检查跟踪文件中的错误信息，其错误情况与客户的情况完全相符，最后数据库实例崩溃：

```
Mon Aug 09 16:12:59 2010
ORA-600 signalled during: create tablespace enmo datafile size 10M...
Mon Aug 09 16:13:00 2010
Errors in file d:\oracle\admin\enmo\udump\enmo_ora_4736.trc:
ORA-00600: 内部错误代码, 参数: [25015], [8], [6], [5], [], [], [], []
ORA-00600: 内部错误代码, 参数: [25013], [0], [8], [ENMO], [ENMO], [5], [6], []

Mon Aug 09 16:13:00 2010
Errors in file d:\oracle\admin\enmo\udump\enmo_ora_4736.trc:
ORA-00600: 内部错误代码, 参数: [kccocx_01], [], [], [], [], [], [], []
ORA-00600: 内部错误代码, 参数: [25015], [8], [6], [5], [], [], [], []
ORA-00600: 内部错误代码, 参数: [25013], [0], [8], [ENMO], [ENMO], [5], [6], []

Mon Aug 09 16:17:34 2010
ORACLE Instance enmo (pid = 8) - Error 600 encountered while recovering transaction
(6, 42).
Mon Aug 09 16:17:34 2010
Errors in file d:\oracle\admin\enmo\bdump\enmo_smon_4824.trc:
ORA-00600: internal error code, arguments: [25015], [8], [6], [5], [], [], [], []

Mon Aug 09 16:27:39 2010
Errors in file d:\oracle\admin\enmo\bdump\enmo_pmon_4252.trc:
ORA-00474: SMON process terminated with error

Instance terminated by PMON, pid = 4252
```

在下次启动数据库时，告警日志中出现如下错误序列，随后数据库实例 Crash，也完全符合用户的情况：

```
Mon Aug 09 17:04:28 2010
Errors in file d:\oracle\admin\enmo\bdump\enmo_smon_3808.trc:
ORA-00600: internal error code, arguments: [25015], [8], [6], [5], [], [], [], []

Mon Aug 09 17:04:30 2010
```

```
Errors in file d:\oracle\admin\enmo\bdump\enmo_smon_3808.trc:
ORA-00600: internal error code, arguments: [kccocx_01], [], [], [], [], [], [], []
ORA-00600: internal error code, arguments: [25015], [8], [6], [5], [], [], [], []

ORACLE Instance enmo (pid = 8) - Error 600 encountered while recovering transaction
(6, 42).
Mon Aug 09 17:04:30 2010
Errors in file d:\oracle\admin\enmo\bdump\enmo_smon_3808.trc:
ORA-00600: internal error code, arguments: [25015], [8], [6], [5], [], [], [], []
```

至此，我们已经完全能够解析出用户故障的来龙去脉。数据库中回滚段 6 上 Slot 42 存在一个死事务，也就是表空间创建失败后的事务无法回滚：

```
SQL> select ADDR,KTUXEUSN,KTUXESLT,KTUXESQN,KTUXESIZ,KTUXECFL
  2  from x$ktuxe where ktuxeusn=6 and ktuxeslt=42;
ADDR        KTUXEUSN   KTUXESLT   KTUXESQN   KTUXESIZ KTUXECFL
-------- ---------- ---------- ---------- ---------- --------------------
094878F0          6         42        332          1 DEAD
SQL> select distinct KTUXECFL,count(*) from x$ktuxe group by KTUXECFL;
KTUXECFL                  COUNT(*)
------------------------ ----------
DEAD                              1
NONE                            577
```

将 UNDO Header 转储出来，可以验证这个判断：

```
SQL> select * from v$rollname;
       USN NAME
---------- ------------------------------
         0 SYSTEM
         1 _SYSSMU1$
         2 _SYSSMU2$
         3 _SYSSMU3$
         4 _SYSSMU4$
         5 _SYSSMU5$
         6 _SYSSMU6$
```

```
         7 _SYSSMU7$
         8 _SYSSMU8$
         9 _SYSSMU9$
        10 _SYSSMU10$

已选择 11 行。
SQL> alter system dump undo header "_SYSSMU6$";
系统已更改。
```

从跟踪文件中可以找到 6 号回滚段的内容，其中 2a 正是 10 进制的 42，此处的活动事务不能回退导致数据库实例的挂起和故障，由于我们指定了损坏参数来打开数据库，这个回滚段头被标记为损坏：

```
Block Checking: DBA = 8388697, Block Type = System Managed Segment Header Block
ERROR: SMU Segment Header Corrupted.  Error Code = 38508
ktu4smck: starting extent(0x11) of txn slot #0x2a is  invalid.
  valid value (0 - 0x10)
*******************************************************************************
Undo Segment:  _SYSSMU6$ (6)
*******************************************************************************
  Extent Control Header
  -----------------------------------------------------------------
°°°°°°°°°
  TRN TBL::

  index  state cflags  wrap#    uel         scn            dba     nub     stmt_num
cmt
  ------------------------------------------------------------------------------
°°°°°°°°°
   0x26    9    0x00  0x014c  0x002b  0x0000.000ddc57  0x008003ee  0x00000007
1281340831
   0x27    9    0x00  0x014c  0x0029  0x0000.000ddc4a  0x00000000  0x00000000
1281340831
   0x28    9    0x00  0x014c  0x002d  0x0000.000e7d51  0x00000000  0x00000000
1281346061
   0x29    9    0x00  0x014c  0x002e  0x0000.000ddc4c  0x008003ea  0x00000001
```

```
1281340831
   0x2a   10    0x10  0x014c  0x0011  0x0000.000ddd7e  0x008003ed  0x00000001  0
   0x2b    9    0x00  0x014c  0x0028  0x0000.000e2dac  0x00000000  0x00000000
1281344675
   0x2c    9    0x00  0x014c  0x0026  0x0000.000ddc55  0x008003ed  0x00000003
1281340831
   0x2d    9    0x00  0x014c  0xffff  0x0000.000e7ebc  0x00000000  0x00000000
1281346136
   0x2e    9    0x00  0x014c  0x0025  0x0000.000ddc4f  0x00000000  0x00000000
1281340831
   0x2f    9    0x00  0x014b  0x0003  0x0000.000dcea2  0x008003de  0x00000001
1281332703
```

如果此时我们将 6 号回滚段标记为损坏，则可以避免回滚时出现的问题，正常无误的启动数据库：

```
SQL> alter system set "_corrupted_rollback_segments"='_SYSSMU6$' scope=spfile;
系统已更改。
SQL> alter system set "_offline_rollback_segments"='_SYSSMU6$' scope=spfile;
系统已更改。
SQL> shutdown immediate;
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL> startup
ORACLE 例程已经启动。

Total System Global Area  612368384 bytes
Fixed Size                  1298160 bytes
Variable Size             167772432 bytes
Database Buffers          436207616 bytes
Redo Buffers                7090176 bytes
数据库装载完毕。
数据库已经打开。
SQL> show parameter rollback
NAME                                 TYPE        VALUE
```

```
---------------------------------- ----------- ------------------------------
_corrupted_rollback_segments         string      _SYSSMU6$
_offline_rollback_segments           string      _SYSSMU6$
fast_start_parallel_rollback         string      LOW
rollback_segments                    string
transactions_per_rollback_segment    integer     5
```

如果此时尝试 DROP 回滚段，则数据库还会出现 600 错误：

```
SQL> drop rollback segment "_SYSSMU6$";
drop rollback segment "_SYSSMU6$"
*
第 1 行出现错误:
ORA-00607: 当更改数据块时出现内部错误
ORA-00600: 内部错误代码, 参数: [kddummy_blkchk], [2], [89], [38508], [], [], [], []
SQL> drop tablespace enmo;
表空间已删除。
```

告警日志信息如下，这里 kddummy_blkchk 不必检索文档，大致可以猜测是数据块检查出现问题。此处检查的文件 2，数据块 89 正是我们的 6 号回滚段：

```
Mon Aug 09 17:46:41 2010
drop rollback segment "_SYSSMU6$"
Mon Aug 09 17:46:42 2010
Errors in file d:\oracle\admin\enmo\udump\enmo_ora_2432.trc:
ORA-00600: 内部错误代码, 参数: [kddummy_blkchk], [2], [89], [38508], [], [], [], []

Mon Aug 09 17:46:43 2010
Doing block recovery for file 2 block 89
Block recovery from logseq 4, block 405 to scn 970707
Mon Aug 09 17:46:43 2010
Recovery of Online Redo Log: Thread 1 Group 1 Seq 4 Reading mem 0
  Mem# 0: D:\ORACLE\ORADATA\ENMO\REDO01.LOG
Block recovery completed at rba 4.407.16, scn 0.970708
ORA-607 signalled during: drop rollback segment "_SYSSMU6$"...
Mon Aug 09 17:46:43 2010
```

```
Corrupt Block Found
         TSN = 1, TSNAME = UNDOTBS1
         RFN = 2, BLK = 89, RDBA = 8388697
         OBJN = 0, OBJD = -1, OBJECT = C_TS#, SUBOBJECT =
         SEGMENT OWNER = SYS, SEGMENT TYPE = Cluster Segment
Mon Aug 09 17:46:43 2010
Errors in file d:\oracle\admin\enmo\bdump\enmo_smon_6092.trc:
ORA-00600: internal error code, arguments: [kddummy_blkchk], [2], [89], [38508], [],
[], [], []
```

此后通过如下一系列的处理，数据库可以成功被打开，但是根据之前的分析，我们应当知道，数据库因此被强制放弃了一些事务的一致性，最好通过导出/导入进行数据库重构：

```
SQL> alter system set "_allow_resetlogs_corruption"=true scope=spfile;
系统已更改。
SQL> shutdown immediate;
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL> startup mount;
ORACLE 例程已经启动。

Total System Global Area  612368384 bytes
Fixed Size                  1298160 bytes
Variable Size             167772432 bytes
Database Buffers          436207616 bytes
Redo Buffers                7090176 bytes
数据库装载完毕。
SQL> recover database using backup controlfile until cancel;
ORA-00279: 更改 1011115 (在 08/09/2010 17:52:04 生成) 对于线程 1 是必需的
ORA-00289: 建议: D:\ORACLE\10.2.0\RDBMS\ARC00006_0726573063.001
ORA-00280: 更改 1011115 (用于线程 1) 在序列 #6 中

指定日志: {<RET>=suggested | filename | AUTO | CANCEL}
cancel
```

介质恢复已取消。

SQL> alter database open resetlogs;

数据库已更改。

SQL> create undo tablespace undotbs2 datafile size 10M;

表空间已创建。

SQL> alter system set undo_tablespace=undotbs2;

系统已更改。

SQL> alter tablespace undotbs1 offline;

表空间已更改。

SQL> drop tablespace undotbs1;

表空间已删除。

这就是一个字符引发的灾难。

一个盘符引发的灾难

判断失误导致的误格式化故障

我们众多的案例都表明，在数据环境中，任何一个微弱的疏忽，都可能导致数据环境的万劫不复。

这个故障是关于盘符的。

灾难描述

用户是这样描述这次灾难的：

1. 一个 1TB 的核心数据库无备份
2. 客户扩展了一块 2TB 的硬盘用于备份
3. 工作在凌晨进行，安装新硬盘后重新启动
4. 分区，格式化后启动数据库
5. ASM 报错无法启动
6. 工程师介入检查，发现盘符发生变化，原有数据盘被格式化
7. 灾难形成

在这个案例发生的过程中，客户操作人员曾经询问厂商人员，是这块盘没错吧？结果厂商人员给了一个模棱两可的答复，应该没错。结果恰恰是错了。

案例警示

这个案例给我们的警示是：

1. **专业的工作应当由专业的人来完成**

与核心数据相关的工作，哪怕是再简单的工作，也没有小事情，也不容疏忽，专业技术人员的经验就是最好的回报。

所以对于数据系统的各层面维护工作，都应当尽量由专业的人员来完成专业的工作，如果不能获得专业支持，那么做好备份。

2. 在疲惫的时刻不要轻易做重要的判断

在这个案例中，当格式化硬盘前，客户曾经提问，似乎有问题，但是工程师在凌晨多个小时加班之后，下意识的期望早点完成工作，草率的未加详细验证就判断没问题，可以进行格式化，轻率判断的结果是 1T 左右的数据荡然无存。

所以我们建议，在疲惫时刻要尽量稍作判断，工作的步骤应该来自预先规划，一旦遇到存在疑问的地方，不要担心浪费时间，一定得到精确的确认后才能执行操作。

3. 重要维护应当做好测试准备工作

在核心系统中的重要操作，应当提前**做好测试工作，并且明确可能存在的故障点**，并且准备回退方案。在重要的任务点上，设置两个角色，互为备份、补充和审核，避免出现误操作的概率。

这一节我们主要说的是人的问题，机器过劳要出故障，人过劳同样会犯低级错误，所以要防范人的错误，要从工作时间、工作监督保障入手。

技术回放

在 Linux、UNIX 系统中，当系统的磁盘发生变化时，如果未作绑定，预先分配的磁盘盘符可能发生变化，引发不必要的麻烦。

以下是当时客户的磁盘情况：

```
[root@smsdb ~]# fdisk -l
Disk /dev/sda: 145.9 GB, 145999527936 bytes
255 heads, 63 sectors/track, 17750 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1         129     1036161   83  Linux
/dev/sda2             130        7013    55295730   83  Linux
/dev/sda3            7014        9563    20482875   83  Linux
```

```
/dev/sda4            9564       17750    65762077+   5  Extended
/dev/sda5            9564       10825    10136983+  83  Linux
/dev/sda6           10826       12087    10136983+  83  Linux
/dev/sda7           12088       13094     8088696   83  Linux
/dev/sda8           13095       13223     1036161   83  Linux
/dev/sda9           13224       15312    16779861   82  Linux swap

Disk /dev/sdb: 968 MB, 968884224 bytes
30 heads, 62 sectors/track, 1017 cylinders
Units = cylinders of 1860 * 512 = 952320 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1        1017      945779   83  Linux

Disk /dev/sdc: 897 MB, 897581056 bytes
28 heads, 62 sectors/track, 1009 cylinders
Units = cylinders of 1736 * 512 = 888832 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdc1               1        1009      875781   83  Linux

Disk /dev/sdd: 779 MB, 779091968 bytes
24 heads, 62 sectors/track, 1022 cylinders
Units = cylinders of 1488 * 512 = 761856 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdd1               1        1022      760337   83  Linux

Disk /dev/sde: 664 MB, 664797184 bytes
21 heads, 61 sectors/track, 1013 cylinders
Units = cylinders of 1281 * 512 = 655872 bytes

   Device Boot      Start         End      Blocks   Id  System
```

```
/dev/sde1               1        1013      648796   83  Linux
```

注意，这里的 /dev/sdf 就是后增加的 1 T 左右的硬盘，由于盘符变化，这个硬盘插入到原来的盘序中间：

```
Disk /dev/sdf: 1209.4 GB, 1209462790144 bytes
255 heads, 63 sectors/track, 147042 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdf1               1      147042  1181114833+  83  Linux

Disk /dev/sdg: 284.5 GB, 284538961920 bytes
255 heads, 63 sectors/track, 34593 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdg1               1       34593   277868241   83  Linux
```

注意，这里的 /dev/sdh 是原有的数据盘之一，客户缺省的认为新增加的硬盘应该位于最后一个盘符，将这个 898 G 的硬盘格式化了：

```
Disk /dev/sdh: 898.3 GB, 898388459520 bytes
255 heads, 63 sectors/track, 109222 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdh1               1      109222   877325683+  83  Linux
```

这个数据库灾难的后续恢复非常复杂，由于 Oracle ASM 直接使用的是裸设备，客户使用了两块硬盘构建磁盘组，现在损失了一块硬盘，但是另外一块完好，ASM 的 AU 是在两块硬盘上均衡分布的，通过残存硬盘和格式化硬盘进行数据拼接和重组，恢复出了大部分数据，但是被文件系统 inode 块覆盖的数据就彻底丢失了，这是一次不完全恢复，恢复率大约为 99%左右。只能以此作为最大限度进行数据挽救。

物尽其用，人尽其才

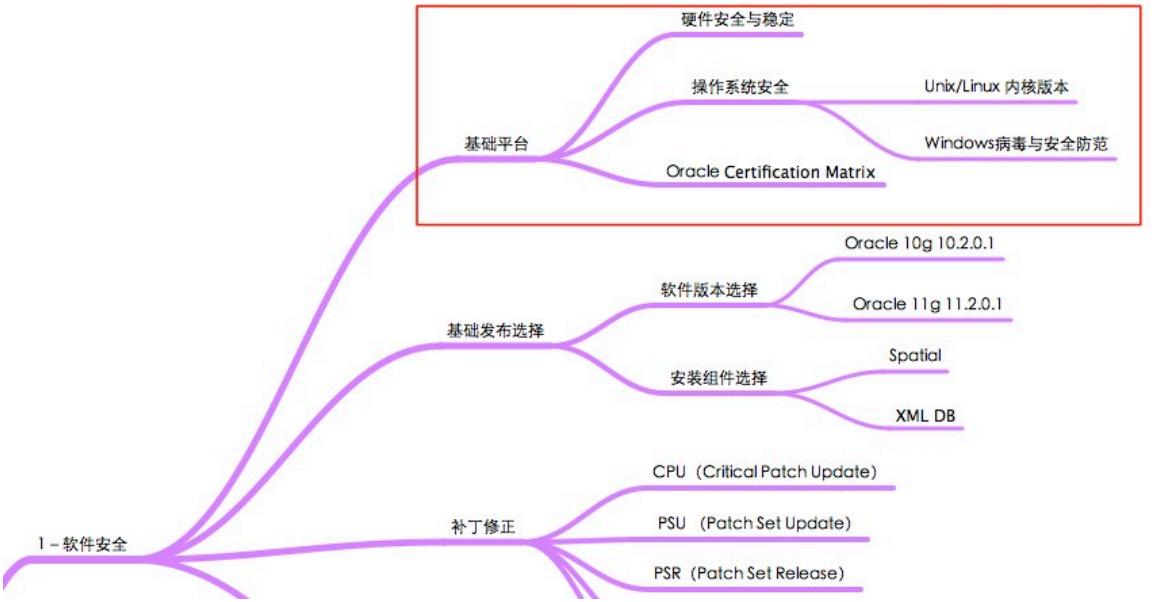
若乃人尽其才；悉用其力。

——《诗经・大雅・荡》

夫尺有所短，寸有所长，物有所不足。

智有所不明，数有所不逮，神有所不通。

——战国・楚・屈原《卜居》



在现实中，通常我们希望能够达到人尽其才，物尽其用的境界。

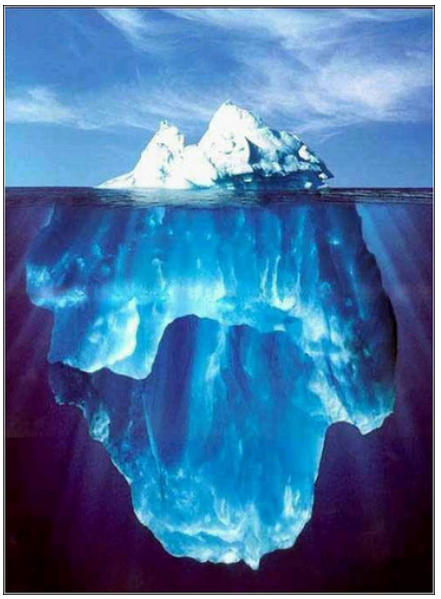
当我们使用了 Oracle 数据库时，你应当了解最为基本的数据库知识，如果你认同将重要的数据存储于数据库当中，那么最基本的了解是对自我数据的负责。虽然 Oracle 数据库经过了 30 年的发展完善，在自动化管理和优化方面已经有了大幅提升，但是最为基本的维护和“保养”仍然需要我们主动去完成。

本篇内容涉及基本的管理安全与基础平台安全，硬件的安全稳定对于数据库系统具有支撑和决定作用。

在整个数据库系统中，我们应当对使用的各类技术不足有所认知，尺有所短，物有不足，只有扬长避短才能充分发挥系统的优势，为业务发展提供有利支撑。

很多企业在选择数据库时，仅仅看到了软件的购置成本，完全不考虑软件的维护、优化等隐性成本，这是完全错误的做法，就如同购买一辆汽车一样，如果不进行定期保养维护，我们是无法对车辆的长期稳定行驶做出确保的。

对于数据库系统，我们不应当仅仅看到冰山一角，我们要时刻认识到，在水面之下，还有很多需要我们思考的问题存在，忽略这些问题就如同一叶障目，可能使我们的数据随时遭遇风险，下图正是冰山之一角，与大家共为警醒的是水面之下，我们更应该关注的部分：



本篇我们将与大家分享关于冰山的几个角落。

关库与关机

强制关机导致的写丢失故障

以下是一则与责任有关的数据库灾难案例，用户甚至对于数据库最为基本的启动和关闭常识都了解不足，几乎一无所知的管理方式最终使得数据库陷入了困境。

灾难描述

在这个案例中，用户从来都使用关机替代关库的方式进行管理，以下是案例的主要描述：

1. 用户数据库在主机重启后无法启动
2. 经检查发现用户半年多来，从来没有正常关闭过数据库
3. 用户的关库就意味着关机
4. 在最近的关机重启后数据库无法启动
5. 数据库提示控制文件不一致，数据库需要恢复
6. 灾难形成

这则案例看起来用户太缺少数据库维护的基本常识了，通过简单的培训和学习，获得基本的数据库运维知识，这是对于数据库最基本的“尊重”和重视。

案例警示

这一则案例给我们的警示是：

1. 尊重数据库即是保护数据资产

企业以 Oracle 承载数据，即是以 Oracle 作为重要的数据资产存储地。在 IT 技术快速发展的今天，企业对于数据的依赖越来越强，尤其是电子商务企业，**数据即是企业的命脉**。

根据 2007 年 Gartner 的一组调查数据显示：在经历了数据完全丢失而导致系统停运的企业中，有 2/5 再也没能恢复运营，余下的企业也有 1/3 在两年内宣告破产。由此可见数据灾难对于企业的影响是多么巨大而深远。

另据报告显示，公开披露数据丢失的企业预计将导致其客户量及相关收入降低 8%；对于上市企业而言，每股股价会下降 8%；平均每丢失一个客户记录便会造成 100 美元的额外损失。

由此可见，数据即资产，数据即财富，如果用户认同自我数据的价值，那么必须对数据库持有一定的敬畏和尊重，至少需要了解数据库最为基本的启动关闭步骤和最简单常规的备份方式。

如果完全忽视数据库的工作机制与原理，则数据安全是完全没有保障的。

2. 必要的培训和制度规范需要严格遵守

企业在部署数据库软件时，必须同时储备数据运维人才或者通过培训获得基本的运维技能，技术与维护相配合才能够充分发挥 IT 系统的价值。

目前很多企业将培训视为过场，不能够从培训中获得必要的技能，很多集成商也将培训视为可有可无。

我们认为，负责任的企业，应当将最为核心和基本的数据维护技能，通过简单的方式方法传达给用户，并且将生产库的重要操作步骤规章制度化，帮助用户减少问题出现的概率可能，唯有如此，才能够保证最低层次的数据安全。

尊重与重视，使得数据库能够物尽其用，这是数据库应用的最基本原则。

恢复过程

以下是从告警日志中分析得来的一些数据，用户的数据库自 2011 年 6 月创建，以模板方式建立：

```
bogon: eygle$ strings alert_sxxhdts.log |head -5
Sun Jun 12 18:30:33 2011
alter database rename global_name to sxxhdts
Completed: alter database rename global_name to sxxhdts
```

半年多间，数据库有 326 次启动信息：

```
bogon: eygle$ strings alert_sxxhdts.log |grep "Starting ORACLE instance (normal)"|wc -l
      326
```

但是数据库从未正常关闭过，以下正常关闭的信息，来自创建数据库时：

```
bogon: eygle$ strings alert_sxxhdts.log |grep "Shutting down instance"
Shutting down instance: further logons disabled
```

```
Shutting down instance (normal)
```

控制文件一致性维护

我们尝试启动这个数据库，首先报出的是控制文件不一致，其中第三个控制文件版本比较新，版本号为 2623：

```
SQL> startup pfile=initora9i.ora
ORACLE instance started.

Total System Global Area  126950956 bytes
Fixed Size                   454188 bytes
Variable Size              92274688 bytes
Database Buffers           33554432 bytes
Redo Buffers                 667648 bytes
ORA-00214: controlfile 'D:\ORACLE\ORADATA\SXXHDTS\CONTROL03.CTL' version 2623
inconsistent with file 'D:\ORACLE\ORADATA\SXXHDTS\CONTROL02.CTL' version 2619
```

通常 Oracle 的三个控制文件是互为镜像的，其内容完全相同，Oracle 通过并行写入来更新控制文件，如果出现不一致现象，那么意味着存储的写入出现丢失，损失了物理 IO。

我们稍微来解析一下控制文件的更新维护过程，以下通过 dbms_monitor 来跟踪一下 CKPT 后台进程对于控制文件的维护操作：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle Database 10g Enterprise Edition Release 10.2.0.5.0 - 64bi
SQL> select sid,serial#,program from v$session where program like '%CKPT%';
       SID    SERIAL# PROGRAM
---------- ---------- ------------------------------------------------
    55       1 oracle@hpserver2.enmotech.com (CKPT)
SQL> select spid,program from v$process where program like '%CKPT%';
SPID      PROGRAM
------------ ------------------------------------------------
6168      oracle@hpserver2.enmotech.com (CKPT)
SQL> exec dbms_monitor.session_trace_enable(55,1,TRUE,TRUE);
```

```
PL/SQL procedure successfully completed.
SQL> alter system checkpoint;
System altered.
SQL> exec dbms_monitor.session_trace_disable;
PL/SQL procedure successfully completed.
```

在 CKPT 进程的跟踪文件中，可以找到执行 Checkpoint 检查点后，对于控制文件的写操作，以下是摘录信息：

```
WAIT #0: nam='db file sequential read' ela= 18 file#=1 block#=1 blocks=1 obj#=-1
tim=1297696398228494
WAIT #0: nam='db file single write' ela= 4952 file#=1 block#=1 blocks=1 obj#=-1
tim=1297696398233526
WAIT #0: nam='db file sequential read' ela= 24 file#=2 block#=1 blocks=1 obj#=-1
tim=1297696398233664
WAIT #0: nam='db file single write' ela= 4206 file#=2 block#=1 blocks=1 obj#=-1
tim=1297696398237931
WAIT #0: nam='db file sequential read' ela= 24 file#=3 block#=1 blocks=1 obj#=-1
tim=1297696398238070
WAIT #0: nam='db file single write' ela= 9713 file#=3 block#=1 blocks=1 obj#=-1
tim=1297696398247844
WAIT #0: nam='db file sequential read' ela= 19 file#=4 block#=1 blocks=1 obj#=-1
tim=1297696398247979
WAIT #0: nam='db file single write' ela= 4774 file#=4 block#=1 blocks=1 obj#=-1
tim=1297696398252817
WAIT #0: nam='db file sequential read' ela= 19 file#=5 block#=1 blocks=1 obj#=-1
tim=1297696398252949
WAIT #0: nam='db file single write' ela= 9004 file#=5 block#=1 blocks=1 obj#=-1
tim=1297696398262018
WAIT #0: nam='db file sequential read' ela= 19 file#=6 block#=1 blocks=1 obj#=-1
tim=1297696398262214
WAIT #0: nam='db file single write' ela= 11076 file#=6 block#=1 blocks=1 obj#=-1
tim=1297696398273357
WAIT #0: nam='control file parallel write' ela= 20004 files=3 block#=12 requests=3
obj#=-1 tim=1297696398293494
```

```
WAIT #0: nam='control file parallel write' ela= 24585 files=3 block#=15 requests=3
obj#=-1 tim=1297696398318200
WAIT #0: nam='control file parallel write' ela= 23752 files=3 block#=10 requests=3
obj#=-1 tim=1297696398342077
WAIT #0: nam='control file parallel write' ela= 24042 files=3 block#=8 requests=3
obj#=-1 tim=1297696398366275
WAIT #0: nam='control file parallel write' ela= 23531 files=3 block#=1 requests=3
obj#=-1 tim=1297696398389955
```

注意后台的控制文件并行写（control file parallel write）操作就是检查点执行时，CKPT 进程更新控制文件的过程，这一等待事件的三个参数内容如下：

```
SQL> select name,parameter1,parameter2,parameter3
  2  from v$event_name where name ='control file parallel write';
NAME                 PARAMETER1           PARAMETER2 PARAMETER3
---------------------------- ------------------------------ ---------- ----------
control file parallel write    files                   block#  requests
```

在跟踪文件显示的参数内容分别如下，其中 files 和 Requests 的参数值相同，请求 3 次 IO 操作，更新了 3 个控制文件，块号显示更新的控制文件块：

等待事件	Files	Block#	Requests
control file parallel write	3	12	3
control file parallel write	3	15	3
control file parallel write	3	10	3
control file parallel write	3	8	3
control file parallel write	3	1	3

如果这发出的三次 IO 丢失了一个或者两个，则控制文件不一致的现象就可能出现了。

继续前面的案例分析，我们可以跟踪一下数据库启动这个过程：

```
SQL> startup nomount pfile=initora9i.ora
ORACLE instance started.

Total System Global Area  126950956 bytes
Fixed Size                   454188 bytes
Variable Size              92274688 bytes
```

```
Database Buffers           33554432 bytes
Redo Buffers                 667648 bytes
SQL> alter session set events '10046 trace name context forever,level 12';
Session altered.
SQL> alter database mount;
alter database mount
*
ERROR at line 1:
ORA-00214: controlfile 'D:\ORACLE\ORADATA\SXXHDTS\CONTROL03.CTL' version 2623
inconsistent with file 'D:\ORACLE\ORADATA\SXXHDTS\CONTROL02.CTL' version 2619
```

查看后台生成的 10046 跟踪文件，摘录部分信息：

```
*** SESSION ID:(9.1) 2012-01-10 13:16:41.183
APPNAME mod='sqlplus.exe' mh=0 act='' ah=0
=====================
PARSING IN CURSOR #1 len=68 dep=0 uid=0 oct=42 lid=0 tim=12373534530 hv=1346161232
ad='6a3c9d04'
alter session set events '10046 trace name context forever,level 12'
END OF STMT
EXEC #1:c=0,e=32,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=12373520716
WAIT #1: nam='SQL*Net message to client' ela= 6 p1=1111838976 p2=1 p3=0
WAIT #1: nam='SQL*Net message from client' ela= 4508814 p1=1111838976 p2=1 p3=0
XCTEND rlbk=0, rd_only=1
=====================
PARSING IN CURSOR #1 len=20 dep=0 uid=0 oct=35 lid=0 tim=12378053544 hv=1379354989
ad='6a3c8d64'
alter database mount
END OF STMT
PARSE #1:c=0,e=3026,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=12378053538
BINDS #1:
WAIT #1: nam='reliable message' ela= 27 p1=1760935332 p2=1760905580 p3=1761534860
WAIT #1: nam='rdbms ipc reply' ela= 2724 p1=5 p2=900 p3=0
WAIT #1: nam='control file sequential read' ela= 254 p1=0 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 219 p1=1 p2=1 p3=1
```

```
WAIT #1: nam='control file sequential read' ela= 208 p1=2 p2=1 p3=1
WAIT #1: nam='reliable message' ela= 996893 p1=1760935332 p2=1760905876 p3=1761534860
WAIT #1: nam='reliable message' ela= 999155 p1=1760935332 p2=1760905876 p3=1761534860
WAIT #1: nam='reliable message' ela= 990131 p1=1760935332 p2=1760905876 p3=1761534860
EXEC #1:c=0,e=3010793,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=12381070548
ERROR #1:err=214 tim=1237671
WAIT #1: nam='SQL*Net break/reset to client' ela= 5 p1=1111838976 p2=1 p3=0
WAIT #1: nam='SQL*Net break/reset to client' ela= 83 p1=1111838976 p2=0 p3=0
WAIT #1: nam='SQL*Net message to client' ela= 6 p1=1111838976 p2=1 p3=0
```

我们可以看到数据库在 Mount 过程中，顺序读取了 3 个控制文件的第一个数据块，进行比较，如果控制文件的版本不一致，就会抛出前面的异常。以下是一个控制文件的转储，头块上记录的 Seq 就是控制文件的版本号，控制文件的校验仅读取这个信息就可以完成：

```
*** 2012-01-10 11:13:27.919
*** SESSION ID:(9.3) 2012-01-10 11:13:27.868
DUMP OF CONTROL FILES, Seq # 2626 = 0xa42
 FILE HEADER:
    Software vsn=153092096=0x9200000, Compatibility Vsn=134217728=0x8000000
    Db ID=615401347=0x24ae4783, Db Name='SXXHDTS'
    Activation ID=0=0x0
    Control Seq=2626=0xa42, File size=246=0xf6
    File Number=0, Blksiz=8192, File Type=1 CONTROL
```

Oracle 的文件恢复判断

正常情况下，Oracle 的控制文件是完全相同的三个拷贝，由于出现不一致现象，三号控制文件较新，所以我们使用该控制文件来尝试启动数据库。

这时数据库提示 UNDO 文件需要恢复：

```
SQL> startup pfile=initora9i.ora
ORACLE instance started.

Total System Global Area  126950956 bytes
Fixed Size                   454188 bytes
```

```
Variable Size              92274688 bytes
Database Buffers           33554432 bytes
Redo Buffers                 667648 bytes
Database mounted.
ORA-01113: file 2 needs media recovery
ORA-01110: data file 2: 'D:\ORACLE\ORADATA\SXXHDTS\UNDOTBS01.DBF'
```

首先我们来分析一下 Oracle 如何判断 UNDOTBS01.DBF 文件需要恢复，通过 10046 事件跟踪一下数据库 Open 过程：

```
SQL> alter session set events '10046 trace name context forever,level 12';
Session altered.
SQL> alter database open;
alter database open
*
ERROR at line 1:
ORA-01113: file 2 needs media recovery
ORA-01110: data file 2: 'D:\ORACLE\ORADATA\SXXHDTS\UNDOTBS01.DBF'
```

查看跟踪文件获得如下信息：

```
*** 2012-01-10 11:49:02.328
APPNAME mod='sqlplus.exe' mh=0 act='' ah=0
=====================
PARSING IN CURSOR #1 len=68 dep=0 uid=0 oct=42 lid=0 tim=7114430174 hv=1346161232
ad='6a3cbec0'
alter session set events '10046 trace name context forever,level 12'
END OF STMT
EXEC #1:c=0,e=29,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=7114401350
WAIT #1: nam='SQL*Net message to client' ela= 7 p1=1111838976 p2=1 p3=0
*** 2012-01-10 11:49:17.259
WAIT #1: nam='SQL*Net message from client' ela= 14920900 p1=1111838976 p2=1 p3=0
XCTEND rlbk=0, rd_only=1
=====================
PARSING IN CURSOR #1 len=19 dep=0 uid=0 oct=35 lid=0 tim=7129363012 hv=2631704207
ad='6a3c6d48'
```

```
alter database open
END OF STMT
PARSE #1:c=0,e=205,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=7129363006
BINDS #1:
WAIT #1: nam='control file sequential read' ela= 745 p1=0 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 148 p1=1 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 142 p1=2 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 144 p1=0 p2=239 p3=1
WAIT #1: nam='control file sequential read' ela= 147 p1=0 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 155 p1=1 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 144 p1=2 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 144 p1=0 p2=239 p3=1
WAIT #1: nam='rdbms ipc reply' ela= 90731 p1=3 p2=910 p3=0
WAIT #1: nam='control file sequential read' ela= 188 p1=0 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 147 p1=1 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 144 p1=2 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 141 p1=0 p2=239 p3=1
WAIT #1: nam='control file sequential read' ela= 158 p1=0 p2=12 p3=1
WAIT #1: nam='direct path read' ela= 23 p1=1 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=2 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 5 p1=3 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=4 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=5 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=6 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 5 p1=7 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=8 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 7 p1=9 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=10 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=201 p2=1 p3=1
WAIT #1: nam='rdbms ipc reply' ela= 171 p1=3 p2=2147483647 p3=0
EXEC #1:c=10014,e=101420,p=11,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=7129464473
ERROR #1:err=1113 tim=712510
WAIT #1: nam='SQL*Net break/reset to client' ela= 5 p1=1111838976 p2=1 p3=0
```

```
WAIT #1: nam='SQL*Net break/reset to client' ela= 63 p1=1111838976 p2=0 p3=0
WAIT #1: nam='SQL*Net message to client' ela= 4 p1=1111838976 p2=1 p3=0
```

对以上跟踪信息进行一点深入分析，以下一段通过'control file sequential read'读取控制文件中的数据块：

```
WAIT #1: nam='control file sequential read' ela= 745 p1=0 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 148 p1=1 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 142 p1=2 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 144 p1=0 p2=239 p3=1
WAIT #1: nam='control file sequential read' ela= 147 p1=0 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 155 p1=1 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 144 p1=2 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 144 p1=0 p2=239 p3=1
WAIT #1: nam='rdbms ipc reply' ela= 90731 p1=3 p2=910 p3=0
WAIT #1: nam='control file sequential read' ela= 188 p1=0 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 147 p1=1 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 144 p1=2 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 141 p1=0 p2=239 p3=1
WAIT #1: nam='control file sequential read' ela= 158 p1=0 p2=12 p3=1
```

等待事件 control file sequential read 包含三个参数输出，这三个参数分别代表文件号，块号和读取的块的数量：

```
SQL> select name,parameter1,parameter2,parameter3
  2  from v$event_name where name ='control file sequential read';

NAME                           PARAMETER1           PARAMETER2           PARAMETER3
------------------------------ -------------------- -------------------- -----------
control file sequential read   file#                block#               blocks
```

以上输出说明启动过程中，数据库读取了控制文件的第 1、12、239 块。再然后，读取了每个数据文件的第一个数据块：

```
WAIT #1: nam='direct path read' ela= 23 p1=1 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=2 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 5 p1=3 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=4 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=5 p2=1 p3=1
```