。进一步的我们可以通过 dbms_system 包跟踪该进程，或者通过 AWR 获得该 SQL 的执行计划等。

1.1.6. 参数文件的选择

接下来关注一下启动过程中 Oracle 选择参数文件的顺序。

从 Oracle 9i 开始，spfile 被引入 Oracle 数据库，Oracle 首选 spfile<ORACLE_SID>.ora 文件作为启动参数文件；如果该文件不存在，Oracle 选择 spfile.ora 文件；如果前两者都不存在，Oracle 将会选择 init<ORACLE_SID>.ora 文件；如果以上三个文件都不存在，Oracle 将无法创建和启动 instance。Oracle 在启动过程中，会在特定的路径中寻找参数文件，在 Unix/Linux 下的路径为$ORACLE_HOME/dbs 目录，在 WINDOWS 上的路径为$ORACLE_HOME\database 目录。

可以在 SQL*PLUS 中通过 show parameter spfile 命令来检查数据库是否使用了 spfile 文件，如果 value 不为 Null，则数据库使用了 spfile 文件：

```
SQL> show parameter spfile
NAME    TYPE    VALUE
------- ------- ------------------------------------------------
spfile  string  ?/dbs/spfile@.ora
```

注意这里的"?"代表 ORACLE_HOME，@代表数据库的 sid。现在如果更名 spfile<ORACLE_SID>.ora 文件，此后 Oracle 将选择 spfile.ora 文件启动数据库：

```
[oracle@jumper dbs]$ mv spfileconner.ora spfileconner.ora.bak
SQL> startup nomount
SQL> show parameter spfile
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
spfile                               string      ?/dbs/spfile.ora
```

进一步的如果再更名 spfile.ora 文件，此后 Oracle 将选择 init<ORACLE_SID>.ora 文件启

动数据库:

```
[oracle@jumper dbs]$ mv spfile.ora spfile.ora.bak
SQL> startup nomount
SQL> show parameter spfile
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
spfile                               string
```

如果这三个文件都不存在，Oracle 将无法启动：

```
[oracle@jumper dbs]$ mv initconner.ora initconner.ora.bak
SQL> startup
ORA-01078: failure in processing system parameters
LRM-00109:          could          not          open          parameter          file
'/opt/oracle/product/9.2.0/dbs/initconner.ora'
```

注意这里出现的错误提示，报告无法找到参数文件 init<ORACLE_SID>.ora，这正是 Oracle 在启动过程中最后一个查找的参数文件。

在 Oracle 整个启动过程中，参数文件是写在应用程序中的硬代码，按照前面描述的顺序进行查找，以下是来自源码中关于参数文件及其查找顺序的定义：

?/dbs/spfile@.ora

?/dbs/spfile.ora

?/dbs/init@.ora

虽然不能改变 Oracle 对于参数文件的搜索路径及行为，但是如果参数文件不在相应的位置，在 Linux/Unix 系统上，可以通过符号链接来进行重定位，以满足一些特殊需要（具体请参考第三章内容）。

1.1.7.　实例启动最小参数需求

在参数文件中，通常需要最少的参数是 db_name ，设置了这个参数之后，数据库实例就可以启动，来看一个简单的测试。

可以随意命名一个 ORACLE_SID（测试来自于 Linux 下，适用于 Linux/Unix，对于 Windows 平台，需要通过 oradim.exe 工具创建服务），然后尝试启动到 nomount 状态：

```
[oracle@jumper dbs]$ export ORACLE_SID=julia
[oracle@jumper dbs]$ sqlplus "/ as sysdba"
SQL*Plus: Release 9.2.0.4.0 - Production on Mon May 8 11:08:36 2006
Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.
Connected to an idle instance.

SQL> startup nomount;
```

```
ORA-01078: failure in processing system parameters
LRM-00109: could not open parameter file
'/opt/oracle/product/9.2.0/dbs/initjulia.ora'
```

参数文件查找失败会给出提示信息，此时创建一个最简单的参数文件（仅包含 DB_NAME 初始化参数），然后就可以启动实例：

```
SQL> ! echo "db_name=julia" > /opt/oracle/product/9.2.0/dbs/initjulia.ora
SQL> startup nomount;
ORACLE instance started.

Total System Global Area   97588504 bytes
Fixed Size                   451864 bytes
Variable Size              46137344 bytes
Database Buffers           50331648 bytes
Redo Buffers                 667648 bytes
```

缺省的，如果不设置，background_dump_dest 目录（告警日志文件 alert_<ORACLE_SID>.log 的存放地点）位于$ORACLE_HOME/rdbms/log 目录下：

```
SQL> show parameter background_dump
NAME                                 TYPE                 VALUE
------------------------------------ -------------------- ----------------------
background_dump_dest                 string               ?/rdbms/log
```

顺便看下其他几个缺省路径的地点：

```
SQL> show parameter dump_dest
NAME                                 TYPE                 VALUE
------------------------------------ -------------------- ------------------------
background_dump_dest                 string               ?/rdbms/log
core_dump_dest                       string               ?/dbs
user_dump_dest                       string               ?/rdbms/log
SQL> show parameter control_files
NAME                                 TYPE                 VALUE
------------------------------------ -------------------- ----------------------
control_files                        string               ?/dbs/cntrl@.dbf
```

这样，通过以上步骤就以最少的参数需求启动了 Oracle 实例。

1.1.8. ORACLE_SID 的含义

回顾一下前面的内容可以注意到，SID 和 ORACLE_SID 已经多次出现，那么 SID 是什么？在数据库启动过程中又起到什么作用呢？

SID 是 **S**ystem **ID**entifier 的缩写，而 ORACLE_SID 就是 Oracle System IDentifier 的缩写，

在 Oracle 系统中，ORACLE_SID 以环境变量的形式出现，**当 Oracle 实例启动时，在操作系统上 fork 的进程就依据 ORACLE_SID 这个环境变量来创建**，这就是 SID 的作用。

Oracle 的实例(instance)是由一块共享内存区域(SGA)和一组后台进程(background processes)共同组成，而后台进程正是数据库和操作系统进行交互的通道，这些进程的名称就是通过 ORACLE_SID 决定的。

通过前面的讨论可以知道，实例的启动需要一个参数文件，参数文件的名称就是由 ORACLE_SID 决定的，对于 init 文件，缺省的文件名称是 init<ORACLE_SID>.ora，对于 spfile 文件，缺省的文件名为 spfile<ORACLE_SID>.ora，Oracle 依据 ORACLE_SID 来决定和寻找参数文件启动实例。

在同一个$ORACLE_HOME 下，通过参数文件，Oracle 能够根据 ORACLE_SID 将实例区分开来；但是注意如果在不同的$ORACLE_HOME 下，即使在同一台主机上，Oracle 也是能够创建相同 ORACLE_SID 的实例的。

以下一个测试，首先启动一个 Oracle 8i 下 ORACLE_SID 为 eygle 的实例：

```
$ export ORACLE_SID=eygle
$ sqlplus "/ as sysdba"
SQL*Plus: Release 8.1.7.0.0 - Production on Fri Feb 16 10:23:58 2007
(c) Copyright 2000 Oracle Corporation.  All rights reserved.
Connected to an idle instance.

SQL> startup nomount;
ORACLE instance started.
SQL> ! ps -ef|grep ora_smon_eygle
oracle8 11123 11076  0 10:24:15 pts/1    0:00 grep ora_smon_eygle
oracle8 11092     1  0 10:24:02 ?        0:00 ora_smon_eygle
```

接下来又可以启动另外$ORACLE_HOME 下 ORACLE_SID 为 eygle 的实例：

```
$ export ORACLE_SID=eygle
$ sqlplus "/ as sysdba"
SQL*Plus: Release 9.2.0.4.0 - Production on Fri Feb 16 10:24:43 2007
Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.
Connected to an idle instance.

SQL> startup nomount;
ORACLE instance started.
SQL> ! ps -ef|grep ora_smon_eygle
oracle9 11214 11172  0 10:24:58 pts/1    0:00 grep ora_smon_eygle
 oracle8 11092     1  0 10:24:02 ?        0:00 ora_smon_eygle
 oracle9 11188     1  0 10:24:48 ?        0:00 ora_smon_eygle
```

现在这同一台主机上就启动了两个相同名称的实例，在操作系统上，Oracle 能够通过 ID 标示将共享内存或信号量区分开来：

```
$ ipcs -i
IPC status from <running system> as of Fri Feb 16 10:30:02 CST 2007
T         ID      KEY        MODE        OWNER    GROUP
Message Queues:
q          0   0x2e781d5  --rw-r--r--     root     root
T         ID      KEY        MODE        OWNER    GROUP ISMATTCH
Shared Memory:
m       4096   0xabdc9b64 --rw-r-----  oracle8      dba       12
m       1025   0x79552064 --rw-r-----  oracle9      dba       11
Semaphores:
s    1245184   0x79978bac --ra-r-----  oracle8      dba
s     458753   0xa0e9f594 --ra-r-----  oracle9      dba
```

通过 Oracle 提供的一个小工具 sysresv，我们可以找到对应于不同的 ORACLE_SID，操作系统上创建的共享内存段 ID(Shared Memory)和信号量 ID(Semaphores)等信息:

```
$ sysresv -l eygle julia
IPC Resources for ORACLE_SID "eygle" :
Shared Memory:
ID              KEY
2560            0x79552064
Semaphores:
ID              KEY
720896          0xa0e9f594
Oracle Instance alive for sid "eygle"
IPC Resources for ORACLE_SID "julia" :
Shared Memory:
ID              KEY
514             0xab281214
Semaphores:
ID              KEY
196610          0xa7645a54
Oracle Instance alive for sid "julia"
```

1.1.9. INSTANCE_NAME 的含义

在数据库内部和 ORACLE_SID 相关联的概念就是 INSTANCE_NAME。

Oracle **数据库内部存在一个初始化参数 INSTANCE_NAME，用于标示数据库实例的名称，其缺省值通常就是 ORACLE_SID**；但是初始化参数 INSTANCE_NAME 和

ORACLE_SID 可以不同，不同实例可以拥有相同的 INSTANCE_NAME。

在同一个 ORACLE_HOME 下，只要 ORACLE_SID 不同，数据库并不校验 INSTANCE_NAME 参数；通过简单的参数文件复制，我们就可以在同一台服务器上创建多个具有相同 instance_name 的实例（注意以下测试来自 Oracle 9i 数据库）：

```
bash-2.03$ cd $ORACLE_HOME/dbs
bash-2.03$ cp initeygle.ora initjulia.ora
bash-2.03$ export ORACLE_SID=julia
bash-2.03$ sqlplus "/ as sysdba"
SQL> startup nomount;
ORACLE instance started.
```

此时同一主机上就可以启动多个实例，ORACLE_SID 不同，但是拥有了相同的 instance_name：

```
SQL> show parameter instance_name
NAME                                 TYPE                 VALUE
------------------------------------ -------------------- ----------------------
instance_name                        string               eygle
```

但是注意，在数据库内部视图 **V$INSTANCE 中也记录着一个 INSTANCE_NAME，这个 INSTANCE_NAME 来自数据库实例的 SID，始终和 ORACLE_SID 保持一致**，这就可能出现数据库中这两个 INSTANCE_NAME 不一致的情况：

```
SQL> select instance_name from v$instance;
INSTANCE_NAME
--------------------------------
julia
SQL> show parameter instance_name
NAME                                 TYPE                 VALUE
------------------------------------ -------------------- ------------------
instance_name                        string               eygle
```

存在这种歧义是因为在 Oracle 9i 中，当创建数据库进行相关配置时，数据库将 INSTANCE_NAME 参数写入了参数文件，这就导致了当修改参数文件名称变更 ORACLE_SID 时可能并不修改 INSTANCE_NAME 参数的情况；值得注意的是，从 Oracle 10g 开始，参数文件中缺省不再记录 INSTANCE_NAME，此时 INSTANCE_NAME 可以动态从系统获得，从而消除了以前可能常见的歧义：

```
D:\oracle\product>grep instance_name 9.2.0\database\SPFILEEEYGLE.ORA
*.instance_name='eeygle'
D:\oracle\product>grep instance_name 10.2.0\database\SPFILEEYGLE.ORA
D:\oracle\product>grep instance_name 11.1.0\database\SPFILEEYGLEE.ORA
```

INSTANCE_NAME 除了用来标识实例名称之外，在监听器动态注册时还会用于向监听器注册。比如在一个数据库 db_name=julia，instance_name=eygle 的数据库中，监听器动态注册会包含如下信息，这里的 Instance 内容就来自 INSTANCE_NAME 参数设置：

```
Services Summary...
Service "julia" has 1 instance(s).
  Instance "eygle", status READY, has 1 handler(s) for this service...
```

V$INSTANCE 视图和数据库实例的生命周期相关，用于显示当前实例的状态，通过这个视图可以获得包括实例的启动时间、运行主机等重要信息，通过以下一段 SQL 可以获得数据库的 UPTIME 信息：

```
SQL> COLUMN STARTED_AT format a25
SQL> COLUMN UPTIME format a50
SQL> SELECT TO_CHAR (startup_time, 'DD-MON-YYYY HH24:MI:SS') started_at,
  2       TRUNC (SYSDATE - (startup_time))
  3   || ' day(s), ' || TRUNC (  24 * ((SYSDATE - startup_time) -
  4   TRUNC (SYSDATE - startup_time)))
  5   || ' hour(s), '|| MOD (TRUNC (  1440 * (  (SYSDATE - startup_time) -
  6   TRUNC (SYSDATE - startup_time))),60)
  7   || ' minute(s), '|| MOD (TRUNC (  86400 * (  (SYSDATE - startup_time) -
  8   TRUNC (SYSDATE - startup_time))),60)
  9   || ' seconds' uptime
  10  FROM v$instance;
STARTED_AT                UPTIME
------------------------- --------------------------------------------------
05-JUL-2005 10:36:58      803 day(s), 2 hour(s), 27 minute(s), 55 seconds
```

1.1.10. DB_NAME 与 INSTANCE_NAME

相较 INSTANCE_NAME 参数来说，对于 Oracle 数据库更为重要的一个参数是 DB_NAME。**DB_NAME 代表了实例即将挂接的数据库名称，关系到具体的物理文件**。通常缺省的数据库 instance_name 和 db_name 可以设置相同（在 RAC 环境下，由于多个实例对应一个数据库，所以 instance_name 和 db_name 不同）。

在创建数据库的过程中，图 1-3 是用于定义数据库名称（db_name）和影响 INSTANCE_NAME 的 SID：

图 1-0-3 数据库创建的标识定义页面

Oracle 文档中对于 db_name 的定义如下：

> DB_NAME 用来定义数据库名称，必须是一个不超过 8 个字符的文本串，在数据库创建过程中，db_name 被记录在数据文件，日志文件和控制文件中。如果数据库实例启动过程中参数文件中的 db_name 和控制文件中的数据库名称不一致，则数据库不能启动。

此外常见的几个结论有：

1. 一个实例可以 mount 并打开任何数据库，但是同一时间一个实例只能打开一个数据库
2. 一个数据库可以被一个或多个实例所 mount 并打开（在 OPS/RAC 环境下，一个数据库可以被多个实例所打开）。

DB_NAME 的另外一个作用是在监听器动态注册时作为缺省服务名注册，以下是 Oracle 10g 的动态注册监听示范：

```
Services Summary...
Service "julia" has 1 instance(s).
  Instance "eygle", status READY, has 1 handler(s) for this service...
```

通过下面的测试来看一下 DB_NAME 与数据库的关系。首先 initeygle.ora 文件代表了一个数据库实例：

```
[oracle@jumper oracle]$ cd $ORACLE_HOME/dbs
[oracle@jumper dbs]$ grep name initeygle.ora
*.db_name='eygle'
```

```
*.instance_name='eygle'
```

这个实例以及当前数据库的相关参数如下：

```
SQL> show parameter db_name
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_name                              string      eygle
SQL> show parameter instance_name
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
instance_name                        string      eygle
```

现在创建另外一个实例，通过复制创建一个 pfile 文件为名为 julia 这个新的实例使用：

```
[oracle@jumper oracle]$ cd $ORACLE_HOME/dbs
[oracle@jumper dbs]$ cp initeygle.ora initjulia.ora
[oracle@jumper dbs]$ ll init*
-rw-r--r--    1 oracle   dba           982 Jul 25 14:03 initeygle.ora
-rw-r--r--    1 oracle   dba           982 Jul 25 14:04 initjulia.ora
```

修改这个文件，更改 instance_name 参数，设置 instance_name = julia，修改后的参数设置如下所示:

```
[oracle@jumper dbs]$ grep name initjulia.ora
*.db_name='eygle'
*.instance_name='julia'
```

现在来启动这个实例名称为 julia 的 instance：

```
[oracle@jumper dbs]$ export ORACLE_SID=julia
[oracle@jumper dbs]$ sqlplus "/ as sysdba"
SQL> startup mount;
ORACLE instance started.

Total System Global Area  139531744 bytes
Fixed Size                   452064 bytes
Variable Size             121634816 bytes
Database Buffers           16777216 bytes
Redo Buffers                 667648 bytes
ORA-01102: cannot mount database in EXCLUSIVE mode
```

注意，当试图加载数据库时出现错误，因为当前数据库被另外一个实例(instance)加载。在非并行模式(OPS/RAC)下，一个数据库同时只能被一个实例加载。

此时已经启动了两个数据库实例，从后台进程可以看出：

```
[oracle@jumper dbs]$ ps -ef|grep dbw
oracle   27323     1  0 Jul14 ?        00:00:00 ora_dbw0_eygle
oracle   15447     1  0 14:04 ?        00:00:00 ora_dbw0_julia
oracle   25030 25000  0 18:38 pts/2    00:00:00 grep dbw
```

关闭 eygle 这个数据库实例:

```
[oracle@jumper dbs]$ export ORACLE_SID=eygle
[oracle@jumper dbs]$ sqlplus "/ as sysdba"
SQL> shutdown immediate;
```

然后就可以通过实例 julia 加载并打开 db_name=eygle 的数据库了,这也就是前面所说的,一个数据库可以被任何一个实例挂接打开（当然是有条件限制的）:

```
[oracle@jumper dbs]$ export ORACLE_SID=julia
[oracle@jumper dbs]$ sqlplus "/ as sysdba"
SQL> alter database mount;
alter database mount
*
ERROR at line 1:
ORA-01990: error opening password file '/opt/oracle/product/9.2.0/dbs/orapw'
ORA-27037: unable to obtain file status
Linux Error: 2: No such file or directory
Additional information: 3
SQL> alter database open;
Database altered.
SQL> select name from v$datafile;
NAME
--------------------------------------------------------------------------------
/opt/oracle/oradata/eygle/system01.dbf
/opt/oracle/oradata/eygle/undotbs01.dbf
/opt/oracle/oradata/eygle/users01.dbf
/opt/oracle/oradata/eygle/eygle01.dbf
SQL> show parameter instance_name
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
instance_name                        string      julia
SQL> show parameter db_name
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_name                              string      eygle
```

进一步的，再来研究一下如果参数文件中的 db_name 和控制文件中的 db_name 不一致会出现什么错误。

修改参数文件中的 db_name 参数：

```
[oracle@jumper dbs]$ grep name initjulia.ora
*.db_name='julia'
*.instance_name='julia'
```

在 nomount 环节不存在任何问题，而在 mount 阶段，数据库会对参数文件和控制文件进行比较，如果两者记录的 db_name 不一致，则数据库无法启动，错误提示指定的数据库名称和控制文件中记录的名称不符：

```
SQL> startup nomount;
SQL> alter database mount;
alter database mount
*
ERROR at line 1:
ORA-01103: database name 'EYGLE' in controlfile is not 'JULIA'
```

1.1.11. RMAN 的缺省实例

在使用 RMAN（Recovery Manager）时存在更为特殊的情况，Oracle 允许在不存在参数文件的情况下启动一个实例，数据库的 db_name 会被缺省的命名为 DUMMY，这是最为极端的情况，在某些恢复过程中，这个功能可以帮助我们减少很多麻烦：

```
[oracle@jumper dbs]$ rman target /
Recovery Manager: Release 9.2.0.4.0 - Production
Copyright (c) 1995, 2002, Oracle Corporation.  All rights reserved.
connected to target database (not started)
RMAN> startup  nomount;
startup failed: ORA-01078: failure in processing system parameters
LRM-00109: could not open parameter file '/opt/oracle/product/9.2.0/dbs/initconner.ora'
trying to start the Oracle instance without parameter files ...
Oracle instance started
Total System Global Area      97588504 bytes
Fixed Size                      451864 bytes
Variable Size                 46137344 bytes
Database Buffers              50331648 bytes
Redo Buffers                    667648 bytes

RMAN> host ;
[oracle@jumper dbs]$ sqlplus "/ as sysdba"
SQL*Plus: Release 9.2.0.4.0 - Production on Tue Mar 12 14:17:07 2006
```

```
Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.
SQL> show parameter db_name
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_name                              string      DUMMY
```

此时警告日志文件中会记录如下信息：

```
Starting up ORACLE RDBMS Version: 9.2.0.4.0.
System parameters with non-default values:
  remote_login_passwordfile= EXCLUSIVE
  db_name                  = DUMMY
PMON started with pid=2
DBW0 started with pid=3
…..
```

总结一下，数据库的 Nomount 过程实质上就是在创建实例，这个步骤只和参数文件相关，在完成实例的创建之后，Oracle 就可以逐步导航，完成数据库的加载，打开等工作。

1.1.12.　Nomount 案例两则

在创建数据库时，如果在这一步骤就出现问题,那么通常可能是系统配置（如内核参数等）存在问题，你需要检查是否分配了足够的系统资源等。

以下是一个启动到 nomount 状态可能会遇到的常见错误：

```
$ export ORACLE_SID=julia
$ sqlplus "/ as sysdba"
SQL*Plus: Release 9.2.0.4.0 - Production on Wed Feb 28 09:55:24 2007
Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.
Connected to an idle instance.

SQL> startup nomount;
ORA-00600: internal error code, arguments: [OSDEP_INTERNAL], [], [], [], [], [], [],
[]
ORA-27302: failure occurred at: skgpwreset1
ORA-27303: additional information: invalid shared ctx
ORA-27146: post/wait initialization failed
ORA-27300: OS system dependent operation:semget failed with status: 28
ORA-27301: OS failure message: No space left on device
ORA-27302: failure occurred at: sskgpsemsper
```

（注意：ORA-00600 是 Oracle 内部错误的一个集合，其具体含义要看后面的参数提示，数据库出现 ORA-00600 错误应当引起 DBA 的充分重视，很多 600 错误可能会导致数据损失。）

在 Nomount 状态就出现问题，通常是系统问题，OS 类错误一般说明是系统资源不足，这在 Linux/Unix 下和信号量等参数设置有关，多出现在同一主机运行多个数据库实例的情况（在 Solaris 上需要修改/etc/system 文件中的内核参数，重起系统后修改生效）。在这个错误提示中，600 错误的第一个参数是 **OSDEP_INTERNAL**，我们大致可以猜测到这是一个 OS Dependent/Internal Error。很多 Oracle 的提示可以根据缩写猜到大致的含义，但是如果是错误号那就要依赖 Oracle 的文档来寻找答案。

此外错误提示 OS 相关的操作是：semget 。这是一个标准的操作系统调用，通过手册可以获得其含义信息，可以看到这是和信号量相关的系统调用：

```
bogon:Eygle eygle$ man semget

SEMGET(2)                  BSD System Calls Manual                  SEMGET(2)

NAME
     semget -- obtain a semaphore id

SYNOPSIS
     #include <sys/sem.h>

     int
     semget(key_t key, int nsems, int semflg);
```

具体就可以判断，问题和操作系统的内核参数设定有关，可能是信号量设置不足导致的，根据不同的操作系统找到相应的参数，调整之后即可解决问题。

这个案例给我们的提示是：**应该认真细致的阅读每一行错误提示信息，往往从直接的提示就可以找到真实的错误原因。**

在另外一个客户现场，遭遇过另外一个案例，当时客户的服务器异常断电，当系统重新启动后，数据库无法启动（提示：**重启主机对于 DBA 来说应当极其慎重，很多隐藏的故障可能在重启时爆发出来，在没有做好充分 之前，不要贸然从事**）。

数据库的症状是，启动主机到 Nomount 状态后，后台进程会立即将实例中止，也就是说数据库实例都无法稳定创建，告警日志文件信息如下：

```
Mon Dec  3 14:24:30 2007
Errors in file /oraclehx/app/admin/sxlss/bdump/sxlss_pmon_360454.trc:
ORA-07445: exception encountered: core dump [] [] [] [] [] []
PSP0 started with pid=3, OS id=422106
MMAN started with pid=4, OS id=303332
DBW0 started with pid=5, OS id=299324
```

```
.........
SMON started with pid=11, OS id=278882
RECO started with pid=12, OS id=319898
CJQ0 started with pid=13, OS id=295404
MMON started with pid=14, OS id=303428
MMNL started with pid=15, OS id=438776
Mon Dec  3 14:24:33 2007
PSP0: terminating instance due to error 472
Instance terminated by PSP0, pid = 422106
```

综合前面介绍的知识，如果实例都无法创建，那通常是在 OS 方面存在问题，这些问题在系统重新启动后才体现出来，经过检查，发现客户系统是 AIX 操作系统补丁应用不完全，最后导致了数据库无法启动，应用完整的系统补丁后，数据库恢复正常：

```
instfix -i|grep ML
All filesets for 5.3.0.0_AIX_ML were found.
All filesets for 5300-01_AIX_ML were found.
All filesets for 5300-02_AIX_ML were found.
All filesets for 5300-03_AIX_ML were found.
All filesets for 5300-04_AIX_ML were found.
All filesets for 5300-05_AIX_ML were found.
Not all filesets for 5300-06_AIX_ML were found.
```

这个案例给我们的经验是：**当进行 OS 补丁应用时一定要认真确认，对关键补丁应当进行服务器重启验证，不能掉以轻心**。

1.2启动数据库到 mount 状态

实例启动到 nomount 状态以后，Oracle 就可以从参数文件中获得控制文件的位置信息，然后找到控制文件，并且根据控制文件中记录的数据文件位置进行数据文件的存在性判断。

1.2.1 控制文件的定位

在 Oracle 10g 之前，通常 Oracle 缺省的会创建 3 个控制文件，这三个控制文件的内容完全一致，是 Oracle 为了安全而采用的镜像手段，在生产环境中，通常我们应该将控制文件存放在不同的物理硬盘上，避免因为介质故障而同时损坏三个控制文件。

控制文件信息在参数文件中记录类似如下所示：

```
*.control_files='/opt/oracle/oradata/conner/control01.ctl',
                '/opt/oracle/oradata/conner/control02.ctl',
                '/opt/oracle/oradata/conner/control03.ctl'
```

从 Oracle 10g 开始，如果设置了闪回恢复区（Flashback Recovery Area，通常闪回区和数

据区位于不同硬盘存储），则 Oracle 缺省的就会将控制文件分布到不同的磁盘组，至此 Oracle 才算完成了控制文件的真正镜像安全保护，以下是 Oracle 10g 中的一个输出示范：

```
SQL> show parameter control_files
NAME            TYPE        VALUE
--------------- ----------- ------------------------------------------------
control_files   string      +ORADG/smsdb/controlfile/current.256.642339925,
                            +FLHDG/smsdb/controlfile/current.256.642339925
```

在 nomount 状态，可以查询 v$parameter 视图，获得控制文件信息，这部分信息来自启动的参数文件；当数据库 mount 之后，可以查询 v$controlfile 视图获得关于控制文件的信息，此时，这部分信息来自控制文件：

```
SQL> startup nomount;
SQL> select * from v$controlfile;
no rows selected
SQL> show parameter control_files
NAME                TYPE              VALUE
------------------ --------------- ---------------------------------------
control_files       string           /opt/oracle/oradata/conner/control01.ctl,
                                    /opt/oracle/oradata/conner/control02.ctl,
                                    /opt/oracle/oradata/conner/control03.ctl
SQL> alter database mount;
Database altered.
SQL> select * from v$controlfile;
STATUS  NAME
------- ----------------------------------------
       /opt/oracle/oradata/conner/control01.ctl
       /opt/oracle/oradata/conner/control02.ctl
       /opt/oracle/oradata/conner/control03.ctl
```

在 mount 数据库的过程中，Oracle 需要找到控制文件，锁定控制文件。如果控制文件全部丢失此时就会报出如下错误：

```
ORA-00205: error in identifying controlfile, check alert log for more info
```

这时候告警日志文件中通常会记录更为详细的信息：

```
ORA-00202: controlfile: '/opt/oracle/oradata/conner/control01.ctl'
ORA-27037: unable to obtain file status
Linux Error: 2: No such file or directory
Additional information: 3
```

因为 Oracle 的三个(缺省的)控制文件内容完全相同，如果只是损失了其中 1~2 个，可以复

制完好的控制文件，更改为相应的名称，就可以启动数据库；如果丢失了所有的控制文件，那么就需要恢复或重建控制文件来打开数据库。

1.2.2 数据文件的存在性判断

在启动了实例之后，实际上数据库的后台进程已经运行，那么当进一步的 Mount 数据库之后，后台进程就可以根据控制文件中记录的数据文件信息来验证数据文件是否存在，如果数据文件不存在，则后台进程将在告警日志文件中记录文件缺失信息，并且在动态视图中记录这些信息。

对以下数据库进行一个简单测试：

```
SQL> select name from v$datafile;
NAME
--------------------------------------------------------------------------------
/opt/oracle/oradata/eygle/system01.dbf
/opt/oracle/oradata/eygle/undotbs01.dbf
/opt/oracle/oradata/eygle/eygle01.dbf
/opt/oracle/oradata/eygle/users.dbf
```

通过以下步骤，移除一个测试文件：

```
SQL> shutdown immediate;
SQL> ! mv /opt/oracle/oradata/eygle/eygle01.dbf /opt/oracle/oradata/eygle/eygle01.dbf.b
SQL> startup mount;
```

此时检查告警日志文件，则可以发现数据文件的缺失信息：

```
[oracle@jumper bdump]$ tail -20 alert_eygle.log
ALTER DATABASE   MOUNT
Mon Sep 15 21:21:01 2008
Successful mount of redo thread 1, with mount id 1484653049.
Mon Sep 15 21:21:01 2008
Database mounted in Exclusive Mode.
Completed: ALTER DATABASE   MOUNT
Mon Sep 15 21:21:31 2008
Errors in file /opt/oracle/admin/eygle/bdump/eygle_dbw0_17074.trc:
ORA-01157: cannot identify/lock data file 3 - see DBWR trace file
ORA-01110: data file 3: '/opt/oracle/oradata/eygle/eygle01.dbf'
ORA-27037: unable to obtain file status
Linux Error: 2: No such file or directory
Additional information: 3
Mon Sep 15 21:21:41 2008
Errors in file /opt/oracle/admin/eygle/bdump/eygle_dbw0_17074.trc:
ORA-01157: cannot identify/lock data file 3 - see DBWR trace file
ORA-01110: data file 3: '/opt/oracle/oradata/eygle/eygle01.dbf'
```

```
ORA-27037: unable to obtain file status
Linux Error: 2: No such file or directory
Additional information: 3
```

此时查询数据的动态视图v$recover_file可以发现数据库记录了FILE NOT FOUND的错误信息：

```
SQL> select * from v$recover_file;
     FILE# ONLINE  ONLINE_ ERROR                          CHANGE# TIME
---------- ------- ------- ------------------------------ ---------- ---------
         3 ONLINE  ONLINE  FILE NOT FOUND                         0
SQL> select name from v$datafile where file#=3;
NAME
--------------------------------------------------------------------------------
/opt/oracle/oradata/eygle/eygle01.dbf
```

不过在较新版本（比如 Oracle 11gR2）中，启动数据库到 Mount 状态时，在告警日志文件中不再提示数据文件缺失信息。

1.2.3控制文件的 HeartBeat

在正常 Mount 数据库的过程中，数据库的警报日志文件仅记录如下信息：

```
alter database mount
Sat Apr 29 10:20:42 2006
Successful mount of redo thread 1, with mount id 1408096182.
Sat Apr 29 10:20:42 2006
Database mounted in Exclusive Mode.
Completed: alter database mount
```

在这一步骤中，数据库需要计算 Mount id 并将其记录在控制文件中，然后开始启动心跳（heartbeat），每 3 秒更新一次控制文件。可以用以下命令间隔 3 秒转储 2 次控制文件信息：

alter session set events 'immediate trace name CONTROLF level 8' ;

在 Linux 上用 diff 命令比较 2 个文件可以发现，控制文件在 Mount 状态下发生改变的只有这个 Heartbeat:

```
[oracle@jumper udump]$ diff conner_ora_25542.trc conner_ora_25706.trc
...
64c63
< heartbeat: 588983634 mount id: 1408096182
---
> heartbeat: 588983636 mount id: 1408096182
```

Heartbeat 表明实例已经被特定例程所 Mount，这个属性主要用于 OPS/RAC 环境。但是

Heartbeat 在单实例环境中同样存在。可以从一个内部表（需要以 SYS 用户登录）中查询到当前的 Heartbeat 值（X$KCCCP 的含义为[**K**]ernel [**C**]ache [**C**]ontrolfile management [**C**]heckpoint [**P**]rogress）：

SELECT CPHBT from X$KCCCP;

从 Oracle 9i 开始，Oracle 在数据库内部通过等待事件 control file heartbeat 来记录这个事件的相关等待；在 Oracle 10g 中，如果使用自动存储管理技术（ASM - Automatic Storage Management），那么还会增加一个 ASM 实例的心跳事件；以下输出来自 Oracle 10g：

```
SQL> select event#,name
  2  from v$event_name where name like '%heart%';
    EVENT# NAME
---------- ----------------------------------------------------------------
       282 ASM mount : wait for heartbeat
       423 control file heartbeat
```

了解了启动的各个步骤，我们也就可以在发生问题的时候，快速定位，准确判断，从而快速解决问题。

1.2.4口令文件的作用

在 Oracle 10g 之前，启动到 Mount 状态，数据库需要具备的另外一个重要文件是口令文件，在 Unix/Linux 上，该文件位于$ORACLE_HOME/dbs 目录下，缺省的名称为 orapw<ORACLE_SID>。而在 Windows 上，文件位于%ORACLE_HOME%\database 目录下，缺省的名称为 PWD<ORACLE_SID>.ora，注意这二者的区别，还要注意 ORACLE_SID 的大小写。

口令文件中存放 SYSDBA/SYSOPER 用户的用户名及口令，在 Unix/Linux 下，通过 strings 命令可以将口令文件中的文本串提出出来，以下是一个口令文件中的字符内容（来自 Oracle 9iR2 环境）：

```
[oracle@jumper dbs]$ strings orapwconner
]\[Z
ORACLE Remote Password file
CONNER
INTERNAL
AB27B53EDC5FEF41
8A8F025737A9097A
EYGLE
B726E09FE21F8E83
```

口令文件可以用于对具备 SYSDBA/SYSOPER 身份的用户进行登录认证，如果不存在该文件，则远程用户将无法使用这两个身份登录数据库。在 Oracle 9i 中，口令文件在数据库的 Mount 阶段被检测。

如果丢失了口令文件，在 mount 阶段就会出现错误，给出提示:

```
SQL> alter database mount;
alter database mount
*
ERROR at line 1:
ORA-01990: error opening password file '/opt/oracle/product/9.2.0/dbs/orapw'
ORA-27037: unable to obtain file status
Linux Error: 2: No such file or directory
Additional information: 3
```

对于口令文件，Oracle 缺省查找 orapw<ORACLE_SID>文件，如果该文件不存在，则继续查找 orapw 文件，如果两者都不存在，则数据库将会出现错误（虽然会出现错误提示，数据库仍然可以继续打开）。如果口令文件丢失，可以通过 orapw 工具即可重建，所以在通常的备份策略中可以不必包含口令文件：

```
[oracle@jumper dbs]$ orapwd
Usage: orapwd file=<fname> password=<password> entries=<users>

  where
    file - name of password file (mand),
    password - password for SYS (mand),
    entries - maximum number of distinct DBA and OPERs (opt),
  There are no spaces around the equal-to (=) character.
```

初始化参数 remote_login_passwordfile 和口令文件的使用有关。

从 Oracle 10g 开始，当口令文件丢失后，在启动过程中，Oracle 将不再提示错误，只是和口令文件相关的部分功能将无法使用。比如之后进行 SYSDBA 的授权或者尝试远程通过 SYSDBA 身份登录都会出现错误：

```
SQL> connect sys/oracle@eygle as sysdba
ERROR:
ORA-01031: insufficient privileges
Warning: You are no longer connected to ORACLE.
```

以下是丢失口令文件的授权示例，系统将提示无法找到口令文件的错误：

```
SQL> grant sysdba to test;
grant sysdba to test
*
ERROR at line 1:
ORA-01994: GRANT failed: password file missing or disabled
```

数据库里具有 SYSDBA/SYSOPER 权限的用户可以通过 v$pwfile_users 视图查询得到。

1.2.5 lk<ORACLE_SID>文件及作用

通常在 Linux/Unix 平台下，在$ORACLE_HOME/dbs 目录下，还会存在另外一个文件，该文件命名规则为 lk<ORACLE_SID>，lk 指 lock，该文件在数据库启动时创建，用于操作系统对数据库的锁定。当数据库启动时获得锁定，数据库关闭时释放。

有时在系统出现异常时，可能数据库已经关闭，但是锁定并未释放，或者因为后台进程未正常停止等原因，会导致下次数据库无法启动，一次相关案例的错误信息引用如下：

```
Sun Apr 30 06:08:58 2006
ALTER DATABASE   MOUNT
Sun Apr 30 06:08:58 2006
scumnt: failed to lock /export/product/oracle/app/dbs/lkBILL exclusive
Sun Apr 30 06:08:58 2006
ORA-09968: scumnt: unable to lock file
SVR4 Error: 11: Resource temporarily unavailable
Additional information: 20169
```

如果遇到这种情况，可以通过重启服务器或者手工释放共享内存段等方法来释放锁定。该文件内容通常只有一行，提示不要删除，该文件仅仅用于锁定：

```
bash-2.03$ more lkBILL
DO NOT DELETE THIS FILE!
```

1.2.6Mount 相同 db_name 的数据库

通过前面的讨论我们知道，在同一台数据库服务器上，可以启动多个具有相同实例名称的实例，那么在同一主机上是否可以启动具有相同 db_name 的数据库呢？如果可以，我们就可以在同一台服务器上 Clone 出一个备用数据库（Standby Database），用于测试或基于时间点的恢复等操作。

答案当然是可以的，但是在同一主机打开相同 DB_NAME 的数据库需要设置另外一个参数，在 Oracle 9i 中这个参数是 lock_name_space，而在 Oracle 10g 中引入了 db_unique_name 参数替代了原来的 lock_name_space。

以下测试来自 Oracle 9iR2，首先获得原数据库的控制文件创建脚本：

```
SQL> alter database backup controlfile to trace;
Database altered.
```

然后关闭初始数据库（此处是 ORACLE_SID 为 eygle 的数据库），意图复制为目标 ORACLE_SID 是 julia 的数据库。

```
[oracle@jumper oradata]$ cp -R eygle julia
[oracle@jumper oradata]$ ls -l
drwxr-xr-x    2 oracle   dba          4096 Feb 16 11:03 eygle
drwxr-xr-x    2 oracle   dba          4096 Feb 16 13:15 julia
```

为新实例创建口令及参数文件：

```
[oracle@jumper dbs]$ ll *eygle*
-rw-r--r--    1 oracle   dba           898 Feb 16 13:10 initeygle.ora
-rwSr-----    1 oracle   dba          1536 Jan 13 19:39 orapweygle
-rw-r-----    1 oracle   dba          2560 Feb 16 13:10 spfileeygle.ora
[oracle@jumper dbs]$ orapwd file=orapwjulia password=oracle entries=5
[oracle@jumper dbs]$ cp spfileeygle.ora spfilejulia.ora
```

更改参数：

```
[oracle@jumper dbs]$ export ORACLE_SID=julia
[oracle@jumper dbs]$ sqlplus "/ as sysdba"
SQL> startup nomount;
SQL> show parameter control
NAME                           TYPE    VALUE
------------------------------ ------- ------------------------------------------
control_file_record_keep_time  integer 7
control_files                  string  /opt/oracle/oradata/eygle/control01.ctl
SQL>   alter   system   set   control_files='/opt/oracle/oradata/julia/control01.ctl'
scope=spfile;
System altered.
SQL> alter system set lock_name_space=julia scope=spfile;
System altered.
```

关闭实例，重新创建控制文件：

```
SQL> startup nomount;
SQL> CREATE CONTROLFILE REUSE DATABASE "EYGLE" NORESETLOGS  ARCHIVELOG
  2  --  SET STANDBY TO MAXIMIZE PERFORMANCE
  3      MAXLOGFILES 5
  4      MAXLOGMEMBERS 3
  5      MAXDATAFILES 100
  6      MAXINSTANCES 1
  7      MAXLOGHISTORY 1134
  8  LOGFILE
  9    GROUP 3 '/opt/oracle/oradata/julia/redo03.log'  SIZE 1M,
 10    GROUP 4 '/opt/oracle/oradata/julia/redo04.dbf'  SIZE 1M,
 11    GROUP 5 '/opt/oracle/oradata/julia/redo05.dbf'  SIZE 1M
 12  -- STANDBY LOGFILE
 13  DATAFILE
 14    '/opt/oracle/oradata/julia/system01.dbf',
 15    '/opt/oracle/oradata/julia/undotbs01.dbf',
```

```
 16    '/opt/oracle/oradata/julia/eygle01.dbf'
 17  CHARACTER SET ZHS16GBK;
Control file created.
SQL> alter database open;
Database altered.
SQL> select name from v$datafile;
NAME
--------------------------------------------------------------------------------
/opt/oracle/oradata/julia/system01.dbf
/opt/oracle/oradata/julia/undotbs01.dbf
/opt/oracle/oradata/julia/eygle01.dbf
```

关闭数据库，修改原实例 eygle 的参数：

```
SQL> alter system set lock_name_space=eygle scope=spfile;
System altered.
```

启动 eygle 数据库：

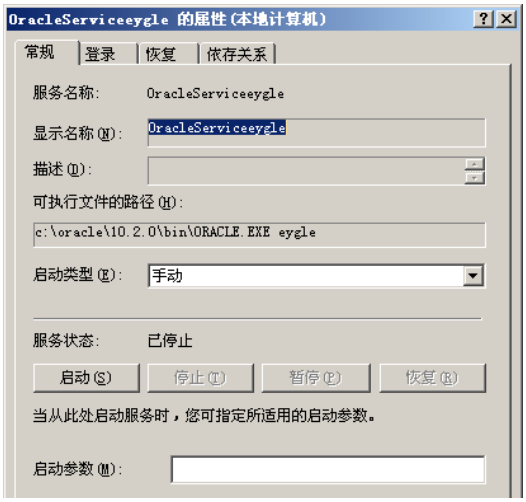
```
[oracle@jumper udump]$ export ORACLE_SID=eygle
[oracle@jumper udump]$ sqlplus '/ as sysdba'
SQL> startup
Database opened.
```

接下来启动 julia 实例：

```
[oracle@jumper udump]$ export ORACLE_SID=julia
[oracle@jumper udump]$ sqlplus '/ as sysdba'
SQL> startup
Database opened.
```

这样在同一台服务器上，就同时拥有了 db_name 和 instance_name 完全相同的两个数据库，当然，如果要真正使用这两个数据库，参数文件中还有一些路径需要修改。

在 Windows 上，情况与 Linux/Unix 上类似，Windows 上的 Oracle 环境依赖于服务而存在：



我们注意到 Oracle 环境的初始化是通过 ORACLE.EXE eygle 来完成的，至于实例和数据库是否随服务启动要依赖于注册表中的设置。

通过手动在命令行执行类似命令可以初始化任意的 Oracle 环境：

```
C:\>oracle julia
```

```
Press CTRL-C to exit server:
```

此后就可以连接到这个环境启动实例：

```
C:\>set ORACLE_SID=julia
C:\>sqlplus "/ as sysdba"
SQL*Plus: Release 10.2.0.1.0 - Production on 星期六 2月 17 10:11:13 2007
Copyright (c) 1982, 2005, Oracle.  All rights reserved.
已连接到空闲例程。
SQL> startup nomount;
ORA-01078: failure in processing system parameters
LRM-00109: could not open parameter file 'C:\ORACLE\10.2.0\DATABASE\INITJULIA.ORA'
```

当然我们还需要创建参数文件和口令文件等：

```
C:\>cp c:\oracle\10.2.0\database\SPFILEEYGLE.ORA
c:\Oracle\10.2.0\database\spfilejulia.ora
C:\>orapwd file=c:\oracle\10.2.0\database\PWDjulia.ora password=oracle entries=5
```

此后，实例可以顺利启动，并可以挂接和打开数据库：

```
C:\>sqlplus "/ as sysdba"
SQL*Plus: Release 10.2.0.1.0 - Production on 星期六 2月 17 10:13:10 2007
Copyright (c) 1982, 2005, Oracle.  All rights reserved.
已连接到空闲例程。
SQL> startup nomount;
SQL> show parameter instance_name
NAME                                 TYPE                 VALUE
------------------------------------ -------------------- ---------------
instance_name                        string               julia
SQL> show parameter db_name
NAME                                 TYPE                 VALUE
------------------------------  ----------------- ------------------------------
db_name                              string               eygle
SQL> alter database mount;
数据库已更改。
SQL> alter database open;
数据库已更改。
```

如果在环境窗口 CTRL+C 退出，则数据库将异常中断。

1.3 启动数据库 open 阶段

我们知道，控制文件中记录了数据库中数据文件、日志文件的位置信息，检查点信息等重要信息，在数据库的 Open 阶段，Oracle 根据控制文件中记录的这些信息找到这些文件，然后进行检查点及完整性检查。如果不存在问题就可以启动数据库，如果存在不一致或文件丢失则需要进行恢复。

1.3.1　Open 阶段的一致性校验

在数据库 Open 的过程中，Oracle 将会读取数据文件头块和控制文件信息，将两者进行对比，如果满足校验，则可以正常打开数据库；如果存在异常，则可能抛出相应异常信息，要求用户介入处理。

Oracle 在 Open 阶段将要进行很多校验检查，其中主要的校验包括以下两项：

第一次检查数据文件头中的检查点计数（Checkpoint cnt）是否和控制文件中的检查点计数（Checkpoint cnt）一致。此步骤检查用以确认数据文件是来自同一版本，而不是从备份中恢复而来（因为 Checkpoint Cnt 不会被冻结，会一直被修改）。

可以通过一个简单的测试来说明一下 Checkpoint Cnt（为了节省篇幅，省略了部分转储信息）的作用（以下实验测试信息来自 Oracle 9iR2）。

首先通过如下命令在不同条件下转储控制文件，第一步转储正常状态下的控制文件：

```
SQL> alter session set events 'immediate trace name CONTROLF level 10';
Session altered.
```

将系统表空间置于热备份状态（热备状态会冻结表空间数据文件的检查点）：

```
SQL> alter tablespace system begin backup;
Tablespace altered.
```

再来转储控制文件：

```
SQL> alter session set events 'immediate trace name CONTROLF level 10';
Session altered.
```

手工执行检查点并转储控制文件：

```
SQL> alter system checkpoint;
System altered.
SQL> alter session set events 'immediate trace name CONTROLF level 10';
Session altered.
```

结束表空间的热备状态，再次转储控制文件：

```
SQL> alter tablespace system end backup;
Tablespace altered.
SQL> alter session set events 'immediate trace name CONTROLF level 10';
```

```
Session altered.
```

注意：在 Oracle 10g 中，转储控制文件的事件有些变化，我们可以使用 Level 8 级转储获得控制文件信息:

```
alter session set events 'immediate trace name controlf level 8';
```

Oracle 10g 中转储的内容包括如下部分，增强了 Enabled Thread 的位图信息，这部分信息是用来支持 Oracle 10g 网格并行计算的，以下信息来自单实例系统:

```
Database checkpoint: Thread=1 scn: 0x0000.0007276a
Threads: #Enabled=1, #Open=1, Head=1, Tail=1
enabled threads: 01000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
……
```

对于双实例 RAC 系统的信息显示如下:

```
Database checkpoint: Thread=2 scn: 0x0000.008a8531
Threads: #Enabled=2, #Open=2, Head=2, Tail=1
enabled threads: 01100000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
…….
```

简要的来看一下前面测试过程中生成的跟踪文件信息(仅研究 system 表空间记录):

A）正常情况下转储控制文件

```
**********************************************************************
DATA FILE RECORDS
**********************************************************************
(blkno = 0x6, size = 180, max = 100, in-use = 24, last-recid= 574)
DATA FILE #1:
(name #4) /opt/oracle/oradata/hsjf/system01.dbf
creation size=32000 block size=8192 status=0xe head=4 tail=4 dup=1
tablespace 0, index=1 krfil=1 prev_file=0
unrecoverable scn: 0x0000.00000000 04/23/2004 01:20:52
Checkpoint cnt:1567 scn: 0x0000.0148181c 06/22/2004 18:58:46
Stop scn: 0xffff.ffffffff 06/22/2004 18:58:05
Creation Checkpointed at scn: 0x0000.000000ae 07/16/2003 03:40:10
```

注意这里记录的检查点计数器及 SCN。

B）执行 Begin backup 以后的

注意到 Checkpoint cnt 增加了 1，同时检查点 SCN 增加，**对表空间执行 Begin Backup 会触发一次表空间检查点**：

```
**********************************************************************
DATA FILE RECORDS
**********************************************************************
(blkno = 0x6, size = 180, max = 100, in-use = 24, last-recid= 574)
DATA FILE #1:
(name #4) /opt/oracle/oradata/hsjf/system01.dbf
creation size=32000 block size=8192 status=0xe head=4 tail=4 dup=1
tablespace 0, index=1 krfil=1 prev_file=0
unrecoverable scn: 0x0000.00000000 04/23/2004 01:20:52
Checkpoint cnt:1568 scn: 0x0000.01481939 06/22/2004 19:02:22
Stop scn: 0xffff.ffffffff 06/22/2004 18:58:05
Creation Checkpointed at scn: 0x0000.000000ae 07/16/2003 03:40:10
```

C）执行手工检查点

在表空间热备份模式下，手工执行检查点后，可以观察到，此时 Checkpoint cnt 增加，但是 SCN 不再改变。这是由于表空间处于热备份模式，数据文件检查点被冻结（热备模式下，数据库会生成额外的 redo 日志，在本书后面章节会有详细介绍）：

```
**********************************************************************
DATA FILE RECORDS
**********************************************************************
(blkno = 0x6, size = 180, max = 100, in-use = 24, last-recid= 574)
DATA FILE #1:
(name #4) /opt/oracle/oradata/hsjf/system01.dbf
creation size=32000 block size=8192 status=0xe head=4 tail=4 dup=1
tablespace 0, index=1 krfil=1 prev_file=0
unrecoverable scn: 0x0000.00000000 04/23/2004 01:20:52
Checkpoint cnt:1569 scn: 0x0000.01481939 06/22/2004 19:02:22
Stop scn: 0xffff.ffffffff 06/22/2004 18:58:05
Creation Checkpointed at scn: 0x0000.000000ae 07/16/2003 03:40:10
```

D）End backup 后的情况

此时数据文件头的冻结被取消,SCN 开始变化

```
**********************************************************************
DATA FILE RECORDS
**********************************************************************
(blkno = 0x6, size = 180, max = 100, in-use = 24, last-recid= 574)
DATA FILE #1:
(name #4) /opt/oracle/oradata/hsjf/system01.dbf
creation size=32000 block size=8192 status=0xe head=4 tail=4 dup=1
```

```
tablespace 0, index=1 krfil=1 prev_file=0
unrecoverable scn: 0x0000.00000000 04/23/2004 01:20:52
Checkpoint cnt:1570 scn: 0x0000.01481941 06/22/2004 19:02:39
Stop scn: 0xffff.ffffffff 06/22/2004 18:58:05
Creation Checkpointed at scn: 0x0000.000000ae 07/16/2003 03:40:10
.............
```

这就是检查点计数器及其在不同模式下的变化。

如果检查点计数检查通过，则数据库进行第二次检查。

第二次检查数据文件头的开始 SCN 和控制文件中记录的该文件的结束 SCN 是否一致，如果控制文件中记录的结束 SCN 等于数据文件头的开始 SCN，则不需要对那个文件进行恢复（如果此前数据库异常崩溃，则结束 SCN 会保持在最大值（无穷大），数据库必须执行实例恢复以确保一致性）。

对每个数据文件都完成检查后，打开数据库，锁定数据文件，同时将每个数据文件的结束 SCN 设置为无穷大（稍后将详细解释这个过程）。

继续 Mount 阶段的测试，如果数据库中的某个文件丢失，在 Mount 数据库时早期版本的 Oracle 会在后台将文件丢失信息记录在告警日志文件中，但是并不会在前台给出提示；而在 Open 阶段，如果数据库无法锁定该文件，则会在前台发出错误警告，数据库将停止启动：

```
SQL> alter database open;
alter database open
*
ERROR at line 1:
ORA-01157: cannot identify/lock data file 3 - see DBWR trace file
ORA-01110: data file 3: '/opt/oracle/oradata/eygle/eygle01.dbf'
```

> 注意：仅在 open 阶段，Oracle 才尝试打开并锁定数据文件，如果丢失或出现问题，则会给出错误提示。这时候就需要 dba 的介入进行处理，根据不同情况进行相应的恢复。

现在来看看告警日志文件中记录的 Open 过程中提示的错误信息.

```
Mon Sep 15 21:47:53 2008
alter database open
Mon Sep 15 21:47:53 2008
Errors in file /opt/oracle/admin/eygle/bdump/eygle_dbw0_17074.trc:
ORA-01157: cannot identify/lock data file 3 - see DBWR trace file
ORA-01110: data file 3: '/opt/oracle/oradata/eygle/eygle01.dbf'
ORA-27037: unable to obtain file status
Linux Error: 2: No such file or directory
Additional information: 3
Mon Sep 15 21:47:53 2008
```

```
ORA-1157 signalled during: alter database open...
```

在数据库出现问题的时候，提示中给出的可能是不完整的信息，而告警日志中则记录了完整的错误信息和错误号。所以当数据库出现故障时，应该优先检查 alert_<ORACLE_SID>.log，从中发现关于故障的详细信息。alert_<ORACLE_SID>.log 通常称为告警日志文件，位置由参数 background_dump_dest 定义：

```
SQL> show parameter background_dump_dest
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
background_dump_dest                 string      /opt/oracle/admin/eygle/bdump
```

1.3.2　Oracle 11g Automatic Diagnostic Repository 新特性

从 Oracle 11g 开始，alert 文件的格式发生了变化，除原有的文本格式外，还引入了 XML 格式。现在告警日志文件的存储位置受到一个新的参数影响，这个参数是 diagnostic_dest：

```
SQL> select * from v$version where rownum <2;
BANNER
--------------------------------------------------------------------------------
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
SQL> show parameter diag
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
diagnostic_dest                      string      /opt/oracle
```

diagnostic_dest 是 Oracle 11g 的新特性自动诊断库（Automatic Diagnostic Repository – ADR）的设置，该目录用于存放数据库诊断日志、跟踪文件等，通常称为称作 ADR base，在启用了 diagnostic_dest 之后，以前所熟知的 background_dump_dest、core_dump_dest、user_dump_dest 都被弃用。diagnostic_dest 参数的缺省值和环境变量 ORACLE_BASE 有关：

- **如果设置了 ORACLE_BASE 则 DIAGNOSTIC_DEST = ORACLE_BASE**
- **如果未设置 ORACLE_BASE ，则 DIAGNOSTIC_DEST = ORACLE_HOME/log**

伴随这一约束，Oracle 将以前在环境变量中设置的参数 OACLE_BASE 引入到数据库内部，增加了一个隐含参数用于记录：

```
SQL> SELECT x.ksppinm NAME, y.ksppstvl VALUE, x.KSPPDESC PDESC
  2    FROM SYS.x$ksppi x, SYS.x$ksppcv y
  3  WHERE x.indx = y.indx AND x.ksppinm LIKE '%&par%';
Enter value for par: oracle_base
NAME                           VALUE                PDESC
------------------------------ -------------------- ----------------------------
__oracle_base                  /opt/oracle          ORACLE_BASE
```

ADR 信息可以通过 V$DIAG_INFO 视图查询，其中 Diag Alert 和 Diag Trace 对应的目录

分别存储了 XML 和文本格式的告警日志文件：

```
SQL> select * from v$diag_info;
 INST_ID NAME                      VALUE
-------- ------------------------- ----------------------------------------
       1 Diag Enabled              TRUE
       1 ADR Base                  /opt/oracle
       1 ADR Home                  /opt/oracle/diag/rdbms/11gtest/eygle
       1 Diag Trace                /opt/oracle/diag/rdbms/11gtest/eygle/trace
       1 Diag Alert                /opt/oracle/diag/rdbms/11gtest/eygle/alert
       1 Diag Incident             /opt/oracle/diag/rdbms/11gtest/eygle/incident
       1 Diag Cdump                /opt/oracle/diag/rdbms/11gtest/eygle/cdump
       1 Health Monitor            /opt/oracle/diag/rdbms/11gtest/eygle/hm
       1 Default Trace File
                     /opt/oracle/diag/rdbms/11gtest/eygle/trace/eygle_ora_19939.trc
       1 Active Problem Count      0
       1 Active Incident Count     0
11 rows selected.
```

新的 XML 格式日志不便于直接阅读，通常我们仍然习惯阅读文本格式的告警日志文件：

```
[oracle@localhost ~]$ ls -l /opt/oracle/diag/rdbms/11gtest/eygle/alert
total 96
-rw-r-----  1 oracle dba 88285 Jul  6 15:39 log.xml
[oracle@localhost ~]$ tail -10 /opt/oracle/diag/rdbms/11gtest/eygle/alert/log.xml
 module='' pid='17457'>
 <txt>Setting  Resource  Manager  plan  SCHEDULER[0x2C0F]:DEFAULT_MAINTENANCE_PLAN  via
scheduler window
 </txt>
</msg>
<msg time='2008-07-06T15:39:51.147+08:00' org_id='oracle' comp_id='rdbms'
 client_id='' type='UNKNOWN' level='16'
 module='' pid='17457'>
 <txt>Setting Resource Manager plan DEFAULT_MAINTENANCE_PLAN via parameter
 </txt>
</msg>
```

随着 ADR 的引入，一个新的工具 ADRCI（ADR Command Interpreter）随之提供，这个工具可以用于管理 Oracle 11g 的诊断数据，当然也可以用于友好阅读 XML 格式的警告日志文件。在命令行输入 adrci 可以进入 ADRCI 工具，如果需要查看 alert 日志，可以通过 help show alert 来查看相关帮助：

adrci> help show alert

Usage: SHOW ALERT [-p <predicate_string>]　[-term]

[[-tail [num] [-f]] | [-file <alert_file_name>]]

Purpose: Show alert messages.

.....

Examples:

show alert

show alert -p "message_text like '%incident%'"

show alert -tail 20

如果当前 ADR 包含多个实例等日志信息，show alert 会提示进行选择，然后打开该日志文件：

```
adrci> show alert
Choose the alert log from the following homes to view:
1: diag/rdbms/11gtest/11gtest
2: diag/rdbms/11gtest/eygle
3: diag/clients/user_oracle/host_61728193_11
4: diag/clients/user_unknown/host_411310321_11
5: diag/tnslsnr/localhost/listener
Q: to quit
Please select option:
```

也可以预先指定 HOMEPATH，然后操作相关的告警日志文件：

```
adrci> set homepath diag/rdbms/phsdb/phsdb
adrci> show alert -tail 10
2008-07-28 02:33:08.415000 +08:00
ORA-609 : opiodr aborting process unknown ospid (27073_2181264)
2008-07-28 03:14:39.068000 +08:00
Stopping background process CJQ0
2008-07-28 07:00:03.598000 +08:00
Thread 1 advanced to log sequence 379
  Current log# 4 seq# 379 mem# 0: /data1/oradata/phsdb/redo04.log
2008-07-28 10:49:40.660000 +08:00
Starting background process CJQ0
CJQ0 started with pid=91, OS id=9205
```

那么 ADR 引入的目的何在呢？

大家知道在 Oracle Database 11g 之前，Oracle 的各类跟踪文件、日志文件等诊断文件的存

储位置并不统一，我们在进行诊断时需要到不同目录来查找相关文件，这在一定程度上带来不便，从 11g 开始，Oracle 开始统一规划这些文件的存储，ADR 之于诊断文件，就类似于 OFA（Optimal Flexible Architecture ）之于数据库文件，FRA（Flash Recovery Area）之于备份文件。

1.3.3　Oracle 11g Fault Diagnosability Infrastructure 新特性

```
D:\oracle>tree diag
卷 PRIVAT 的文件夹 PATH 列表
卷序列号为 5CB3-FCF7
D:\ORACLE\DIAG
├─asm
├─clients
│  └─user_SYSTEM
│      └─host_1390027684_11
│          ├─alert
│          ├─cdump
│          ├─incident
│          ├─incpkg
│          ├─lck
│          ├─metadata
│          ├─stage
│          ├─sweep
│          └─trace
├─crs
├─diagtool
├─lsnrctl
├─netcman
├─ofm
├─rdbms
│  └─eyglee
│      └─eyglee
│          ├─alert
│          ├─cdump
│          ├─hm
│          ├─incident
│          ├─incpkg
│          ├─ir
│          ├─lck
│          ├─metadata
│          ├─stage
│          ├─sweep
│          └─trace
└─tnslsnr
    └─eygle
        └─listener
            ├─alert
            ├─cdump
            ├─incident
            ├─incpkg
            ├─lck
            ├─metadata
            ├─stage
            ├─sweep
            └─trace
```

图 1-5 ADR 目录结构

前面介绍的 ADR 其实不过是 Oracle 另外一个重要新特性的一角，这个新特性就是故障诊断基础架构-Fault Diagnosability Infrastructure。

一直以来，当用户的数据库出现故障以后，向 Oracle 请求协助时，反复繁杂的交互与数据收集折磨了无数的用户，而这一工作又是不可缺少的。Oracle 也一直试图简化这些工作，提高故障的分析和解决效率。现在新引入的 FDI 就是来完成这一使命的。

这一架构用来收集和管理诊断数据，通常的诊断数据包括：

- 跟踪文件-trace files
- 转储文件-dumps
- 内核转储文件-core files
- 其他诊断数据等

FDI 的一个核心组件就是 ADR，左图是一个 ADR 目录所包含的内容结构，我们看到 Oracle 将数据库所有的日志等文件进行了归类汇总集合到 ADR 架构之下。

FDI 的引入目的在于更快速的问题诊断与解决、更少的用户交互以及减少和缩减故障的影响。

实现这一架构设想的一项技术是：第一时间自动诊断数据捕获- Automatic capture of diagnostic data upon first failure。当故障出现时，能够第一时间收集到故障信息对于问题的诊断与解决都是至关重要的，Oracle 通过一直打开的（always-on）基于内存的（memory-base）的跟踪系统进行信息收集，这是发现问题根源的有利保障。

当数据库故障、错误出现并被检测到时，故障诊断架构就被激活收集诊断数据，并且将

数据库不同组件记录的相关数据收集并存储到数据库之外的诊断库中。这一诊断原理非常类似飞机飞行中的"黑匣子（Black Box）"，Oracle 因此也将 FDI 称为 Oracle 数据库的黑匣子，通过这个黑匣子的引入，Oracle 希望能够收集到完善完整的数据用于故障诊断。

配合 FDI 的另外一项技术是**事件打包服务-Incident packaging service (IPS)**。同一次错误或故障相关的数据可能很多，包括 traces, dumps, health check reports 以及其他数据，手工收集和整理这些数据曾经是用户必需完成的工作，现在 IPS 服务帮助我们自动打包压缩这些数据。用户需要做的就是将这些打包数据传输给技术支持即可。

一个 Incident package 是一个代表一个或多个问题的逻辑结构，缺省情况下一个问题的第一次和最后三次信息被包含在 package 中。为了能够生成物理 Package，你还需要对逻辑结构进行实体化，这一工作可以通过 ADRCI 进行。

在 ADRCI 下，通过 show incident 命令可以显示数据库当前记录的 Incident：

```
[oracle@localhost ~]$ adrci
ADRCI: Release 11.1.0.6.0 - Beta on Mon Aug 4 11:55:09 2008
Copyright (c) 1982, 2007, Oracle.  All rights reserved.

ADR base = "/opt/oracle"
adrci> show incident;

ADR Home = /opt/oracle/diag/rdbms/11gtest/11gtest:
*************************************************************************
INCIDENT_ID          PROBLEM_KEY                  CREATE_TIME
-------------------- ---------------------------- ---------------------------------
14601                ORA 7445 [qerfxGCol()+3553]  2008-08-01 16:20:46.067780 +08:00
14545                ORA 7445 [qerfxGCol()+3553]  2008-08-01 16:21:23.318532 +08:00
2 rows fetched
```

注意当前数据库记录了两次 INCIDENT，通过具体的 INCIDENT_ID 可以查看更为详细的信息：

```
adrci> show incident -mode DETAIL -p "incident_id=14601";

ADR Home = /opt/oracle/diag/rdbms/11gtest/11gtest:
*************************************************************************

*************************************************************
INCIDENT INFO RECORD 1
*************************************************************
   INCIDENT_ID                   14601
   STATUS                        ready
```

```
   CREATE_TIME                    2008-08-01 16:20:46.067780 +08:00
   PROBLEM_ID                     1
   CLOSE_TIME                     <NULL>
   FLOOD_CONTROLLED               none
   ERROR_FACILITY                 ORA
   ERROR_NUMBER                   7445
   ERROR_ARG1                     qerfxGCol()+3553
   ERROR_ARG2                     SIGBUS
   ERROR_ARG3                     ADDR:0x20800000
   ERROR_ARG4                     PC:0xE4A0E09
   ERROR_ARG5                     Non-existent physical address
   ERROR_ARG6                     <NULL>
   ERROR_ARG7                     <NULL>
   ERROR_ARG8                     <NULL>
   SIGNALLING_COMPONENT           SQL_Execution
   SIGNALLING_SUBCOMPONENT        <NULL>
   SUSPECT_COMPONENT              <NULL>
   SUSPECT_SUBCOMPONENT           <NULL>
   ECID                           <NULL>
   IMPACTS                        0
   PROBLEM_KEY                    ORA 7445 [qerfxGCol()+3553]
   FIRST_INCIDENT                 14601
   FIRSTINC_TIME                  2008-08-01 16:20:46.067780 +08:00
   LAST_INCIDENT                  14545
   LASTINC_TIME                   2008-08-01 16:21:23.318532 +08:00
   IMPACT1                        0
   IMPACT2                        0
   IMPACT3                        0
   IMPACT4                        0
   KEY_NAME                       PQ
   KEY_VALUE                      (16777216, 1217578841)
   KEY_NAME                       Client ProcId
   KEY_VALUE                                  oracle@localhost.localdomain  (TNS
V1-V3).6813_3086902976
   KEY_NAME                       SID
   KEY_VALUE                      133.809
   KEY_NAME                       ProcId
   KEY_VALUE                      25.34
   OWNER_ID                       1
```

```
   INCIDENT_FILE
/opt/oracle/diag/rdbms/11gtest/11gtest/trace/11gtest_ora_6813.trc
   OWNER_ID                      1
   INCIDENT_FILE
/opt/oracle/diag/rdbms/11gtest/11gtest/incident/incdir_14601/11gtest_ora_6813_i14601.
trc
1 rows fetched
```

在相应的目录下，可以找到异常事件的跟踪信息：

```
[oracle@localhost incdir_14601]$ pwd
/opt/oracle/diag/rdbms/11gtest/11gtest/incident/incdir_14601
[oracle@localhost incdir_14601]$ ls -l
total 2140
-rw-r-----  1 oracle dba 2118047 Aug  1 16:20 11gtest_ora_6813_i14601.trc
-rw-r-----  1 oracle dba   50332 Aug  1 16:20 11gtest_ora_6813_i14601.trm
```

使用如下命令可以为 14601 号 Incident 创建逻辑 Package：

```
adrci> set homepath diag/rdbms/11gtest/11gtest
adrci> ips create package incident 14601
Created package 1 based on incident id 14601, correlation level typical
```

进一步的可以通过如下命令生成 Incident 的物理 Package，现在这个物理包就是需要发送出去用于技术支持进行诊断所需的文件：

```
adrci> ips generate package 1 in /opt/oracle/diag
Generated package 1 in file /opt/oracle/diag/ORA7445qe_20080804120317_COM_1.zip, mode
complete
```

如果想了解 Oracle 自动收集了哪些文件，可以解压缩一下这个压缩包来查看。

我们看到，FDI 实际上又是一系列自动化服务的增强，这些增强旨在更快的问题分析诊断、更少的用户交互，从而实现更高的可用性。**自动化是 Oracle 一直坚持不懈的方向。**

以下是 FDI 的几项关键组件：

- Automatic Diagnostic Repository (ADR)
- Alert Log
- Trace Files, Dumps, and Core Files
- Other ADR Contents
- Enterprise Manager Support Workbench
- ADRCI Command-Line Utility

从 FRA 到 ADR 以及 FDI，可以看到 Oracle 每一个进步都是精心设计的，模块化、

规范化正在被不断的加强。

1.3.4　关于诊断的建议

我曾经在论坛上看到无数类似这样的提问:

- 我的数据库启动不了了,怎么办?
- 我的数据库慢，怎么办？
- 我的数据库 down 了，怎么办？

面对这样的问题，我们往往无能为力，这里面没有任何实质性的信息，无从判断。

所以，我们说，在提问之前，你应该想想你想传达怎样的信息给别人。如果你只想说，我的数据库出问题了，那么别人只能了解这个事实，没有办法帮你。学会提问，也需要智慧。

在这里我提醒大家需要记住的是，**在数据库出现问题的时候，首先检查你的告警日志文件，研究其中的警告信息或者提供给他人寻求帮助，这通常是解决问题的第一个步骤**。

从警报日志文件来看一下完整的数据库启动过程：

```
Sat Apr 29 11:44:45 2006
alter database open
Sat Apr 29 11:44:45 2006
Thread 1 opened at log sequence 124
  Current log# 1 seq# 124 mem# 0: /opt/oracle/oradata/eygle/redo01.log
Successful open of redo thread 1.
Sat Apr 29 11:44:45 2006
SMON: enabling cache recovery
Sat Apr 29 11:44:46 2006
Undo Segment 1 Onlined
Undo Segment 2 Onlined
Undo Segment 3 Onlined
Undo Segment 4 Onlined
Undo Segment 5 Onlined
Undo Segment 6 Onlined
Undo Segment 7 Onlined
Undo Segment 8 Onlined
Undo Segment 9 Onlined
Undo Segment 10 Onlined
Successfully onlined Undo Tablespace 1.
Sat Apr 29 11:44:46 2006
SMON: enabling tx recovery
```

```
Sat Apr 29 11:44:46 2006
Database Characterset is ZHS16GBK
replication_dependency_tracking turned off (no async multimaster replication found)
Completed: alter database open
```

在完成数据库的验证和恢复过程后，数据库处于一致的状态，数据库还需要进行一系列的处理过程：将 Undo 段在线等操作，然后数据库可以提供访问，同时 SMON 可以开始进行事务回滚等。

在启动日志里你可能注意到了这样一行：

```
Database Characterset is ZHS16GBK
```

在每次数据库的启动过程中，Oracle 都需要判断控制文件中记录的字符集和数据库中的字符集是否相符，如果相符，则记录如上一行日志；如果不相符合，则以数据库中的字符集为准更新控制文件中的字符集记录，类似的日志如下：

```
Updating character set in controlfile to ZHS16CGB231280
```

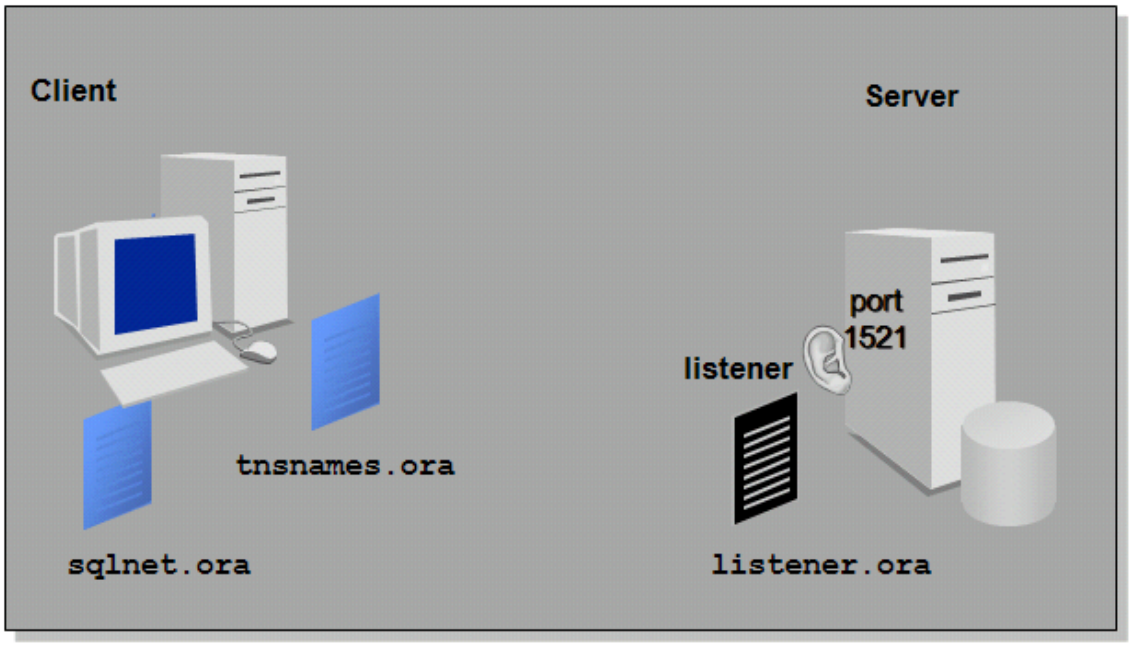
> 提示：在 Oracle 8i 之前，可以通过 Update props$表的方式修改字符集，从 Oracle 8i 开始，切记绝对不要使用同样的方式修改字符集。

如果细致一些，**启动日志中的每条信息都是值得研究的**。

2. 数据库的访问

在前面的测试环节中，我们一直使用的数据库连接方式是本地连接，即通过 SQL*Plus 工具在数据库服务器本地，通过 SYSDBA 身份进行登陆，这种登陆方式在进行数据库管理时需要经常用到。

而数据库启动之后，要想提供网络服务，通过远程连接访问，我们还需要启动数据库的监听器（配置文件通常为 listener.ora），监听器用于在特定的端口上（缺省的端口是 1521）提供监听，接收来自客户端的访问请求（客户端请求通过 tnsnames.ora 文件定义发送）。在专用服务器模式下（Dedicated Server），监听器会为每个请求衍生一个服务器进程相对应，通过这个服务器进程将客户端与数据库联系起来。下图描述了建立网络连接时客户端以及服务器端的相关文件及关系：



接下来我们介绍一下客户端以及服务器端为提供网络应用必须执行的相关配置，这些配置可以通过 Oracle 的工具 Net Configuration Assistant 来进行。

2.1客户端的 TNSNAMES.ORA 文件配置

客户端为了连接 Oracle 数据库服务器，需要安装客户端软件，并在客户端配置网络服务名文件（当然 Oracle 支持多种连接方式，这里主要介绍网络服务名文件方式连接，也就是 tnsnames.ora 文件），这个配置文件位于$ORACLE_HOME/network/admin 目录下，在 Windows 和 Unix/Linux 上的位置是相同的，文件名为 tnsnames.ora。

一段典型的配置为：

```
EYGLE=
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 172.16.33.11)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = eygle)
    )
  )
```

这里的 ADDRESS 部分包含了服务器的地址及监听端口信息，CONNECT_DATA 部分包含了连接信息，用于定义目标服务的名称。SERVICE_NAME 在这里用于识别访问的数据库服务；SERVICE_NAME 在这里也经常可以用 SID 来替代，从 Oracle 9i 开始，Oracle 推荐使用 SERVICE_NAME 而不是 SID。

配置完成之后，可以通过 tnsping 工具进行连通性测试：

```
[oracle@jumper oracle]$ tnsping eygle
Used TNSNAMES adapter to resolve the alias
Attempting to contact (DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)(HOST =
172.16.33.11)(PORT = 1521))) (CONNECT_DATA = (SERVICE_NAME = eygle)))
OK (10 msec)
```

如果能够顺利通讯，则可以通过 SQL*Plus 或其他工具通过网络服务名进行数据库网络访问：

```
SQL> connect system/oracle@eygle
Connected.
SQL> select sid,username,machine from v$session where username='SYSTEM';
       SID USERNAME                       MACHINE
---------- ------------------------------ ------------------------------
        11 SYSTEM                         eygle.com
```

注意：tnsping 工具只是测试监听能否连通，如果 tnsping 返回的结果为 OK，表明客户端到服务器的网络是通畅的，同时表明监听的 TCP 端口与客户端配置的 TCP 端口是一致的。实际上 tnsping 测试时，监听并不检查客户端输入的 SERVICE_NAME 或 SID，所以 tnsping 成功并不意味着 tnsname 的配置就是完全正确的，也不意味着就一定能连接上数据库，也许数据库根本还没启动。

特殊的，除了 tnsnames.ora 文件之外，Oracle 支持通过 SQL*Plus 直接连接远程或本地数据库，当然需要在连接时指定必须的参数信息，例如：

```
SQL>conn
eygle/eygle@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=172.16.33.32)(PORT=1521))(CONNE
CT_DATA=(SERVICE_NAME=hsbill)))
Connected.
SQL> select instance_name,startup_time,version from v$instance;
INSTANCE_NAME    STARTUP_T VERSION
---------------- --------- -----------------
hsbill           07-MAY-08 9.2.0.4.0
```

连接本地数据库可以使用类似如下方式：

```
conn
eygle/eygle@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=localhost)(Port=1521))(CONNECT_
DATA=(SID=eygle)))
```

从 Oracle 10g 开始，还可以使用如下方式进行连接（这个特性在本章后面“简捷的 Easy Connect 方式”一节有详细说明）：

```
SQL> connect eygle/eygle@localhost:1521/eygle
Connected.
SQL> select sid,program from v$session where username='EYGLE';
       SID PROGRAM
---------- ------------------------------------------------
       158 sqlplus@test126 (TNS V1-V3)
```

前面说的是客户端，现在说说服务器端；在服务器端数据库中存在一个初始化参数 SERVICE_NAMES，这个参数就是用于定义客户端请求的数据库服务名。

Oracle 文档中这样定义这个初始化参数：

SERVICE_NAMES 为实例所连接的数据库定义一个或多个服务名，可以通过定义多个服务名将不同用户连接区分开来。这个参数的缺省格式为：DB_NAME.DB_DOMAIN，如果定义了 DB_DOMAIN 那么定义的服务名就类似：

```
SERVICE_NAMES = sales.eygle.com, news.eygle.com
```

通过这样的定义，销售用户可以通过在客户端 TNSNAME 定义中使用 SALES 服务名来建立连接，而新闻用户则可以通过 NEWS 服务名进行连接，最终用户可以不必关注数据库是哪一个，他们只需要关心服务名。

注意：在 RAC 环境中，tnsnames.ora 文件中的参数设置会有所不同。

2.2 服务器端的监听器文件 listener.ora 配置

说完了客户端再来看一下，如果数据库端设置了 SERVICE_NAMES 后，监听器应该怎样配置。例如如下一个数据库系统，为数据库设置多个服务名（通过 SCOPE=both 设置，同时修改了参数文件）：

```
SQL> show parameter service_name
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
service_names                        string      eygle
SQL> alter system set service_names='eygle,julia' scope=both;
System altered.
SQL> show parameter service_names
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
service_names                        string      eygle,Julia
```

同样在$ORACLE_HOME/network/admin 目录下可以找到 listener.ora 文件，以下是一个监听器文件的典型配置：

```
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
```

```
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
      )
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = 172.16.33.11)(PORT = 1521))
      )
    )
  )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = /opt/oracle/product/9.2.0)
      (PROGRAM = extproc)
    )
    (SID_DESC =
      (GLOBAL_DBNAME = eygle)
      (ORACLE_HOME = /opt/oracle/product/9.2.0)
      (SID_NAME = eygle)
    )
    (SID_DESC =
      (GLOBAL_DBNAME = julia)
      (ORACLE_HOME = /opt/oracle/product/9.2.0)
      (SID_NAME = eygle)
    )
  )
```

监听器文件主要包括两个部分：

第一部分 LISTENER 信息，这部分包含了监听的协议、地址以及端口等信息。

第二部分 SID_LIST_LISTENER 信息，这部分信息用于提供对外的数据库服务列表。第一个 SID_DESC 部分（SID_NAME = PLSExtProc）是数据库缺省就包含的对外部存储过程提供的本地监听，此外两个 SID_DESC 部分就是对数据库的两个 SERVICE_NAMES 所设置的监听服务，对于同一个 SID 对应的数据库，可以对外提供多个服务名供客户端访问。

设置服务名的参数为 GLOBAL_DBNAME，当处理客户端连接请求时，监听器首先尝试将 GLOBAL_DBNAME 和客户端请求中的 SERVICE_NAME 相匹配；如果客户端连接请求的是 SID 信息，则 Oracle 不检查 GLOBAL_DBNAME 设置，而是对监听器中设置的 SID_NAME 进行匹配。

启动这个监听器后，可以看到对于不同服务名 Oracle 所启动的监听信息。首先输出的信息显示了监听器文件地址以及监听日志文件位置(监听器日志在诊断数据库异常或攻击信息时非常有用)：

```
[oracle@jumper admin]$ lsnrctl start
LSNRCTL for Linux: Version 9.2.0.4.0 - Production on 16-FEB-2007 16:50:37
Copyright (c) 1991, 2002, Oracle Corporation.  All rights reserved.
Starting /opt/oracle/product/9.2.0/bin/tnslsnr: please wait...

TNSLSNR for Linux: Version 9.2.0.4.0 - Production
System parameter file is /opt/oracle/product/9.2.0/network/admin/listener.ora
Log messages written to /opt/oracle/product/9.2.0/network/log/listener.log

Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=EXTPROC)))
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=172.16.33.11)(PORT=1521)))
```

最后部分包含了启动的相关服务信息，本例包含三个服务：

```
Services Summary...
Service "PLSExtProc" has 1 instance(s).
  Instance "PLSExtProc", status UNKNOWN, has 1 handler(s) for this service...
Service "eygle" has 1 instance(s).
  Instance "eygle", status UNKNOWN, has 1 handler(s) for this service...
Service "julia" has 1 instance(s).
  Instance "eygle", status UNKNOWN, has 1 handler(s) for this service...
The command completed successfully
```

2.3 通过不同服务器名对数据库的访问

服务器端完成上述配置后，现在客户端可以通过不同的网络服务名配置来访问这个数据库，下面是一个配置示范：

```
EYGLE=
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 172.16.33.11)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = eygle)
    )
  )
JULIA=
  (DESCRIPTION =
```

```
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 172.16.33.11)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = julia)
    )
  )
```

通过这两个服务名，都可以顺利的访问到数据库：

```
[oracle@jumper admin]$ sqlplus /nolog
SQL*Plus: Release 9.2.0.4.0 - Production on Fri Feb 16 16:54:48 2007
Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.

SQL> connect eygle/eygle@eygle
Connected.
SQL> show parameter service_names
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
service_names                        string      eygle,julia
SQL> connect eygle/eygle@julia
Connected.
SQL> show parameter service_names
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
service_names                        string      eygle,Julia
```

通过服务名，Oracle 可以将客户端和服务器彻底隔离开来，对于客户端来说，它不用关心数据库的名字、实例名到底是什么，它只需要知道数据库对外提供的服务名就行了，这个名字可能和实例名相同，也可能并不相同。

2.4 动态监听器注册服务

从 Oracle 8i 开始，Oracle 引入了动态服务注册（Dynamic Service Registration）的功能，所谓动态注册是指当实例启动之后，由后台进程 PMON 在监听器中注册数据库服务信息。在动态注册机制下，原来监听器中的 SID_LIST 部分将不再需要。

通过服务注册可以获得如下收益：

1．简化配置

服务注册可以减化监听器的配置，SID_LIST_<listener_name>参数将不再需要，

2．连接时 Failover

在动态注册时，由数据库主动向监听器注册实例，因此监听器总是可以知道实例的状

态，在 RAC 环境下，当某个数据库实例出现故障时，动态服务注册功能可以快速自动的 Failover 客户端请求到其他实例；而如果在静态注册模式下，监听器将首先启动一个专用服务器进程接受客户端请求，然后向数据库服务器发起连接，随后才能发现实例已经停止，给出“Oracle not available”的错误提示，这个过程要缓慢低效得多。

3．运行时连接负载均衡

在 RAC 环境下，服务注册使得监听器能够向负载最轻的实例转发连接请求，从而实现运行时连接的负载均衡。

动态注册在 Oracle 9i 里是自动启用的，监听器文件可以不再需要，或者可以配置一个经过极大简化的监听器文件。现在一个简单的监听器配置可能类似如下示例（缺省的监听 PLSExtProc 是为外部存储过程调用而配置的）：

```
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = 172.16.33.50)(PORT = 1521))
      )
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
      )
    )
  )
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = /opt/oracle9/product/9.2.0)
      (PROGRAM = extproc)
    )
    )
```

这样监听器启动后可以看到如下信息：

```
bash-2.03$ lsnrctl start
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=172.16.33.50)(PORT=1521)))
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for Solaris: Version 9.2.0.4.0 - Production
Start Date                16-FEB-2007 20:42:27
```

```
Listener Parameter File   /opt/oracle9/product/9.2.0/network/admin/listener.ora
Listener Log File         /opt/oracle9/product/9.2.0/network/log/listener.log
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=172.16.33.50)(PORT=1521)))
  (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=EXTPROC)))
Services Summary...
Service "PLSExtProc" has 1 instance(s).
  Instance "PLSExtProc", status UNKNOWN, has 1 handler(s) for this service...
The command completed successfully
```

已经启动的实例随后会将服务名（初始化参数中定义的 SERVICE_NAMES）注册到监听器中：

```
bash-2.03$ lsnrctl status
°°°°°°°°°°°°°°°°°°°°
Services Summary...
Service "PLSExtProc" has 1 instance(s).
  Instance "PLSExtProc", status UNKNOWN, has 1 handler(s) for this service...
Service "eygle" has 1 instance(s).
  Instance "testora9", status READY, has 1 handler(s) for this service...
Service "julia" has 1 instance(s).
  Instance "testora9", status READY, has 1 handler(s) for this service...
Service "testora9XDB" has 1 instance(s).
  Instance "testora9", status READY, has 1 handler(s) for this service...
The command completed successfully
```

动态注册的服务名，由于监听器确切地知道实例的状态，所以正常状态通常显示为 READY，而对于静态注册的服务名，则状态显示为 UNKNOW，这也是我们经常看到某些数据库的监听器会有如下显示的原因：

```
Services Summary...
Service "PLSExtProc" has 1 instance(s).
  Instance "PLSExtProc", status UNKNOWN, has 1 handler(s) for this service...
Service "hsbill" has 2 instance(s).
  Instance "hsbill", status UNKNOWN, has 1 handler(s) for this service...
  Instance "hsbill", status READY, has 1 handler(s) for this service...
```

缺省情况下，实例使用数据库服务器主机名对应的 IP 地址和 1521 端口连接监听进行动态注册，如果监听器使用了服务器主机名或主机名对应的 IP 地址、缺省的 1521 端口及 TCP 协议，则无需任何特殊配置，Oracle 就能执行动态注册。否则需要设置 LOCAL_LISTENER 参数。

对于专用服务器模式，参数可以设置为：

```
LOCAL_LISTENER=listener_alias
```

对于共享服务器模式，参数可以设置为：

```
DISPATCHERS="(PROTOCOL=tcp)(LISTENER=listener_alias)"
```

Listener_alias 随后通过 Oracle 命名方式（例如 tnsnames.ora 文件）解析为其他协议地址。例如如果监听器监听端口为 1522，可以设置初始化参数为：

```
LOCAL_LISTENER=listener1
```

对于共享服务期模式，可以设置为：

```
DISPATCHERS="(PROTOCOL=tcp)(LISTENER=listener1)"
```

在 tnsnames.ora 文件中 listener1 可以按如下方式解析：

```
listener1=
(DESCRIPTION=
  (ADDRESS=(PROTOCOL=tcp)(HOST=prod-server)(PORT=1522)))
```

同样，监听器还可以向远程服务器注册，例如在 RAC 环境中，配置监听器远程注册需要设置 REMOTE_LISTENER 参数，假定两个实例的实例名称分别为 prod1 和 prod2，那么两个实例的 REMOTE_LISTENER 参数应该分别设置如下。

对于 prod1 服务器设置：

```
REMOTE_LISTENER=listener_prod2
```

对于 prod2 服务器设置为：

```
REMOTE_LISTENER=listener_prod1
```

在 prod1 服务器上的 tnsnames.ora 文件中可以如下配置 listener_prod2 命名：

```
listener_prod2=
(DESCRIPTION=
(ADDRESS=(PROTOCOL=tcp)(HOST=prod2-server)(PORT=1521)))
```

在 prod2 服务器上的 tnsnames.ora 文件中可以如下配置 listener_prod1 命名：

```
listener_prod1=
(DESCRIPTION=
(ADDRESS=(PROTOCOL=tcp)(HOST=prod1-server)(PORT=1521)))
```

而为了进一步简化，REMOTE_LISTENER 参数的配置在 RAC 环境中可以相同，以下是来自 Oracle 10g RAC 环境中的示例，多个实例的参数设置相同：

```
SQL> show parameter remote_lis
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
remote_listener                      string      LISTENERS_SMSDB
```

这个设置可以通过手工方式修改，类似如下命令可以用于完成这一工作：

alter system set REMOTE_LISTENER = 'LISTENERS_ALIAS' scope=both sid='';*

然后 tnsnames.ora 文件配置包含如下信息：

```
LISTENERS_SMSDB =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.200.13)(PORT = 1521))
    (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.200.14)(PORT = 1521))
  )
```

这样监听器启动之后就会同时自动在远程和本地进行注册，这个 RAC 数据库的初始化参数 SERVICE_NAMES 设置如下：

```
SQL> show parameter service_name
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
service_names                        string      smsrac, smsdb
```

以下输出是 RAC 环境中数据库的注册信息（做了适当简化）：

```
[oracle@smsdbrac2 admin]$ lsnrctl status
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for Linux: Version 10.2.0.4.0 - Production
Start Date                05-MAY-2008 16:04:44
------------------------
Services Summary...
Service "+ASM" has 1 instance(s).
  Instance "+ASM2", status BLOCKED, has 1 handler(s) for this service...
Service "smsdb" has 2 instance(s).
  Instance "smsdb1", status READY, has 1 handler(s) for this service...
  Instance "smsdb2", status READY, has 2 handler(s) for this service...
Service "smsrac" has 2 instance(s).
  Instance "smsdb1", status READY, has 1 handler(s) for this service...
  Instance "smsdb2", status READY, has 2 handler(s) for this service...
The command completed successfully
```

对应于 RAC 环境，客户端的 tnsnames.ora 文件配置也有所不同，以下是一段 RAC 环境下客户端的配置示例。与单实例的不同之处在于地址列表段包含多个实例的地址信息，同时支持负载均衡和在多实例之间的 FailOver 切换：

```
SMSRAC =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.200.13)(PORT = 1521))
```

```
    (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.200.14)(PORT = 1521))
    (LOAD_BALANCE = yes)
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = smsrac)
      (FAILOVER_MODE =
        (TYPE = SELECT)
        (METHOD = BASIC)
        (RETRIES = 180)
        (DELAY = 5)
      )
    )
  )
```

2.5 简捷的 Easy Connect 方式

从 Oracle Database 10g 开始，一种称为 Easy Connect Naming Method 被引入到数据库中来，通过**在客户端和数据库服务器端的 sqlnet.ora 文件中设置 NAMES.DIRECTORY_PATH 命名方法可以启用这个特性**，例如如下设置，指定数据库可以接受 EZCONNECT 方式的连接：

NAMES.DIRECTORY_PATH= (TNSNAMES, EZCONNECT)

进行了这样的设定之后，可以在客户端使用如下方式进行数据库连接：

```
[oracle@drac1 admin]$ sqlplus eygle/eygle@172.16.3.248:1521/rac

SQL*Plus: Release 10.2.0.4.0 - Production on Tue Feb 8 19:08:42 2011
Copyright (c) 1982, 2007, Oracle.  All Rights Reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.4.0 - 64bit Production
With the Partitioning, Real Application Clusters, Data Mining and Real Application Testing
options

SQL>
```

在不存在 TNSNAMES 命名设置时，客户端连接会自动尝试 EZCONNECT 方式，如果可能就据此建立连接，这里 EZ 是 Easy 的含义。这一特性的连接语法如下：

CONNECT username/password@[//]host[:port][/service_name]

其中 Host 既可以是可解析的域名也可以是具体的 IP 地址，以下命令可以和前面的测试语句达到同样的效果 sqlplus eygle/eygle@rac1:1521/rac　：

```
[oracle@rac1 admin]$ sqlplus eygle/eygle@rac1:1521/rac
```

```
SQL*Plus: Release 10.2.0.4.0 - Production on Tue Feb 8 19:10:03 2011
Copyright (c) 1982, 2007, Oracle.  All Rights Reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.4.0 - 64bit Production
With the Partitioning, Real Application Clusters, Data Mining and Real Application Testing
options

SQL>
```

使用 tnsping 工具可以看出以上连接的创建过程：

```
[oracle@rac1 admin]$ tnsping rac.eygle.com

TNS Ping Utility for Solaris: Version 10.2.0.4.0 - Production on 08-FEB-2011 19:06:30
Copyright (c) 1997,  2007, Oracle.  All rights reserved.

Used parameter files:
/opt/oracle/product/10.2.0/db10g/network/admin/sqlnet.ora
Used EZCONNECT adapter to resolve the alias
Attempting to contact (DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=rac1.eygle.com))
(ADDRESS=(PROTOCOL=TCP)(HOST=172.16.3.248)(PORT=1521)))
OK (10 msec)
```

结合 sqlnet.ora 文件的设置，客户端和服务器端的连接可以设定很多内容，选项是极其丰富的。

3. 数据库的关闭

数据库的启动，通常只需要一个命令 startup 就完成了，实际上在后台 Oracle 是通过 nomount、mount、open 三个步骤来完成的；将这个过程逆向过来，那么实际上当我们通过 shutdown 来关闭数据库时，实际上数据库也就经历了 close、dismount、shutdown 三个步骤。

3.1 数据库关闭的步骤

以下是 Oracle 10g 中数据库关闭的分步骤操作：

```
SQL> alter database close;
Database altered.
```

注意 Close 数据库仅允许在没有连接的情况下进行，否则可能遇到如下错误：

```
ORA-01093: ALTER DATABASE CLOSE only permitted with no sessions connected
```

这一过程的告警日志信息显示如下：

```
Tue Jul 18 11:08:10 2006
```

```
alter database close
Tue Jul 18 11:08:10 2006
SMON: disabling tx recovery
Tue Jul 18 11:08:10 2006
Stopping background process CJQ0
Tue Jul 18 11:08:10 2006
Stopping background process QMNC
Tue Jul 18 11:08:14 2006
Stopping Job queue slave processes
Tue Jul 18 11:08:22 2006
Waiting for Job queue slaves to complete
Tue Jul 18 11:08:55 2006
Job queue slave processes stopped
Tue Jul 18 11:08:55 2006
SMON: disabling cache recovery
Tue Jul 18 11:08:56 2006
Shutting down archive processes
Archiving is disabled
Archive process shutdown avoided: 0 active
Thread 1 closed at log sequence 39
Successful close of redo thread 1
Tue Jul 18 11:08:56 2006
Completed: alter database close
```

接下来可以将数据库卸载：

```
SQL> alter database dismount;
Database altered.
```

这一过程的告警日志信息如下所示：

```
Tue Jul 18 11:09:49 2006
alter database dismount
Tue Jul 18 11:09:49 2006
Completed: alter database dismount
```

最后一个步骤是彻底关闭数据库实例，可以通过发出 Shutdown 命令完成：

```
SQL> shutdown;
ORA-01507: database not mounted


ORACLE instance shut down.
```

最后的实例关闭记录如下日志：

```
Tue Jul 18 11:10:59 2006
Shutting down instance: further logons disabled
Tue Jul 18 11:10:59 2006
Stopping background process MMNL
Tue Jul 18 11:11:00 2006
Stopping background process MMON
Tue Jul 18 11:11:01 2006
Shutting down instance (normal)
License high water mark = 2
Tue Jul 18 11:11:01 2006
ALTER DATABASE CLOSE NORMAL
ORA-1507 signalled during: ALTER DATABASE CLOSE NORMAL...
ARCH: Archival disabled due to shutdown: 1090
Shutting down archive processes
Archiving is disabled
Archive process shutdown avoided: 0 active
ARCH: Archival disabled due to shutdown: 1090
Shutting down archive processes
Archiving is disabled
Archive process shutdown avoided: 0 active
```

在使用 shutdown 命令关闭数据库时，还有几个可选参数，这几个参数分别是：NORMAL、IMMEDIATE、TRANSACTIONAL、ABORT。

3.2 几种关闭方式的对比

以下是几种关库方式的对比：

3.2.1 SHUTDOWN NORMAL

SHUTDOWN NORMAL 是数据库关闭 SHUTDOWN 命令的缺省选项，当我们执行 SHUTDOWN 时，Oracle 即以正常方式关闭数据库。发出该命令后，任何新的连接都将不再允许连接到数据库，但是在数据库关闭之前，Oracle 需要等待当前连接的所有用户都从数据库中退出。

采用这种方式关闭数据库，在下一次启动时不需要进行任何的实例恢复，但是由于 Normal 方式要等所有用户断开连接后才能关闭数据库，所以等待时间可能超长；在生产环境中，**这种方式几乎无法关闭有大量用户连接的数据库，所以很少被采用**。

3.2.2 SHUTDOWN IMMEDIATE

SHUTDOWN IMMEDIATE 方式是最为常用的一种关闭数据库的方式，使用这个命令时，当前正在被 Oracle 处理的事务立即中断，未提交的事务将全部回滚，系统不等待连接到数据库的用户退出，强制断开所有的连接用户。然后执行检查点，将变更数据全部写回数据

文件，关闭数据库。使用这种方式关闭数据库，在下次启动数据库时不需要进行实例恢复，是一种安全的数据库关闭方式。

但是注意，如果数据库系统繁忙，当前有大量事务执行（甚至是大事务正在处理），那么使用此选项关闭数据库也可能需要大量时间。

3.2.3 SHUTDOWN TRANSACTIONAL

SHUTDOWN TRANSACTIONAL 选项仅在 Oracle 8i 后可用，使用该命令时，数据库不再允许建立新的连接，禁止新事务的进行，但是允许当前活动事务执行完毕。

在所有活动的事务完成后，数据库将和 SHUTDOWN IMMEDIATE 同样的方式关闭数据库。

3.2.4 SHUTDOWN ABORT

SHUTDOWN ABORT 是最不推荐采用的关闭数据库的方法，使用改选项，数据库会立即终止所有用户连接、中断所有事务、立即关闭数据库，使用这种方式关闭数据库，未完成事务不会回滚，数据库也不会执行检查点，所以在下次启动时，数据库必须执行实例恢复，实例恢复可能会需要大量时间，数据库的启动因此可能需要等候很长时间。

Abort 的方式关闭数据库，就类似于数据库服务器突然断电，可能会导致不一致的情况出现，所以除非不得已，轻易不要使用这种方式关闭数据库。

那么在什么情况下需要使用 Shutdown abort 方式关闭数据库呢？以下是一些常见的场景：

- 数据库或应用异常，其他方式无法关闭数据库
- 因为马上到来的断电或其他维护情况，需要快速关闭数据库
- 启动异常后需要重新尝试启动
- 当使用 Shutdown Immediate 无法关闭时
- 需要快速重新启动数据库
- Shutdown 超时或异常

以下是一个使用 Shutdown Abort 情况的案例。

当尝试使用 shutdown immediate 方式关闭数据库时，长时间未获得响应：

```
SQL> shutdown immediate;
```

此时检查数据库的告警日志文件，发现出现如下错误信息：

```
Tue Jul 18 13:18:40 2006
Process OS id : 28507 alive after kill
Errors in file
Tue Jul 18 13:18:55 2006
```

```
PMON failed to acquire latch, see PMON dump
PMON failed to acquire latch, see PMON dump
PMON failed to acquire latch, see PMON dump
```

报错信息反复出现，数据库无法关闭，此时只有通过 Shutdown Abort 的方式强制关闭数据库：

```
[oracle@test126 bdump]$ sqlplus "/ as sysdba"
SQL*Plus: Release 10.2.0.1.0 - Production on Tue Jul 18 13:20:38 2006
Copyright (c) 1982, 2005, Oracle.  All rights reserved.
Connected.
SQL> shutdown abort;
ORACLE instance shut down.
```

此时告警日志文件记录了这个过程，输出信息显示由于不能 Kill 一个或多个进程，实例终止失败，最后被用户中止：

```
Tue Jul 18 13:20:42 2006
Shutting down instance (abort)
License high water mark = 11
Termination issued to instance processes. Waiting for the processes to exit
Tue Jul 18 13:20:52 2006
Instance termination failed to kill one or more processes
Instance terminated by USER, pid = 28892
```

除了异常情况之外，有时候需要快速重新启动数据库，很多人习惯用 abort 方式来进行操作，但是需要注意的是，Abort 之后重启数据库需要进行恢复，启动的时间可能很长，所以如果时间允许，可以在关闭数据库之前执行一次 Checkpoint，如：

alter system checkpoint

如果此后再使用 Abort 关闭数据库，那么在下次启动恢复时，需要恢复的数据就可以减少，当然如果能够不使用 ABORT 方式是最好的。

有时候在使用常规命令关闭数据库时（如 Shutdown Normal 或 Immediate），可能会遇到由于个别事务或进程阻塞无法关闭的情况，如果数据库等待一小时仍然无法关闭，就会以超时抛出错误信息：

```
ORA-01013: user requested cancel of current operation.
```

此外在关闭过程中，如果用户发出 CTRL-C，则关闭也可以被手工取消：

```
SQL> shutdown immediate;
ORA-01013: user requested cancel of current operation
```

但是无论超时还是用户手工取消，数据库都可能陷于两种状态，一种是正常状态，可以继续运行；一种是未知状态，数据库无法正常运行。如果数据库无法正常运行，那么此时我们

将被迫使用 ABORT 方式关闭数据库。

下表列举了不同方式关闭数据库的区别，供参考：

关闭方式	A	I	T	N
允许新的连接	×	×	×	×
等待活动会话中止	×	×	×	√
等待活动事务中止	×	×	√	√
强制 CheckPoint，关闭所有文件	×	√	√	√

3.3数据库关闭的诊断案例一则

在一个客户案例中，客户尝试关闭数据库启动归档模式，但是发现通过 immediate 方式无法关闭数据库，只能 abort 关闭，但是再次重新启动数据库之后，发现仍然无法正常关闭数据库，还是只能 abort 关闭，但是 abort 关闭数据库之后，下次启动需要执行数据库的实例恢复（Instance Recovery），无法启用归档模式。

诊断这个问题，首先在会话级别启用 10046 跟踪，然后执行 shutdown 操作，示范如下：

```
[eygle@hpserver2 ~]$ sqlplus / as sysdba
SQL*Plus: Release 10.2.0.5.0 - Production on Fri Jun 15 17:24:23 2012
Copyright (c) 1982, 2010, Oracle.  All Rights Reserved.
SQL> alter session set events '10046 trace name context forever,level 12';

Session altered.

SQL> shutdown immediate;
Database closed.
Database dismounted.
ORACLE instance shut down.
```

分析生成的跟踪文件，即可进行数据库关闭相关的分析。如下内容是在客户案例中生成的跟踪文件，跟踪文件显示，在 immediate 过程中，数据库以 instance state change 等待事件处于循环等待。该事件含义为：等待实例状态改变，此处是等待实例由 Open 变更为 Close。这个等待只说明当前状况，造成等待的原因来自之前的 SQL：

```
=====================
PARSING IN CURSOR #2 len=71 dep=1 uid=0 oct=7 lid=0 tim=0 hv=2341485949 ad='a95882e0'
delete from sys.mon_mods$ where obj# not in (select obj# from sys.obj$)
END OF STMT
PARSE #2:c=0,e=0,p=0,cr=52,cu=0,mis=1,r=0,dep=1,og=4,tim=0
BINDS #2:
EXEC #2:c=0,e=0,p=0,cr=0,cu=4,mis=0,r=0,dep=1,og=4,tim=0
STAT #2 id=1 cnt=1 pid=0 pos=0 obj=0 op='DELETE MON_MODS$ '
STAT #2 id=2 cnt=1 pid=1 pos=1 obj=0 op='FILTER '
STAT #2 id=3 cnt=1 pid=2 pos=1 obj=369 op='TABLE ACCESS FULL MON_MODS$ '
STAT #2 id=4 cnt=0 pid=2 pos=2 obj=18 op='TABLE ACCESS FULL OBJ$ '
WAIT #1: nam='instance state change' ela= 0 p1=1 p2=0 p3=1
WAIT #1: nam='instance state change' ela= 0 p1=1 p2=0 p3=2
WAIT #1: nam='instance state change' ela= 0 p1=1 p2=0 p3=3
WAIT #1: nam='instance state change' ela= 0 p1=1 p2=0 p3=4
WAIT #1: nam='instance state change' ela= 0 p1=1 p2=0 p3=5
WAIT #1: nam='instance state change' ela= 0 p1=1 p2=0 p3=6
WAIT #1: nam='instance state change' ela= 0 p1=1 p2=0 p3=7
WAIT #1: nam='instance state change' ela= 0 p1=1 p2=0 p3=8
```

我们注意 DELETE 语句对 SYS 用户的 mon_mods$表进行操作，该表的创建语句如下，可以看出其作用是为了记录对于数据库对象的修改操作，包括增（INSERT）删（DELETE）改（UPDATE）的次数：

```
create table mon_mods$
(
  obj#              number,                                  /* object number */
  inserts           number,  /* approx. number of inserts since last analyze */
  updates           number,  /* approx. number of updates since last analyze */
  deletes           number,  /* approx. number of deletes since last analyze */
  timestamp         date,     /* timestamp of last time this row was changed */
  flags             number,                                        /* flags */
                                          /* 0x01 object has been truncated */
  drop_segments     number   /* number of segemnt in part/subpartition table */
)
  storage (initial 200K next 100k maxextents unlimited pctincrease 0)
/
```

数据库关闭时需要执行的 DELETE 操作,目的是将那些 OBJ$不存在的对象内容删除（可能有些对象被 DROP 掉），由于早期版本中，NOT IN 操作的执行计划走了全表扫描，导致该操作可能极其缓慢。

在这个案例中，有两个选择，一是手工删除这些记录，或者多等一段时间，数据库完成这个删除操作之后，自然就可以正常关闭了。这个问题作为 BUG，在 Oracle 9i 之后的版本中已经被修正。

参考文献：

Oracle® Database Administrator's Guide 10g Release 2 (10.2)　B14231-01

Oracle® Database Concepts 10g Release 2 (10.2)　B14220-02

Oracle® Database Concepts 11g Release 1 (11.1)　B28318-03

Oracle® Database Administrator's Guide 11g Release 1 (11.1)　B28310-04