第2章 控制文件与数据库初始化

在上一章中我们探讨了数据库的启动和关闭过程，在这一过程中，Oracle 的控制文件起着极其重要的作用，我习惯打的一个比喻是：**控制文件是数据库的大脑，而 SYSTEM 表空间是数据库的心脏**。在这一章里，我们将继续对控制文件以及数据库的初始化过程进行进一步的探讨。

2.1 控制文件的内容

既然控制文件在数据库中扮演着重要的角色，那么控制文件中到底存储了哪些重要信息，在数据库运行过程中又是如何发挥重要作用的呢？

首先从文档上得知控制文件中保存着下列信息：

- 数据库名称以及数据库创建时间等
- 所有数据文件和重做日志文件的名称和位置信息
- 表空间信息
- OFFLINE 数据文件信息
- 重做日志及归档日志信息
- 备份集及备份文件信息、
- 检查点（checkpoint）及 SCN 信息等

当然这些只是一个粗略的介绍，由于控制文件是个二进制文件，无法直接打开查阅，但是通过上一章介绍的如下命令可以将控制文件内容转储出来便于查看：

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

以下是来自 Oracle Database 11g 的转储测试：

```
SQL> alter session set events 'immediate trace name controlf level 8';
Session altered.
SQL> select value from v$diag_info where name='Default Trace File';
VALUE
--------------------------------------------------------------------------------
/opt/oracle/diag/rdbms/11gtest/11gtest/trace/11gtest_ora_5910.trc
```

注意：从 11g 开始，可以通过 v$diag_info 获得当前会话转储文件的名称。

打开这个跟踪文件现在就可以清晰的看到控制文件的内容，最开始的一段是关于数据库 ID、名称等的概要信息：

```
V10 STYLE FILE HEADER:
        Compatibility Vsn = 185597952=0xb100000
        Db ID=1478080230=0x5819b6e6, Db Name='11GTEST'
```

```
        Activation ID=0=0x0
        Control Seq=1707=0x6ab, File size=594=0x252
        File Number=0, Blksiz=16384, File Type=1 CONTROL
```

接下来是数据库条目的详细信息，包括了数据库的名称、数据文件及日志文件的数量、数据库的检查点及 SCN 信息等：

```
*******************************************************************************
DATABASE ENTRY
*******************************************************************************
 (size = 316, compat size = 316, section max = 1, section in-use = 1,
  last-recid= 0, old-recno = 0, last-recno = 0)
 (extent = 1, blkno = 1, numrecs = 1)
 07/04/2008 13:51:34
 DB Name "11GTEST"
 Database flags = 0x00404001 0x00001200
 Controlfile Creation Timestamp  07/04/2008 13:51:35
 Incmplt recovery scn: 0x0000.00000000
 Resetlogs scn: 0x0000.00000001 Resetlogs Timestamp  07/04/2008 13:51:34
 Prior resetlogs scn: 0x0000.00000000 Prior resetlogs Timestamp  01/01/1988 00:00:00
 Redo Version: compatible=0xb100000
 #Data files = 4, #Online files = 4
 Database checkpoint: Thread=1 scn: 0x0000.0008718a
 Threads: #Enabled=1, #Open=1, Head=1, Tail=1
 enabled  threads:  01000000 00000000 00000000 00000000 00000000 00000000
....
 Max log members = 3, Max data members = 1
 Arch list: Head=1, Tail=1, Force scn: 0x0000.0007fe17scn: 0x0000.0008ebd0
 Activation ID: 1478075366
 Controlfile Checkpointed at scn:  0x0000.0008ec14 07/07/2008 08:00:29
 thread:0 rba:(0x0.0.0)
 enabled  threads:  00000000 00000000 00000000 00000000 00000000 00000000
....
```

再接下来是检查点记录信息,这部分内容包含了 Low Cache RBA 和 On Disk RBA 信息,在执行数据库实例恢复时,前者是恢复的起点,后者是恢复的终点,其分别指向了日志文件中的确定地址:

```
*******************************************************************************
CHECKPOINT PROGRESS RECORDS
*******************************************************************************
 (size = 8180, compat size = 8180, section max = 11, section in-use = 0,
  last-recid= 0, old-recno = 0, last-recno = 0)
```

```
 (extent = 1, blkno = 2, numrecs = 11)
THREAD #1 - status:0x2 flags:0x0 dirty:688
low cache rba:(0x1b.16c04.0) on disk rba:(0x1c.a1c.0)
on disk scn: 0x0000.0008ed61 07/07/2008 08:01:59
resetlogs scn: 0x0000.00000001 07/04/2008 13:51:34
heartbeat: 659440589 mount id: 1478276654
```

好了，我们先引用到这里，大家在学习时应该仔细阅读接下来的每个条目。

在上面的引用中，已经频繁出现了 checkpoint 和 SCN 信息，Oracle 数据库在内部通过 SCN 和检查点来保证数据库的一致性、可恢复性等重要属性，下面就让我们来详细了解一下 Oracle 的 SCN 与 检查点机制。

2.2　SCN 的说明

SCN 在 Oracle 的文档上以多种形式出现，一种是 System Change Number，一种是 System Commit Number，在大多数情况下，Systems Change Numbers 的定义更为确切。

2.2.1 SCN 的定义

SCN(System Change Number)，也就是通常我们所说的系统改变号，是数据库中非常重要的一个数据结构，用以标识数据库在某个确切时刻提交的版本。在事务提交时，它被赋予一个唯一的标示事务的 SCN。SCN 同时被作为 Oracle 数据库的内部时钟机制，可以被看作逻辑时钟，每个数据库都有一个全局的 SCN 生成器。

作为数据库内部的逻辑时钟，数据库事务依 SCN 而排序，Oracle 也依据 SCN 来实现一致性读(Read Consistency)等重要数据库功能，另外对于分布式事务（Distributed Transactions），SCN 也极为重要。

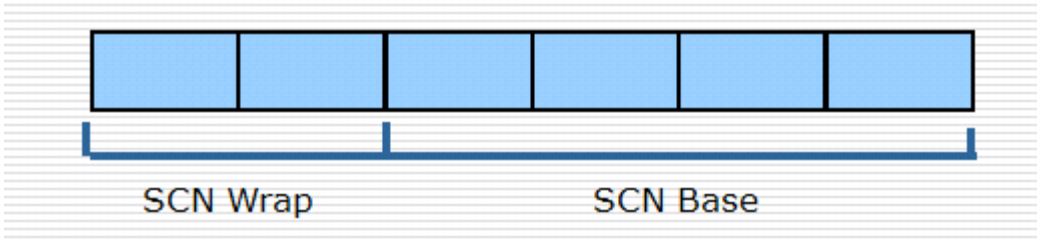
SCN 在数据库中是唯一的，并随时间而增加，但是可能并不连贯。除非重建数据库，SCN 的值永远不会被重置为 0。

一直以来，对于 SCN 有很多争议，很多人认为 SCN 是指，System Commit Number，而通常 SCN 在提交时才变化，所以很多时候，这两个名词经常在文档中反复出现。即使在 Oracle 的官方文档中，SCN 也常以 System Change/Commit Number 两种形式出现。

到底是哪个词其实不是最重要的，重要的是我们需要知道 SCN 是 Oracle 内部的时钟机制，Oracle 通过 SCN 来维护数据库的一致性，并通过 SCN 实施 Oracle 至关重要的恢复机制。

SCN 在数据库中是无处不在的，常见的事务表、控制文件、数据文件头、日志文件、数据块头等都记录有 SCN 值。冠以不同前缀，SCN 也有了不同的名称，比如检查点 SCN（checkpoint scn）, Resetlogs SCN 等等。

SCN 由两部分组成，高位 SCN Wrap 由 2 Bytes 记录，低位 SCN Base 由 4 Bytes 记录：



SCN 的 6 Bytes 记录,理论上可以存储 281 trillion (兆)的数值,这是 SCN 的极限值:

```
SQL> select power(2,48) from dual;

 POWER(2,48)
--------------------
     281474976710656
```

虽然 SCN 理论上可以容纳如此大的值，但是为了控制 SCN 的异常增长，Oracle 也做出了一些限制，在 Oracle 11g 以前，每秒增进的 SCN 值不能超过 16K，这个数字在 11g 中增加到 32K/秒，最大值可以达到 256K/秒。据此计算,在 11g 之前,SCN 至少可以用 500 年左右:

```
SQL> select trunc(power(2,48)/12/31/24/3600/16/1024,2) Year from dual;

     YEAR
--------------------
      534.51
```

基于每秒产生 SCN 的限制,SCN 在任意时间的最大允许值就可以被计算出来.

以下两个参数在 11g 中引入以控制 SCN 的可能合理值，_max_reasonable_scn_rate 用于限制每秒最大产生 SCN 的数量，_reasonable_scn_offset_seconds 用于设定一个用于计算时间的偏移量：

```
_max_reasonable_scn_rate       32768                Max reasonable SCN rate
_reasonable_scn_offset_seconds 0                    Reasonable SCN offset seconds
```

最大可能 SCN 的计算是基于一个固定时间，Oracle 内部使用了一个 4G 范围的数据来表示 01/01/1988 00:00:00 ~ 08/18/2121 06:28:15 这段时间,它的算法简单,每个月都是用 31 天来表示时间,每增加 1 秒,这个数值就增加 1,有了这个时间起点，再加上每秒允许产生 16384 个 SCN（11g 之前），就可以计算当前最大的允许 SCN：

```
col scn for 999,999,999,999,999,999
select
(
    (
        (
            (
                (
                    (
                        to_char(sysdate,'YYYY')-1988
                    )*12+
```

```
                to_char(sysdate,'mm')-1
                )*31+to_char(sysdate,'dd')-1
            )*24+to_char(sysdate,'hh24')
        )*60+to_char(sysdate,'mi')
    )*60+to_char(sysdate,'ss')
) * to_number('ffff','XXXXXXXX')/4 scn
from dual
/
SYSDATE                         SCN
--------- ------------------------
16-JUN-12       12,879,847,825,800
```

当数据库的 SCN 超过合理值意外增长后，将会出现 ORA-00600 2552 错误。以下信息就是在数据库出现 SCN 异常之后抛出的警告:

```
Mon May 14 17:55:54 2012
Errors in file /t3/orat3/product/admin/ora1020410/bdump/ora1020410_mmon_25386.trc:
ORA-00600: internal error code, arguments: [2252], [2988], [9], [], [], [], [], []
Mon May 14 17:56:06 2012
Errors in file /t3/orat3/product/admin/ora1020410/bdump/ora1020410_smon_23805.trc:
ORA-00600: internal error code, arguments: [2252], [2988], [13], [], [], [], [], []
```

SCN 的大小问题在很长时间内并未得到关注,但是自 2012 年初,Oracle 发布了一个重要的补丁修正,声明数据库可能遇到 SCN 异常增长导致耗尽的问题,这一问题应当引起关注.

2.2.2 SCN 的获取方式

可以通过如下几种方式获得数据库的当前或近似 SCN。

1．从 Oracle9i 开始

可以通过可以使用 dbms_flashback.get_system_change_number 来获得

```
SQL> select dbms_flashback.get_system_change_number from dual;
GET_SYSTEM_CHANGE_NUMBER
------------------------
                 2982184
```

2．Oracle9i 前

可以通过查询 x$ktuxe 获得系统最接近当前值的 SCN：

X$KTUXE----------[**K**]ernel [**T**]ransaction [**U**]ndo Transa[**x**]tion [**E**]ntry (table)

```
SQL> select max(ktuxescnw*power(2,32)+ktuxescnb) from x$ktuxe;
MAX(KTUXESCNW*POWER(2,32)+KTUXESCNB)
------------------------------------
                             2980613
```

3.从 Oracle10g 开始

在 v$database 视图中增加了 current_scn 字段，通过查询该字段可以获得数据库的当前 SCN 值：

```
SQL> select current_scn from v$database;
CURRENT_SCN
-----------
     612842
```

4.从内存中取得 SCN 信息

通过 oradebug 工具可以直接读取内存中用于记录 SCN 的内存变量：

```
SQL> oradebug setmypid
Statement processed.
SQL> oradebug DUMPvar SGA kcsgscn_
kcslf kcsgscn_ [2000C848, 2000C868) = 0000000 000959EA 00000000 00000000 00000000 00000000
00000000 2000C654
SQL> select to_number('959EA','xxxxxx') SCN from dual;
SCN
---------------------------
                     612842
```

2.2.3 SCN 的进一步说明

系统当前 SCN 并不是在任何的数据库操作发生时都会改变，SCN 通常在事务提交或回滚时改变,在控制文件,数据文件头,数据块,日志文件头,日志文件 change vector 中都有 SCN,但其作用各不相同。

1．数据文件头中包含了该数据文件的检查点信息

其中包括 Checkpoint SCN,表示该数据文件最近一次执行检查点操作时的 SCN。

从控制文件的 dump 文件中，我们可以得到以下内容:

```
DATA FILE #1:
  (name #4) /opt/oracle/oradata/conner/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 01/01/1988 00:00:00
 Checkpoint cnt:273 scn: 0x0000.0023aff1 11/22/2004 17:10:11
 Stop scn: 0xffff.ffffffff 11/22/2004 16:58:49
 Creation Checkpointed at scn:  0x0000.00000008 10/20/2004 20:59:35
 thread:1 rba:(0x1.3.10)
。。。。。
```

对于每一个数据文件都包含一个这样的条目，记录该文件的检查点 SCN 的值以及检查点

发生的时间，这里的 Checkpoint SCN、Stop SCN 以及 Checkpoint Cnt 都是非常重要的数据结构，我们将会在下面检查点部分详细介绍。

同样可以通过命令转储数据文件头，观察其具体信息及检查点记录等：

```
SQL> alter session set events 'immediate trace name file_hdrs level 8';
Session altered.
SQL> @gettrcname
TRACE_FILE_NAME
------------------------------------------------------------------------------
/opt/oracle/admin/conner/udump/conner_ora_5862.trc
```

从跟踪文件中摘取 SYSTEM 表空间的记录作为参考（摘要信息）：

```
DATA FILE #1:
  (name #4) /opt/oracle/oradata/conner/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 01/01/1988 00:00:00
 Checkpoint cnt:319 scn: 0x0000.002e3016 12/03/2004 06:42:18
 Stop scn: 0xffff.ffffffff 12/01/2004 23:37:33
 Creation Checkpointed at scn:  0x0000.00000008 10/20/2004 20:59:35
 thread:1 rba:(0x1.3.10)
 Offline scn: 0x0000.001cff67 prev_range: 0
 Online Checkpointed at scn:  0x0000.001cff68 11/16/2004 14:10:35
 thread:1 rba:(0x1.2.0)
 Hot Backup end marker scn: 0x0000.00000000
 aux_file is NOT DEFINED
 FILE HEADER:
        Software vsn=153092096=0x9200000, Compatibility Vsn=134217728=0x8000000
        Db ID=3152029224=0xbbe02628, Db Name='CONNER'
        Activation ID=0=0x0
        Control Seq=1093=0x445, File size=32000=0x7d00
        File Number=1, Blksiz=8192, File Type=3 DATA
Tablespace #0 - SYSTEM  rel_fn:1
Creation   at   scn: 0x0000.00000008 10/20/2004 20:59:35
Backup taken at scn: 0x0000.001aca21 11/14/2004 09:08:34 thread:1
 reset logs count:0x20541edb scn: 0x0000.001cff68 recovered at 12/01/2004 23:07:30
 status:0x4 root dba:0x004001a1 chkpt cnt: 319 ctl cnt:318
Checkpointed at scn:  0x0000.002e3016 12/03/2004 06:42:18
 thread:1 rba:(0x35.2.10)
Backup Checkpointed at scn:  0x0000.001aca21 11/14/2004 09:08:34
 thread:1 rba:(0xc6.4fff.10)
```

注意，在以上输出中，FILE HEADER 部分之前信息来自控制文件，之后信息来自数据文件头，在数据库的启动过程中，需要依赖两部分信息进行比对判断，从而确保数据库的一致性和判断是否需要进行恢复。

2．日志文件头中包含了 Low SCN，Next SCN

这两个 SCN 标示该日志文件包含有介于 Low SCN 到 Next SCN 的重做信息，对于 Current 的日志文件（当前正在被使用的 Redo Logfile），其最终 SCN 不可知，所以 Next SCN 被置为无穷大，也就是 ffffffff。

我们来看一下日志文件的情况:

```
SQL> select * from v$log;
GROUP#  THREAD#  SEQUENCE# BYTES   MEMBERS ARC STATUS   FIRST_CHANGE# FIRST_TIM
------- -------- ---------- ---------- ---------- --- -------- ------------- ---------
      1        1         50   10485760          1 YES ACTIVE         2973017 02-DEC-04
      2        1         51   10485760          1 NO  CURRENT      2984378 02-DEC-04
      3        1         49   10485760          1 YES INACTIVE       2966611 01-DEC-04
SQL> select dbms_flashback.get_system_change_number from dual;
GET_SYSTEM_CHANGE_NUMBER
------------------------
                 2984476
SQL> alter system switch logfile;
System altered.
SQL> select * from v$log;
GROUP#  THREAD# SEQUENCE#  BYTES   MEMBERS ARC STATUS   FIRST_CHANGE# FIRST_TIM
------- -------- ---------- ---------- ---------- --- -------- ------------- ---------
      1        1         50   10485760          1 YES INACTIVE       2973017 02-DEC-04
      2        1         51   10485760          1 YES INACTIVE       2984378 02-DEC-04
      3        1         52   10485760          1 NO  CURRENT      2984481 02-DEC-04
```

我们看到，SCN **2984476** 显然位于 Log Group#为 2 的日志文件中，该日志文件包含了 SCN 自 **2984378** 至 **2984481** 的 redo 信息。Oracle 在进行恢复时就需要根据低 SCN 和高 SCN 来确定需要的恢复信息位于哪一个日志或归档文件中。

如果通过控制文件转储，我们可以在控制文件中找到关于日志文件的信息:

```
LOG FILE #1:
  (name #1) /opt/oracle/oradata/conner/redo01.log
 Thread 1 redo log links: forward: 2 backward: 0
 siz: 0x5000 seq: 0x00000011 hws: 0x2 bsz: 512 nab: 0x2 flg: 0x1 dup: 1
 Archive links: fwrd: 0 back: 0 Prev scn: 0x0000.0023ac36
 Low scn: 0x0000.0023afee 11/22/2004 17:10:06
 Next scn: 0x0000.0023aff1 11/22/2004 17:10:11
LOG FILE #2:
```

```
 (name #2) /opt/oracle/oradata/conner/redo02.log
 Thread 1 redo log links: forward: 3 backward: 1
 siz: 0x5000 seq: 0x00000012 hws: 0x2 bsz: 512 nab: 0x19 flg: 0x1 dup: 1
 Archive links: fwrd: 0 back: 0 Prev scn: 0x0000.0023afee
 Low scn: 0x0000.0023aff1 11/22/2004 17:10:11
 Next scn: 0x0000.0023b01e 11/22/2004 17:10:54
LOG FILE #3:
 (name #3) /opt/oracle/oradata/conner/redo03.log
 Thread 1 redo log links: forward: 0 backward: 2
 siz: 0x5000 seq: 0x00000013 hws: 0x1 bsz: 512 nab: 0xffffffff flg: 0x8 dup: 1
 Archive links: fwrd: 0 back: 0 Prev scn: 0x0000.0023aff1
 Low scn: 0x0000.0023b01e 11/22/2004 17:10:54
 Next scn: 0xffff.ffffffff 01/01/1988 00:00:00
```

从以上信息可以注意到，Log File 3 是当前的日志文件，该文件拥有的 Next SCN 为无穷大。同样我们可以通过直接 dump 日志文件的方式来进行转储:

```
SQL> select * from v$logfile;
 GROUP# STATUS   TYPE            MEMBER
------- -------- -------    ------------------------------------------------------
      1          ONLINE     /opt/oracle/oradata/conner/redo01.log
      2          ONLINE     /opt/oracle/oradata/conner/redo02.log
      3          ONLINE     /opt/oracle/oradata/conner/redo03.log
SQL> alter system dump logfile '/opt/oracle/oradata/conner/redo01.log';
System altered.
```

在 trace 文件中我们可以看到关于 SCN 的详细的内容：

```
DUMP OF REDO FROM FILE '/opt/oracle/oradata/conner/redo01.log'
 Opcodes *.*
 DBA's: (file # 0, block # 0) thru (file # 65534, block # 4194303)
 RBA's: 0x000000.00000000.0000 thru 0xffffffff.ffffffff.ffff
 SCN's scn: 0x0000.00000000 thru scn: 0xffff.ffffffff
 Times: creation thru eternity
 FILE HEADER:
        Software vsn=153092096=0x9200000, Compatibility Vsn=153092096=0x9200000
        Db ID=3152029224=0xbbe02628, Db Name='CONNER'
        Activation ID=3154332244=0xbc034a54
        Control Seq=1084=0x43c, File size=20480=0x5000
        File Number=1, Blksiz=512, File Type=2 LOG
 descrip:"Thread 0001, Seq# 0000000050, SCN 0x0000002d5d59-0x0000002d89ba"
 thread: 1 nab: 0x15be seq: 0x00000032 hws: 0x2 eot: 0 dis: 0
 reset logs count: 0x20541edb scn: 0x0000.001cff68
```

```
Low scn: 0x0000.002d5d59 12/02/2004 11:25:40
Next scn: 0x0000.002d89ba 12/02/2004 15:29:42
Enabled scn: 0x0000.001cff68 11/16/2004 14:10:35
Thread closed scn: 0x0000.002d5d59 12/02/2004 11:25:40
Log format vsn: 0x8000000 Disk cksum: 0xd79c Calc cksum: 0xd79c
```

我们不打算详细介绍具体命令的用法及更进一步的内容，因为对于一本书来说，这些内容还是太广阔了。在这里只希望给大家直观的认识，有兴趣的自然可以由此开始进一步的探索。

2.3 检查点-Checkpoint

许多文档把 Checkpoint 描述得非常复杂，为正确理解检查点带来了障碍，结果现在检查点变成了一个非常复杂的问题。

实际上，检查点只是一个数据库事件，它存在的根本意义在于**减少崩溃恢复（Crash Recovery）时间**。检查点事件由 CKPT 后台进程触发，当检查点发生时，CKPT 进程会负责通知 DBWR 进程将脏数据（Dirty Buffer）写出到数据文件上;CKPT 进程的另外一个职责是负责更新数据文件头及控制文件上的检查点信息。

2.3.1 检查点（checkpoint）的工作原理

在 Oracle 数据库中，当进行数据修改时，需要首先将数据读入内存中（Buffer Cache），修改数据的同时，Oracle 会记录重做信息（redo）用于恢复。因为有了重做信息的存在，Oracle 不需要在事务提交时（Commit）立即将变化的数据写回磁盘（立即写的效率会很低），重做的存在也正是为了在数据库崩溃之后，数据可以恢复。

最常见的情况，数据库可能因为断电而 Crash，那么内存中修改过的、尚未写入数据文件的数据将会丢失。在下一次数据库启动之后，Oracle 可以通过重做日志（redo）进行事务重演（也就是进行前滚），将数据库恢复到崩溃之前的状态，然后数据库可以打开提供使用，之后 Oracle 可以将未提交的事务进行回滚。

在这个启动过程中，通常大家最关心的是数据库要经历多久才能打开。也就是需要读取多少重做日志才能完成前滚。当然我们希望这个时间越短越好，Oracle 也正是通过各种手段在不断优化这个过程，缩短恢复时间。

检查点的存在就是为了缩短这个恢复时间。

当检查点发生时（此时的 SCN 被称为 Checkpoint SCN）Oracle 会通知 DBWR 进程，把修改过的数据，也就是此 Checkpoint SCN 之前的脏数据（dirty data）从 Buffer Cache 写入磁盘，当写入完成之后，CKPT 进程则会相应更新控制文件和数据文件头，记录检查点信息，标识变更。

因此检查点操作可以分为三个阶段：

第一阶段，CKPT 进程初始化检查点，同时捕获检查点 RBA，通常就是当前的 RBA

第二阶段，DBWR 进程写出所有满足条件的 Buffer，也即所有 RBA<=Checkpoint RBA 的

Buffer

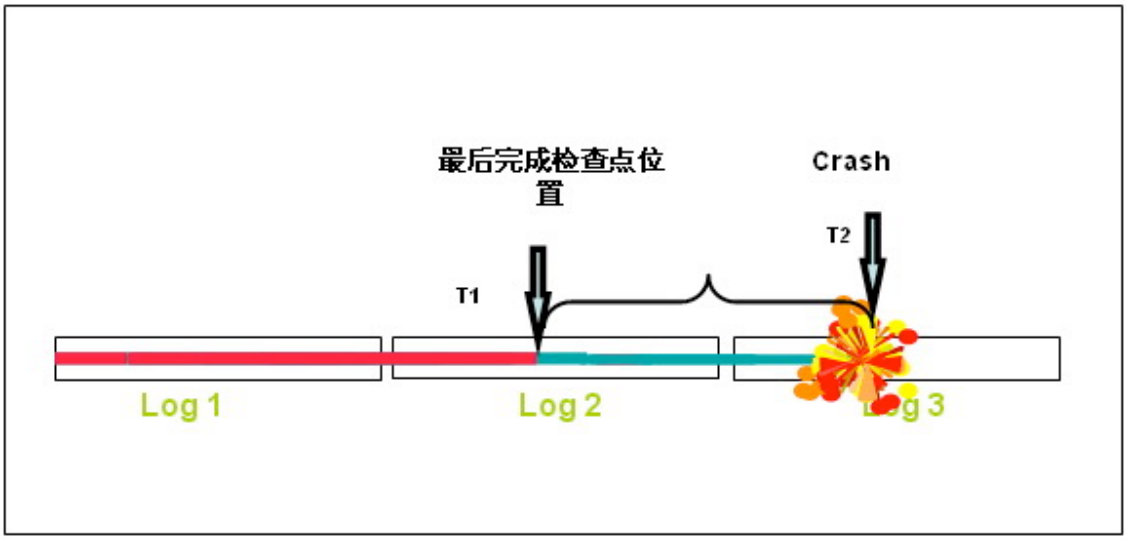
第三阶段，当所有 Buffer 写出后，CKPT 进程在控制文件中记录完成的检查点。

Checkpoint SCN 可以从数据库中查询得到：

```
SQL> select file#,checkpoint_change#,to_char(checkpoint_time,'yyyy-mm-dd hh24:mi:ss') CPT
  2  from v$datafile;
    FILE# CHECKPOINT_CHANGE#  CPT
---------- ------------------ ------------------------------
         1      8904572779065 2006-06-05 16:25:19
         2      8904572779065 2006-06-05 16:25:19
         3      8904572779065 2006-06-05 16:25:19
……
13 rows selected
SQL> select dbid,CHECKPOINT_CHANGE# from v$database;
      DBID CHECKPOINT_CHANGE#
---------- ------------------
3965153484      8904572779065
```

在检查点完成之后，此检查点之前修改过的数据都已经写回磁盘，重做日志文件中的相应重做记录对于崩溃/实例恢复不再需要。

下图中标记了三个日志组，假定在 T1 时间点，数据库完成并记录了最后一次检查点，在 T2 时刻数据库 Crash。那么在下次数据库启动时，T1 时间点之前的 Redo 不再需要进行恢复，Oracle 需要重新应用的就是 T1 至 T2 之间数据库生成的重做日志（Redo）。



从上图也可以很容易的看出，检查点的频度对于数据库的恢复时间具有极大的影响，如果检查点的频率高，那么恢复时需要应用的重做日志就相对的少，恢复时间就可以缩短。然而，需要注意的是，数据库的内部操作的相关性极强，过于频繁的检查点同样会带来性能问题，尤

其是更新频繁的数据库。所以**数据库的优化是一个系统工程，不能草率**。

更进一步的可以知道，如果 Oracle 可以在性能允许的情况下，使得检查点的 SCN 逐渐逼近 Redo 的最新变更，那么最终可以获得一个最佳平衡点，使得 Oracle 可以最大化的减少恢复时间。

为了实现这个目标，Oracle 在不同版本中一直在改进检查点的算法。

2.3.2 常规检查点与增量检查点

为了理解常规检查点和增量检查点的概念，首先需要介绍一下脏缓冲列表（LRUW List）。前面提到，当数据在 Buffer Cache 中被修改之后，Dirty Buffer 会被转移到 LRUW List，以便将来执行的检查点可以将这些修改过的 Buffer 写出到数据文件上。

但是注意，由于 LRUW List 上的 Buffer 并没有严格顺序，有的 Buffer 反复被修改，在写出之前，可能会被从 LRUW 移动回 AUXILIARY lru list,当这样的 Dirty Buffer 再次回到 LRUW 列表时，会倍添加到链表尾部，也就是说一个 Buffer 在 LRUW 链表上的位置可能发生变化，所以当检查点发生时，Oracle 需要将脏缓冲列表上的数据全部写出到数据文件。为了区分，在 Oracle8 之前，Oracle 实施的这类检查点通常被称为常规检查点（Conventional Checkpoint），由于检查点时需要写出全部的脏数据，所以也被称为完全检查点（Complete Checkpoint）。常规检查点按特定的条件触发（log_checkpoint_interval，log_checkpoint_timeout 参数设置及 log switch 等条件触发），触发时会同时更新数据文件头以及控制文件记录检查点信息。

从 Oracle8 开始，Oracle 演进了新的算法，进而引入了增量检查点（Incremental Checkpoint）的概念。和以前的版本相比，在新版本中，主要的变化是引入了检查点队列（Buffer Checkpoint Queue - CKPTQ）机制。在数据库内部，每一个脏数据块都会被**记录到检查点队列**，按照 LRBA（Low RBA - 第一次对此数据块修改对应的 Redo Byte Address）的顺序来排列，如果一个数据块进行过多次修改，该数据块在检查点队列上的顺序并不会发生变化（相对于 LRBA，后面修改的 RBA 被称为 HRBA）。

检查点队列的内存存储空间在 Shared Pool 内存中分配:

```
SQL> select * from v$sgastat where upper(name) like '%CHECKPOINT%';
POOL         NAME                            BYTES
------------ -------------------------- ----------
shared pool Checkpoint queue         410624
shared pool   log_checkpoint_timeout        12360
```

当执行增量检查点时，DBWR 从检查点队列按照 Low RBA 的顺序写出，此时先修改的数据就可以被按顺序优先写出，实例检查点因此可以不断增进，阶段性的，CKPT 进程使用非常轻量级的控制文件更新协议，将当前的最低 RBA(也即 Low Cache RBA)写入控制文件，为了减少频繁增量检查点的性能影响，**CKPT 在进行轻量级更新时，并不会改写控制文件中数据文件的检查点信息以及数据文件头信息**，而只是记录控制文件检查点 SCN（Controlfile Checkpointed at scn）并且根据增量检查点的写出增进 RBA 信息，同时不需要更改数据文件头信息。

通过增量检查点，数据库可以将以前的全量写出变更为增量渐进写出，从而可以极

大的减少对于数据库性能的影响；而检查点队列则进一步的将 RBA 和检查点关联起来，从而可以通过检查点来确定恢复的起点。

增量检查点的进度可以通过 X$KCBBES 表来查询，该表的含义为：

```
X$KCBBES–
          [K]ernel [C]ache [B]uffer Management Buffer Event Statistics
```

该视图记录了内存中 Buffer 的写出统计，其中 INDX 为 4 的条目即增量检查点写出的 Buffer 数量：

```
SQL> desc x$kcbbes
 Name                           Null?      Type
 ------------------------------ -------- ----------------------------
 ADDR                                      RAW(8)
 INDX                                      NUMBER
 INST_ID                        NUMBER
 REASON                         NUMBER
 PRIORITY                       NUMBER
 SAVECODE                       NUMBER

SQL> select * from v$version;
BANNER
--------------------------------------------------------------------------------
Oracle Database 11g Enterprise Edition Release 11.2.0.2.0 - 64bit Production

SQL> select * from x$kcbbes where reason >0;

ADDR             INDX    INST_ID     REASON  PRIORITY   SAVECODE
---------------- ----- ---------- --------- ---------- ----------
00002B37AA4429E8     2       1          480  199      71
00002B37AA4429E8     4       1          199    0       7
00002B37AA4429E8     9       1            6    0       0
00002B37AA4429E8    11       1       784125    0       0
00002B37AA4429E8    12       1          284    0       0
```

反复查询该视图可以观察增量检查点的进度，如果 REASON 值增加则意味着检查点在增进，如果不变，则意味着检查点未发生（注意，即便是增量检查点其发生也可能间隔较长时间）。

```
SQL> select * from x$kcbbes where indx=4;
ADDR             INDX    INST_ID     REASON   PRIORITY   SAVECODE
---------------- ---------- ---------- ---------- ---------- ----------
00002AC45797C2F8    4       1        5884    0       7
SQL> select * from x$kcbbes where indx=4;
```

```
ADDR             INDX    INST_ID    REASON   PRIORITY   SAVECODE
---------------- ---------- ---------- ---------- ---------- ----------
00002AC4579578A0     4       1        5893    0       7
```

检查点队列在数据库内部通过 Latch 保护：

```
SQL> select name,gets,misses from v$latch where name='checkpoint queue latch';
NAME                                                GETS     MISSES
---------------------------------- ---------- ----------
checkpoint queue latch                      14710851          0
```

Checkpoint Queue Latch 存在多个子 Latch，可以通过 V$LATCH_CHILDREN 视图查询：

```
SQL> select name,gets,misses from v$latch_children where name='checkpoint queue latch';
NAME                                          GETS     MISSES
---------------------------------------- ---------- ----------
checkpoint queue latch                        917660          0
checkpoint queue latch                        917660          0
checkpoint queue latch                        917660          0
checkpoint queue latch                        917660          0
checkpoint queue latch                        917660          0
checkpoint queue latch                        917660          0
checkpoint queue latch                        917660          0
checkpoint queue latch                        917660          0
checkpoint queue latch                        917660          0
checkpoint queue latch                        917660          0
checkpoint queue latch                        924547          0
checkpoint queue latch                        927484          0
checkpoint queue latch                        920711          0
checkpoint queue latch                        920711          0
checkpoint queue latch                        920711          0
checkpoint queue latch                        920711          0
```

除了检查点队列（CKPTQ）之外，数据库中还存在另外一个队列和检查点相关，这就是文件检查点队列-FILE QUEUE，通常缩写为 FILEQ，文件检查点队列的引入提高了表空间检查点（Tablespace Checkpoint）的性能。每个 Dirty Buffer 同时链接到这两个队列， CKPTQ 包含实例所有需要执行检查点的 Buffer，FILEQ 包含属于特定文件需要执行检查点的 Buffer，每个文件都包含一个文件队列，在执行表空间检查点请求时需要使用 FILEQ，通常当对表空间执行 Offline 等操作时会触发表空间检查点。

在 Buffer Chache 中，每个 Buffer 的 Header 上都存在 CKPTQ 以及 FILEQ 队列信息，通过如下命令可以转储 Buffer Cache 信息（注意应当仅在测试环境中尝试）：

```
alter session set events 'immediate trace name buffers level 10';
```

以下 BH 信息来自 Oracle9i 9.2.0.4 数据库环境：

```
BH (0x0x55fba950) file#: 2 rdba: 0x0080008c (2/140) class 34 ba: 0x0x553d8000
```

```
set: 3 dbwrid: 0 obj: -1 objn: 0
hash: [54ee90b0,57476608] lru: [55fbaa54,55fba8dc]
LRU flags:
ckptq: [55feb234,55ffce68] fileq: [55ffcf2c,55ffce70]
st: XCURRENT md: NULL rsop: 0x(nil) tch: 31
flags: buffer_dirty gotten_in_current_mode block_written_once
       redo_since_read
LRBA: [0x1d.38e.0] HSCN: [0x081b.dc808434] HSUB: [1] RRBA: [0x0.0.0]
buffer tsn: 1 rdba: 0x0080008c (2/140)
scn: 0x081b.dc808434 seq: 0x01 flg: 0x00 tail: 0x84340201
frmt: 0x02 chkval: 0x0000 type: 0x02=KTU UNDO BLOCK
```

注意摘录信息中的 CKPTQ 和 FILEQ，这就是检查点队列和文件队列。每个队列后面记录了两个地址信息，分别是前一块以及下一块的地址，通过这个信息 CKPTQ 和 FILEQ 构成了双向链表。注意仅 Dirty Buffer 才会包含 CKPTQ 信息，否则为 NULL，信息类似：ckptq: [NULL] fileq: [NULL]。

同样对上一个跟踪文件进行 grep 信息输出，来看一下这两个队列：

```
[oracle@jumper udump]$ grep ckptq eygle_ora_1467.trc |grep -v NULL
      ckptq: [55fba708,56374c18] fileq: [55fba710,56374c70]
      ckptq: [55ffccf0,55ffd504] fileq: [55ffccf8,55ffd50c]
      ckptq: [55fbaab4,55fba880] fileq: [55fbaabc,55fba888]
      ckptq: [55fba880,55fba7c4] fileq: [55fba888,55fba7cc]
      ckptq: [55ffd09c,55ffcfe0] fileq: [55ffd0a4,55ffcfe8]
      ckptq: [55fba64c,55feb234] fileq: [55fba654,55fbaa00]
      ckptq: [55feb3ac,55ffcf24] fileq: [574d0c08,55ffcf2c]
      ckptq: [55fba9f8,574d0bb0] fileq: [55fbaa00,574d0c08]
      ckptq: [55feb234,55ffce68] fileq: [55ffcf2c,55ffce70]
      ckptq: [55fba7c4,55fba708] fileq: [55fba7cc,55fba710]
      ckptq: [56374c18,55ffd09c] fileq: [56374c70,55ffd0a4]
      ckptq: [55ffcfe0,55ffccf0] fileq: [55ffcfe8,55ffccf8]
      ckptq: [55ffcf24,55fba9f8] fileq: [55feb3b4,574d0bdc]
      ckptq: [574d0bb0,55fba64c] fileq: [574d0bdc,55feb23c]
```

简单整理一下 CKPTQ，其顺序就是（对于）：

```
55fbaab4->55fba880->55fba7c4->55fba708->56374c18->55ffd09c->55ffcfe0->55ffccf0->55ffd504
55feb3ac->55ffcf24->55fba9f8->574d0bb0->55fba64c->55feb234->55ffce68
```

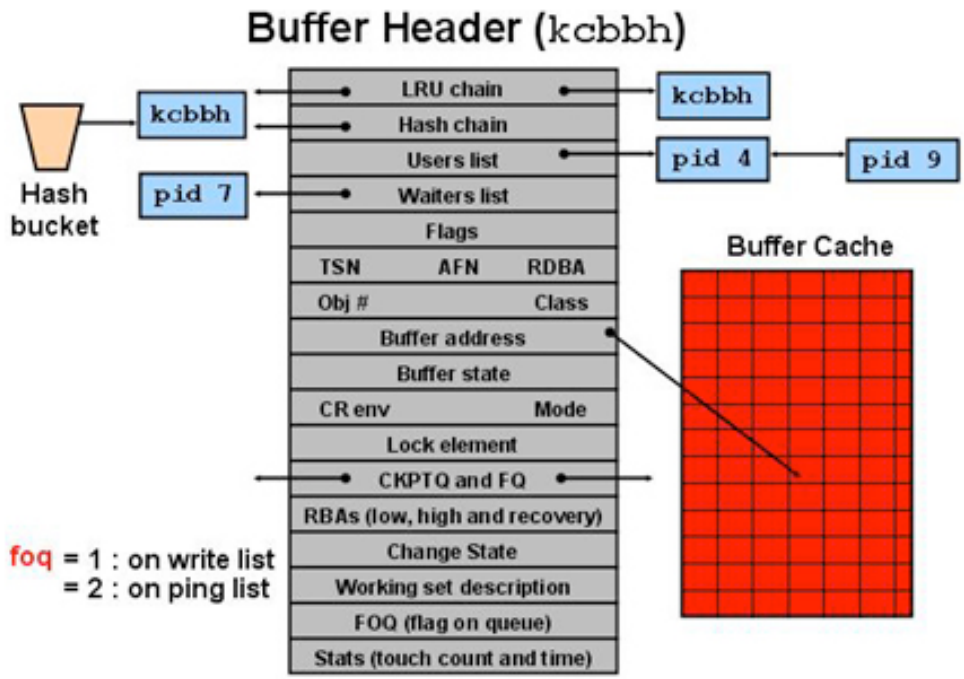
在 SGA 中存在一块内存区域用于记录这个检查点队列：

```
SQL> select name,bytes from v$sgastat where upper(name) like '%CHECKPOINT%';
NAME                                         BYTES
---------------------------------------- ----------
Checkpoint queue                             282304
```

从 Oracle10g 开始，数据库中额外增加了对象检查点队列（Object Queue - OBJQ）用于记录对象检查点信息：

```
BH (0x273f092c) file#: 1 rdba: 0x00401009 (1/4105) class: 1 ba: 0x271ec000
  set: 6 blksize: 8192 bsi: 0 set-flg: 2 pwbcnt: 133
  dbwrid: 0 obj: 517 objn: 517 tsn: 0 afn: 1
  hash: [5ebae9b8,5ebae9b8] lru: [273f0a30,273f08d0]
  lru-flags:
  ckptq: [243ea704,277f0c14] fileq: [5e4b08d4,277f0c1c] objq:
[277f0c94,273f0874]
  st: XCURRENT md: NULL tch: 2
  flags: buffer_dirty gotten_in_current_mode block_written_once
         redo_since_read
  LRBA: [0x38.11a97.0] HSCN: [0x81b.8f844f3e] HSUB: [1]
  buffer tsn: 0 rdba: 0x00401009 (1/4105)
  scn: 0x081b.8f844f3e seq: 0x01 flg: 0x02 tail: 0x4f3e0601
  frmt: 0x02 chkval: 0x0000 type: 0x06=trans data
```

下图是 Buffer Header 的结构示意图，对照以上 Buffer Header 转储信息可以更加清晰的了解 BH 的结构：



共享池中分配了相关内存用于 OBJECT QUEUE:

```
SQL> select * from v$sgastat where name like 'object queue%';
POOL         NAME                            BYTES
------------ -------------------------- ----------
shared pool  object queue hash table d        6080
shared pool  object queue hash buckets      139264
shared pool  object queue                    49056
```

了解了几种队列之后，下面让我们来看一下控制文件以及增量检查点的协同工作。以下输出来自 Oracle 11g 测试环境，两次 Level 8 级控制文件转储，时间间隔有 8 分钟左右（去除了一点次要信息）。

第一部分重要信息是控制文件的 Seq 号，控制文件随着数据库的变化而增进版本：

```
[oracle@localhost trace]$ diff 11gtest_ora_24951.trc 11gtest_ora_25106.trc
< DUMP OF CONTROL FILES, Seq # 1781 = 0x6f5
---
> DUMP OF CONTROL FILES, Seq # 1782 = 0x6f6
42c28
<       Control Seq=1781=0x6f5, File size=594=0x252
---
>       Control Seq=1782=0x6f6, File size=594=0x252
```

接下来是控制文件检查点 SCN，增量检查点不断增进的内容之一：

```
85c71
<  Controlfile Checkpointed at scn:  0x0000.00095a4c 07/07/2008 11:27:56
---
>  Controlfile Checkpointed at scn:  0x0000.00095ae6 07/07/2008 11:33:46
```

检查点记录之后是 RBA 信息，检查点和 Redo 相关联在这里实现，通过以下信息可以注意到，通过增量检查点之后，Dirty Buffer 数量从 47 降低到 7，而 Low Cache RBA 从 0x1d.4b1.0 增进到 0x1d.518.0，LOW Cache RBA 是下一次恢复的起点，而 On Disk RBA 则是指已经写入磁盘（Redo Log File）的 RBA 地址，这是前滚恢复能够到达的终点，增量检查点的作用由此体现：

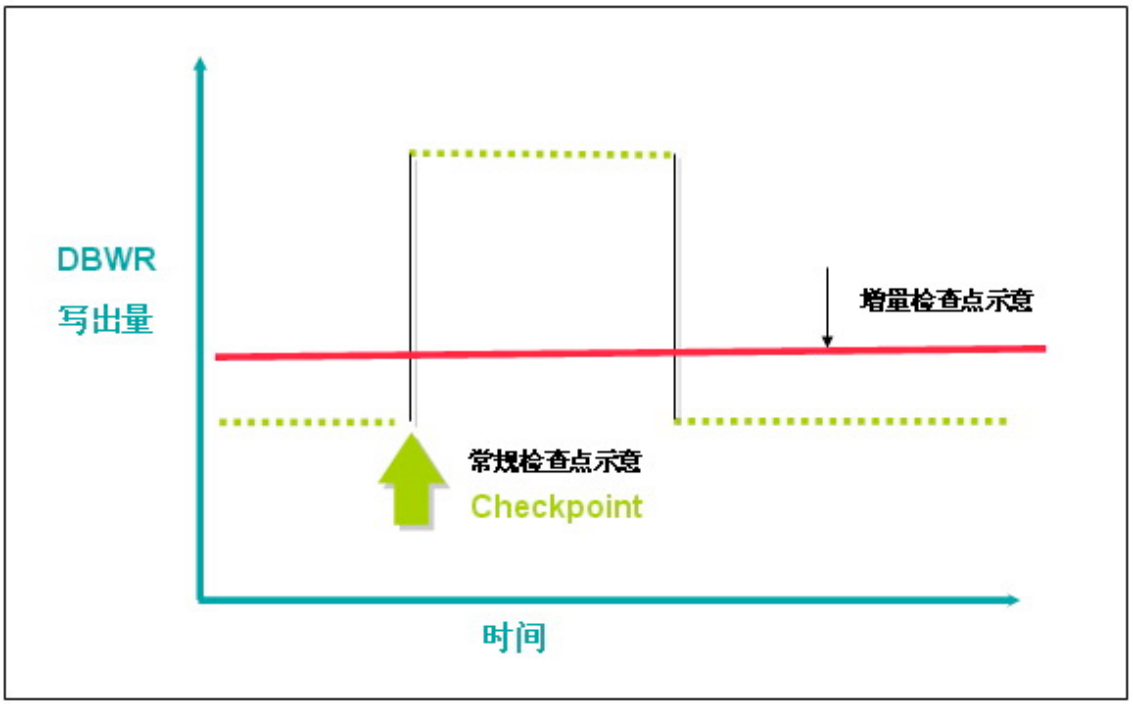
```
114,116c100,102
< THREAD #1 - status:0x2 flags:0x0 dirty:47
< low cache rba:(0x1d.4b1.0) on disk rba:(0x1d.515.0)
< on disk scn: 0x0000.00095a78 07/07/2008 11:28:18
---
> THREAD #1 - status:0x2 flags:0x0 dirty:7
> low cache rba:(0x1d.518.0) on disk rba:(0x1d.525.0)
> on disk scn: 0x0000.00095ada 07/07/2008 11:33:07
```

最后一部分是 Heartbeat 心跳信息，每 3 秒更新一次用于验证实例的存活性，第一章中已经有所提及：

```
118c104
< heartbeat: 659472703 mount id: 1478358445
---
> heartbeat: 659472842 mount id: 1478358445
```

通过以上分析可以清晰的看到增量检查点的实施过程，因为增量检查点可以连续的进行，所以检查点 RBA 可以比常规检查点更接近数据库的最后状态，从而在数据库的实例恢复中可以极大的减少恢复时间。

而且，通过增量检查点，DBWR 可以持续进行写出，从而避免了常规检查点出发的峰值写入对于 I/O 的过度争用，通过下图可以清楚的看到这一改进的意义：



显而易见的是，增量检查点明显优于常规的完全检查点，所以在引入检查点队列之后，数据库正常情况下执行的都是增量检查点，从 Oracle8i 开始，完全检查点仅在以下两种情况下出现：

- ALTER SYSTEM CHECKPOINT
- SHUTDOWN (除 ABORT 方式外)

LOG SWITCH 事件同样触发的是增量检查点，但是在 LOG SWITCH 触发的检查点会促使数据文件头与控制文件信息的同步（数据文件头的写操作并非每次 Log Switch 检查点都会发生）。

如前所述，我们知道每个 Buffer 都可能和 ckptq、fileq、objq 相关联，从而检查点也就有了不同的分类，常见的分类有：Full Checkpoint、Thread Checkpoint、File Checkpoint、Object Checkpoint、Parallel Query Checkpoint、Incremental Checkpoint、Log Switch Checkpoint.

2.3.3 LOG_CHECKPOINT_TO_ALERT 参数

在数据库中，可以设置初始化参数 log_checkpoints_to_alert 为 True，则数据库会将检查点的执行情况记入告警日志文件，这个参数的初始值为 FALSE:

```
SQL> show parameter checkpoints_to
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
log_checkpoints_to_alert             boolean     FALSE
SQL> alter system set log_checkpoints_to_alert=true;
System altered.
```

当数据库执行各类检查点时，日志文件中会记录详细信息，以下是来自 Oracle10g 告警日志文件中的信息摘录。

注意以下信息中，共发生了两次检查点，触发条件都是 LOG SWITCH，在日志中，注意 RBA 信息和检查点 SCN 同时出现，这就是检查点队列的作用，LOG SWITCH 检查点的特别之处在于需要同时在控制文件和数据文件头上标记检查点进度：

```
Wed Jul 19 17:33:05 2006
Thread 1 cannot allocate new log, sequence 44
Private strand flush not complete
  Current log# 3 seq# 43 mem# 0: /opt/oracle/oradata/alexhell/redo03.log
Beginning log switch checkpoint up to RBA [0x2c.2.10], SCN: 8914464526139
Thread 1 advanced to log sequence 44
  Current log# 4 seq# 44 mem# 0: /opt/oracle/oradata/alexhell/redo04.log
Wed Jul 19 17:34:33 2006
Beginning log switch checkpoint up to RBA [0x2d.2.10], SCN: 8914464533295
Thread 1 advanced to log sequence 45
  Current log# 5 seq# 45 mem# 0: /opt/oracle/oradata/alexhell/redo05.log
Wed Jul 19 17:38:30 2006
Completed checkpoint up to RBA [0x2c.2.10], SCN: 8914464526139
Wed Jul 19 17:39:39 2006
Completed checkpoint up to RBA [0x2d.2.10], SCN: 8914464533295
```

从以上信息还可以观察到，检查点的触发和检查点完成具有一定的时间间隔，这进一步说明，检查点仅仅是一个数据库事件，**发生检查点时 CKPT 进程负责通知 DBWR 执行写出，但是检查点不会等待写出完成，它会在下一次触发时写出上一次成功完成的检查点信息**。

在告警日志文件你可能还会看到类似如下信息：

```
Thu Jul 20 16:35:33 2006
```

```
Incremental checkpoint up to RBA [0x31.6f79.0], current log tail at RBA [0x31.7108.0]
Thu Jul 20 17:05:39 2006
Incremental checkpoint up to RBA [0x31.77f8.0], current log tail at RBA [0x31.7f42.0]
Thu Jul 20 17:06:23 2006
Beginning log switch checkpoint up to RBA [0x32.2.10], SCN: 8914464684949
Thread 1 advanced to log sequence 50
  Current log# 5 seq# 50 mem# 0: /opt/oracle/oradata/eygle/redo05.log
Thu Jul 20 17:11:27 2006
Completed checkpoint up to RBA [0x32.2.10], SCN: 8914464684949
Thu Jul 20 17:35:46 2006
Incremental checkpoint up to RBA [0x32.652.0], current log tail at RBA [0x32.809.0]
```

这些信息和检查点的另外一个触发条件有关。为了保证检查点不会滞后整个日志文件，Oracle 限制最长的检查点跨度不超过最小日志大小的 90%。所以数据库在运行过程中会根据 LOG TAIL 进行计算，主动触发增量检查点。

2.3.4 控制文件与数据文件头信息

前面曾经提到，CKPT 的一项任务是更新数据文件头和控制文件，记录检查点信息，这些信息对于数据库的恢复和完整性校验都至关重要，接下来我们来看一下控制文件和数据文件头都记录了哪些信息。

通过以下内部命令可以转储 Oracle 的数据文件头信息：

alter session set events 'immediate trace name file_hdrs level 10';

首先以 immediate 方式关闭数据库，在 mount 状态下执行该命令，研究一下此时转储的文件头信息：

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

选取一个文件的信息（这里选择 eygle01.dbf 文件）注意，以下就是 trace 文件摘录的信息，这类 Trace 文件的信息包含两个部分，一部分来自控制文件，一部分来自数据文件:

```
DATA FILE #4:
  (name #4) /opt/oracle/oradata/eygle/eygle01.dbf
creation size=0 block size=8192 status=0xe head=4 tail=4 dup=1
 tablespace 4, index=4 krfil=4 prev_file=0
 unrecoverable scn: 0x0000.00000000 01/01/1988 00:00:00
 Checkpoint cnt:53 scn: 0x0000.002ac5f9 08/10/2006 20:58:21
 Stop scn: 0x0000.002ac5f9 08/10/2006 20:58:21
 Creation Checkpointed at scn:  0x0000.0015078d 06/06/2006 09:41:54
.........
 FILE HEADER:
```

```
        Software vsn=153092096=0x9200000, Compatibility Vsn=134217728=0x8000000
        Db ID=1407686520=0x53e79778, Db Name='EYGLE'
        Activation ID=0=0x0
        Control Seq=973=0x3cd, File size=1280=0x500
        File Number=4, Blksiz=8192, File Type=3 DATA
Tablespace #4 - EYGLE  rel_fn:4
Creation   at   scn: 0x0000.0015078d 06/06/2006 09:41:54
Backup taken at scn: 0x0000.00000000 01/01/1988 00:00:00 thread:0
 reset logs count:0x232bee1f scn: 0x0000.0007c781 recovered at 08/10/2006 20:57:53
 status:0x0 root dba:0x00000000 chkpt cnt: 53 ctl cnt:52
begin-hot-backup file size: 0
Checkpointed at scn:  0x0000.002ac5f9 08/10/2006 20:58:21
 thread:1 rba:(0x35.1275.10)
```

注意，**这其中"FILE HEADER"开始的信息就是来自数据文件头，之前的相关内容来自控制文件**。通过如下手段可以验证这个结论，继续以上的试验，在 mount 状态下将 eygle01.dbf 文件移除，重复转储命令：

```
[oracle@jumper eygle]$ mv eygle01.dbf eygle01.dbf.n
[oracle@jumper eygle]$ sqlplus "/ as sysdba"
SQL> alter session set events 'immediate trace name file_hdrs level 10';
Session altered.
```

检查现在生成的跟踪文件可以看到，由于文件丢失，"FILE HEADER"部分信息将无法获得：

```
DATA FILE #4:
  (name #4) /opt/oracle/oradata/eygle/eygle01.dbf
creation size=0 block size=8192 status=0xe head=4 tail=4 dup=1
 tablespace 4, index=4 krfil=4 prev_file=0
 unrecoverable scn: 0x0000.00000000 01/01/1988 00:00:00
 Checkpoint cnt:53 scn: 0x0000.002ac5f9 08/10/2006 20:58:21
 Stop scn: 0x0000.002ac5f9 08/10/2006 20:58:21
 Creation Checkpointed at scn:  0x0000.0015078d 06/06/2006 09:41:54
 thread:0 rba:(0x0.0.0)
。。。。。。。
ORA-01157: cannot identify/lock data file 4 - see DBWR trace file
ORA-01110: data file 4: '/opt/oracle/oradata/eygle/eygle01.dbf'
*** Error 1157 in open/read file # 4 ***
DUMP OF TEMP FILES: 0 files in database
```

此时报出的错误是，文件无法找到，也就是说当我们执行 trace file_hdrs 时需要读取数据文件头，获得相关信息。

回过头看一下来自控制文件部分的信息，其中包含:

```
Checkpoint cnt:53 scn: 0x0000.002ac5f9 08/10/2006 20:58:21
```

在"FILE HEADER"部分信息中包括了如下部分：

```
status:0x0 root dba:0x00000000 chkpt cnt: 53 ctl cnt:52
begin-hot-backup file size: 0
Checkpointed at scn:  0x0000.002ac5f9 08/10/2006 20:58:21
```

其中控制文件中记录的 SCN 指最后一次成功完成的检查点 SCN；数据文件头中记录的 Checkpointed at scn 指数据文件中记录的最后一次成功完成的检查点 SCN；这两者在正常情况下是相等的。此外在控制文件和数据文件头都记录一个检查点计数（chkpt cnt），而且数据文件头还记录了一个控制文件检查点计数（ctl cnt），在以上输出中 ctl cnt:52 比控制文件中的 checkpoint cnt 小 1，这是为什么呢？

这是因为当检查点更新控制文件和数据文件头上的 chkpt cnt 信息时，在更新控制文件之前，可以获得当前的 ctl cnt，这个信息被记入了数据文件，也就是 ctl cnt:52，为什么要写这个到数据文件呢？因为不能保证当前更新控制文件上的 checkpoint cnt 一定会成功（数据库可能突然 crash 掉了），记录之前成功的 ctl cnt 可以确保上一次的 checkpoint 是成功完成的，从而省略了校验步骤。

2.3.5 数据库的启动验证

在前文我们还提到，在数据库启动过程中的检验包含以下两个步骤：

- 第一次检查数据文件头中的 Checkpoint cnt 是否与对应控制文件中的 Checkpoint cnt 一致，如果相等,进行第二次检查;
- 第二次检查数据文件头的开始 SCN 和对应控制文件中的结束 SCN 是否一致如果结束 SCN 等于开始 SCN,则不需要对那个文件进行恢复。

对每个数据文件都完成检查后,打开数据库.同时将每个数据文件的结束 SCN 设置为无穷大。

在上一节中，我们说过：当使用 file_hdrs 事件来转储数据文件头信息时，Oracle 会转储两部分信息，一部分来自控制文件，一部分来自数据文件，在数据库启动过程中，这两部分信息要用来进行启动验证。通过以下过程可以进一步来深入探讨一下这个内容。

首先来看以下来自 Mount 状态控制文件部分转储（选取一个文件测试）信息：

```
DATA FILE #4:
  (name #4) /opt/oracle/oradata/eygle/eygle01.dbf
creation size=0 block size=8192 status=0xe head=4 tail=4 dup=1
 tablespace 4, index=4 krfil=4 prev_file=0
 unrecoverable scn: 0x0000.00000000 01/01/1988 00:00:00
 Checkpoint cnt:58 scn: 0x0000.002ac8ee 08/11/2006 09:48:29
 Stop scn: 0x0000.002ac8ee 08/11/2006 09:48:29
 Creation Checkpointed at scn:  0x0000.0015078d 06/06/2006 09:41:54
 thread:0 rba:(0x0.0.0)
```

这部分中包含的重要信息有:

- 检查点计数: Checkpoint cnt:58
- 检查点 SCN: scn: 0x0000.002ac8ee 08/11/2006 09:48:29
- 数据文件 Stop SCN:Stop scn: 0x0000.002ac8ee 08/11/2006 09:48:29

接下来再来看看来自数据文件头的信息：

```
FILE HEADER:
        Software vsn=153092096=0x9200000, Compatibility Vsn=134217728=0x8000000
        Db ID=1407686520=0x53e79778, Db Name='EYGLE'
        Activation ID=0=0x0
        Control Seq=979=0x3d3, File size=1280=0x500
        File Number=4, Blksiz=8192, File Type=3 DATA
Tablespace #4 - EYGLE  rel_fn:4
Creation   at   scn: 0x0000.0015078d 06/06/2006 09:41:54
Backup taken at scn: 0x0000.00000000 01/01/1988 00:00:00 thread:0
 reset logs count:0x232bee1f scn: 0x0000.0007c781 recovered at 08/10/2006 20:57:53
 status:0x0 root dba:0x00000000 chkpt cnt: 58 ctl cnt:57
begin-hot-backup file size: 0
Checkpointed at scn:  0x0000.002ac8ee 08/11/2006 09:48:29
```

这部分中包含的重要信息有:

- 检查点 SCN: Checkpointed at scn: 0x0000.002ac8ee 08/11/2006 09:48:29
- 检查点计数: chkpt cnt: 58 ctl cnt:57

这两者都和控制文件中所记录的一致。如果这两者一致，数据库启动时就能通过验证，启动数据库。

那么如果不一致，Oracle 则会请求进行恢复；以下是从备份中恢复 eygle01.dbf 文件。首先第一部分从控制文件中获得的信息是相同的:

```
DATA FILE #4:
  (name #4) /opt/oracle/oradata/eygle/eygle01.dbf
creation size=0 block size=8192 status=0xe head=4 tail=4 dup=1
 tablespace 4, index=4 krfil=4 prev_file=0
 unrecoverable scn: 0x0000.00000000 01/01/1988 00:00:00
 Checkpoint cnt:58 scn: 0x0000.002ac8ee 08/11/2006 09:48:29
 Stop scn: 0x0000.002ac8ee 08/11/2006 09:48:29
 Creation Checkpointed at scn:  0x0000.0015078d 06/06/2006 09:41:54
```

相关信息同样为：

- 检查点计数: Checkpoint cnt:58
- 检查点 SCN: scn: 0x0000.002ac8ee 08/11/2006 09:48:29
- 数据文件 Stop SCN:Stop scn: 0x0000.002ac8ee 08/11/2006 09:48:29

而从文件头中获得的备份文件信息则是：

```
FILE HEADER:
        Software vsn=153092096=0x9200000, Compatibility Vsn=134217728=0x8000000
        Db ID=1407686520=0x53e79778, Db Name='EYGLE'
        Activation ID=0=0x0
        Control Seq=973=0x3cd, File size=1280=0x500
        File Number=4, Blksiz=8192, File Type=3 DATA
Tablespace #4 - EYGLE  rel_fn:4
Creation   at   scn: 0x0000.0015078d 06/06/2006 09:41:54
Backup taken at scn: 0x0000.00000000 01/01/1988 00:00:00 thread:0
 reset logs count:0x232bee1f scn: 0x0000.0007c781 recovered at 08/10/2006 20:57:53
 status:0x0 root dba:0x00000000 chkpt cnt: 53 ctl cnt:52
begin-hot-backup file size: 0
Checkpointed at scn:  0x0000.002ac5f9 08/10/2006 20:58:21
```

此时备份文件的信息：

- 检查点是：Checkpointed at scn: 0x0000.002ac5f9 08/10/2006 20:58:21
- 检查点计数为:chkpt cnt: 53 ctl cnt:52

这两者不再一致，首先是检查点计数不一致，当前文件的 chkpt cnt 为 53，小于控制文件中记录的 58，Oracle 可以判断文件是从备份中恢复的，或者文件故障，需要进行介质恢复。如果此时我们试图打开数据库，则 Oracle 提示文件需要介质恢复:

```
SQL> alter database open;
alter database open
*
ERROR at line 1:
ORA-01113: file 4 needs media recovery
ORA-01110: data file 4: '/opt/oracle/oradata/eygle/eygle01.dbf'
```

执行恢复:

```
SQL> recover datafile 4;
Media recovery complete.
```

有必要再来考察一下恢复完成之后，控制文件和数据文件的变化。

首先看控制文件的变化:

```
DATA FILE #4:
  (name #4) /opt/oracle/oradata/eygle/eygle01.dbf
creation size=0 block size=8192 status=0xe head=4 tail=4 dup=1
 tablespace 4, index=4 krfil=4 prev_file=0
 unrecoverable scn: 0x0000.00000000 01/01/1988 00:00:00
 Checkpoint cnt:59 scn: 0x0000.002ac8ee 08/11/2006 09:48:29
 Stop scn: 0x0000.002ac8ed 08/11/2006 09:48:29
 Creation Checkpointed at scn:  0x0000.0015078d 06/06/2006 09:41:54
```

检查点计数: Checkpoint cnt:59，执行了恢复之后，检查点计数较前增加了 1

- 检查点 SCN: scn: 0x0000.002ac8ee 08/11/2006 09:48:29
- 数据文件 Stop scn: 0x0000.002ac8ed 08/11/2006 09:48:29

数据文件 Stop scn 和数据文件进行了同步。

以下是数据文件头信息：

```
 FILE HEADER:
        Software vsn=153092096=0x9200000, Compatibility Vsn=134217728=0x8000000
        Db ID=1407686520=0x53e79778, Db Name='EYGLE'
        Activation ID=0=0x0
        Control Seq=983=0x3d7, File size=1280=0x500
        File Number=4, Blksiz=8192, File Type=3 DATA
Tablespace #4 - EYGLE  rel_fn:4
Creation   at   scn: 0x0000.0015078d 06/06/2006 09:41:54
Backup taken at scn: 0x0000.00000000 01/01/1988 00:00:00 thread:0
 reset logs count:0x232bee1f scn: 0x0000.0007c781 recovered at 08/11/2006 10:11:26
 status:0x0 root dba:0x00000000 chkpt cnt: 59 ctl cnt:58
begin-hot-backup file size: 0
Checkpointed at scn:  0x0000.002ac8ed 08/11/2006 09:48:29
```

此时数据文件的信息：

检查点是：Checkpointed at scn: 0x0000.002ac8ed 08/11/2006 09:48:29

这个检查点和控制文件中记录的 stop scn 一致，数据库启动可以顺利进行。检查点计数为:chkpt cnt: 59 ctl cnt:58。

打开数据库，看一看 OPEN 阶段的变化：

```
SQL> alter database open;
Database altered.
SQL> alter session set events 'immediate trace name file_hdrs level 10';
Session altered.
```

此时数据库恢复正常运行，控制文件信息如下：

```
DATA FILE #4:
  (name #4) /opt/oracle/oradata/eygle/eygle01.dbf
creation size=0 block size=8192 status=0xe head=4 tail=4 dup=1
 tablespace 4, index=4 krfil=4 prev_file=0
 unrecoverable scn: 0x0000.00000000 01/01/1988 00:00:00
 Checkpoint cnt:60 scn: 0x0000.002ac8ef 08/11/2006 10:19:30
 Stop scn: 0xffff.ffffffff 08/11/2006 09:48:29
 Creation Checkpointed at scn:  0x0000.0015078d 06/06/2006 09:41:54
```

此时 stop scn 被置为最大值（0xffff.ffffffff）。

数据文件头信息如下，其中检查点信息和控制文件中记录的 Checkpoint 信息一致：

```
 FILE HEADER:
        Software vsn=153092096=0x9200000, Compatibility Vsn=134217728=0x8000000
        Db ID=1407686520=0x53e79778, Db Name='EYGLE'
        Activation ID=0=0x0
        Control Seq=984=0x3d8, File size=1280=0x500
        File Number=4, Blksiz=8192, File Type=3 DATA
Tablespace #4 - EYGLE  rel_fn:4
Creation   at   scn: 0x0000.0015078d 06/06/2006 09:41:54
Backup taken at scn: 0x0000.00000000 01/01/1988 00:00:00 thread:0
 reset logs count:0x232bee1f scn: 0x0000.0007c781 recovered at 08/11/2006 10:11:26
 status:0x4 root dba:0x00000000 chkpt cnt: 60 ctl cnt:59
begin-hot-backup file size: 0
Checkpointed at scn:  0x0000.002ac8ef 08/11/2006 10:19:30
```

关于 CheckPoint Cnt 作用更为直接的验证需要使用 BBED 工具(在本章后面会有更为详细的说明),通过如下命令简单修改 Checkpoint cnt 为较小的值:

```
C:\>bbed    filename='C:\ORACLE\ORADATA\ORA9I\UNDOTBS01.DBF'    blocksize=8192    mode=edit
password=blockedit

BBED: Release 2.0.0.0.0 - Limited Production on 星期一 6月 18 22:33:33 2012
Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.
************* !!! For Oracle Internal Use only !!! ***************

BBED> set count 32
        COUNT           32
BBED> p kcvfh.kcvfhcpc   <---CheckPoint Count
ub4 kcvfhcpc                                @176      0x000000c5

BBED> dump
 File: C:\ORACLE\ORADATA\ORA9I\UNDOTBS01.DBF (0)
 Block: 1                Offsets:  176 to  207           Dba:0x00000000
------------------------------------------------------------------------
 c5000000 8145de2e c4000000 eb190400 00002e04 3305de2e 01007d02 51000000

 <32 bytes per line>

BBED> modify /x c2
 File: C:\ORACLE\ORADATA\ORA9I\UNDOTBS01.DBF (0)
 Block: 1                Offsets:  176 to  207           Dba:0x00000000
------------------------------------------------------------------------
```

```
c2000000 8145de2e c4000000 eb190400 00002e04 3305de2e 01007d02 51000000

 <32 bytes per line>

BBED> sum apply
Check value for File 0, Block 1:
current = 0xa797, required = 0xa797
```

此时启动数据库,数据库自动检测到检查点计数不一致,提示文件需要介质恢复:

```
SQL> startup
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: 'C:\ORACLE\ORADATA\ORA9I\UNDOTBS01.DBF'
```

由于手工修改了 chkpt cnt,此时尝试恢复 2 号文件,就会收到控制文件的检查点计数与数据文件不一致的提示,控制文件记录的值为 194(c2),数据文件记录的 Ctl Cnt 信息为 196(c4),两者不符,就出现了内部错误:

```
SQL> recover datafile 2;
ORA-00283: recovery session canceled due to errors
ORA-00600: internal error code, arguments: [kfhpfh_03-1210], [fno =], [2],
[fhcpc =], [194], [fhccc =], [196], []
ORA-01110: data file 2: 'C:\ORACLE\ORADATA\ORA9I\UNDOTBS01.DBF'
```

关于控制文件、数据文件以及在启动过程的校验，我们可以从另外一个角度进行进一步的验证。通过以下步骤跟踪数据库的启动过程，可以获得跟踪文件（在本章后面部分还会用到这个测试过程）：

```
SQL> startup nomount;
SQL> alter session set events='10046 trace name context forever,level 12';
SQL> alter database mount;
SQL> alter database open;
```

跟踪文件（以下信息来自 Oracle10g，在 user_dump_dest 参数设置目录下可以找到跟踪文件）里包含了重要的提示信息，在数据库 open 的过程中，首先需要读取控制文件，顺序获取控制文件中记录的检查点、数据文件信息等，注意以下输出中，不同的 block#就代表着在读取不同的内容：

```
PARSING IN CURSOR #2 len=19 dep=0 uid=0 oct=35 lid=0 tim=1130985988950724 hv=1907384048
```

```
ad='3f7a16b4'
alter database open
END OF STMT
PARSE #2:c=10000,e=736,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,tim=1130985988950715
BINDS #2:
WAIT #2: nam='rdbms ipc reply' ela= 69 from_process=7 timeout=2147483647 p3=0 obj#=-1
WAIT #2: nam='control file sequential read' ela= 32 file#=0 block#=1 blocks=1 obj#=-1
WAIT #2: nam='control file sequential read' ela= 16 file#=1 block#=1 blocks=1 obj#=-1
WAIT #2: nam='control file sequential read' ela= 17 file#=2 block#=1 blocks=1 obj#=-1
WAIT #2: nam='control file sequential read' ela= 14 file#=0 block#=15 blocks=1 obj#=-1
WAIT #2: nam='control file sequential read' ela= 13 file#=0 block#=17 blocks=1 obj#=-1
WAIT #2: nam='rdbms ipc reply' ela= 1649 from_process=5 timeout=910 p3=0 obj#=-1
WAIT #2: nam='control file sequential read' ela= 21 file#=0 block#=1 blocks=1 obj#=-1
WAIT #2: nam='control file sequential read' ela= 14 file#=1 block#=1 blocks=1 obj#=-1
WAIT #2: nam='control file sequential read' ela= 13 file#=2 block#=1 blocks=1 obj#=-1
WAIT #2: nam='control file sequential read' ela= 12 file#=0 block#=15 blocks=1 obj#=-1
WAIT #2: nam='control file sequential read' ela= 13 file#=0 block#=17 blocks=1 obj#=-1
WAIT #2: nam='control file sequential read' ela= 14 file#=0 block#=296 blocks=1 obj#=-1
。。。。。。。。
```

读取控制文件之后，Oracle 顺序读取每个数据文件头，获取数据文件的信息，注意以下输出中，direct path read 读取了每个数据文件的第一个 Block：

```
WAIT #2: nam='direct path read' ela= 18 file number=1 first dba=1 block cnt=1 obj#=-1
WAIT #2: nam='direct path read' ela= 2 file number=2 first dba=1 block cnt=1 obj#=-1
WAIT #2: nam='direct path read' ela= 2 file number=3 first dba=1 block cnt=1 obj#=-1
WAIT #2: nam='direct path read' ela= 2 file number=4 first dba=1 block cnt=1 obj#=-1
WAIT #2: nam='direct path read' ela= 2 file number=6 first dba=1 block cnt=1 obj#=-1
WAIT #2: nam='direct path read' ela= 2 file number=8 first dba=1 block cnt=1 obj#=-1
WAIT #2: nam='direct path read' ela= 2 file number=10 first dba=1 block cnt=1 obj#=-1
WAIT #2: nam='direct path read' ela= 2 file number=14 first dba=1 block cnt=1 obj#=-1
WAIT #2: nam='direct path read temp' ela= 2 file number=201 first dba=1 block cnt=1 obj#=-1
```

通过这些信息的比较，数据库才能确定文件的版本与一致性，这是启动过程中必须进行的校验。从不同角度来观察、研究和领会数据库的机制，是学习 Oracle 的一个重要方法。

当然这个跟踪日志中展现的内容原不止于此，大家可以由此开始进一步的研究和探索。

2.3.6 使用备份的控制文件

继续上一节的介绍，我们可以想象，如果控制文件是从备份中恢复的，那么数据库在 open 过程中又将如何呢?

首先备份控制文件，打开数据库，增进检查点：

```
[oracle@jumper eygle]$ cp control01.ctl control01.ctl.bak
[oracle@jumper eygle]$ sqlplus "/ as sysdba"
……
SQL> startup
SQL> alter system checkpoint;
System altered.
SQL> shutdown immediate;
```

然后恢复旧的控制文件，mount 数据库，转储数据文件头：

```
[oracle@jumper eygle]$ mv control01.ctl control01.ctl.n
[oracle@jumper eygle]$ mv control01.ctl.bak control01.ctl
[oracle@jumper eygle]$ sqlplus "/ as sysdba"
SQL> startup mount;
SQL> alter session set events 'immediate trace name file_hdrs level 10';
Session altered.
```

检查此时的控制文件的信息（选择一个文件）：

```
DATA FILE #4:
  (name #4) /opt/oracle/oradata/eygle/eygle01.dbf
creation size=0 block size=8192 status=0xe head=4 tail=4 dup=1
 tablespace 4, index=4 krfil=4 prev_file=0
 unrecoverable scn: 0x0000.00000000 01/01/1988 00:00:00
 Checkpoint cnt:61 scn: 0x0000.002acb1e 08/11/2006 10:44:38
 Stop scn: 0x0000.002acb1e 08/11/2006 10:44:38
 Creation Checkpointed at scn:  0x0000.0015078d 06/06/2006 09:41:54
```

再看数据文件头信息:

```
 FILE HEADER:
        Software vsn=153092096=0x9200000, Compatibility Vsn=134217728=0x8000000
        Db ID=1407686520=0x53e79778, Db Name='EYGLE'
        Activation ID=0=0x0
        Control Seq=989=0x3dd, File size=1280=0x500
        File Number=4, Blksiz=8192, File Type=3 DATA
Tablespace #4 - EYGLE  rel_fn:4
Creation   at   scn: 0x0000.0015078d 06/06/2006 09:41:54
Backup taken at scn: 0x0000.00000000 01/01/1988 00:00:00 thread:0
 reset logs count:0x232bee1f scn: 0x0000.0007c781 recovered at 08/11/2006 10:11:26
 status:0x0 root dba:0x00000000 chkpt cnt: 64 ctl cnt:63
begin-hot-backup file size: 0
Checkpointed at scn:  0x0000.002acb98 08/11/2006 10:46:24
```

注意到数据文件的 chkpt cnt: 64 要大于控制文件的 Checkpoint cnt:61，也就是说控制文件是旧的（控制文件以及数据文件上记录的 Control Seq 也可以用来判断控制文件与数据库是否

匹配）。

此时尝试打开数据库就会出现如下错误:

```
SQL> alter database open;
alter database open
*
ERROR at line 1:
ORA-01122: database file 1 failed verification check
ORA-01110: data file 1: '/opt/oracle/oradata/eygle/system01.dbf'
ORA-01207: file is more recent than controlfile - old controlfile
```

Oracle 告诉我们，控制文件是旧的。此时我们可以通过重建控制文件或者从旧的数据备份开始恢复。

2.3.7 FAST_START_MTTR_TARGET

继续检查点的探讨。在数据库中，增量检查点是通过 Fast-Start Checkpointing 特性来实现的，从 Oracle8i 开始，这一特性包含在 Oracle 企业版的 Fast-Start Fault recovery 组件之中，通过查询 v$option 视图我们了解一下这一特性：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle8i Enterprise Edition Release 8.1.7.4.0 - 64bit Production
SQL> select * from v$option where  Parameter='Fast-Start Fault Recovery';
PARAMETER                      VALUE
------------------------------ --------------------
Fast-Start Fault Recovery      TRUE
```

该选件包含三个主要特性，可以加快系统在故障后的恢复，提高系统的可用性：

- Fast-Start Checkpointing
- Fast-Start On-Demand Rollback
- Fast-Start Parallel Rollback.

Fast-Start Checkpointing 特性在不同版本中的控制方式不同:

- Oracle8i 中主要通过参数 FAST_START_IO_TARGET 来实现；
- Oracle9i 中，Fast-Start Checkpointing 主要通过 FAST_START_MTTR_TARGET 参数来实现;
- 从 Oracle10g 开始，数据库可以实现自动调整的检查点（SelfTune Chechpoint）

以下简要介绍一下不同版本中相关参数的设定和实现.

FAST_START_MTTR_TARGET 参数从 Oracle9i 开始被引入，该参数定义数据库进行 Crash 恢复的时间，单位是秒，取值范围是在 0 到 3600 秒之间。

在 Oracle9i 中，Oracle 推荐设置这个参数代替 FAST_START_IO_TARGET、LOG_CHECKPOINT_TIMEOUT 及 LOG_CHECKPOINT_INTERVAL 等参数。

缺省的，在 Oracle9i 中，FAST_START_IO_TARGET 和 LOG_CHECKPOINT_INTERVAL 参数已经被设置为 0。

```
SQL> show parameter fast_start_io
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
fast_start_io_target                 integer     0
SQL> show parameter interval
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
log_checkpoint_interval              integer     0
```

从 Oracle9iR2 开始，Oracle 引入了一个新的视图提供 MTTR 建议：

```
SQL> select MTTR_TARGET_FOR_ESTIMATE MttrEst,
  2         ADVICE_STATUS AD,
  3         DIRTY_LIMIT DL,
  4         ESTD_CACHE_WRITES ESTCW,
  5         ESTD_CACHE_WRITE_FACTOR EstCWF,ESTD_TOTAL_WRITES ESTW,
  6         ESTD_TOTAL_WRITE_FACTOR ETWF,ESTD_TOTAL_IOS ETIO
  7  from v$mttr_target_advice;
 MTTREST AD            DL      ESTCW     ESTCWF       ESTW       ETWF       ETIO
  ------ ----- ---------- ---------- ---------- ---------- ---------- ----------
      34 ON          1000   22610363     1.3382   86049188     1.0711 1658237817
      90 ON          7157   18841862     1.1151   82280687     1.0242 1654469316
     180 ON         17081   16896371          1   80335196          1 1652523825
     270 ON         27005   16136315      0.955   79575140     0.9905 1651763769
     360 ON         36929   15662363      0.927   79101188     0.9846 1651289817
```

该视图评估在不同 FAST_START_MTTR_TARGET 设置下，系统需要执行的 I/O 次数等操作。可以根据数据库的建议，对 FAST_START_MTTR_TARGET 进行相应调整。

这个建议信息的收集受到 Oracle9i 新引入的初始化参数 statistics_level 的控制（关于 statistics_level 参数，我们在其他章节会进行详细介绍），当该参数设置为 Typical 或 ALL 时，MTTR 建议信息被收集：

```
SQL> show parameter statistics_level
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
statistics_level                     string      TYPICAL
```

也可以通过 v$statistics_level 视图来查询 MTTR Advice 的当前设置：

```
SQL> select STATISTICS_NAME,DESCRIPTION
  2  from v$statistics_level where STATISTICS_NAME='MTTR Advice';
STATISTICS_NAME DESCRIPTION
--------------- ----------------------------------------------------------------------
```

```
MTTR Advice     Predicts the impact of different MTTR settings on number of physical I/Os
```

数据库当前的实例恢复状态可以通过视图 v$instance_recovery 查询得到：

```
SQL> select RECOVERY_ESTIMATED_IOS REIO,
  2         ACTUAL_REDO_BLKS ARB,TARGET_REDO_BLKS TRB,LOG_FILE_SIZE_REDO_BLKS LFSRB,
  3         LOG_CHKPT_TIMEOUT_REDO_BLKS LCTRB,LOG_CHKPT_INTERVAL_REDO_BLKS LCIRB,
  4         FAST_START_IO_TARGET_REDO_BLKS FSIOTRB,TARGET_MTTR TMTTR,
  5         ESTIMATED_MTTR EMTTR,CKPT_BLOCK_WRITES CBW
  6  from v$instance_recovery;
REIO     ARB      TRB    LFSRB    LCTRB    LCIRB  FSIOTRB TMTTR   EMTTR       CBW
------  -------  ------ -------  -----    ------- ------- ------ --------  -----
10138   26582    26582   184320   26582                   180      108    3725700
```

从 v$instance_recovery 视图我们可以看到当前数据库估计的平均恢复时间(MTTR)－ESTIMATED_MTTR。ESTIMATED_MTTR 的估算值基于 Dirty Buffer 的数量和日志块数量做出的，这个参数值告所我们，如果此时数据库崩溃，那么进行实例恢复将会需要的时间。

在 v$instance_recovery 视图中，TARGET_MTTR 代表的是期望的平均恢复时间，通常该参数应该等于 FAST_START_MTTR_TARGET 参数设置值（但是如果 FAST_START_MTTR_TARGET 参数定义的值极大或极小，TARGET_MTTR 可能不等于 FAST_START_MTTR_TARGET 的设置）。

当 ESTIMATED_MTTR 接近或超过 FAST_START_MTTR_TARGET 参数设置（v$instance_recovery. TARGET_MTTR）时,系统就会触发检查点，执行写出，写出之后，系统恢复信息将会重新计算（继续执行以上查询，观察数据变化）：

```
SQL> /
REIO  ARB      TRB  LFSRB    LCTRB    LCIRB    FSIOTRB   TMTTR  EMTTR     CBW
---------- ---------- ---------- ----------   ------ --------  -----   ----------
7063   23728    23729 184320      23729                  180      80       3725701
```

在繁忙的系统中，很容易观察到 ESTIMATED_MTTR > TARGET_MTTR，这可能是因为 DBWR 正忙于写出，甚或出现 Checkpoint 不能及时完成的情况。

2.3.8 关于检查点执行的案例

以下案例来自一个生产系统。当执行查询时（查询脚本同前）发现 ESTIMATED_MTTR > TARGET_MTTR：

```
SQL>/
 REIO     ARB      TRB    LFSRB    LCTRB  LCIRB  FSIOTRB  TMTTR   EMTTR      CBW
------- ------- ------- ------- ------- ------- ------- ------ ------- --------
  30337  614392  184320  184320  614392                  180      264  3727074
```

继续查询，发现 ESTIMATED_MTTR 继续升高：

```
SQL> /
  REIO     ARB      TRB    LFSRB    LCTRB  LCIRB  FSIOTRB  TMTTR  EMTTR      CBW
```

```
-------  -------  -----  -------  ------ ------- ------- ------  ------ --------
 24059    614392 184320   184320  614392                    180     303  3727076
```

此时查询 v$session_wait，我们发现数据库处于 checkpoint incomplete 等待：

```
SQL> select sid,seq#,event from v$session_wait;
       SID       SEQ# EVENT
---------- ---------- ----------------------------------------------------
        20       6918 log file switch (checkpoint incomplete)
         2      40139 db file parallel write
         3      32433 db file parallel write
         6       6296 smon timer
。。。。
```

查询 V$LOG 视图，发现除了 Current 日志组外，所有日志组都处于 Active 状态：

```
SQL> select * from v$Log;
 GROUP# THREAD#  SEQUENCE#      BYTES MEMBERS ARCHIVED STATUS   FIRST_CHANGE#
------ ------- ---------- ---------- ------- -------- -------- -------------
     1       1      12104  104857600       1 NO       ACTIVE   8903567335116
     2       1      12105  104857600       1 NO       ACTIVE   8903567337657
     3       1      12106  104857600       1 NO       CURRENT  8903567340158
     4       1      12102  104857600       1 NO       ACTIVE   8903567324018
     5       1      12103  104857600       1 NO       ACTIVE   8903567326497
```

此时，通过 OS 查看 iostat 状态信息，可以发现系统 swap 已经很严重(si,sw 较高),CPU 等待 IO(wa)也很高:

```
[root@neirong root]# vmstat 2
procs                      memory      swap          io     system         cpu
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id wa
 0  3 217184  18160  19680 3383068    1    0     0     4    1     0  2  0  4  1
 0  4 224764  17944   4884 3401520    4 1172  3858  9144  859  1114 12  2 30 55
 1  4 226060  17948   4876 3402996   20  726  4302  9762  944  1181 13  6 17 64
 0  4 226436  21556   4904 3401188   16  368  3646  8782  814  1118 12  2 18 68
 2  2 226828  18124   4912 3405220   16  628  1978  9294  543   840 13  1 20 66
 0  3 227068  19336   4928 3404364   48  162  1174  5682  434   784  7  1 38 54
 0  3 227236  21392   4928 3402748    2  324   642  1856  450   873  0  0 34 66
 0  3 227236  17888   4932 3406344    0   44   770  1660  453   898  0  0 35 64
 0  3 227320  17900   4952 3406516   78  118   720  1680  449   873  0  0 27 73
 0  3 227488  20232   4924 3404608   16  222   784  1710  443   849  0  1 36 63
 0  4 227468  17968   4992 3406640   66   56   794  1584  455   867  0  1 32 67
 0  5 228036  17880   4996 3406764  314  138  1976  8388  549   898 11  2 18 69
 1  4 228340  18052   5000 3407628   40  386  1340  8878  468   798 12  1 18 69
 1  3 228812  18088   5028 3407476  144  394  1398  9038  448   771 11  2 17 70
```

这意味着系统 IO 存在瓶颈,或者系统有突发的大规模写操作。

通过 CKPT_BLOCK_WRITES 字段，可以看出检查点已经写出的数据块的数量，增量检查点的触发以及 DBWR 的持续写出，都会促使该值增加，继续查询，可以观察到随着 CKPT_BLOCK_WRITES 的增加，ESTIMATED_MTTR 开始减少：

```
SQL> /
 REIO     ARB     TRB    LFSRB   LCTRB  LCIRB  FSIOTRB  TMTTR  EMTTR      CBW
------ ------- ------- -------- ------- ------- ------- ------ ------ --------
 23132   614392  184320    184320  614392               180     238    3727077
SQL> /
 REIO     ARB     TRB    LFSRB   LCTRB  LCIRB  FSIOTRB  TMTTR  EMTTR      CBW
------ ------- ------- -------- ------- ------- ------- ------ ------ --------
 17059    614392  184320   184320   614392               180     183  3727088
```

当系统完成一个检查点之后：

```
SQL> select * from v$Log;
GROUP# THREAD#  SEQUENCE#      BYTES    MEMBERS ARCHIVED STATUS   FIRST_CHANGE#
------ ------- ---------- ---------- ---------- -------- -------- -------------
     1       1      12104  104857600          1 NO       INACTIVE 8903567335116
     2       1      12105  104857600          1 NO       ACTIVE   8903567337657
     3       1      12106  104857600          1 NO       ACTIVE   8903567340158
     4       1      12107  104857600          1 NO       ACTIVE   8903567342713
     5       1      12108  104857600          1 NO       CURRENT  8903567345267
```

可以看到 ESTIMATED_MTTR 逐渐恢复到一个较为正常的状态

```
SQL> /
  REIO   ARB     TRB  LFSRB   LCTRB  LCIRB  FSIOTRB  TMTTR   EMTTR      CBW
------- ------- ------- ------ ------- ---- ------- ------ ------- --------
  13076  183735  184320  184320  468895              180     138     3727103
```

2.3.9 Oracle10g 自动检查点调整

从 Oracle10g 开始，数据库可以实现自动调整的检查点（SelfTune Checkpoint），使用自动调整的检查点，Oracle 数据库可以利用系统的低 I/O 负载时段写出内存中的脏数据，从而提高数据库的效率。因此，即使数据库管理员设置了不合理的检查点相关参数，Oracle 仍然能够通过自动调整将数据库的 Crash Recovery 时间控制在合理的范围之内。

当 FAST_START_MTTR_TARGET 参数未设置时，自动检查点调整生效。

通常，如果我们必须严格控制实例或节点恢复时间，那么我们可以设置 FAST_START_MTTR_TARGET 为期望时间值；如果恢复时间不需要严格控制，那么我们可以不设置 FAST_START_MTTR_TARGET 参数，从而启用 Oracle10g 的自动检查点调整特性。

当取消 FAST_START_MTTR_TARGET 参数设置之后：

```
SQL> show parameter fast_start_mttr
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
fast_start_mttr_target               integer     0
```

在启动数据库的时候，我们可以从 alert 文件中看到如下信息：

```
Wed Jan 11 16:28:12 2006
MTTR advisory is disabled because FAST_START_MTTR_TARGET is not set
```

检查 v$instance_recovery 视图，我们可以发现 Oracle10g 中的改变：

```
SQL> select RECOVERY_ESTIMATED_IOS REIOS,TARGET_MTTR TMTTR,
  2  ESTIMATED_MTTR EMTTR,WRITES_MTTR WMTTR,WRITES_OTHER_SETTINGS WOSET,
  3  CKPT_BLOCK_WRITES CKPTBW,WRITES_AUTOTUNE WAUTO,WRITES_FULL_THREAD_CKPT WFTCKPT
  4  from v$instance_recovery;
 REIOS   TMTTR     EMTTR      WMTTR    WOSET     CKPTBW    WAUTO    WFTCKPT
------   -----    -----      ------   -----    ---------- -------  --------
 49407   0         68          0          0      3649819   3506125   3130700
```

在以上视图中，WRITES_AUTOTUNE 字段值就是指由于自动调整检查点执行的写出次数，而 CKPT_BLOCK_WRITES 指的则是由于检查点写出的 Block 的数量。

以下是来自生产环境的查询输出，注意第一个查询返回的 DBWR checkpoint buffers written 信息就是指 DBWR 执行检查点写出的 Buffer 数量：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Prod
SQL> select name,value from v$sysstat where upper(name) like '%DBWR%';
NAME                                                                  VALUE
---------------------------------------------------------------- ----------
DBWR checkpoint buffers written                                     8262301
DBWR thread checkpoint buffers written                                    0
DBWR tablespace checkpoint buffers written                                0
DBWR parallel query checkpoint buffers written                            0
DBWR object drop buffers written                                          0
DBWR transaction table writes                                        337168
DBWR undo block writes                                              1398013
DBWR revisited being-written buffer                                       0
DBWR make free requests                                                   0
DBWR lru scans                                                            0
DBWR checkpoints                                                       4770
DBWR fusion writes                                                        0
```

在自动调整检查点下，DBWR checkpoint buffers written 和 V$INSTANCE_RECOVERY 中

的 WRITES_AUTOTUNE 相等，正说明数据库的检查点都是通过自动调整实现的：

```
SQL> select RECOVERY_ESTIMATED_IOS REIOS,TARGET_MTTR TMTTR,
  2  ESTIMATED_MTTR EMTTR,WRITES_MTTR WMTTR,WRITES_OTHER_SETTINGS WOSET,
  3  CKPT_BLOCK_WRITES CKPTBW,WRITES_AUTOTUNE WAUTO,WRITES_FULL_THREAD_CKPT WFTCKPT
  4  from v$instance_recovery;
     REIOS      TMTTR      EMTTR      WMTTR      WOSET     CKPTBW      WAUTO    WFTCKPT
---------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
       207          0         38          0          0    7634863    8262301          0
```

进一步的，以下来自 Oracle11g 生产环境的查询可以更为详细的反应检查点信息：

```
SQL> select name,value from v$sysstat where upper(name) like '%DBWR%';
NAME                                                                  VALUE
---------------------------------------------------------------- ----------
DBWR checkpoint buffers written                                    6004076
DBWR thread checkpoint buffers written                                   0
DBWR tablespace checkpoint buffers written                           23866
DBWR parallel query checkpoint buffers written                           0
DBWR object drop buffers written                                         6
DBWR transaction table writes                                       623681
DBWR undo block writes                                             1819188
DBWR revisited being-written buffer                                      7
DBWR lru scans                                                           0
DBWR checkpoints                                                     33358
DBWR fusion writes                                                       0
SQL> select RECOVERY_ESTIMATED_IOS REIOS,TARGET_MTTR TMTTR,
ESTIMATED_MTTR EMTTR,WRITES_MTTR WMTTR,WRITES_OTHER_SETTINGS WOSET,
  2    3  CKPT_BLOCK_WRITES CKPTBW,WRITES_AUTOTUNE WAUTO,WRITES_FULL_THREAD_CKPT WFTCKPT
  4  from v$instance_recovery;
     REIOS      TMTTR      EMTTR      WMTTR      WOSET     CKPTBW      WAUTO    WFTCKPT
---------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
        35          0         19          0          0    4544246    5980210          0
```

注意这里的输出显示，表空间检查点的写出不是通过自动调整检查点完成的：

```
WRITES_AUTOTUNE = DBWR checkpoint buffers written - DBWR tablespace checkpoint buffers written
SQL> select 6004076 -23866 - 5980210 from dual;
6004076-23866-5980210
---------------------
                    0
```

这是由于 FILEQ 队列的存在，Oracle 可以通过 FILEQ 而不是 CKPTQ 来针对表空间检查点事件进行写出。和自我调整检查点相关的参数有如下几个：

```
SQL> @GetHidPar
```

```
Enter value for par: selftune
NAME                              VALUE PDESC
--------------------------------- ----- --------------------------------------------------
_selftune_checkpoint_write_pct  3     Percentage of total physical i/os for self-tune ckpt
_disable_selftune_checkpointing FALSE Disable self-tune checkpointing
_selftune_checkpointing_lag     300   Self-tune checkpointing lag the tail of the redo log
```

这里使用到一个获取隐含参数的脚本 GetHidPar.sql，该脚本在 SYS 用户下执行可以获得数据库的隐含参数及常规参数，其内容如下，在本书中该脚本在多处用到：

```
set linesize 120
col name for a30
col value for a10
col PDESC for a50
SELECT x.ksppinm NAME, y.ksppstvl VALUE, x.KSPPDESC PDESC
  FROM SYS.x$ksppi x, SYS.x$ksppcv y
WHERE x.indx = y.indx
   AND x.ksppinm LIKE '%&par%'
/
```

关于检查点的机制问题，我们侧重介绍了原理，至于具体的算法实现，可以不需要去追究过多，只要明白了这些原理性的规则，理解 Oracle 就会变成轻松的事情。

Oracle 的算法改进是一种优化，我们对于数据库的调整优化也不外如此，借鉴 Oracle 的优化对于我们理解和优化 Oracle 数据库具有极大的好处。

2.3.10检查点信息及恢复起点

在控制文件的转储中，可以看到关于检查点进程进度的记录：

```
**************************************************************************
CHECKPOINT PROGRESS RECORDS
**************************************************************************
 (blkno = 0x4, size = 104, max = 1, in-use = 1, last-recid= 0)
THREAD #1 - status:0x2 flags:0x0 dirty:20
low cache rba:(0x10.1d6.0) on disk rba:(0x10.1e1.0)
on disk scn: 0x0000.0023af04 11/22/2004 17:04:55
resetlogs scn: 0x0000.001cff68 11/16/2004 14:10:35
heartbeat: 542912976 mount id: 3154897498
```

这里 **low cache rba**（recovery block address）指在 cache 中，最低的 rba 地址，在实例恢复或者崩溃恢复中，需要从这里开始恢复。on disk rba 是磁盘上的最高的重做值，在进行恢复时应用重做至少要达到这个值。

2.3.11 正常关闭数据库的状况

在第一章中曾经提到，在关闭数据库的多种方式中，除 shutdown abort 之外，其他方式在关闭数据库后下次启动时都不需要执行恢复，这是因为 shutdown 时执行了完全检查点（Full Checkpoint），所有脏数据已经写入到数据文件。在控制文件的数据文件信息部分，对于每个数据文件都有一个"Checkpoint SCN"和"Stop SCN"，用于进行启动时校验判断。

下面是来自一个 Clean Shutdown 的数据库的控制文件和数据文件头的内容。因为数据库在关闭之前执行了完全检查点，所以线程检查点 SCN 和所有数据文件检查点 SCN 和数据文件 Stop SCN 都一致。

首先通过 shutdown immediate 关闭数据库，然后在 Mount 状态转储获取控制文件内容：

```
SQL> shutdown immediate
SQL> startup mount;
SQL> alter session set events 'immediate trace name CONTROLF level 12';
Session altered.
SQL> @gettrcname
TRACE_FILE_NAME
--------------------------------------------------------------------------------
/opt/oracle/admin/conner/udump/conner_ora_23234.trc
```

这个 trace 文件里就记录了控制文件的详细内容。

数据库的相关信息

Database Entry 部分记录有数据库上次成功完成的检查点信息以及控制文件的检查点信息：

```
**********************************************************************
DATABASE ENTRY
**********************************************************************
Database checkpoint: Thread=1 scn: 0x0000.00235d1f  --这里是检查点 SCN
 Threads: #Enabled=1, #Open=0, Head=0, Tail=0
。。。。。。
Arch list: Head=0, Tail=0, Force scn: 0x0000.002179cascn: 0x0000.001caf68
 Controlfile Checkpointed at scn:  0x0000.0022cb87 11/22/2004 03:02:15
```

Redo 检查点信息

REDO 线程记录中同样记录了检查点信息，这个检查点与 Database 检查点一致：

```
**********************************************************************
REDO THREAD RECORDS
**********************************************************************
 (blkno = 0x4, size = 104, max = 1, in-use = 1, last-recid= 0)
THREAD #1 - status:0xe thread links forward:0 back:0
 #logs:3 first:1 last:3 current:2 last used seq#:0xf
 enabled at scn: 0x0000.001cff68 11/16/2004 14:10:35
```

```
 disabled at scn: 0x0000.00000000 01/01/1988 00:00:00
 opened at 11/16/2004 14:10:37 by instance conner
Checkpointed at scn:  0x0000.00235d1f 11/22/2004 16:12:54  --检查点信息
 thread:1 rba:(0xf.490a.10)
```

数据文件检查点信息

抽取一个数据文件的信息作为示例，注意其检查点信息和 Database Entry 部分的数据库检查点一致：

```
********************************************************************************
DATA FILE #4:
  (name #8) /opt/oracle/oradata/conner/eygle01.dbf
creation size=25600 block size=8192 status=0xe head=8 tail=8 dup=1
 tablespace 4, index=5 krfil=4 prev_file=0
 unrecoverable scn: 0x0000.00000000 01/01/1988 00:00:00
 Checkpoint cnt:223 scn: 0x0000.00235d1f 11/22/2004 16:12:54 --检查点 SCN
 Stop scn: 0x0000.00235d1f 11/22/2004 16:12:54               --Stop SCN
 Creation Checkpointed at scn:  0x0000.0005fd18 10/25/2004 23:34:32
```

注意这里，数据库正常关闭后，由于执行了完全检查点，数据文件处于一致的状态，检查点 SCN 在此等于 Stop SCN。

在此情况下，由于数据库处于一致状态，如果数据文件没有损失，下次启动 Oracle 就能够通过验证，顺利启动。

2.3.12数据库异常关闭的情况

如果数据库异常关闭，则不会执行任何检查点。通过 Shutdown abort 可以模拟一次异常，当使用 shutdown abort 方式关闭数据库时，Oracle 会立即中断所有事务，关闭当前所有数据库连接，不执行检查点，立即关闭数据库。使用这种方式关闭数据库和断电故障类似，数据库在下一次启动时必须执行实例恢复才能够启动。除非在特别紧急的情况下，否则我们通常不建议使用这种方式关闭数据库。

以下测试来自 Oracle 11.2.0.3 ,详细说明了异常关闭数据库的可能影响：

```
SQL> shutdown abort;
SQL> startup mount;
SQL> alter session set events 'immediate trace name CONTROLF level 12';
Session altered.
SQL> @gettrcname
TRACE_FILE_NAME
--------------------------------------------------------------------------------
/u01/app/oracle/diag/rdbms/eygle/eygle/trace/eygle_ora_14786.trc
```

下面来看看此时控制文件的内容。

2.3.12.1 数据库的相关信息

在 Database Entry 部分，我们可以看到数据库的 Thread Checkpoint 信息：

```
**********************************************************************
DATABASE ENTRY
**********************************************************************
 (size = 316, compat size = 316, section max = 1, section in-use = 1,
  last-recid= 0, old-recno = 0, last-recno = 0)
 (extent = 1, blkno = 1, numrecs = 1)
 06/14/2012 17:02:30
 DB Name "EYGLE"
 Database flags = 0x00404000 0x00001000
 Controlfile Creation Timestamp  06/14/2012 17:02:30
 Incmplt recovery scn: 0x0000.00000000
 Resetlogs scn: 0x0000.00000001 Resetlogs Timestamp  06/14/2012 17:02:30
 Prior resetlogs scn: 0x0000.00000000 Prior resetlogs Timestamp  01/01/1988 00:00:00
 Redo Version: compatible=0xb200000
 #Data files = 3, #Online files = 3
 Database checkpoint: Thread=1 scn: 0x0000.000038e9
 Threads: #Enabled=1, #Open=1, Head=1, Tail=1
 enabled  threads:  01000000 00000000 00000000 00000000 00000000 00000000
 Max log members = 2, Max data members = 1
 Arch list: Head=0, Tail=0, Force scn: 0x0000.00000000scn: 0x0000.00000000
 Activation ID: 1605014374
 Controlfile Checkpointed at scn:  0x0000.000285e5 06/19/2012 16:22:23
 thread:0 rba:(0x0.0.0)
 enabled  threads:  00000000 00000000 00000000 00000000 00000000 00000000
```

2.3.12.2 检查点信息记录

以下部分记录了检查点信息,其中包含了 Low Cache RBA 和 On Disk RBA 信息:

```
**********************************************************************
CHECKPOINT PROGRESS RECORDS
**********************************************************************
 (size = 8180, compat size = 8180, section max = 4, section in-use = 0,
  last-recid= 0, old-recno = 0, last-recno = 0)
 (extent = 1, blkno = 2, numrecs = 4)
THREAD #1 - status:0x2 flags:0x0 dirty:7
```

```
low cache rba:(0x1.1006e.0) on disk rba:(0x1.10077.0)
on disk scn: 0x0000.000285fa 06/19/2012 16:23:07
resetlogs scn: 0x0000.00000001 06/14/2012 17:02:30
heartbeat: 786373933 mount id: 1605518565
```

2.3.12.3 控制文件记录的 redo 信息

在控制文件中，也可以找到 REDO THREAD 的检查点信息：

```
**************************************************************************
REDO THREAD RECORDS
**************************************************************************
 (size = 256, compat size = 256, section max = 1, section in-use = 1,
  last-recid= 0, old-recno = 0, last-recno = 0)
 (extent = 1, blkno = 5, numrecs = 1)
THREAD #1 - status:0xf thread links forward:0 back:0
 #logs:2 first:1 last:2 current:1 last used seq#:0x1
 enabled at scn: 0x0000.00000001 06/14/2012 17:02:36
 disabled at scn: 0x0000.00000000 01/01/1988 00:00:00
 opened at 06/14/2012 17:02:37 by instance eygle
Checkpointed at scn:  0x0000.000038e9 06/14/2012 17:02:55
 thread:1 rba:(0x1.c8b2.10)
```

2.3.12.4 数据文件检查点信息

同样，以下是控制文件中记录的数据文件检查点信息：

```
**************************************************************************
DATA FILE RECORDS
**************************************************************************
 (size = 520, compat size = 520, section max = 30, section in-use = 3,
  last-recid= 3, old-recno = 0, last-recno = 0)
 (extent = 1, blkno = 7, numrecs = 30)
DATA FILE #1:
  name #3: /home/oracle/oradata/EYGLE/datafile/o1_mf_system_7xmb5ff1_.dbf
creation size=12800 block size=8192 status=0xe head=3 tail=3 dup=1
 tablespace 0, index=1 krfil=1 prev_file=0
 unrecoverable scn: 0x0000.00000000 01/01/1988 00:00:00
 Checkpoint cnt:3 scn: 0x0000.000038e9 06/14/2012 17:02:55
 Stop scn: 0xffff.ffffffff 06/14/2012 17:02:37
 Creation Checkpointed at scn:  0x0000.00000007 06/14/2012 17:02:37
 thread:1 rba:(0x1.3.10)
```

注意此处，由于数据库是异常关闭，数据库没有完成最后的检查点，数据文件的 Stop SCN 仍然为无穷大（**ffffffff**）。

在以上的信息中，各部分的 Checkpoint SCN 都一致，停留在时间点: 06/14/2012 17:02:55. 但是数据文件的 Stop SCN 不等于 Checkpoint SCN，这意味着数据库上一次关闭没有执行完全检查点，是异常关闭,此时启动数据库需要进行恢复。

那么恢复执行到的位置是哪里呢?

根据增量检查点的记录,On Disk RBA 已经记录到时间 06/19/2012 16:23:07 :

```
low cache rba:(0x1.1006e.0) on disk rba:(0x1.10077.0)
on disk scn: 0x0000.000285fa 06/19/2012 16:23:07
```

也就是说,由于数据库缺乏活动,数据文件头上自 06/14/2012 17:02:55 之后再未执行检查点信息写入更新.

2.3.12.5 数据库的实例恢复

在数据库异常关闭之后，下次启动时，Oracle 会自动执行实例恢复（Instance Recovery），实例恢复包括两个步骤:Cache Recovery 和 Transaction Recovery。

继续以上的测试，在启动数据库之后可以从 alert_<sid>.log 文件中获得数据库关于恢复的相关信息:

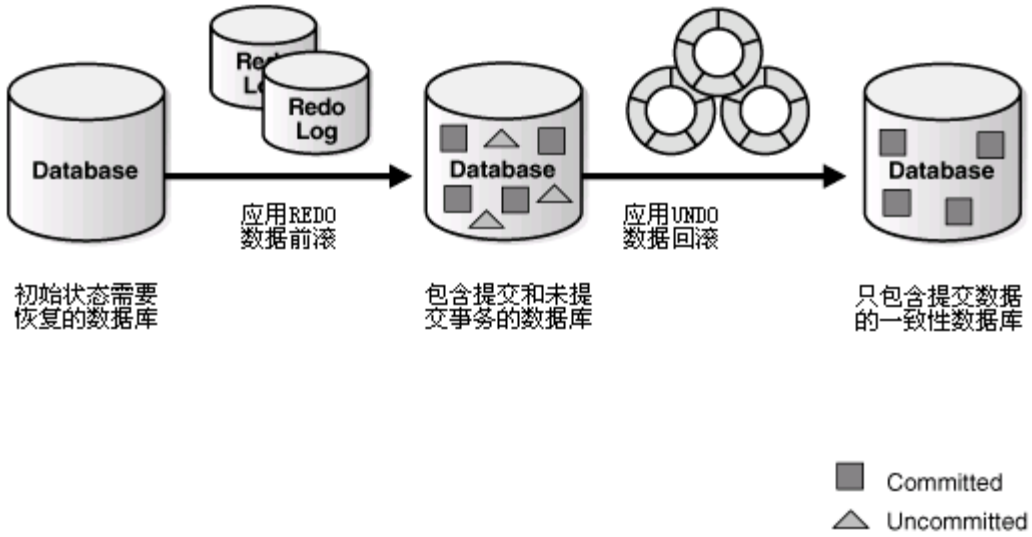
```
Tue Jun 19 16:41:17 2012
alter database open
Beginning crash recovery of 1 threads
 parallel recovery started with 2 processes
Started redo scan
Completed redo scan
 read 4 KB redo, 7 data blocks need recovery
Started redo application at
 Thread 1: logseq 1, block 65646
Recovery of Online Redo Log: Thread 1 Group 1 Seq 1 Reading mem 0
  Mem# 0: /home/oracle/oradata/EYGLE/onlinelog/o1_mf_1_7xmb56lg_.log
Completed redo application of 0.00MB
Completed crash recovery at
 Thread 1: logseq 1, block 65655, scn 185370
 7 data blocks read, 7 data blocks written, 4 redo k-bytes read
```

计算一下,日志中提示的恢复结束点日志块信息 65655 转换为 16 进制就是 0x10077,正是 On Disk RBA 的恢复终点.

我们注意到 Oracle 在恢复过程中，首先读取日志，从最后完成的检查点开始，应用所有重做记录，这个过程叫前滚（Rolling Forward），也就是 Cache Recovery 过程，完成前滚之后，数据库可以被打开提供访问和使用，但是此时数据库可能包含已提交和未提交事务。

此后进入实例恢复的第二阶段事务恢复（Transaction Recovery），在这一阶段 Oracle 将回滚未提交事务，使数据库恢复到一致状态。

下图说明了 Cache Recovery 以及 Transaction Recovery 的过程：



Oracle 使用两个特点来增加事务恢复阶段的效率,这两个特点是:Fast-Start On-Demand rollback 和 Fast-Start Parallel Rollback(这些特点是 Fast-Start Fault Recovery 的组成部分，仅在 Oracle8i 之后的企业版中可用)。

使用 fast-start on-demand rollback 特点，Oracle 自动允许在数据库打开之后开始新的事务，这通常只需要很短的 Cache Recovery 时间。如果一个用户试图访问被异常中止进程锁定的记录，Oracle 回滚那些新事务请求的记录，也就是说，因需求而回滚。因而，新事务不需要等待漫长的事务回滚时间。

在 fast-start parallel rollback 中, 后台进程 SMON 充当一个调度员，使用多个服务器进程并行回滚一个事务集。

Fast-start parallel rollback 主要对于长时间运行的未提交事务有效，尤其是并行 INSERT, UPDATE, 和 DELETE 等操作。SMON 自动决定何时开始并行回滚并且自动在多个进程之间分散工作。Fast-start parallel rollback 的一个特殊形式是内部事务恢复(intra-transaction recovery)。在内部事务恢复中，一个大的事务可以被拆分，分配给几个服务器进程并行回滚。

可以通过初始化参数 FAST_START_PARALLEL_ROLLBACK 来控制并行回滚，该参数有三个参数值：

- FALSE-禁用 fast-start parallel rollback
- LOW-限制恢复进程不能超过 2 倍的 CPU_COUNT
- HIGH-限制恢复进程不能超过 4 倍的 CPU_COUNT

2.3.13 数据库并行恢复案例一则

前面一节提到，数据库在发生故障之后，在恢复阶段可以进行 Fast-start parallel rollback。以下是和并行恢复相关的一个具体案例。

故障起因是由于工程师对一个 10G 大表进行 imp 操作，imp 语句如下

```
imp username/passwd file=G:\mobile.dmp log=G:\ mobile_old.log ignore=y buffer=8192000
feedback=10000
```

这里导出文件 mobile.dmp 的大小有 6G，由于失误未使用 commit=y 参数，这导致导入数据没有分批量提交，当导入执行了几个小时后，由于对业务系统产生了极大的压力和影响，imp 操作被强行中断，这个中断导致了这个大事务的回滚，噩梦从此开始。

随后对该对象的 Trucate/Drop 操作都随之挂起，系统开始回滚，检查回滚段可以找到这个大事务的回滚段：

```
SQL> select * from v$rollstat where xacts >0;
USN LATCH EXTENTS     RSSIZE WRITES XACTS   GETS WAITS OPTSIZE    HWMSIZE SHRINKS  WRAPS
EXTENDS  AVESHRINK  AVEACTIVE STATUS CUREXT CURBLK
--- ----- ------- ---------- ------ ----- ------ ----- ------- ---------- -------- -----
------- ---------- ---------- ------ ------ ------
  3     3    3571 2066300928      0     1 445173    41         2066300928       0      0
0          0          0 ONLINE   3569    159
```

注意到这个回滚段已经扩展到 2,066,300,928 Bytes 大小，也就是约 2G 大小，这个回滚段所涵盖的数据都需要回滚。初始的系统参数 FAST_START_PARALLEL_ROLLBACK 设置为 LOW：

```
SQL> show parameter parallel_rollback
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
fast_start_parallel_rollback         string      LOW
SQL> show parameter cpu_count
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
cpu_count                            integer     4
```

系统启动了 8 个并行进程进行并行回滚，但是回滚执行了 10 个多小时仍然未完成（v$session 的等待事件显示并行恢复产生了竞争，在此类情况下，通常串行恢复是更适宜的）：

```
SQL>  select sid,program,event from v$session where PROGRAM like '%P%';
       SID PROGRAM              EVENT
---------- -------------------- ----------------------------------------
       533 ORACLE.EXE (P003)    db file sequential read
       534 ORACLE.EXE (P006)    db file sequential read
       535 ORACLE.EXE (P007)    wait for a undo record
       536 ORACLE.EXE (P004)    db file sequential read
       537 ORACLE.EXE (P005)    db file sequential read
       538 ORACLE.EXE (P002)    wait for a undo record
       540 ORACLE.EXE (P000)    wait for a undo record
       541 ORACLE.EXE (P001)    db file sequential read
```

```
550 ORACLE.EXE (CKPT)    rdbms ipc message
554 ORACLE.EXE (PSP0)    rdbms ipc message
555 ORACLE.EXE (PMON)    pmon timer
```

随后尝试修改了 FAST_START_PARALLEL_ROLLBACK 设置为 HIGH，然后数据库启动了 16 个并行进程进行恢复：

```
Wed Jan 24 09:21:06 2007
SMON: parallel recovery restart with degree=16 (!=8)
Wed Jan 24 09:21:18 2007
SMON: Restarting fast_start parallel rollback
Wed Jan 24 09:21:19 2007
ALTER SYSTEM SET fast_start_parallel_rollback='HIGH' SCOPE=BOTH;
```

这个恢复最终持续了一个小时左右，以 SMON 出错而结束，最后 SMON 终止了 16 个并行进程，启动了一个恢复进程进行恢复，smon 的 TRACE 文件报了如下信息：

```
*** 2007-01-24 09:21:06.334
*** SERVICE NAME:(SYS$BACKGROUND) 2007-01-24 09:21:06.318
*** SESSION ID:(549.1) 2007-01-24 09:21:06.318
*** 2007-01-24 09:21:06.334
SMON: parallel recovery restart with degree=16 (!=8)
Parallel Transaction recovery caught exception 30312
*** 2007-01-24 09:21:18.553
Parallel Transaction recovery caught error 30312
*** 2007-01-24 09:21:18.553
SMON: Restarting fast_start parallel rollback
*** 2007-01-24 11:16:17.697
Parallel Transaction recovery caught exception 12801
Parallel Transaction recovery caught error 370
*** 2007-01-24 11:16:20.885
SMON: Restarting fast_start parallel rollback
Dead transaction 0x0003.028.00017d2a recovered by 1 server(s)
```

最后两行至为关键，Oracle 通过 1 个 Server 进程进行恢复，通过查询视图 V$FAST_START_TRANSACTIONS，可以发现恢复的速度加快了许多：

```
SQL> select usn,undoblockdone,undoblockstotal,cputime from V$FAST_START_TRANSACTIONS;
  USN       UNDOBLOCKSDONE      UNDOBLOCKSTOTAL      CPUTIME
---------- ---------------      -------------------  ---------------
  3          1036                203328              8208
SQL> select * from V$FAST_START_SERVERS;
STATE       UNDOBLOCKSDONE        PID XID
----------- -------------- ---------- ----------------
RECOVERING          1036         16 030028002A7D0100
```

UNDOBLOCKSDONE 表示已经恢复的回滚段块，UNDOBLOCKSTOTAL 表示需要恢复的回滚段块总量，可以看出，回滚段 3 在参与恢复的操作，203328 中有 1036 块已经恢复，cpu 占用了 8208 秒。最终再通过近 3 个小时的恢复，数据库完成了这次灾难性的回滚。

从 v$fast_start_transactions 视图我们可以看到具体信息，恢复共花费了 10255 秒的时间，而平均每秒数据库仅对约 20 个 UNDO BLOCK 完成了回滚：

```
SQL> select * from v$fast_start_transactions;
USN SLT   SEQ STATE     UNDOBLOCKSDONE UNDOBLOCKSTOTAL CPUTIME  XID              RCVSERVERS
--- --- ----- --------- -------------- --------------- -------  ---------------- ----------
 3  40 97578 RECOVERED         203328          203328   10255  030028002A7D0100          1
```

通过这个案例我们知道 Oracle 的并行回滚有时候并不可靠，而且的确还存在大量的 Bug，从 Oracle 的官方网站上可以找到大量类似的案例。

SMON 对于死事务执行恢复，如果事务很大，Oracle 将首先尝试并行恢复，并行恢复 Oracle 将采用 intra-transaction 并行恢复，也就是使用多个并行进程恢复死事务，如果 intra-transaction 并行恢复失败，SMON 将指定单进程进行恢复，这个恢复过程将会十分缓慢，特别是当事务很大时。在恢复期间，SMON 进程还可能停止其他类别的服务，如排序段请求、实例恢复请求等，同时还可能导致查询十分缓慢（由于大量的 CR 回滚等），从外在看来，数据库就可能处于停顿状态。这些问题最终被定义为一个 Bug，Oracle 在 Oracle 11g 中进行了改进。

然而这种问题应该极少被遇到，做为一个 DBA 也应该避免陷于这样的境地，将这个案例收录于此，供大家借鉴。

2.3.14判断一个死事务的恢复进度

很多时候 DBA 在处理类似上一节案例的时候，往往先是通过 shutdown abort 方式重新打开数据库，期望能够缓解性能问题，然而实际情况是，Oracle 的回滚无法终止，必须等待回滚完成，相关的锁定和资源才能释放。但是一旦重启数据库之后，v$transaction 事务表中的事务信息将会消失，恢复事务变成了死事务（Dead Transaction）。

那么如何来判断一个死事务的恢复进度呢？

这可以通过 Oracle 的一个内部表 x$ktuxe 来进行查询，该表会记录 Dead 事务的恢复进度。首先可以根据状态信息查看事务情况：

```
SQL> select distinct KTUXECFL,count(*) from x$ktuxe group by KTUXECFL;
KTUXECFL                  COUNT(*)
------------------------ ----------
DEAD                              1
NONE                           2393
SCO|COL                           8
```

通过观察 KTUXESIZ 字段可以来评估恢复进度：

```
SQL> select ADDR,KTUXEUSN,KTUXESLT,KTUXESQN,KTUXESIZ
  2 from x$ktuxe where  KTUXEUSN=10 and KTUXESLT=39;
ADDR             KTUXEUSN  KTUXESLT  KTUXESQN  KTUXESIZ
```

```
---------------- ---------- ---------- ---------- ----------
FFFFFFFF7D07B91C         10         39    2567412    1086075
SQL> select ADDR,KTUXEUSN,KTUXESLT,KTUXESQN,KTUXESIZ
  2 from x$ktuxe where  KTUXEUSN=10 and KTUXESLT=39;
ADDR             KTUXEUSN  KTUXESLT  KTUXESQN  KTUXESIZ
---------------- ---------- ---------- ---------- ----------
FFFFFFFF7D07B91C         10         39    2567412    1086067
```

根据评估，这个事务回滚需要大约 2.55 天：

```
SQL> declare
 2  l_start number;
 3  l_end    number;
 4  begin
 5    select ktuxesiz into l_start from x$ktuxe where  KTUXEUSN=10 and KTUXESLT=39;
 6    dbms_lock.sleep(60);
 7    select ktuxesiz into l_end from x$ktuxe where  KTUXEUSN=10 and KTUXESLT=39;
 8    dbms_output.put_line('time est Day:'|| round(l_end/(l_start -l_end)/60/24,2));
 9  end;
 10  /
time est Day:2.55
```

2.4数据库的初始化

在了解了控制文件与数据库起停的关系之后，现在再继续深入探讨一下在数据库 Open 的过程中，后台所执行的深层次的初始化操作。

2.4.1　bootstrap$及数据库初始化过程

数据库的信息都是存放在数据文件当中的，但是当数据库尚未打开之前，Oracle 是无法获得这部分数据的。那么 Oracle 是怎样完成这个从数据文件到内存的初始化过程的呢?

首先通过以下步骤对数据库的 OPEN 过程进行跟踪，研究获得的跟踪文件：

```
SQL> startup mount;
SQL> alter session set sql_trace = true;
SQL> alter database open;
```

以上通过 SQL_TRACE 获得一个跟踪文件，跟踪文件里将记录从 mount 到 open 的过程中，Oracle 所执行的后台操作。可以通过 tkprof 工具对跟踪文件进行格式化，使得其中的信息更便于阅读。首先我们来参考跟踪文件的前面部分（以下跟踪文件来自 Oracle9iR2），这是第一个对象的创建：

```
create table bootstrap$ ( line# number not null, obj#
number not null, sql_text varchar2(4000) not null) storage (initial
```

```
50K objno 56 extents (file 1 block 377))
```

注意，在这一步骤中，实际上 Oracle 是在内存中创建 bootstrap$的结构，然后从数据文件的 file 1 block 377 读取数据到内存中，完成第一次初始化。

> 提示：file 1 block 377 子句是内部语句，该语法对用户是不可用的。

从数据库的创建脚本 $ORACLE_HOME/rdbms/admin/sql.bsq 文件中，可以获得 bootstrap$表的初始创建语句：

```
create table bootstrap$
( line#         number not null,                        /* statement order id */
  obj#          number not null,                             /* object number */
  sql_text      varchar2("M_VCSZ") not null)                    /* statement */
  storage (initial 50K)           /* to avoid space management during IOR I */
//                                                  /* "//" required for bootstrap */
```

接下来从数据库中查询一下，file 1 block 377 上存储的是什么对象：

```
SQL> select segment_name,file_id,block_id
  2  from dba_extents where block_id=377;
SEGMENT_NAME       FILE_ID   BLOCK_ID
--------------- ----------  ---------
BOOTSTRAP$               1        377
```

File 1 Block 377 开始存放的正是 Bootstrap$对象。

继续查看 Trace 文件的内容，Oracle 进一步执行的是如下操作：

```
select line#, sql_text from bootstrap$ where obj# != :1
```

在创建并从数据文件中装载了 bootstrap$的内容之后，Oracle 开始递归的从该表中读取信息，加载数据。那么 bootstrap$中记录的是什么信息呢？

在数据库中，bootstrap$是一张实际存在的系统表：

```
SQL> desc bootstrap$
 Name            Null?    Type
 --------------- -------- --------------------
 LINE#           NOT NULL NUMBER
 OBJ#            NOT NULL NUMBER
 SQL_TEXT        NOT NULL VARCHAR2(4000)
```

来看一下这张表的具体内容:

```
SQL> select * from bootstrap$ where rownum <5;
 LINE#  OBJ# SQL_TEXT
------ ----- --------------------------------------------------------------------------
    -1    -1 8.0.0.0.0
     0     0 CREATE ROLLBACK SEGMENT SYSTEM STORAGE (  INITIAL 112K NEXT 1024K MINE
               XTENTS 1 MAXEXTENTS 32765 OBJNO 0 EXTENTS (FILE 1 BLOCK 9))
     8     8 CREATE CLUSTER C_FILE#_BLOCK#("TS#" NUMBER,"SEGFILE#" NUMBER,"SEGBLOCK
```

```
#" NUMBER) PCTFREE 10 PCTUSED 40 INITRANS 2 MAXTRANS 255 STORAGE (  IN
ITIAL 24K NEXT 1024K MINEXTENTS 1 MAXEXTENTS 2147483645 PCTINCREASE 0
OBJNO 8 EXTENTS (FILE 1 BLOCK 73)) SIZE 225
9     9 CREATE INDEX I_FILE#_BLOCK# ON CLUSTER C_FILE#_BLOCK# PCTFREE 10 INITR
ANS 2 MAXTRANS 255 STORAGE (  INITIAL 64K NEXT 1024K MINEXTENTS 1 MAXE
XTENTS 2147483645 PCTINCREASE 0 OBJNO 9 EXTENTS (FILE 1 BLOCK 81))
```

以上输出只显示了表中的 4 条记录，大家可以自行研究一下其他记录的内容。从这些语句中可以看出，bootstrap$ 中实际上是记录了一些数据库系统基本对象的创建语句。Oracle 通过 bootstrap$进行引导，进一步创建相关的重要对象，从而启动了数据库。

如果向前追溯，可以继续考察一下 bootstrap$的创建过程。查看一下创建数据库的脚本，可以发现数据库在创建过程中最先运行的是一个叫做 CreateDB.sql 的脚本。这个脚本发出 CREATE DATABASE 的命令，具体类似如下的例子：

```
CREATE DATABASE eygle
MAXINSTANCES 1 MAXLOGHISTORY 1 MAXLOGFILES 5 MAXLOGMEMBERS 3 MAXDATAFILES 100
DATAFILE '/opt/oracle/oradata/eygle/system01.dbf'
SIZE 250M REUSE AUTOEXTEND ON NEXT  10240K MAXSIZE UNLIMITED EXTENT MANAGEMENT
LOCAL
DEFAULT TEMPORARY TABLESPACE TEMP TEMPFILE '/opt/oracle/oradata/eygle/temp01.dbf'
SIZE 40M REUSE AUTOEXTEND ON NEXT  640K MAXSIZE UNLIMITED
UNDO TABLESPACE "UNDOTBS1" DATAFILE '/opt/oracle/oradata/eygle/undotbs01.dbf'
SIZE 200M REUSE AUTOEXTEND ON NEXT  5120K MAXSIZE UNLIMITED
CHARACTER SET ZHS16GBK NATIONAL CHARACTER SET AL16UTF16
LOGFILE GROUP 1 ('/opt/oracle/oradata/eygle/redo01.log') SIZE 10240K,
GROUP 2 ('/opt/oracle/oradata/eygle/redo02.log') SIZE 10240K,
GROUP 3 ('/opt/oracle/oradata/eygle/redo03.log') SIZE 10240K;
exit;
```

在这个创建过程中，Oracle 会隐含的调用$ORACLE_HOME/rdbms/admin/sql.bsq 脚本，用于创建数据字典。这个文件的位置受到一个隐含的初始化参数 （_init_sql_file ）的控制:

```
SQL> @GetParDescrb.sql
Enter value for par: init_sql
NAME            VALUE                DESCRIB
--------------- -------------------- ------------------------------------------------------
_init_sql_file  ?/rdbms/admin/sql.bsq File containing SQL statements to execute upon database
                                     creation
```

如果在创建过程中，Oracle 无法找到 sql.bsq 文件，则数据库创建将会出错，我们可以测试一下移除 sql.bsq 文件，再看这样一个数据库创建过程:

```
SQL> startup nomount;
SQL> @CreateDB.sql
CREATE DATABASE eygle
```

```
*
ERROR at line 1:
ORA-01092: ORACLE instance terminated. Disconnection forced
```

此时日志中会记录如下信息:

```
Fri Aug 18 15:45:49 2006
Errors in file /opt/oracle/admin/eygle/udump/eygle_ora_3632.trc:
ORA-01501: CREATE DATABASE failed
ORA-01526: error in opening file '?/rdbms/admin/sql.bsq'
ORA-07391: sftopn: fopen error, unable to open text file.
Error 1526 happened during db open, shutting down database
USER: terminating instance due to error 1526
```

这就是 sql.bsq 文件在数据库创建过程中的作用。知道了这个内容之后，我们甚至可以通过手工修改 sql.bsq 文件来更改数据库字典对象参数，从而实现特殊要求数据库的创建或测试自定义库。

2.4.2 bootstrap$的定位

上一节曾经提到，通过修改 sql.bsq 文件，字典表的存储是可以变更的，那么在数据库的引导过程中，又该如何去定位 bootstrap$的位置呢？

这就不得不提到了 SYSTEM 表空间，在系统表空间文件头存在一个重要的数据结构 root dba，我们可以通过转储数据文件头获得这个信息，从生成的 trace 文件中，我们可以获得如下信息（Oracle9i 环境信息摘录）：

```
FILE HEADER:
        Software vsn=153092096=0x9200000, Compatibility Vsn=134217728=0x8000000
        Db ID=1407686520=0x53e79778, Db Name='EYGLE'
        Activation ID=0=0x0
        Control Seq=1299570=0x13d472, File size=27017=0x6989
        File Number=1, Blksiz=8192, File Type=3 DATA
Tablespace #0 - SYSTEM  rel_fn:1
Creation   at   scn: 0x0000.00000007 04/24/2006 11:34:39
Backup taken at scn: 0x0004.6c2d657e 02/12/2007 15:54:52 thread:1
 reset logs count:0x24dc1f7d scn: 0x0004.6c432ec0 recovered at 04/07/2007 21:04:11
 status:0x4 root dba:0x004001a1 chkpt cnt: 6939 ctl cnt:6938
```

root dba 仅在 SYSTEM 表空间的文件头存在，用于定位数据库引导的 bootstrap$信息。Root dba 存储的是用 16 进制表示的二进制数，其中包含十位的文件号以及 22 位的数据块号，我们可以将 **0x004001a1** 转换为二进制就是 0000 0000 0100 0000 0000 0001 1010 0001，前 10 位为 1，代表文件号为 1，后 22 位转换为 10 进制为 417，代表数据文件 1 上的 417 号数据块。

当然在数据库中无须如此复杂，Oracle 提供工具用于数据块及文件号的转换：

```
SQL> variable file# number
SQL> execute :file#:=dbms_utility.data_block_address_file(to_number('4001a1','xxxxxxx'));
```

```
PL/SQL procedure successfully completed.
SQL> variable block# number
SQL> execute :block#:=dbms_utility.data_block_address_block(to_number('4001a1','xxxxxxx'))
PL/SQL procedure successfully completed.
SQL>print file#
     FILE#
----------
         1
SQL> print block#
    BLOCK#
----------
       417
```

那么这个 Block 上存放的是什么对象呢？

2.4.3 Oracle 9i 中独一无二的 Cache 对象

可以查询一下数据库中 file 1 block 417 上存放的对象：

```
SQL> select segment_name,segment_type,header_file,header_block
  2  from dba_segments where segment_type='CACHE';
SEGMENT_NA SEGMENT_TYPE       HEADER_FILE HEADER_BLOCK
---------- ------------------ ----------- ------------
1.417      CACHE                        1          417
```

在 Oracle9i 中这里存放的是 Oracle 数据库中独一无二的 Cache 对象（从 Oracle 10g 开始，这个 Cache 对象被取消，Oracle 将 root dba 直接指向了 bootstrap$ 表）。这个对象的名称来自于文件号和数据块号，1.417 正好就是文件 1 的第 417 个数据块。这个 Cache 对象在 Oracle 数据库中的含义非同一般，在数据库启动的 bootstrap 过程中，这个对象之前的所有对象都需要用来 bootstrap：

```
SQL> select b.object_id,a.segment_name,a.segment_type,a.header_block
2 from dba_segments a,dba_objects b
  3  where a.segment_name=b.object_name(+) and a.header_file=1 and a.header_block <= 417
  4  order by a.header_block;
 OBJECT_ID SEGMENT_NAME                   SEGMENT_TYPE       HEADER_BLOCK
---------- ------------------------------ ------------------ ------------
           SYSTEM                         ROLLBACK                      9
         2 C_OBJ#                         CLUSTER                      25
         3 I_OBJ#                         INDEX                        49
         6 C_TS#                          CLUSTER                      57
         7 I_TS#                          INDEX                        65
        15 UNDO$                          TABLE                       105
        17 FILE$                          TABLE                       113
```

```
        18 OBJ$                         TABLE                       121
。。。。。。。
        55 I_CCOL2                      INDEX                       369
        56 BOOTSTRAP$                   TABLE                       377
           1.417                        CACHE                       417
45 rows selected.
```

这些对象在 CREATE DATABASE 过程中通过 sql.bsq 文件创建，其对象号同样代表了这些对象的创建顺序。我们再来看看 1.417 对象中存储的信息，转储数据块可以使用如下命令：

alter system dump datafile 1 block 417

检查生成的跟踪文件，可以获得主要信息如下：

```
Start dump data blocks tsn: 0 file#: 1 minblk 417 maxblk 417
buffer tsn: 0 rdba: 0x004001a1 (1/417)
scn: 0x0004.6c4f907a seq: 0x01 flg: 0x04 tail: 0x907a0d01
frmt: 0x02 chkval: 0xc5e1 type: 0x0d=Compatibility segment
Header:  size 12   next rdba 0x0   entries 25   offset 536f
Compatibility entry for 'COMPATSG':
   Size: 24  Release 0x134217728  By 0x153092096
Dump of memory from 0x0AA84E34 to 0x0AA84E38
AA84E30          00000000                       [....]
Compatibility entry for 'BOOTSTRP':
   Size: 24  Release 0x134217728  By 0x153092096
Dump of memory from 0x0AA84E4C to 0x0AA84E50
AA84E40                          00400179              [y.@.]
```

这个信息记录了 BOOTSTRP 的信息，00400179 正好指向的是 file 1 block 377：

```
SQL> variable block# number
SQL> execute :block#:=dbms_utility.data_block_address_block(to_number('400179','xxxxxxx'))
PL/SQL procedure successfully completed.
SQL> print block#
    BLOCK#
----------
       377
```

这一数据块上存储的正是数据库的 BOOTSTRP$引导表：

```
SQL> select segment_name from dba_extents
  2  where block_id between 377 and blocks + 377 -1;
SEGMENT_NAME
--------------------------------------------------------------------
BOOTSTRAP$
```

而 1.417 对象的前一个对象正是 BOOTSTRAP$，这个对象中存放的就是创建这些对象的语句，这些语句在数据库启动时，需要被获取以在内存中创建这些对象，再从硬盘上加载启动

数据库必须的元数据从而启动 Oracle 数据库。

1.417 给我们的另外一个启示是，Oracle 对于对象的命名可以采用 file#.first_block#的方式进行。对于很多中间操作产生的临时对象，Oracle 经常采用类似的命名规则，有朋友在数据库空间异常使用后进行检查，发现正是由于某个异常的数据加载（sql *loader）导致了空间的高额耗用：

```
select segment_name,sum(bytes)/1048576 from dba_segments
where tablespace_name='TEST' group by segment_name order by 2;
299.20715                                    202，681
```

而这个临时对象的命名正好符合以上的介绍，对于这些临时对象，如果出现异常，数据库会自动清除来释放空间。

2.4.4 Oracle 数据库的引导

现在可以全面的来回顾一下数据库的内部引导过程（大家应当已经熟悉了第一章中介绍的 nomount、mount、open 的过程），通过 10046 事件可以跟踪一下数据库的启动过程，使用前面曾经提到过的步骤：

```
SQL> startup nomount;
SQL> alter session set events='10046 trace name context forever,level 12';
SQL> alter database mount;
SQL> alter database open;
```

从跟踪文件（以下跟踪文件来自 Oracle9iR2）中我们可以获得如下重要信息：

```
PARSING IN CURSOR #1 len=19 dep=0 uid=0 oct=35 lid=0 tim=1149404325198521 hv=2631704207
ad='533986f8'
alter database open
END OF STMT
PARSE #1:c=0,e=926,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=1149404325198499
BINDS #1:
……
WAIT #1: nam='direct path read' ela= 43 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= 3 p1=3 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 2 p1=4 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 2 p1=201 p2=1 p3=1
……
WAIT #1: nam='db file sequential read' ela= 88 p1=1 p2=417 p3=1
WAIT #1: nam='db file sequential read' ela= 91 p1=1 p2=377 p3=1
```

从以上信息中可以注意到，Oracle 首先通过 direct path read 方式从每个数据文件头读取了第一个 Block 的信息，然后通过 db file sequential read 的单块读方式分别读取了数据文件 1 的第 417 个 block 和第 377 个 block。

417 上存放的正是 1.417 号对象，通过 1.417 对象进而找到 bootstrap$对象，也就是 block

377，找到了 block 377，Oracle 进而读取其内容，在内存中创建了这个对象：

```
PARSING IN CURSOR #2 len=188 dep=1 uid=0 oct=1 lid=0 tim=1149404325230977 hv=0 ad='b700de24'
create table bootstrap$ ( line#          number not null,   obj#            number not null,
sql_text    varchar2(4000) not null)    storage (initial 50K objno 56 extents (file 1 block 377))
END OF STMT
```

再接下来，Oracle 通过递归查询，从 bootstrap$中获取其它对象的创建语句：

```
PARSING IN CURSOR #2 len=55 dep=1 uid=0 oct=3 lid=0 tim=1149404325233023 hv=3952717224
ad='533984d8'
select line#, sql_text from bootstrap$ where obj# != :1
END OF STMT
```

进而创建这些对象：

```
PARSING IN CURSOR #2 len=129 dep=1 uid=0 oct=36 lid=0 tim=1149404325255090 hv=0 ad='b700de24'
CREATE ROLLBACK SEGMENT SYSTEM STORAGE (   INITIAL 112K NEXT 1024K MINEXTENTS 1 MAXEXTENTS 32765
OBJNO 0 EXTENTS (FILE 1 BLOCK 9))
END OF STMT
```

在读取了 1.417 号对象之前的所有对象后，Oracle 将可以正常打开数据库。

注意，自 Oracle10g 开始，Oracle 将 root dba 直接指向了 bootstrap$对象，从而消除了 Oracle 数据库中这个唯一的 Cache 对象,以下是 10g 的 SYSTEM 文件头信息:

```
V10 STYLE FILE HEADER:
        Compatibility Vsn = 169869568=0xa200100
        Db ID=3965153484=0xec5770cc, Db Name='DANALY'
        Activation ID=0=0x0
        Control Seq=2912565=0x2c7135, File size=84180=0x148d4
        File Number=1, Blksiz=8192, File Type=3 DATA
Tablespace #0 - SYSTEM  rel_fn:1
status:0x2004 root dba:0x00400179 chkpt cnt: 45507 ctl cnt:45506
begin-hot-backup file size: 0
```

了解了 SYSTEM 表空间的重要作用，也就可以理解，为什么系统表空间的文件头损坏，或者如果启动对象的数据块损坏后，Oracle 数据库就将无法启动。

我们曾经见过很多案例，很多用户的数据库运行在非归档模式下，又没有备份，最后当 SYSTEM 表空间出现故障后，数据库就无法打开了，这是最为严重的情况，通常的手段是没有办法恢复数据的。

所以我们经常反复建议，SYSTEM 表空间极其重要，备份重于一切，希望通过我们的不断呼吁，数据库的安全能够更加引起重视，用户的数据能够更加安全。

2.4.5 系统对象与 bootstrap$

ORA-00701 错误可能是很多朋友都遇到过的：

```
ORA-00701: object necessary for warmstarting database cannot be altered
```

这类错误发生在我们试图移动或变更系统对象的时候：

```
SQL> alter table obj$ move;
alter table obj$ move
            *
ERROR at line 1:
ORA-00701: object necessary for warmstarting database cannot be altered
```

这个提示是告诉我们，这些对象是数据库启动所必须的，Oracle 不允许移动或修改。Obj$在bootstrap$表中存在：

```
CREATE TABLE OBJ$("OBJ#" NUMBER NOT NULL,"DATAOBJ#" NUMBER,"OWNER#" NUMBER NOT NULL,"NAME"
VARCHAR2(30) NOT NULL,"NAMESPACE" NUMBER NOT NULL,"SUBNAME" VARCHAR2(30),"TYPE#" NUMBER NOT
NULL,"CTIME" DATE NOT NULL,"MTIME" DATE NOT NULL,"STIME" DATE NOT NULL,"STATUS" NUMBER NOT
NULL,"REMOTEOWNER"     VARCHAR2(30),"LINKNAME"     VARCHAR2(128),"FLAGS"     NUMBER,"OID$"
RAW(16),"SPARE1"  NUMBER,"SPARE2"  NUMBER,"SPARE3"  NUMBER,"SPARE4"  VARCHAR2(1000),"SPARE5"
VARCHAR2(1000),"SPARE6" DATE) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 STORAGE (  INITIAL
16K NEXT 1024K MINEXTENTS 1 MAXEXTENTS 2147483645 PCTINCREASE 0 OBJNO 18 EXTENTS (FILE 1 BLOCK
121))
```

不过仍然有一些例外，以下介绍仅为了使大家增进一点对于数据库启动及内部机制的认识，这些方法并不推荐采用。

一个朋友遇到这样一个案例：

用户…将所有用户表都建立在 System 表空间了，…System 增长到 30G，后来…删除了一个大小 8G 的表的数据，尝试 resize 系统表空间…

从常规来说，如果我们希望 resize 一个文件，这个文件能够 resize 的部分必然是没有数据，未被使用的，也就是说，我们需要找到一个文件最高的 Extent 号，这个 Extent 之外的空间是可以被 resize 的，通过以下一个查询可以找到一个文件最末端的对象:

```
col segment_name for a30
col owner for a10
SELECT *
  FROM (SELECT   owner, segment_name,segment_type,block_id, blocks
           FROM dba_extents
          WHERE tablespace_name = 'SYSTEM' and file_id='&fileid'
       ORDER BY block_id DESC)
WHERE ROWNUM < 11;
```

这个朋友找到的最高对象为：

```
1    SYS  I_H_OBJ#_COL#      1195145  128  INDEX
2    SYS  I_HH_OBJ#_INTCOL#  1195049  8    INDEX
3    SYS  I_HH_OBJ#_COL#     1195041  8    INDEX
4    SYS  PLAN_TABLE         1143417  8    TABLE
5    SYS  C_OBJ#_INTCOL#      921225  128  CLUSTER
6    SYS  I_OBJ1               911753  128  INDEX
```

```
7    SYS  C_OBJ#_INTCOL#       890121 128  CLUSTER
8    SYS  SMON_SCN_TO_TIME     872457 128  CLUSTER
9    SYS  C_OBJ#_INTCOL#       872329 128  CLUSTER
10   SYS  I_OBJ#_INTCOL#       872297 8    INDEX
```

前几个索引对象我们是可以通过 rebuild 使其移动，从而可以释放文件末尾的空间，但是同样的，正常情况下 Oracle 以 ORA-00701 阻止用户的这一行为：

```
SQL> alter index I_H_OBJ#_COL# rebuild;
alter index I_H_OBJ#_COL# rebuild
*
ERROR at line 1:
ORA-00701: object necessary for warmstarting database cannot be altered
```

有两种方式可以使得这些对象允许被改变:

1.通过 migrate 模式

```
SQL> shutdown immediate;
SQL> startup migrate;
SQL> alter index I_H_OBJ#_COL# rebuild;
Index altered.
SQL> shutdown immediate;
SQL> startup
```

2.通过一个内部事件

```
SQL> alter system set event='38003 trace name context forever, level 10' scope=spfile;
System altered.
SQL> shutdown immediate;
SQL> startup
SQL> alter index i_h_obj#_col# rebuild;
Index altered.
```

38003 事件的作用是:CBO Disable column stats for the dictionary objects in recursive SQL，也就是说可以将部分对象从启动的 bootstrap$需要里剥离出来，从而可以被在线 rebuild。这两种方法可以被用来处理一些系统对象的损坏，如坏块等问题，在处理完毕之后应当注销这些内部事件。

我们已知的是，bootstrap$内数据库启动依赖的对象是不能被修改或变更的，另外在启动过程中 Recursive SQL 所依赖的对象，如索引等也是不能修改或变更的，38003 事件的作用就在于使得这些和 column stats 相关的字典对象从启动依赖里脱离出来，从而实现其维护。

这个事件能够影响的对象很有限，在 Oracle9iR2 里，主要有以下对象受影响：hist_head$、histgrm$、i_hh_obj#_col#、i_hh_obj#_intcol#、i_obj#_intcol#、i_h_obj#_col#、c_obj#_intcol#

从 Oracle10g 开始，以下对象被加入进来：fixed_obj$、tab_stats$、ind_stats$、i_fixed_obj$_obj#、i_tab_stats$_obj#、i_ind_stats$_obj#

Oracle 在不同版本中内部事件都会有所变更，以上信息仅供参考，对于 SYSTEM 对象的调整应该十分谨慎并经过严密测试。

2.4.6　数据库引导的分解

前面提到的数据库的引导过程，可以通过 GDB 工具在 Linux、Unix 上进行跟踪，分步骤来观察这个启动过程，以下输出可以帮助读者进一步了解这些内部操作。

首先将数据库启动到 Mount 状态，找到进程 SPID：

```
SQL> startup mount;
ORACLE instance started.
Database mounted.
SQL> select spid from v$process where addr in (select paddr from v$session where sid=(select
distinct sid from v$mystat));
SPID
------------
1518
```

然后通过 gdb 跟踪这个进程：

```
localhost:~ oracle$ gdb $ORACLE_HOME/bin/oracle 1518
GNU gdb 6.3.50-20050815 (Apple version gdb-1518) (Sat Feb 12 02:52:12 UTC 2011)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "x86_64-apple-darwin"...Reading symbols for shared libraries ..
Attaching to program: `/oracle/product/10.2.0/bin/oracle', process 1518.
Reading symbols for shared libraries .++++++++++++ done
0x00007fff80616984 in read ()
(gdb)
```

然后跟踪两个内部指令：

```
(gdb) break kcrf_commit_force
Breakpoint 1 at 0x1025a2d4c
(gdb) break kqlobjlod
Breakpoint 2 at 0x1006c78b4
```

此时执行数据库 OPEN 操作会被挂起：

```
SQL> alter database open;
```

然后重新开启一个 SQL*Plus 进程，查询此时数据库加载的 ROWCACHE 对象：

```
SQL> select parameter,count,gets from v$rowcache where count!=0;

no rows selected
```

然后继续执行,我们看到在第三个步骤之后，数据库加载了一个 ROW Cache 对象：

```
(gdb) c
Continuing.

Breakpoint 1, 0x00000001025a2d4c in kcrf_commit_force ()
(gdb) c
Continuing.

Breakpoint 1, 0x00000001025a2d4c in kcrf_commit_force ()
(gdb) c
Continuing.

Breakpoint 2, 0x00000001006c78b4 in kqlobjlod ()

SQL> select parameter,count,gets from v$rowcache where count!=0;
PARAMETER                COUNT     GETS
-------------------------------- ---------- ----------
dc_objects                    1       1
```

这个对象是什么呢？

```
SQL>  select  address,cache_name,existent,lock_mode,saddr,substr(key,1,40)  keystr  from
v$rowcache_parent;

ADDRESS    CACHE_NAME          E  LOCK_MODE SADDR         KEYSTR
---------------       --------------------       -       ----------       ----------------
----------------------------------------
00000001942E9080 dc_objects        N    3                                    0000000194782EB0
000000000A00424F4F5453545241502400000000
```

解析其 KEY 值，正是 bootstrap$,这就是数据库初始化时加载的第一个对象：

```
SQL> select dump('BOOTSTRAP$',16) from dual;
DUMP('BOOTSTRAP$',16)
-----------------------------------------
Typ=96 Len=10: 42,4f,4f,54,53,54,52,41,50,24
```

然后数据库将递归查询该对象中的数据，向内存中加载其他对象。更进一步：

```
(gdb) c
Continuing.
Breakpoint 2, 0x00000001006c78b4 in kqlobjlod ()
```

```
ADDRESS       CACHE_NAME       E  LOCK_MODE SADDR    KEYSTR
--------------- ------------------- - ---------- --------------- ----------------------------------------
00000001942E30D8 dc_tablespaces       N     0 00           00000000000000000000000000000000000000000
00000001942DE9B8 dc_rollback_segments Y     0 00           00000000000000000000000000000000000000000
00000001942E9080 dc_objects          Y      0 00           000000000A00424F4F5453545241502400000000
00000001942DE2D0 dc_objects          N      3 0000000194782EB0 000000000600435F4F424A230000000000000000
00000001942E3340 dc_object_ids       Y      0 00           38000000000000000000000000000000000000000
```

这里可以看到数据库加载了回滚段信息，首先加载的是 SYSTEM 的回滚段，转储 Row Cache 信息之后，就可以看到这些详细的内容：

```
SQL>  ALTER SESSION SET EVENTS 'immediate trace name row_cache level 10';

Session altered.
```

这里得到的 BUCKET 37 包含了回滚段信息：

```
BUCKET 37:
  row cache parent object: address=0x1942de9b8 cid=3(dc_rollback_segments)
  hash=5fed2a24 typ=9 transaction=0x0 flags=000000a6
  own=0x1942dea88[0x1942dea88,0x1942dea88] wat=0x1942dea98[0x1942dea98,0x1942dea98] mode=N
  status=VALID/INSERT/-/FIXED/-/-/-/-/-
  data=
  00000000 00000000 00000001 00000009 59530006 4d455453 00000000 00000000
  00000000 00000000 00000000 00000000 00000003 00000000 00000000 00000000
  00000000 00000000 00000000 00000000
  BUCKET 37 total object count=1
ROW CACHE HASH TABLE: cid=4 ht=0x192b2e8a8 size=256
```

其中 53595354454d 正是 SYSTEM 回滚段：

```
SQL> select dump('SYSTEM',16) from dual;

DUMP('SYSTEM',16)
--------------------------------
Typ=96 Len=6: 53,59,53,54,45,4d
```

而另外一个 BUCKET 上正是 BOOTSTRAP$对象：

```
BUCKET 43170:
  row cache parent object: address=0x1942e9080 cid=8(dc_objects)
  hash=f3d1a8a1 typ=11 transaction=0x0 flags=000000a6
  own=0x1942e9150[0x1942e9150,0x1942e9150] wat=0x1942e9160[0x1942e9160,0x1942e9160] mode=N
  status=VALID/INSERT/-/FIXED/-/-/-/-/-
```

```
set=0, complete=TRUE
data=
00000000 4f42000a 5453544f 24504152 00000000 00000000 00000000 00000000
00000000 00000001 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000038 00000001 00000038 066f7802 030d1411
11066f78 78030d14 1411066f 0001030d 00000000 00000000 00000000 00000000
00000000 00000000
BUCKET 43170 total object count=1
```

这就是数据库启动过程中，BOOTSTRAP$的加载与引导过程。

2.4.7 BOOTSTRAP$的重要性

由上面的讨论我们可以知道 bootstrap$表的重要，如果 bootstrap$表发生损坏，则数据库将无法启动。曾经遭遇过以下案例，bootstrap$表被恶意修改，如果关闭数据库，之后将无法启动。

以下测试仅为说明问题需要，请勿模仿，请勿在任何环境下尝试此类操作：

```
SQL> col sql_text for a15
SQL> select * from bootstrap$ where rownum <2;
     LINE#       OBJ# SQL_TEXT
---------- ---------- ---------------
        -1         -1 8.0.0.0.0
SQL> update bootstrap$ set sql_text = '9.0.0.0.0' where line#=-1;
1 row updated.
SQL> commit;
Commit complete.
SQL> select * from bootstrap$ where rownum <2;
     LINE#       OBJ# SQL_TEXT
---------- ---------- ---------------
        -1         -1 9.0.0.0.0
```

如果不关闭数据库，该修改是无影响的，bootstrap$也仅在数据库启动时才发挥其重要作用。继续往下看，如果 bootstrap$损坏或者被恶意修改，在数据库启动时会收到如下错误：

```
SQL> startup
ORACLE instance started.
Database mounted.
```

```
ORA-01092: ORACLE instance terminated. Disconnection forced
```

进一步检查 alert 文件可以发现更为详细的提示信息：

```
Sat Apr 29 17:14:22 2006
Errors in file /opt/oracle/admin/conner/udump/conner_ora_31111.trc:
ORA-00704: bootstrap process failure
ORA-00702: bootstrap verison '9.0.0.0.0' inconsistent with version '8.0.0.0.0'
Sat Apr 29 17:14:22 2006
Error 704 happened during db open, shutting down database
USER: terminating instance due to error 704
Instance terminated by USER, pid = 31111
ORA-1092 signalled during: ALTER DATABASE OPEN...
```

日志给出了详细的错误提示，在这种情况下，最好的方式是从备份中进行不完全恢复，如果没有备份，则恢复将会非常复杂和艰难。

2.4.8 BBED 工具的简要介绍

> 声明：以下我将简要介绍通过 Oracle 内部工具恢复以上故障的方法，但是此方法不受 Oracle 支持，也不受作者推荐，介绍这个工具仅仅为了拓展大家对于 Oracle 的认识，提供另外一种恢复的可能性。

Oracle 随软件发布一个内部工具 BBED （某些版本未包含，部分平台需要自行编译），该工具通常位于$ORACLE_HOME/bin 目录下，其名称为 Block Browser/Editor 的缩写。

也就是说，通过 BBED 工具，我们可以直接打开数据文件，修改其中的内容。

以上故障中由于'8.0.0.0.0'被修改为'9.0.0.0.0'而导致数据库无法启动，我们可以通过 BBED 对文件进行操作，修正该数据，从而使得数据库可以重新启动（以下测试来自 Oracle9i 环境，不同版本可能有所不同）。

在使用 BBED 之前需要配置两个文件，一个包含文件列表，一个包含启动参数等：

```
[oracle@jumper conner]$ more par.bbd
blocksize=8192
listfile=file.lst
mode=edit
[oracle@jumper conner]$ more file.lst
1 /opt/oracle/oradata/conner/system01.dbf       440401920
```

通过 BBED 调用 par.bbd 参数文件进行操作：

```
[oracle@jumper conner]$ bbed parfile=par.bbd
Password:
BBED: Release 2.0.0.0.0 - Limited Production on Sat Apr 29 17:17:53 2006
Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.
```

```
************* !!! For Oracle Internal Use only !!! ***************
```

此处 BBED 需要提供一个口令验证，这个口令目前是 *blockedit* 。

通过前文介绍可以知道 bootstrap$位于 file 1 block 377，这里就是需要搜索的开始。通过 find 命令可以查找被修改的字串，通过搜索定位我们知道'9.0.0.0.0'起始于 file 1 blok 378 offset 1276 :

```
BBED> set file 1 block 378 offset 1276
        FILE#           1
        BLOCK#          378
        OFFSET          1276
BBED> dump
 File: /opt/oracle/oradata/conner/system01.dbf (1)
 Block: 378              Offsets: 1276 to 1787           Dba:0x0040017a
------------------------------------------------------------------------
 392e302e 302e302e 302e302e 303e6466 033e6466 0a31362e 302e302e 302e302c
 000302c1 3202c132 c3435245 41544520 554e4951 55452049 4e444558 20495f43
 4f4e3220 4f4e2043 4f4e2428 434f4e23 29205043 54465245 45203130 20494e49
 5452414e 53203220 4d415854 52414e53 20323535 2053544f 52414745 20282020
 ................................
```

将此处的 9 修改为 8 即可：

```
BBED> modify /c "8" offset 1276
Warning: contents of previous BIFILE will be lost. Proceed? (Y/N) Y
 File: /opt/oracle/oradata/conner/system01.dbf (1)
 Block: 378              Offsets: 1276 to 1787           Dba:0x0040017a
------------------------------------------------------------------------
 382e302e 302e302e 302e302e 303e6466 033e6466 0a31362e 302e302e 302e302c
 000302c1 3202c132 c3435245 41544520 554e4951 55452049 4e444558 20495f43
 4f4e3220 4f4e2043 4f4e2428 434f4e23 29205043 54465245 45203130 20494e49
 5452414e 53203220 4d415854 52414e53 20323535 2053544f 52414745 20282020
 ................................
```

修改过后，Oracle 会认为该块损坏：

```
BBED> verify
DBVERIFY - Verification starting
FILE = /opt/oracle/oradata/conner/system01.dbf
BLOCK = 378

Block 378 is corrupt
***
Corrupt block relative dba: 0x0040017a (file 0, block 378)
Bad check value found during verification
```

```
Data in bad block -
 type: 6 format: 2 rdba: 0x0040017a
 last change scn: 0x0819.004c0295 seq: 0x1 flg: 0x06
 consistency value in tail: 0x02950601
 check value in block header: 0x9f21, computed block checksum: 0x1
 spare1: 0x0, spare2: 0x0, spare3: 0x0
***
```

重新计算和应用校验位后，数据块可以恢复一致：

```
BBED> sum apply
Check value for File 1, Block 378:
current = 0x9f20, required = 0x9f20
BBED> verify
DBVERIFY - Verification starting
FILE = /opt/oracle/oradata/conner/system01.dbf
BLOCK = 378
```

此后数据库可以正常启动。

> 注意：此处能够正常恢复，是因为从 8 至 9 的修改并为改变数据位数，否则 BBED 也很难奏效。本文些微介绍仅为扩展大家的视野，使用该工具需要极为谨慎，并且自行负责。

在前面提到，在某些平台中，Oracle 没有提供 BBED，但是源程序已经包含在发布版本中（在 Oracle11g 中，缺省的已经无法编译这个工具），我们可以自行链接编译出这个工具，以下是 Linux 上的一个 Link 示范：

```
[oracle@app01 ~]$ cd $ORACLE_HOME/rdbms/lib
[oracle@app01 lib]$  make -f ins_rdbms.mk $ORACLE_HOME/rdbms/lib/bbed

Linking BBED utility (bbed)
rm -f /opt/oracle/product/10.2.0/rdbms/lib/bbed
gcc -o /opt/oracle/product/10.2.0/rdbms/lib/bbed ……
[oracle@app01 lib]$ ll bbed
-rwxr-xr-x  1 oracle dba 548129 Sep 16 23:17 bbed
```

Oracle Database 11g 中缺省的未提供 BBED 库文件，但是可以用 10g 的文件编译出来，参考如下步骤：

1.复制 Oracle 10g 文件

copy $ORA10g_HOME/rdbms/lib/ssbbded.o to $ORA11g_HOME/rdbms/lib
copy $ORA10g_HOME/rdbms/lib/sbbdpt.o to $ORA11g_HOME/rdbms/lib

copy $ORA10g_HOME/rdbms/mesg/bbedus.msb to $ORA11g_HOME/rdbms/mesg
copy $ORA10g_HOME/rdbms/mesg/bbedus.msg to $ORA11g_HOME/rdbms/mesg
copy $ORA10g_HOME/rdbms/mesg/bbedar.msb to $ORA11g_HOME/rdbms/mesg

2.编译

make -f $ORA11g_HOME/rdbms/lib/ins_rdbms.mk BBED=$ORACLE_HOME/bin/bbed $ORACLE_HOME/bin/bbed

虽然 BBED 工具的使用存在很多风险，但是如果利用得当，可以以之解决很多棘手的问题。

2.4.9 坏块的处理与恢复

有时候为了学习测试，我们常常希望能够模拟数据块损坏，然后来研究相应问题的处理方法，最常见的是使用第三方软件进行数据块修改，模拟数据块损坏。以下是一种常见的模拟及坏块问题的解决方法及步骤。

首先构造测试数据，创建相应的表空间等：

```
SQL> create tablespace block datafile 'e:\oracle\oradata\eygle\block.dbf' size 1M;
表空间已创建。
```

变更测试用户使用该表空间：

```
SQL> alter user eygle default tablespace block;
用户已更改。
SQL> alter user eygle quota unlimited on block;
用户已更改。
```

创建测试表及构造测试数据：

```
SQL> connect eygle/eygle
已连接。
SQL> create table t as select * from dba_tables;
表已创建。
SQL> insert into t select * from t;
已创建 4096 行。
SQL> /
insert into t select * from t
*
ERROR 位于第 1 行:
ORA-01653: 表 EYGLE.T 无法通过 8（在表空间 BLOCK 中）扩展
SQL> commit;
提交完成。
```

关闭数据库后可以用 Ultredit 等工具编辑数据文件，选择特定的数据部分，更改或者删除一些字符，然后启动数据库（有时候你可能需要多次更改才能成功损坏数据块），此时查询测试表，发现数据块已经损坏：

```
SQL> select count(*) from eygle.t;
select count(*) from eygle.t
```

```
*
ERROR 位于第 1 行:
ORA-01578: ORACLE 数据块损坏（文件号 4，块号 35）
ORA-01110: 数据文件 4: 'E:\ORACLE\ORADATA\EYGLE\BLOCK.DBF'
```

使用 DBV 检查数据文件，可以检测到块损坏：

```
E:\Oracle\oradata\eygle>dbv file=block.dbf blocksize=8192

DBVERIFY - 验证正在开始 : FILE = block.dbf
标记为损坏的页 35
***
Corrupt block relative dba: 0x01000023 (file 4, block 35)
Bad check value found during dbv:
Data in bad block -
type: 6 format: 2 rdba: 0x01000023
last change scn: 0x0000.00049097 seq: 0x1 flg: 0x06
consistency value in tail: 0x90970601
check value in block header: 0xd6cb, computed block checksum: 0x2c0a
spare1: 0x0, spare2: 0x0, spare3: 0x0
***
```

在这种情况下，如果有备份，需要从备份中恢复；如果没有备份，那么坏块部分的数据肯定要丢失了，在这个时候导出是不允许的：

```
E:\>exp eygle/eygle file=t.dmp tables=t
即将导出指定的表通过常规路径 ...
. . 正在导出表 T
EXP-00056: 遇到 ORACLE 错误 1578
ORA-01578: ORACLE 数据块损坏（文件号 4，块号 35）
ORA-01110: 数据文件 4: 'E:\ORACLE\ORADATA\EYGLE\BLOCK.DBF'
导出成功终止，但出现警告。
```

当然，对于不同的情况需要区别对待,首先你需要检查损坏的对象，使用以下 SQL：

```
SQL> SELECT tablespace_name, segment_type, owner, segment_name FROM dba_extents
  2  WHERE file_id = 4 and 35 between block_id AND block_id + blocks - 1;
TABLESPACE_NAME  SEGMENT_TYPE     OWNER      SEGMENT_NAME
---------------- ---------------- ---------- ---------------
BLOCK            TABLE            EYGLE      T
```

如果损失的是数据（如果是索引块损坏，通常可以通过索引重建来解决），我们可以设置内部事件，使 exp 跳过这些损坏的 block：

```
SQL> ALTER SYSTEM SET EVENTS='10231 trace name context forever,level 10';
系统已更改。
```

然后可以导出未损坏的数据

```
E:\>exp eygle/eygle file=t.dmp tables=t
即将导出指定的表通过常规路径 ...
. . 正在导出表 T 8036 行被导出
在没有警告的情况下成功终止导出。
```

这时候数据成功导出，此后可以通过 drop table，重建数据表导入数据。本例中我们损失了 8192 - 8036 = 156 行数据。

除了使用这种方式之外，还可以使用 Oracle 提供的 dbms_repair 包进行坏块标记，从而可以使查询跳过这些坏块，减少坏块对于表扫描访问的影响。

2.4.10 使用 BBED 模拟和修复坏块

其实在模拟坏块的过程中，更好的工具是 BBED，利用 BBED 可以查看或更改数据块的内容，甚至是修复数据块的损坏。以下是一个简单的示范，首先创建测试表：

```
SQL> create table bbed tablespace users as select * from dba_tables;
Table created.
SQL> select count(*) from bbed;
  COUNT(*)
----------
       523
SQL> select segment_name,file_id,block_id from dba_extents where segment_name='BBED';
SEGMENT_NA    FILE_ID   BLOCK_ID
---------- ---------- ----------
BBED                3          9
BBED                3         17
BBED                3         25
```

类似前面介绍创建 BBED 参数文件等

```
[oracle@jumper conner]$ more filelist.txt
         1 /opt/oracle/oradata/conner/system01.dbf   440401920
         2 /opt/oracle/oradata/conner/undotbs01.dbf  104857600
         3 /opt/oracle/oradata/conner/users01.dbf     27262976
[oracle@jumper conner]$ more par.bbd
blocksize=8192
listfile=filelist.txt
mode=edit
```

接下来就可以使用 BBED 来打开数据文件，使用 modify 命令进行相应的数据块内容更改：

```
[oracle@jumper conner]$ bbed parfile=par.bbd
Password:
BBED: Release 2.0.0.0.0 - Limited Production on Sun Sep 11 20:01:01 2005
Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.
```

```
************* !!! For Oracle Internal Use only !!! ***************
BBED> set file 3
        FILE#          3
BBED> modify 1000 file 3 block 17
 File: /opt/oracle/oradata/conner/users01.dbf (3)
 Block: 17              Offsets: 1000 to 1511           Dba:0x00c00011
------------------------------------------------------------------------
 03e80000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 00000000 00000000 002c002f 03535953 18415050 4c59245f 434f4e46 5f48444c
 525f434f 4c554d4e 53065359 5354454d ffff02c1 0b02c129 02c10203 c2033804
 c3073825 ff02c102 06c51630 31252eff 02c10202 c1020359 4553014e ffffffff
 ffffffff 0a202020 20202020 2020310a 20202020 20202020 20310520 2020204e
 07454e41 424c4544 ffff024e 4fff014e 014e024e 4f074445 4641554c 54084449

 <32 bytes per line>
```

使用 verify 命令，可以发现刚才修改的 file 3 block 17 已经被标记为损坏(有时候这样的修改不一定能够损坏一个数据块,**通过 CORRUPT 命令可以直接将一个数据块标记为损坏**)。

```
BBED> verify
DBVERIFY - Verification starting
FILE = /opt/oracle/oradata/conner/users01.dbf
BLOCK = 17

Block 17 is corrupt
***
Corrupt block relative dba: 0x00c00011 (file 0, block 17)
Bad check value found during verification
Data in bad block -
 type: 6 format: 2 rdba: 0x00c00011
 last change scn: 0x0000.20a3b575 seq: 0x1 flg: 0x04
 consistency value in tail: 0xb5750601
 check value in block header: 0x3006, computed block checksum: 0xe803
 spare1: 0x0, spare2: 0x0, spare3: 0x0
***
```

重新启动数据库以后，执行全表扫描，此时错误出现：

```
SQL> select count(*) from bbed;
select count(*) from bbed
                     *
```

```
ERROR at line 1:
ORA-01578: ORACLE data block corrupted (file # 3, block # 17)
ORA-01110: data file 3: '/opt/oracle/oradata/conner/users01.dbf'
```

这就模拟了数据块的损坏，**以上方法仅供测试使用，切记不可用于重要环境**。BBED 不仅可以破坏数据块，当然还可以修复数据块，上面的测试，由于事先备份了相关的数据文件，下面就可以通过 BBED 的 COPY 命令来恢复损坏的数据块。首先需要将备份文件加入 BBED 文集列表，然后打开该文件，通过 copy 命令在两个文件之间进行数据块的复制恢复：

```
BBED> set file 4
        FILE#          4
BBED> copy file 4 block 17 to file 3 block 17;
Warning: contents of previous BIFILE will be lost. Proceed? (Y/N) Y
 File: /opt/oracle/oradata/conner/users01.dbf (3)
 Block: 17              Offsets:    0 to  511           Dba:0x00c00011
------------------------------------------------------------------------
 06020000 1100c000 75b5a320 00000104 06300000 01000000 611e0000 72b5a320
 00000000 03003201 0900c000 ffff0000 00000000 00000000 00000000 00800000
 72b5a320 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 <32 bytes per line>
BBED> verify
DBVERIFY - Verification starting
FILE = /opt/oracle/oradata/conner/users01.dbf
BLOCK = 17


DBVERIFY - Verification complete
```

善用 BBED 可以从很多棘手的问题中拯救我们的数据库，但是由于其使用较为不便，所以不推荐使用。

2.4.11 使用 RMAN 进行坏块修复

出现坏块，正确的途径是通过备份来恢复，使用 Rman 可以很容易的通过备份对坏块进行恢复，以下是一个简单的测试范例，首先备份数据库：

```
RMAN> backup database format='d:\oradata\eygle_fullbk.bak' tag='eygle';
```

如果数据文件上的数据块出现损坏，访问时会给出错误提示：

```
SQL> select count(*)from t;
select count(*)from t
                  *
ERROR 位于第 1 行:
ORA-01578: ORACLE 数据块损坏（文件号 2，块号 14）
ORA-01110: 数据文件 2: 'D:\ORADATA\EYGLE\EYGLE01.DBF'
```

坏块的信息会记录在告警日志文件中，也可以通过 DBV 进行检测；但是注意 RMAN 也

具备检测坏块的功能，在执行以下命令后 RMAN 会在字典视图中记录坏块信息：

```
RMAN> backup validate datafile 2;
```

查询 RMAN 发现的坏块信息

```
SQL> select * from v$database_block_corruption where file#=2;
     FILE#     BLOCK#     BLOCKS CORRUPTION_CHANGE# CORRUPTIO
---------- ---------- ---------- ------------------ ---------
         2         14          1                  0 FRACTURED
```

使用 RMAN 进行坏块修复可以在 Mount 状态进行：

```
RMAN> startup mount;
RMAN> blockrecover datafile 2 block 14 from backupset;
启动 blockrecover 于 12-6 月 -05
正在使用目标数据库控制文件替代恢复目录

通道 ORA_DISK_1: 正在恢复块
通道 ORA_DISK_1: 正在指定要从备份集恢复的块
正在恢复数据文件 00002 的块
通道 ORA_DISK_1: 已从备份段 1 恢复块
段 handle=D:\ORADATA\EYGLE_FULLBK.BAK tag=EYGLE params=NULL
通道 ORA_DISK_1: 块恢复已完成

正在开始介质的恢复
完成介质的恢复

完成 blockrecover 于 12-6 月 -05
```

2.5控制文件与启动校验

在数据库启动过程中，除了要进行本章之前描述的种种校验之外，在某些情况下 Oracle 还会进行一系列其他重要的校验工作，比如将数据字典和控制文件进行交叉检查。

在数据字典中存在数据文件的相关记录信息，而控制文件中也存在数据文件的信息，这两部分信息如果不一致该如何是好呢？

答案是：以数据字典为准，系统表空间是数据库的心脏，一切听从“内心”的指示。

以下是已知的几种校验：

1.检验文件大小和创建时间，将文件头记录的文件创建 SCN 与数据字典（file$）比较，如果数据文件信息与数据字典不符，则出现 ORA-01203 错误

2.检查文件数量，如果数据字典中存在的文件，在控制文件中不存在，就将文件添加入数据文件；如果控制文件中存在的数据文件在数据字典中不存在，就从控制文件

中删除文件。

这种情况通常出现在控制文件重建的时候。

以下是 ORA-01203 错误提示的内容，该提示显示文件的创建 SCN 有误，与数据字典不一致：

```
SQL> alter database open;
alter database open
*
ERROR at line 1:
ORA-01122: database file 4 failed verification check
ORA-01110: data file 4: '/oracle/oradata/eygle/users01.dbf'
ORA-01203: wrong incarnation of this file - wrong creation SCN
```

可以从数据字典 file$中获得 SCN 信息：

```
SQL> select file#,crscnwrp,crscnbas from file$;
     FILE#   CRSCNWRP    CRSCNBAS
---------- ---------- ----------
     1        0           8
     2        0        6448
     3        0        6606
     4        0       10623
     5        0      174768
```

注意 4 号文件的创建 SCN 是 10623,转换为 16 进制：

```
SQL> select to_char('10623','xxxxxx') from dual;

TO_CHAR
-------
   297f
```

然后通过 BBED 去修改这个创建 SCN 信息，在文件头上找到这个位置：

```
BBED> set offset 100
    OFFSET            100

BBED> dump
 File: /oracle/oradata/eygle/users01.dbf (4)
 Block: 1                Offsets:  100 to  131           Dba:0x01000001
------------------------------------------------------------------------
 b0aa0200 00000000 a111ed2c 3a12ed2c 53ab0200 0000e000 00000000 00000000

 <32 bytes per line>
```

```
BBED> modify /x 7f290000
 File: /oracle/oradata/eygle/users01.dbf (4)
 Block: 1                Offsets:  100 to  131           Dba:0x01000001
------------------------------------------------------------------------
 7f290000 00000000 a111ed2c 3a12ed2c 53ab0200 0000e000 00000000 00000000

 <32 bytes per line>

BBED> p kcvfhcrs
struct kcvfhcrs, 8 bytes                    @100
   ub4 kscnbas                              @100      0x0000297f
   ub2 kscnwrp                              @104      0x0000

BBED> sum apply
Check value for File 4, Block 1:
current = 0x9f43, required = 0x9f43
```

然后再次尝试恢复数据文件 4，会收到如下错误：

```
SQL> recover datafile 4;
ORA-00283: recovery session canceled due to errors
ORA-01110: data file 4: '/oracle/oradata/eygle/users01.dbf'
ORA-01122: database file 4 failed verification check
ORA-01110: data file 4: '/oracle/oradata/eygle/users01.dbf'
ORA-01202: wrong incarnation of this file - wrong creation time
```

这里提示的是文件创建时间错误，这是与控制文件的交互校验，如果不通过，说明控制文件和数据文件头上记录的信息不符，可以重建控制文件，重新从数据文件上获得正确的时间信息。这个步骤较为简单：

```
SQL> @cr
CREATE CONTROLFILE REUSE DATABASE "EYGLE" NORESETLOGS  NOARCHIVELOG
*
ERROR at line 1:
ORA-01503: CREATE CONTROLFILE failed
ORA-01200: actual file size of 640 is smaller than correct size of 1280 blocks
ORA-01110: data file 4: '/oracle/oradata/eygle/users01.dbf'
```

这里又出现一个错误，数据文件的实际大小和正确大小不一致：

```
SQL> select to_char('640','xxx') from dual;
TO_C
```

```
----
 280
```

这里遇到的错误指文件的实际大小和文件头记录的信息不一致，这说明数据文件头信息有误，还需要修改数据块的数量：

```
BBED> set offset 44
    OFFSET            44

BBED> p kcvfhhdr
struct kcvfhhdr, 76 bytes                   @20
   ub4 kccfhswv                             @20       0x00000000
   ub4 kccfhcvn                             @24       0x0a200300
   ub4 kccfhdbi                             @28       0x5dbc6478
   text kccfhdbn[0]                         @32       E
   text kccfhdbn[1]                         @33       Y
   text kccfhdbn[2]                         @34       G
   text kccfhdbn[3]                         @35       L
   text kccfhdbn[4]                         @36       E
   text kccfhdbn[5]                         @37
   text kccfhdbn[6]                         @38
   text kccfhdbn[7]                         @39
   ub4 kccfhcsq                             @40       0x0000008b
   ub4 kccfhfsz                             @44       0x00000500
   s_blkz kccfhbsz                          @48       0x00
   ub2 kccfhfno                             @52       0x0004
   ub2 kccfhtyp                             @54       0x0003
   ub4 kccfhacid                            @56       0x00000000
   ub4 kccfhcks                             @60       0x00000000


BBED> set offset 44
    OFFSET            44

BBED> dump
 File: /oracle/oradata/eygle/users01.dbf (4)
 Block: 1                Offsets:   44 to   75           Dba:0x01000001
------------------------------------------------------------------------
 00050000 00200000 04000300 00000000 00000000 00000000 00000000 00000000
```

```
 <32 bytes per line>

BBED> modify /x 8002
 File: /oracle/oradata/eygle/users01.dbf (4)
 Block: 1                Offsets:   44 to   75           Dba:0x01000001
------------------------------------------------------------------------
 80020000 00200000 04000300 00000000 00000000 00000000 00000000 00000000

 <32 bytes per line>

BBED> p kcvfhhdr
struct kcvfhhdr, 76 bytes                   @20
   ub4 kccfhswv                             @20       0x00000000
   ub4 kccfhcvn                             @24       0x0a200300
   ub4 kccfhdbi                             @28       0x5dbc6478
   text kccfhdbn[0]                         @32       E
   text kccfhdbn[1]                         @33       Y
   text kccfhdbn[2]                         @34       G
   text kccfhdbn[3]                         @35       L
   text kccfhdbn[4]                         @36       E
   text kccfhdbn[5]                         @37
   text kccfhdbn[6]                         @38
   text kccfhdbn[7]                         @39
   ub4 kccfhcsq                             @40       0x0000008b
   ub4 kccfhfsz                             @44       0x00000280
   s_blkz kccfhbsz                          @48       0x00
   ub2 kccfhfno                             @52       0x0004
   ub2 kccfhtyp                             @54       0x0003
   ub4 kccfhacid                            @56       0x00000000
   ub4 kccfhcks                             @60       0x00000000
```

应用修改：

```
BBED> verify
DBVERIFY - Verification starting
FILE = /oracle/oradata/eygle/users01.dbf
BLOCK = 1

Block 1 is corrupt
Corrupt block relative dba: 0x01000001 (file 0, block 1)
```

```
Bad check value found during verification
Data in bad block:
 type: 11 format: 2 rdba: 0x01000001
 last change scn: 0x0000.00000000 seq: 0x1 flg: 0x04
 spare1: 0x0 spare2: 0x0 spare3: 0x0
 consistency value in tail: 0x00000b01
 check value in block header: 0x9f43
 computed block checksum: 0x780

BBED> sum apply
Check value for File 4, Block 1:
current = 0x98c3, required = 0x98c3
```

然后可以重建控制文件，恢复一下，Online 加载数据文件，将原本 Offline 丢失的数据文件，加载回数据库中：

```
SQL> startup nomount;
ORACLE instance started.

Total System Global Area  612368384 bytes
Fixed Size            2085872 bytes
Variable Size          167775248 bytes
Database Buffers      436207616 bytes
Redo Buffers            6299648 bytes
SQL> @cr
SQL> CREATE CONTROLFILE REUSE DATABASE "EYGLE" NORESETLOGS  NOARCHIVELOG
  2       MAXLOGFILES 16
  3       MAXLOGMEMBERS 3
  4       MAXDATAFILES 100
  5       MAXINSTANCES 8
  6       MAXLOGHISTORY 292
  7  LOGFILE
  8    GROUP 1 '/oracle/oradata/eygle/redo01.log'  SIZE 50M,
  9    GROUP 2 '/oracle/oradata/eygle/redo02.log'  SIZE 50M,
 10    GROUP 3 '/oracle/oradata/eygle/redo03.log'  SIZE 50M
 11  -- STANDBY LOGFILE
 12  DATAFILE
 13    '/oracle/oradata/eygle/system01.dbf',
 14    '/oracle/oradata/eygle/undotbs01.dbf',
 15    '/oracle/oradata/eygle/sysaux01.dbf',
 16    '/oracle/oradata/eygle/users01.dbf',
```

```
 17    '/oracle/oradata/eygle/user02.dbf'
 18  CHARACTER SET ZHS16GBK
 19  ;

Control file created.

SQL> alter database open;
alter database open
*
ERROR at line 1:
ORA-01113: file 4 needs media recovery
ORA-01110: data file 4: '/oracle/oradata/eygle/users01.dbf'

SQL> select file#,name,status from v$datafile;

     FILE# NAME                      STATUS
---------- ---------------------------------------- -------
     1 /oracle/oradata/eygle/system01.dbf        SYSTEM
     2 /oracle/oradata/eygle/undotbs01.dbf        ONLINE
     3 /oracle/oradata/eygle/sysaux01.dbf        ONLINE
     4 /oracle/oradata/eygle/users01.dbf        RECOVER
     5 /oracle/oradata/eygle/user02.dbf        ONLINE

SQL> recover datafile 4;
Media recovery complete.
SQL> alter database open;

Database altered.

SQL> select file#,name,status from v$datafile;

     FILE# NAME                      STATUS
---------- ---------------------------------------- -------
     1 /oracle/oradata/eygle/system01.dbf        SYSTEM
     2 /oracle/oradata/eygle/undotbs01.dbf        ONLINE
     3 /oracle/oradata/eygle/sysaux01.dbf        ONLINE
     4 /oracle/oradata/eygle/users01.dbf        ONLINE
     5 /oracle/oradata/eygle/user02.dbf        ONLINE
```

以下通过一个更为复杂的恢复案例和读者继续探讨。

在 Oracle 数据库的启动过程中，因为一些特殊的情况，数据库可能要进行数据字典检查，这个字典检查对于数据库至关重要。

先看以下测试，在测试数据库中创建一个新的表空间：

```
SQL> alter system set db_create_file_dest='D:\ORACLE\ORADATA\ENMO\' scope=both;

系统已更改。

SQL> create tablespace eygle datafile size 20M;

表空间已创建。

SQL> select name from v$datafile;

NAME
--------------------------------------------------------------------------------
D:\ORACLE\ORADATA\ENMO\SYSTEM01.DBF
D:\ORACLE\ORADATA\ENMO\UNDOTBS01.DBF
D:\ORACLE\ORADATA\ENMO\SYSAUX01.DBF
D:\ORACLE\ORADATA\ENMO\USERS01.DBF
D:\ORACLE\ORADATA\ENMO\ENMO\DATAFILE\O1_MF_EYGLE_65YQMOHC_.DBF
```

生成一个控制文件的跟踪，很多时候在丢失控制文件之后，我们可能需要通过类似这样一个脚本去重建控制文件：

```
SQL> alter database backup controlfile to trace;

数据库已更改。
```

然后强制关闭数据库，删除所有的控制文件，模拟一次异常：

```
SQL> alter system switch logfile;
SQL> alter system switch logfile;
SQL> alter system switch logfile;
SQL> shutdown abort;
ORACLE 例程已经关闭。
SQL> exit
从 Oracle Database 10g Enterprise Edition Release 10.2.0.4.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options 断开

C:\Users\eygle>del D:\ORACLE\ORADATA\ENMO\*.ctl
```

现在启动数据库会出现 ORA-00205 错误，找不到控制文件，我们可以去找出跟踪文件，重新创建控制文件，此处为了测试的需要，选择了使用 RESETLOGS 方式打开数据库：

```
SQL> startup nomount;
ORACLE instance started.

Total System Global Area  612368384 bytes
Fixed Size                  1298160 bytes
Variable Size             167772432 bytes
Database Buffers          436207616 bytes
Redo Buffers                7090176 bytes
SQL> CREATE CONTROLFILE REUSE DATABASE "ENMO" RESETLOGS  NOARCHIVELOG
  2      MAXLOGFILES 16
  3      MAXLOGMEMBERS 3
  4      MAXDATAFILES 100
  5      MAXINSTANCES 8
  6      MAXLOGHISTORY 292
  7  LOGFILE
  8    GROUP 1 'D:\ORACLE\ORADATA\ENMO\REDO01.LOG'  SIZE 50M,
  9    GROUP 2 'D:\ORACLE\ORADATA\ENMO\REDO02.LOG'  SIZE 50M,
 10    GROUP 3 'D:\ORACLE\ORADATA\ENMO\REDO03.LOG'  SIZE 50M
 11  -- STANDBY LOGFILE
 12  DATAFILE
 13    'D:\ORACLE\ORADATA\ENMO\SYSTEM01.DBF',
 14    'D:\ORACLE\ORADATA\ENMO\UNDOTBS01.DBF',
 15    'D:\ORACLE\ORADATA\ENMO\SYSAUX01.DBF',
 16    'D:\ORACLE\ORADATA\ENMO\USERS01.DBF'
 17  CHARACTER SET ZHS16GBK
 18  ;

Control file created.

SQL> recover database using backup controlfile until cancel;
ORA-00279: change 899492 generated at 08/09/2010 09:49:16 needed for thread 1
ORA-00289: suggestion : D:\ORACLE\10.2.0\RDBMS\ARC00004_0726572838.001
ORA-00280: change 899492 for thread 1 is in sequence #4


Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
D:\ORACLE\ORADATA\ENMO\REDO01.LOG
```

```
Log applied.
Media recovery complete.
SQL> alter database open resetlogs;

Database altered.
```

数据库打开之后，我们发现数据库里增加了一个新的文件：

D:\ORACLE\10.2.0\DATABASE \MISSING00005

这个文件就是在创建控制文件时遗漏的，Oracle 以缺省命名方式加入到数据库中：

```
SQL> select name from v$datafile;
NAME
--------------------------------------------------------------------------------
D:\ORACLE\ORADATA\ENMO\SYSTEM01.DBF
D:\ORACLE\ORADATA\ENMO\UNDOTBS01.DBF
D:\ORACLE\ORADATA\ENMO\SYSAUX01.DBF
D:\ORACLE\ORADATA\ENMO\USERS01.DBF
D:\ORACLE\10.2.0\DATABASE\MISSING00005
```

那么数据库是如何得知这个文件的存在又是如何将这个文件加入到控制文件中呢？

在告警日志中有明确的信息提示，这里所说的 Dictionary Check 就是在将数据字典的内容和控制文件进行比较，当发现字典中存在，但是控制文件中缺失的文件时，就会主动为控制文件增加一个缺省命名的文件，此时'MISSING00005'就出现了：

```
Mon Aug 09 09:51:09 2010
Successfully onlined Undo Tablespace 1.
Dictionary check beginning
Tablespace 'TEMP' #3 found in data dictionary,
but not in the controlfile. Adding to controlfile.
Tablespace 'EYGLE' #6 found in data dictionary,
but not in the controlfile. Adding to controlfile.
File #5 found in data dictionary but not in controlfile.
Creating OFFLINE file 'MISSING00005' in the controlfile.
This file can no longer be recovered so it must be dropped.
Dictionary check complete
Mon Aug 09 09:51:10 2010
SMON: enabling tx recovery
```

数据库找到的'MISSING00005'在 Resetlogs 过程中被离线，并且通常无法再通过正常途径加入到数据库中：

```
SQL> select * from v$recover_file;
     FILE# ONLINE  ONLINE_ ERROR                    CHANGE# TIME
---------- ------- ------- -------------------- ---------- ---------
```

```
         5 OFFLINE OFFLINE UNKNOWN ERROR           899492 09-AUG-10

SQL> alter tablespace eygle online;
alter tablespace eygle online
*
ERROR at line 1:
ORA-01190: control file or data file 5 is from before the last RESETLOGS
ORA-01110: data file 5:'D:\ORACLE\ORADATA\ENMO\ENMO\DATAFILE\O1_MF_EYGLE_65YQMOHC_.DBF'

SQL> recover tablespace eygle;
ORA-00283: recovery session canceled due to errors
ORA-19909: datafile 5 belongs to an orphan incarnation
ORA-01110: data file 5:'D:\ORACLE\ORADATA\ENMO\ENMO\DATAFILE\O1_MF_EYGLE_65YQMOHC_.DBF'
```

当然，如果我们不通过 Resetlogs 方式打开数据库，或者我们拥有足够的归档日志，该文件仍然可以被恢复出来的，从 Oracle 10g 开始，Oracle 数据库支持跨越 Resetlogs 时间点的恢复。

以下是一个我们经常可能遇到的通过 Rename 方式校正文件名，进行恢复的操作：

```
SQL> alter database rename file '/opt/oracle/10.2.0/dbs/MISSING00006'
  2  to '/opt/oracle/oradata/eygle/eygle.dbf';
Database altered.
SQL> alter tablespace eygle online;
alter tablespace eygle online
*
ERROR at line 1:
ORA-01190: control file or data file 8 is from before the last RESETLOGS
ORA-01110: data file 6: '/opt/oracle/oradata/eygle/eygle.dbf'
SQL> recover tablespace eygle;
ORA-00279: change 1623 generated at 01/18/2010 13:20:30 needed for thread 1
ORA-00289: suggestion : /archive/1_29_234595239.dbf
ORA-00280: change 1623 for thread 1 is in sequence #29
Specify log: {=suggested | filename | AUTO | CANCEL}
AUTO
Log applied.
Media recovery complete.
SQL> alter tablespace eygle online;
Tablespace altered.
```

那么这个字典检查是和数据库中哪些字典表进行的校验呢？最近的一次恢复工作使我们可以清晰的明确这个过程。

2.5.1 遭遇 ORA-00600 25013 / 25015 错误

某客户遇到 ORA-00600 25013 / 25015 错误，请求协助，用户提供的主要错误信息如下：

```
Fri Jun 25 09:18:40 2010
Errors in file /oracle/admin/cwgk/udump/cwgk1_ora_20031.trc:
ORA-00600: internal error code, arguments: [25013], [0], [229], [EMIS116N2],
[EMIS116k], [184], [179], []
Fri Jun 25 09:18:45 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_pmon_4050.trc:
ORA-00600: internal error code, arguments: [25015], [229], [179], [184], [], [],
[], []
Fri Jun 25 09:24:06 2010
ORACLE Instance cwgk1 (pid = 30) - Error 600 encountered while recovering
transaction (203, 10).
Fri Jun 25 09:24:06 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_smon_4083.trc:
ORA-00600: internal error code, arguments: [25015], [229], [179], [184], [], [], [], []
Fri Jun 25 09:24:06 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_pmon_4050.trc:
ORA-00474: SMON process terminated with error
Fri Jun 25 09:24:06 2010
PMON: terminating instance due to error 474
Fri Jun 25 15:33:50 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_smon_27702.trc:
ORA-00600: internal error code, arguments: [kccocx_01], [], [], [], [], [], [], []
ORA-00600: internal error code, arguments: [25015], [229], [179], [184], [], [], [], []
Fri Jun 25 15:33:52 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_smon_27702.trc:
ORA-00600: internal error code, arguments: [kccocx_01], [], [], [], [], [], [], []
ORA-00600: internal error code, arguments: [25015], [229], [179], [184], [], [], [], []
```

最后用户尝试重建控制文件，强制打开数据库等手段，均不能使数据库恢复正常。

客户这样描述异常的体现：

```
FILE_NAME                                                TABLESPACE_NAME
-------------------------------------------------------------------------------
/dev/vg02/rlvdata224_8G                                  EMIS116N2
/dev/vg02/rlvdata225_8G                                  EMIS116L
/dev/vg02/rlvdata226_8G                                  EMIS116M
/dev/vg02/rlvdata227_8G                                  EMIS116K2
```

```
/dev/vg02/rlvdata228_8G                                 EMIS116P
/dev/vg02/rlvdata229_8G                                 EMIS116Q
```

从这里可看出：表空间 EMIS116N2 使用了设备 /dev/vg02/rlvdata224_8G
表空间 EMIS116K2 使用了设备 /dev/vg02/rlvdata227_8G

而这两个设备原本是给"EMIS116n"及"EMIS116k"使用的，因此产生冲突。但从日志情况看，创建的语句为：

```
CREATE  TABLESPACE "EMIS116N2" DATAFILE '/dev/vg02/rlvdata088_16G'
SIZE 7900M AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED LOGGING
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
```

应该是使用 /dev/vg02/rlvdata088_16G 这个设备。目前数据库的数据字典不正确，在 dba_data_files 中有 emis116k2 及 emis116n2 的表空间，但在 dba_tablespaces 中却都不存在。

这就是客户的案情描述。

2.5.2 来龙去脉 – 表空间创建

收到这个案例之后，我们首先分析告警日志文件，理清事件的来龙去脉。根据用户提供的信息，故障是因为表空间维护导致的，最初出现问题的是一个名称叫做"EMIS116K"表空间。

在告警日志中，这个表空间最初于 Apr 27 创建：

```
Tue Apr 27 09:05:27 2010
CREATE  TABLESPACE "EMIS116K" DATAFILE '/dev/vg02/rlvdata224_8G'
SIZE 7900M AUTOEXTEND ON NEXT 23M MAXSIZE UNLIMITED LOGGING
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
Tue Apr 27 09:11:05 2010
Completed: CREATE  TABLESPACE "EMIS116K" DATAFILE '/dev/vg02/rlvdata224_8G'
```

后来在 Jun 24 被删除：

```
Thu Jun 24 16:37:53 2010
DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
Thu Jun 24 16:47:48 2010
Deleted file /dev/vg02/rlvdata224_8G
```

再然后，一个新的表空间在该文件上建立，表空间名称与原来“相同”。注意这里的相同是加了引号的，用户在这里指定的表空间名称是"EMIS116k"，这与原来的"EMIS116K"是不同的。大小写在引号之中，会存在不同的含义：

```
Thu Jun 24 16:49:01 2010
CREATE  TABLESPACE "EMIS116k" DATAFILE '/dev/vg02/rlvdata224_8G'
SIZE 7900M AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED LOGGING
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
Thu Jun 24 16:53:39 2010
```

```
Completed: CREATE  TABLESPACE "EMIS116k" DATAFILE '/dev/vg02/rlvdata224_8G'
SIZE 7900M AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED LOGGING
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
```

随后用户可能发现了一些问题，试图去删除这个表空间，出现了 ORA-959 错误：

```
Thu Jun 24 16:57:19 2010
DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
Thu Jun 24 16:57:19 2010
ORA-959 signalled during: DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
...
```

ORA-959 错误的含义是指定的表空间不存在：

```
ORA-00959 tablespace 'string' does not exist
Cause: A statement specified the name of a tablespace that does not exist.
Action: Enter the name of an existing tablespace. For a list of tablespace names, query the
data dictionary.
```

简单的测试一下：

```
SQL> create tablespace "EYGLe" datafile size 10M;
表空间已创建。

SQL> drop tablespace eygle;
drop tablespace eygle
*
第 1 行出现错误:
ORA-00959: 表空间 'EYGLE' 不存在

SQL> select tablespace_name from dba_tablespaces where tablespace_name like 'E%';
TABLESPACE_NAME
------------------------------
EYGLe

SQL> drop tablespace "EYGLe";
表空间已删除。
```

如果用户当时能够进行类似的操作，一切将变得简单。

可是事情朝着危险的方向迈进了，这里给我们的一个重要启示是：

在进行数据库维护，编写脚本时，要注重规范，统一编码，避免不必要的麻烦和风险，有时候规范比技术能力本身更重要。

由于前面的操作报出的提示是表空间不存在，客户认为该表空间没有创建成功，或是已经被删除，于是接下来继续在同样的裸设备上创建新的表空间，这一次的名称是全部大写的"EMIS116K"：

```
Thu Jun 24 19:04:12 2010
CREATE  TABLESPACE "EMIS116K" DATAFILE '/dev/vg02/rlvdata224_8G'
SIZE 7900M AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED LOGGING
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
```

由于上一次创建的表空间仍然存在，在同一个裸设备上创建表空间必然不会成功，这一次出现的是 ORA-1537 号错误：

```
Thu Jun 24 19:04:12 2010
ORA-1537 signalled during: CREATE  TABLESPACE "EMIS116K" DATAFILE '/dev/vg02/rlvdata224_8G'
SIZE 7900M AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED LOGGING
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
...
```

ORA-01537 的错误是指，指定的文件已经是数据库的一部分了，不能再次被使用：

```
ORA-01537 cannot add data file 'string' - file already part of database
Cause: During CREATE or ALTER TABLESPACE, a file being added is already part of the database.
Action: Use a different file name.
```

客户并没有认识到这样的错误，紧跟着一个创建语句再次被发出，此次的表空间名称又变成了大小写混合的"EMIS116k"：

```
Thu Jun 24 19:04:28 2010
CREATE  TABLESPACE "EMIS116k" DATAFILE '/dev/vg02/rlvdata224_8G'
SIZE 7900M AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED LOGGING
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
```

这显然也不会成功，这次的错误提示是 ORA-1543：

```
Thu Jun 24 19:04:28 2010
ORA-1543 signalled during: CREATE  TABLESPACE "EMIS116k" DATAFILE '/dev/vg02/rlvdata224_8G'
SIZE 7900M AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED LOGGING
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
...
```

ORA-01543 错误是指，要创建的表空间已经存在：

```
ORA-01543 tablespace 'string' already exists

Cause: An attempt was made to create a tablespace which already exists.
Action: Use a different name for the new tablespace.
```

好了，到这里打住。

我要提醒大家的一点是：**如果你在工作中遇到了类似的情况，当数据库的表征超出了你的预期，那么最好停下来，仔细检查或咨询他人，避免在生产数据库上进行无把握的试错和尝试。**

这次的事情还要继续，接下来两个 DROP 命令被发出，但是 EMIS116K 和 EMIS116k 都不被数据库认可，两次尝试又失败了：

```
Thu Jun 24 19:05:09 2010
DROP TABLESPACE EMIS116K INCLUDING CONTENTS AND DATAFILES
Thu Jun 24 19:05:09 2010
ORA-959 signalled during: DROP TABLESPACE EMIS116K INCLUDING CONTENTS AND DATAFILES
...
Thu Jun 24 19:05:31 2010
DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
ORA-959 signalled during: DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
...
```

用户开始尝试将该文件离线再 Online，试图通过这样的操作消除这个不明原因的无法 DROP 问题。这个尝试过程又引入了恢复的环节，还好一切顺利，表空间的数据文件能够被重新 Online，这样看起来数据文件是没有问题的：

```
Thu Jun 24 19:20:32 2010
alter database datafile '/dev/vg02/rlvdata224_8G' offline
Thu Jun 24 19:20:33 2010
Completed: alter database datafile '/dev/vg02/rlvdata224_8G' offline
Thu Jun 24 19:21:02 2010
alter database datafile '/dev/vg02/rlvdata224_8G' online
Thu Jun 24 19:21:02 2010
ORA-1113 signalled during: alter database datafile '/dev/vg02/rlvdata224_8G' online...
Thu Jun 24 19:21:31 2010
ALTER DATABASE RECOVER  datafile '/dev/vg02/rlvdata224_8G'
Thu Jun 24 19:21:31 2010
Media Recovery Start
 parallel recovery started with 15 processes
Thu Jun 24 19:21:35 2010
Media Recovery Complete (cwgk1)
Completed: ALTER DATABASE RECOVER  datafile '/dev/vg02/rlvdata224_8G'
Thu Jun 24 19:21:45 2010
alter database datafile '/dev/vg02/rlvdata224_8G' online
Thu Jun 24 19:21:45 2010
Completed: alter database datafile '/dev/vg02/rlvdata224_8G' online
```

再回过头来继续尝试 DROP 表空间的操作，在没找到根本原因之前，这样的操作当然又失败了：

```
Thu Jun 24 19:48:18 2010
DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
Thu Jun 24 19:48:18 2010
ORA-959 signalled during: DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
...
```

```
Thu Jun 24 19:48:54 2010
DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
Thu Jun 24 19:48:54 2010
ORA-959 signalled during: DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
...
```

终于客户发现了大小写的问题，尝试将这个小写的 EMIS116k 更改为大写的 EMIS116K 名称。但是注意，不加入双引号，Oracle 认为这两者是没有区别的：

```
Thu Jun 24 19:52:52 2010
ALTER  TABLESPACE EMIS116k RENAME TO EMIS116K
Thu Jun 24 19:52:52 2010
ORA-710 signalled during: ALTER  TABLESPACE EMIS116k RENAME TO EMIS116K
...
```

ORA-710 是说这两个名字是一样的，无需更改：

```
ORA-00710 new tablespace name is the same as the old tablespace name
Cause：An attempt to rename a tablespace failed because the new name is the same as the old
name.
Action：No action required.
```

这个错误出现在语法校验层面，数据库根本不需要去检查表空间是否存在，比如如下的测试可以说明这个问题：

```
SQL> alter tablespace eygle rename to EYGLE;
alter tablespace eygle rename to EYGLE
*
第 1 行出现错误:
ORA-00710: 新的表空间与旧的表空间同名
```

用户继续 DROP，仍然是 ORA-959：

```
Thu Jun 24 20:00:04 2010
DROP TABLESPACE EMIS116k
Thu Jun 24 20:00:04 2010
ORA-959 signalled during: DROP TABLESPACE EMIS116k
...
```

看着这样的过程，我们是否会有很着急的感觉？我相信在现场时，你可能会是焦急！

终于用户发现了本质问题，发出了带着双引号的删除命令，这本身是没有问题的，可是恰恰此时数据库出了问题，一个异常的 ORA-00600 错误出现了，4348 出现：

```
Thu Jun 24 20:03:59 2010
DROP TABLESPACE "EMIS116k"
Thu Jun 24 20:03:59 2010
Errors in file /oracle/admin/cwgk/udump/cwgk1_ora_25919.trc:
ORA-00600: internal error code, arguments: [4348], [U], [0], [229], [], [], [], []
```

再然后，案情基本成立，任何 DROP 命令都提示表空间不存在了：

```
Thu Jun 24 20:04:12 2010
DROP TABLESPACE "EMIS116k"
Thu Jun 24 20:04:13 2010
ORA-959 signalled during: DROP TABLESPACE "EMIS116k"
...
Thu Jun 24 20:04:30 2010
DROP TABLESPACE "EMIS116k"
Thu Jun 24 20:04:30 2010
ORA-959 signalled during: DROP TABLESPACE "EMIS116k"
...
Thu Jun 24 20:04:40 2010
DROP TABLESPACE "EMIS116n"
Thu Jun 24 20:04:40 2010
ORA-959 signalled during: DROP TABLESPACE "EMIS116n"
...
```

接下来用户继续尝试创建其他的表空间，此时 ORA-00600 25013 错误出现：

```
Fri Jun 25 09:14:00 2010
CREATE  TABLESPACE "EMIS116N2" DATAFILE '/dev/vg02/rlvdata088_16G'
SIZE 7900M AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED LOGGING
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
Fri Jun 25 09:18:40 2010
Errors in file /oracle/admin/cwgk/udump/cwgk1_ora_20031.trc:
ORA-00600: internal error code, arguments: [25013], [0], [229], [EMIS116N2],
[EMIS116k], [184], [179], []
Fri Jun 25 09:18:45 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_pmon_4050.trc:
ORA-00600: internal error code, arguments: [25015], [229], [179], [184], [], [], [], []
Fri Jun 25 09:18:47 2010
Trace dumping is performing id=[cdmp_20100625091847]
Fri Jun 25 09:18:47 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_pmon_4050.trc:
ORA-00600: internal error code, arguments: [kccocx_01], [], [], [], [], [], [], []
ORA-00600: internal error code, arguments: [25015], [229], [179], [184], [], [], [], []
```

伴随这些错误，有一个新的提示出现，Oracle 实例进行事务恢复时遇到 600 错误，这已经给我们一个暗示，数据库内部事务出现了一致性的问题：

```
Fri Jun 25 09:19:01 2010
ORACLE Instance cwgk1 (pid = 30) - Error 600 encountered while recovering transaction (203,
10).
```

此后，后台进程异常终止，数据库挂起，最后被迫 Abort 关闭了数据库：

```
Fri Jun 25 09:24:06 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_pmon_4050.trc:
ORA-00474: SMON process terminated with error
Fri Jun 25 09:24:06 2010
PMON: terminating instance due to error 474
Fri Jun 25 09:24:06 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_lms3_4071.trc:
ORA-00474: SMON process terminated with error
Fri Jun 25 09:24:06 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_lms1_4067.trc:
ORA-00474: SMON process terminated with error
Fri Jun 25 09:24:06 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_lms0_4065.trc:
ORA-00474: SMON process terminated with error
Fri Jun 25 09:24:06 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_lms2_4069.trc:
ORA-00474: SMON process terminated with error
Fri Jun 25 09:24:08 2010
Shutting down instance (abort)
License high water mark = 265
Fri Jun 25 09:24:12 2010
Instance terminated by PMON, pid = 4050
Fri Jun 25 09:24:13 2010
Instance terminated by USER, pid = 24022
```

接下来用户执行了一系列的恢复尝试，包括重建控制文件，强制 Resetlogs 打开数据库等，最终导致情况失控。

我们回过头来分析一下最初的错误出现时机，这里用户执行了正确的语句，但是数据库却不再配合，报出了 ORA-00600 4348 错误，这个错误号在 Metalink 上也没有收录，错误原因未知：

```
Thu Jun 24 20:03:59 2010
DROP TABLESPACE "EMIS116k"
Thu Jun 24 20:03:59 2010
Errors in file /oracle/admin/cwgk/udump/cwgk1_ora_25919.trc:
ORA-00600: internal error code, arguments: [4348], [U], [0], [229], [], [], [], []
```

但是结合后面的出错信息：

```
Fri Jun 25 09:19:01 2010
ORACLE Instance cwgk1 (pid = 30) - Error 600 encountered while recovering transaction (203,
```

10).

我们猜测，DROP TABLESPACE 实质上执行了后台递归操作，但是这个 DDL 事务因为异常，在失败后没有完成回滚，造成了数据字典不一致。

2.5.3 Drop Tablespace Internal

在学习和研究 Oracle 数据库技术时，学会经常性跟踪和分析一下内部操作非常有助于深入理解 Oracle。

此处，我们可以跟踪一下 DROP TABLESPACE 的过程，获得一些内部 SQL，帮助我们进行一下进一步的判断：

```
SQL> create tablespace eygle datafile size 10M;

表空间已创建。

SQL> alter session set events '10046 trace name context forever,level 12';

会话已更改。

SQL> drop tablespace eygle;

表空间已删除。

SQL> alter session set events '10046 trace name context off';

会话已更改。
```

通过 tkprof 工具格式化一下跟踪文件，摘录一下主要的内部 DML 操作如下，主要包含了三个 Update 操作，分别是针对两个底层表 ts$和 file$的操作。

第一个 SQL 更新语句如下：

```
SQL ID:
Plan Hash: 3567703090
update ts$ set name=:2,online$=:3,contents$=:4,undofile#=:5,undoblock#=:6,
  blocksize=:7,dflmaxext=:8,dflinit=:9,dflincr=:10,dflextpct=:11,dflminext=
  :12,dflminlen=:13,owner#=:14,scnwrp=:15,scnbas=:16,pitrscnwrp=:17,
  pitrscnbas=:18,dflogging=:19,bitmapped=:20,inc#=:21,flags=:22,plugged=:23,
  spare1=:24,spare2=:25
where ts#=:1


Rows     Row Source Operation
-------  ---------------------------------------------------
```

```
      0  UPDATE  TS$ (cr=4 pr=0 pw=0 time=176 us)
      1   TABLE ACCESS CLUSTER TS$ (cr=4 pr=0 pw=0 time=75 us)
      1    INDEX UNIQUE SCAN I_TS# (cr=1 pr=0 pw=0 time=17 us)(object id 7)

********************************************************************************
```

我们可以对照格式化之前的跟踪文件信息，获得绑定变量等内容，进行进一步分析（为了简洁，删除了跟踪文件的部分信息）。

以下输出显示了绑定变量，这个 SQL 更新了 TS$字典表，将表空间 6（ts# = 6）更新 name='EYGLE'，Online$=2，完成了这一步骤的操作：

```
PARSING IN CURSOR #16 len=322 dep=1 uid=0 oct=6 lid=0 tim=20747569441 hv=463096017
ad='3028cda4'
update ts$ set name=:2,online$=:3,contents$=:4,undofile#=:5,undoblock#=:6, blocksize=:7,
dflmaxext=:8,dflinit=:9,dflincr=:10,dflextpct=:11,dflminext=:12,dflminlen=:13,owner#=:14,
scnwrp=:15,scnbas=:16,pitrscnwrp=:17,pitrscnbas=:18,dflogging=:19,bitmapped=:20,inc#=:21,
flags=:22,plugged=:23,spare1=:24,spare2=:25 where ts#=:1
END OF STMT
PARSE #16:c=0,e=427,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=4,tim=20747569438
BINDS #16:
kkscoacd
 Bind#0
  oacdty=01 mxl=32(05) mxlc=00 mal=00 scl=00 pre=00
  oacflg=18 fl2=0001 frm=01 csi=852 siz=32 off=0
  kxsbbbfp=33ec347a  bln=32  avl=05  flg=09
  value="EYGLE"
 Bind#1
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=0686ffc4  bln=24  avl=02  flg=05
  value=2
<…… Bind#2 ~ BIND#4 value=0 ……>
Bind#5
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=0686ff2c  bln=24  avl=03  flg=05
  value=8192
 Bind#6
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=0686ff08  bln=24  avl=06  flg=05
```

```
 value=2147483645
Bind#7
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=0686fee4  bln=24  avl=02  flg=05
 value=8
Bind#8
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=0686fec0  bln=24  avl=03  flg=05
 value=128
Bind#9
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=0686fe9c  bln=24  avl=01  flg=05
 value=0
Bind#10
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=0686fe78  bln=24  avl=02  flg=05
 value=1
Bind#11
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=0686fe4c  bln=24  avl=02  flg=05
 value=8
<……Bind#12~Bind15 value=0……>
Bind#16
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=096f04c0  bln=24  avl=04  flg=05
 value=905730
Bind#17
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=096f049c  bln=24  avl=02  flg=05
 value=1
Bind#18
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
```

```
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=096f0478  bln=24  avl=02  flg=05
  value=8
 Bind#19
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=096f0454  bln=24  avl=02  flg=05
  value=3
 Bind#20
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=096f0430  bln=24  avl=02  flg=05
  value=33
<…… Bind#21~Bind#23 value=0……>
 Bind#24
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=0686ffe8  bln=22  avl=02  flg=05
  value=6
=====================
```

那么这个步骤做的是什么工作呢？

参考 sql.bsq 文件中的 ts$表创建的语句。在 Oracle 的源码 KTT.H 中记录了 Online$字段的几个可选状态，其中 1 为 Online，2 为 Offline，3 为 INVALID，数据库通过状态位来标记表空间的变化:

```
create table ts$                                                /* tablespace table */
( ts#           number not null,                  /* tablespace identifier number */
  name          varchar2("M_IDEN") not null,           /* name of tablespace */
  owner#        number not null,                      /* owner of tablespace */
  online$       number not null,                      /* status (see KTT.H): */
                                        /* 1 = ONLINE, 2 = OFFLINE, 3 = INVALID */
  contents$     number not null,      /* TEMPORARY/PERMANENT                    */
  undofile#     number,   /* undo_off segment file number (status is OFFLINE) */
  undoblock#    number,                /* undo_off segment header file number */
  blocksize     number not null,                  /* size of block in bytes */
  inc#          number not null,                /* incarnation number of extent */
  scnwrp        number,      /* clean offline scn - zero if not offline clean */
  scnbas        number,               /* scnbas - scn base, scnwrp - scn wrap */
  dflminext     number not null,        /*  default minimum number of extents */
  dflmaxext     number not null,         /* default maximum number of extents */
  dflinit       number not null,               /* default initial extent size */
  dflincr       number not null,                  /* default next extent size */
  dflminlen     number not null,               /* default minimum extent size */
  dflextpct     number not null,      /* default percent extent size increase */
  dflogging     number not null,
      /* lowest bit: default logging attribute: clear=NOLOGGING, set=LOGGING */
                                     /* second lowest bit: force logging mode */
  affstrength   number not null,                       /* Affinity strength */
  bitmapped     number not null,        /* If not bitmapped, 0 else unit size */
                                                              /* in blocks */
  plugged       number not null,                             /* If plugged */
  directallowed number not null,    /* Operation which invalidate standby are */
                                                                /* allowed */
  flags         number not null,                          /* various flags */
```

前面的第一个 SQL 将表空间的 Online$状态更改为 2，也就是 Offline。这也就告诉我们，对于表空间的 DROP 操作，实际上要先将表空间离线，而将表空间离线，会触发表空间检查点操作，将表空间中的内存脏数据写出到数据文件上。这些从跟踪文件的等待事件上可以清晰的看到。请认真理解体会一下以下等待事件列表：

```
WAIT #6: nam='log file sync' ela= 392 buffer#=6496 p2=0 p3=0 obj#=3 tim=20747572833
WAIT #6: nam='control file sequential read' ela= 488 file#=0 block#=1 blocks=1 obj#=3 tim=20747573396
WAIT #6: nam='control file sequential read' ela= 313 file#=1 block#=1 blocks=1 obj#=3 tim=20747573738
WAIT #6: nam='control file sequential read' ela= 301 file#=2 block#=1 blocks=1 obj#=3 tim=20747574066
WAIT #6: nam='control file sequential read' ela= 1507 file#=0 block#=15 blocks=1 obj#=3 tim=20747575601
WAIT #6: nam='control file sequential read' ela= 309 file#=0 block#=17 blocks=1 obj#=3 tim=20747575952
WAIT #6: nam='control file sequential read' ela= 1243 file#=0 block#=24 blocks=1 obj#=3 tim=20747577246
WAIT #6: nam='db file sequential read' ela= 265 file#=5 block#=1 blocks=1 obj#=3
tim=20747577576
WAIT #6: nam='control file parallel write' ela= 1523 files=3 block#=23 requests=3 obj#=3 tim=20747579133
WAIT #6: nam='control file parallel write' ela= 1176 files=3 block#=18 requests=3 obj#=3 tim=20747580346
WAIT #6: nam='control file parallel write' ela= 1237 files=3 block#=16 requests=3 obj#=3 tim=20747581619
WAIT #6: nam='control file parallel write' ela= 1154 files=3 block#=1 requests=3 obj#=3 tim=20747582808
WAIT #6: nam='control file sequential read' ela= 336 file#=0 block#=1 blocks=1 obj#=3 tim=20747583177
WAIT #6: nam='rdbms ipc reply' ela= 2033 from_process=5 timeout=21474836 p3=0 obj#=3 tim=20747585265
WAIT #6: nam='rdbms ipc reply' ela= 43 from_process=7 timeout=21474836 p3=0 obj#=3 tim=20747585347
```

```
WAIT #6: nam='rdbms ipc reply' ela= 32 from_process=7 timeout=21474836 p3=0 obj#=3 tim=20747585405
WAIT #6: nam='enq: TC - contention' ela= 15 name|mode=1413677062 checkpoint ID=65551 0=0
obj#=3 tim=20747585451
WAIT #6: nam='control file sequential read' ela= 262 file#=0 block#=1 blocks=1 obj#=3 tim=20747585772
WAIT #6: nam='control file sequential read' ela= 257 file#=1 block#=1 blocks=1 obj#=3 tim=20747586055
WAIT #6: nam='control file sequential read' ela= 252 file#=2 block#=1 blocks=1 obj#=3 tim=20747586333
WAIT #6: nam='control file sequential read' ela= 242 file#=0 block#=16 blocks=1 obj#=3 tim=20747586601
WAIT #6: nam='control file sequential read' ela= 246 file#=0 block#=18 blocks=1 obj#=3 tim=20747586873
WAIT #6: nam='control file sequential read' ela= 214 file#=0 block#=23 blocks=1 obj#=3 tim=20747587118
WAIT #6: nam='db file sequential read' ela= 217 file#=5 block#=1 blocks=1 obj#=3
tim=20747587364
WAIT #6: nam='db file single write' ela= 374 file#=5 block#=1 blocks=1 obj#=3 tim=20747587769
WAIT #6: nam='control file parallel write' ela= 918 files=3 block#=24 requests=3 obj#=3 tim=20747588722
WAIT #6: nam='control file parallel write' ela= 949 files=3 block#=17 requests=3 obj#=3 tim=20747589701
WAIT #6: nam='control file parallel write' ela= 916 files=3 block#=15 requests=3 obj#=3 tim=20747590654
WAIT #6: nam='control file parallel write' ela= 713 files=3 block#=1 requests=3 obj#=3 tim=20747591404
WAIT #6: nam='control file sequential read' ela= 315 file#=0 block#=1 blocks=1 obj#=3 tim=20747591745
WAIT #6: nam='rdbms ipc reply' ela= 11728 from_process=5 timeout=2147483647 p3=0 obj#=3 tim=20747736143
```

注意这里的 enq: TC – contention 就是指 Thread Checkpoint。

接下来的操作更新了 file$字典表：

```
SQL ID:
Plan Hash: 3567873039
update file$ set blocks=:2,ts#=DECODE(:3,-1,NULL,:3),status$=:4, relfile#=
  DECODE(:5,0,NULL,:5),maxextend=:6,inc=:7,crscnwrp=:8,crscnbas=:9,spare1=
  DECODE(:10,0,NULL,:10)
where file#=:1

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  UPDATE  FILE$ (cr=1 pr=0 pw=0 time=345 us)
      1   INDEX UNIQUE SCAN I_FILE1 (cr=1 pr=0 pw=0 time=24 us)(object id 41)

********************************************************************************
```

这个 SQL 的绑定变量信息如下：

```
PARSING IN CURSOR #10 len=182 dep=1 uid=0 oct=6 lid=0 tim=20747737469 hv=810151256
ad='3028c9e4'
update file$ set blocks=:2,ts#=DECODE(:3,-1,NULL,:3),status$=:4,
relfile#=DECODE(:5,0,NULL,:5),maxextend=:6,inc=:7,crscnwrp=:8,crscnbas=:9,spare1=DECODE(:
```

```
10,0,NULL,:10)where file#=:1
END OF STMT
PARSE #10:c=15601,e=1129,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=4,tim=20747737461
BINDS #10:
kkscoacd
 Bind#0
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=096f052c  bln=24  avl=03  flg=05
  value=1280
 Bind#1
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=096f0508  bln=24  avl=03  flg=05
  value=-1
 Bind#2
  No oacdef for this bind.
 Bind#3
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=096f04e4  bln=24  avl=02  flg=05
  value=1
 Bind#4
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=096f04c0  bln=24  avl=01  flg=05
  value=0
 Bind#5
  No oacdef for this bind.
 Bind#6
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=096f049c  bln=24  avl=01  flg=05
  value=0
 Bind#7
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=096f0478  bln=24  avl=01  flg=05
  value=0
```

```
Bind#8
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=096f0454  bln=24  avl=01  flg=05
 value=0
Bind#9
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=096f0430  bln=24  avl=04  flg=05
 value=905737
Bind#10
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=096f040c  bln=24  avl=05  flg=05
 value=20971522
Bind#11
 No oacdef for this bind.
Bind#12
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=096f0550  bln=22  avl=02  flg=05
 value=5
```

结合 file$的信息，我们可以很容易的分辨出来，以上 SQL 将 file$表中的 file#=5 文件的状态 STATUS$改成了 1，同时部分字段置空：

```
SQL> select file#,relfile#,inc,status$,blocks,ts#,crscnwrp,crscnbas from file$;

     FILE#    RELFILE#        INC    STATUS$    BLOCKS       TS#  CRSCNWRP  CRSCNBAS
---------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
         1          1       1280          2      38400          0          0          9
         2          2        640          2       3200          1          0     562043
         3          3       1280          2      15360          2          0       6381
         4          4        160          2        640          4          0      10365
         5                     0          1       1280                     0     905737
```

我们知道，Oracle 数据库在删除文件时，是不会将文件信息从 file$中删除的，替代的，数据库只是将文件的状态更新，根据字典记录，status$为 1 的文件是 INVALID 的，数据库在启动时不会检查，而状态为 2 的是 AVAILABLE，这些信息来自 KTS.H 的源码文件：

```
create table file$                                                    /* file table */
( file#         number not null,                        /* file identifier number */
  status$       number not null,                          /* status (see KTS.H): */
                                                  /* 1 = INVALID, 2 = AVAILABLE */
  blocks        number not null,                        /* size of file in blocks */
                                                /* zero for bitmapped tablespaces */
  ts#           number,                             /* tablespace that owns file */
  relfile#      number,                               /* relative file number */
  maxextend     number,                                  /* maximum file size */
  inc           number,                                   /* increment amount */
  crscnwrp      number,                                  /* creation SCN wrap */
  crscnbas      number,                                  /* creation SCN base */
  ownerinstance varchar("M_IDEN"),                      /* Owner instance name */
  spare1        number,       /* tablespace-relative DBA of space file header */
                                      /* NULL for dictionary-mapped tablespaces */
  spare2        number,
  spare3        varchar2(1000),
  spare4        date
)
/
```

完成 file$的更新之后，数据库继续更新了 ts$字典表：

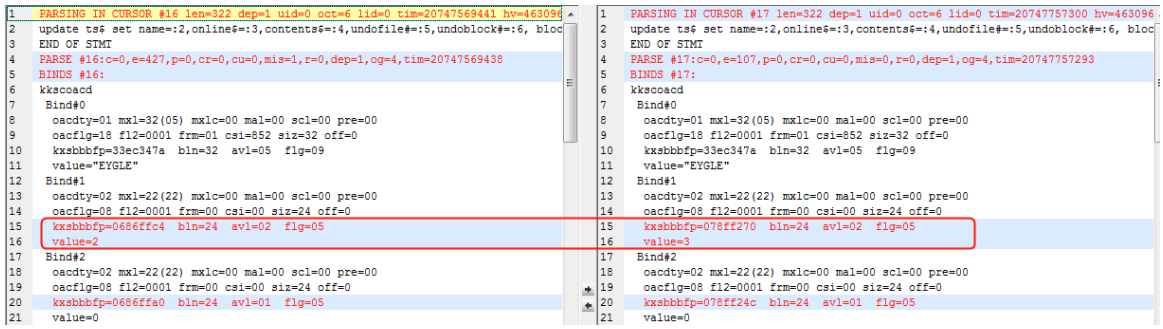
```
SQL ID:
Plan Hash: 3567893424
update ts$ set name=:2,online$=:3,contents$=:4,undofile#=:5,undoblock#=:6,
  blocksize=:7,dflmaxext=:8,dflinit=:9,dflincr=:10,dflextpct=:11,dflminext=
  :12,dflminlen=:13,owner#=:14,scnwrp=:15,scnbas=:16,pitrscnwrp=:17,
  pitrscnbas=:18,dflogging=:19,bitmapped=:20,inc#=:21,flags=:22,plugged=:23,
  spare1=:24,spare2=:25
where ts#=:1

Rows     Row Source Operation
-------  ---------------------------------------------------
      0  UPDATE  TS$ (cr=4 pr=0 pw=0 time=255 us)
      1   TABLE ACCESS CLUSTER TS$ (cr=4 pr=0 pw=0 time=130 us)
      1    INDEX UNIQUE SCAN I_TS# (cr=1 pr=0 pw=0 time=28 us)(object id 7)

********************************************************************************
```

其相关更新值如下：

```
PARSING IN CURSOR #17 len=322 dep=1 uid=0 oct=6 lid=0 tim=20747757300 hv=463096017
ad='3028cda4'
update ts$ set name=:2,online$=:3,contents$=:4,undofile#=:5,undoblock#=:6,
blocksize=:7,dflmaxext=:8,dflinit=:9,dflincr=:10,dflextpct=:11,dflminext=:12,dflminlen=:1
3,owner#=:14,scnwrp=:15,scnbas=:16,pitrscnwrp=:17,pitrscnbas=:18,dflogging=:19,bitmapped=
:20,inc#=:21,flags=:22,plugged=:23,spare1=:24,spare2=:25 where ts#=:1
END OF STMT
```

```
PARSE #17:c=0,e=107,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=20747757293
BINDS #17:
kkscoacd
 Bind#0
  oacdty=01 mxl=32(05) mxlc=00 mal=00 scl=00 pre=00
  oacflg=18 fl2=0001 frm=01 csi=852 siz=32 off=0
  kxsbbbfp=33ec347a  bln=32  avl=05  flg=09
  value="EYGLE"
 Bind#1
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=078ff270  bln=24  avl=02  flg=05
  value=3
 。。。。。。。。。
 Bind#24
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=078ff294  bln=22  avl=02  flg=05
  value=6
```

比较两次更新，仅有一个绑定变量的值发生了改变，就是 Online$从 2（Offline）改变为 3（INVALID），就完成了表空间的删除操作（当然我们还省略了一些其他的次要过程）：



```
PARSING IN CURSOR #16 len=322 dep=1 uid=0 oct=6 lid=0 tim=20747569441 hv=463096
update ts$ set name=:2,online$=:3,contents$=:4,undofile#=:5,undoblock#=:6, bloc
END OF STMT
PARSE #16:c=0,e=427,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=4,tim=20747569438
BINDS #16:
kkscoacd
 Bind#0
  oacdty=01 mxl=32(05) mxlc=00 mal=00 scl=00 pre=00
  oacflg=18 fl2=0001 frm=01 csi=852 siz=32 off=0
  kxsbbbfp=33ec347a  bln=32  avl=05  flg=09
  value="EYGLE"
 Bind#1
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=0686ffc4  bln=24  avl=02  flg=05
  value=2
 Bind#2
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=0686ffa0  bln=24  avl=01  flg=05
  value=0
```

```
PARSING IN CURSOR #17 len=322 dep=1 uid=0 oct=6 lid=0 tim=20747757300 hv=463096
update ts$ set name=:2,online$=:3,contents$=:4,undofile#=:5,undoblock#=:6, bloc
END OF STMT
PARSE #17:c=0,e=107,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=20747757293
BINDS #17:
kkscoacd
 Bind#0
  oacdty=01 mxl=32(05) mxlc=00 mal=00 scl=00 pre=00
  oacflg=18 fl2=0001 frm=01 csi=852 siz=32 off=0
  kxsbbbfp=33ec347a  bln=32  avl=05  flg=09
  value="EYGLE"
 Bind#1
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=078ff270  bln=24  avl=02  flg=05
  value=3
 Bind#2
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=078ff24c  bln=24  avl=01  flg=05
  value=0
```

查询一下 TS$的信息，我们可以看到 6 号表空间 EYGLE 的存在，其表空间状态为 3：

```
SQL> select ts#,name,online$ from ts$;

       TS# NAME                           ONLINE$
---------- ------------------------------ ----------
         0 SYSTEM                                 1
         1 UNDOTBS1                               1
         2 SYSAUX                                 1
         3 TEMP                                   1
         4 USERS                                  1
         5 UNDOTBS2                               3
         6 EYGLE                                  3
         7 EYGLe                                  3

已选择8行。
```

而此时查询 dba_tablespaces 视图时，是看不到这些状态位 3 的表空间信息的：

```
SQL> select tablespace_name from dba_tablespaces;

TABLESPACE_NAME
------------------------------
SYSTEM
UNDOTBS1
SYSAUX
TEMP
USERS
```

这是因为在创建视图时，Oracle 对 online$字段进行了过滤：

```
create or replace view DBA_TABLESPACES
    (TABLESPACE_NAME, BLOCK_SIZE, INITIAL_EXTENT, NEXT_EXTENT, MIN_EXTENTS,
     MAX_EXTENTS, PCT_INCREASE, MIN_EXTLEN,STATUS, CONTENTS, LOGGING,
      FORCE_LOGGING, EXTENT_MANAGEMENT, ALLOCATION_TYPE, PLUGGED_IN,
     SEGMENT_SPACE_MANAGEMENT, DEF_TAB_COMPRESSION, RETENTION, BIGFILE)
as select ts.name, ts.blocksize, ts.blocksize * ts.dflinit,
          decode(bitand(ts.flags, 3), 1, to_number(NULL), ts.blocksize * ts.dflincr),
          ts.dflminext,
          decode(ts.contents$, 1, to_number(NULL), ts.dflmaxext),
          decode(bitand(ts.flags, 3), 1, to_number(NULL), ts.dflextpct),
          ts.blocksize * ts.dflminlen,
          decode(ts.online$, 1, 'ONLINE', 2, 'OFFLINE',
                 4, 'READ ONLY', 'UNDEFINED'),
          decode(ts.contents$, 0, (decode(bitand(ts.flags, 16), 16, 'UNDO',
                 'PERMANENT')), 1, 'TEMPORARY'),
          decode(bitand(ts.dflogging, 1), 0, 'NOLOGGING', 1, 'LOGGING'),
          decode(bitand(ts.dflogging, 2), 0, 'NO', 2, 'YES'),
```

```
        decode(ts.bitmapped, 0, 'DICTIONARY', 'LOCAL'),
        decode(bitand(ts.flags, 3), 0, 'USER', 1, 'SYSTEM', 2, 'UNIFORM', 'UNDEFINED'),
        decode(ts.plugged, 0, 'NO', 'YES'),
        decode(bitand(ts.flags,32), 32,'AUTO', 'MANUAL'),
        decode(bitand(ts.flags,64), 64,'ENABLED', 'DISABLED'),
        decode(bitand(ts.flags,16), 16, (decode(bitand(ts.flags, 512), 512,
               'GUARANTEE', 'NOGUARANTEE')), 'NOT APPLY'),
        decode(bitand(ts.flags,256), 256, 'YES', 'NO')
from sys.ts$ ts
where ts.online$ != 3
and bitand(flags,2048) != 2048
/
```

如果我们进一步的创建一个新的表空间，Oracle 会重用之前 file$中删除的文件号：

```
SQL> create tablespace enmo datafile size 10M;
表空间已创建。
SQL> select ts#,name,online$ from ts$;

       TS# NAME                           ONLINE$
---------- ------------------------------ ----------
         0 SYSTEM                                  1
         1 UNDOTBS1                                1
         2 SYSAUX                                  1
         3 TEMP                                    1
         4 USERS                                   1
         5 UNDOTBS2                                3
         6 EYGLE                                   3
         7 EYGLe                                   3
         8 ENMO                                    1
已选择9行。
SQL> select file#,relfile#,inc,status$,blocks,ts#,crscnwrp,crscnbas from file$;

     FILE#   RELFILE#        INC    STATUS$     BLOCKS        TS#  CRSCNWRP   CRSCNBAS
---------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
         1          1       1280          2      38400          0          0          9
         2          2        640          2       3200          1          0     562043
         3          3       1280          2      15360          2          0       6381
         4          4        160          2        640          4          0      10365
         5          5          0          2       1280          8          0     907947
```

而如果我们重建的表空间与之前删除的同名，则 Oracle 会重用之前的表空间信息：

```
SQL> create tablespace eygle datafile size 20M;
表空间已创建。

SQL> select ts#,name,online$ from ts$ where ts#>4;
       TS# NAME                           ONLINE$
---------- ------------------------------ ----------
         5 UNDOTBS2                                3
         6 EYGLE                                   1
```

```
         7 EYGLe                                3
         8 ENMO                                 1

SQL> select file#,relfile#,inc,status$,blocks,ts#,crscnwrp,crscnbas
  2  from file$ where file#>4;

     FILE#   RELFILE#        INC    STATUS$     BLOCKS        TS#   CRSCNWRP   CRSCNBAS
---------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
         5          5          0          2       1280          8          0     907947
         6          6          0          2       2560          6          0
908003
```

2.5.4 ORA-600 4348 错误的成因

明白了所有的内部操作过程，我们又可以继续分析 ORA-00600 4348 号错误的成因了：

```
ORA-00600: internal error code, arguments: [4348], [U], [0], [229], [], [], [], []
```

我说过，猜测是一种重要的学习能力，现在我们就要运用这种能力了。

这个错误的 4384 后的第一个代码是 U，应该是 Update 的缩写，更新的应该就是 file$，ts$字典表。这里的 229 应该就是更新的表空间号，Update 出错的可能就是被更新的记录不存在了。

那么更新的记录为什么会不存在？

这个我猜测可能是某个 DBA 考虑到是这个表空间的问题，手工执行了对于字典表的 Delete 操作，这就导致了后面的异常；当然也有可能是 Oracle 数据库自身丢失了这条记录（这种可能性不大）。

可以验证一下这个猜测，首先通过手工删除一个表空间信息。当然我们需要严正提示：**这些操作仅可在测试环境，或者说属于你自己的测试环境上进行，测试之前请先备份你的数据库。**

删除可以通过 dba_tablespaces 和 ts$表进行，我们选择了对 ts$表直接操作：

```
SQL> delete from dba_tablespaces where tablespace_name='ENMO';

已删除 1 行。

SQL> select ts#,name,online$ from ts$ ;
       TS# NAME                           ONLINE$
---------- ------------------------------ ----------
         0 SYSTEM                                 1
         1 UNDOTBS1                               1
         2 SYSAUX                                 1
         3 TEMP                                   1
```

```
         4 USERS                                1
         5 UNDOTBS2                             3
         6 EYGLE                                1
         7 EYGLe                                3

已选择 8 行。

SQL> rollback;
回退已完成。

SQL> select ts#,name,online$ from ts$ ;
       TS# NAME                           ONLINE$
---------- ------------------------------ ----------
         0 SYSTEM                                 1
         1 UNDOTBS1                               1
         2 SYSAUX                                 1
         3 TEMP                                   1
         4 USERS                                  1
         5 UNDOTBS2                               3
         6 EYGLE                                  1
         7 EYGLe                                  3
         8 ENMO                                       1
已选择 9 行。
SQL> delete from ts$ where ts#=8;
已删除 1 行。
SQL> commit;
提交完成。
```

删除了字典表信息之后，如果此时再来执行 drop tablespace 的操作，即出现了 4348 错误，后面的参数 8 是指表空间号是 8，也就是说当试图去 Update 修改 8 号表空间的状态时出现了异常，也就是发现这个表空间的信息不存在了，在正常情况下，这种情况绝不应该发生，因为数据库根本不对 ts$执行 DELETE 操作：

```
SQL> drop tablespace enmo;
drop tablespace enmo
*
第 1 行出现错误:
ORA-00600: 内部错误代码, 参数: [4348], [U], [0], [8], [], [], [], []
```

告警日志中也记录了错误信息，并且产生了一个跟踪文件，这里的 8 号表空间也就是 ENMO 表空间：

```
Mon Aug 09 15:58:10 2010
drop tablespace enmo
Mon Aug 09 15:58:10 2010
Errors in file d:\oracle\admin\enmo\udump\enmo_ora_4736.trc:
ORA-00600: 内部错误代码, 参数: [4348], [U], [0], [8], [], [], [], []

Mon Aug 09 15:58:11 2010
ORA-600 signalled during: drop tablespace enmo...
```

跟踪文件中也显示全部的相关信息，绑定变量等：

```
Cursor#3(04D60CA4) state=BOUND curiob=04D7B9B4
 curflg=5 fl2=0 par=04D60C24 ses=34343FE4
 sqltxt(3028CDA4)=
update ts$ set name=:2,online$=:3,contents$=:4,undofile#=:5,undoblock#=:6,
blocksize=:7,dflmaxext=:8,dflinit=:9,dflincr=:10,dflextpct=:11,dflminext=:12,dflminlen=:1
3,owner#=:14,scnwrp=:15,scnbas=:16,pitrscnwrp=:17,pitrscnbas=:18,dflogging=:19,bitmapped=
:20,inc#=:21,flags=:22,plugged=:23,spare1=:24,spare2=:25 where ts#=:1
  hash=29d4143c7a377b40366bdc581b9a48d1
  parent=2FED6560 maxchild=02 plk=31B316EC ppn=n
cursor instantiation=04D7B9B4 used=1281340689
 child#0(3028CC60) pcs=2FED6764
  clk=31B31BBC ci=2FED5D00 pn=00000000 ctx=2F966490
 kgsccflg=0 llk[04D7B9B8,04D7B9B8] idx=0
 xscflg=a0100426 fl2=5000400 fl3=2218c fl4=0
 sharing failure(s)=10
 Bind bytecodes
  Opcode = 5   Bind Rpi Scalar Sql In (not out) Nocopy
…….
kkscoacd
 Bind#0
  oacdty=01 mxl=32(04) mxlc=00 mal=00 scl=00 pre=00
  oacflg=18 fl2=0001 frm=01 csi=852 siz=32 off=0
  kxsbbbfp=30284552  bln=32  avl=04  flg=09
  value="ENMO"
 Bind#1
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=04d734f4  bln=24  avl=02  flg=05
  value=2
 Bind#2~ Bind#4  value=0
```

```
Bind#5
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=04d7345c  bln=24  avl=03  flg=05
 value=8192
Bind#6
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=04d73438  bln=24  avl=06  flg=05
 value=2147483645
Bind#7
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=04d73414  bln=24  avl=02  flg=05
 value=8
Bind#8
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=04d733f0  bln=24  avl=03  flg=05
 value=128
Bind#9
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=04d733cc  bln=24  avl=01  flg=05
 value=0
Bind#10
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=04d733a8  bln=24  avl=02  flg=05
 value=1
Bind#11
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
 kxsbbbfp=04d7337c  bln=24  avl=02  flg=05
 value=8
Bind#12~ Bind#15  value=0
Bind#16
 oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
 oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
```

```
  kxsbbbfp=04d7284c  bln=24  avl=04  flg=05
  value=907939
 Bind#17
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=04d72828  bln=24  avl=02  flg=05
  value=1
 Bind#18
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=04d72804  bln=24  avl=02  flg=05
  value=8
 Bind#19
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=04d727e0  bln=24  avl=02  flg=05
  value=1
 Bind#20
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=04d727bc  bln=24  avl=02  flg=05
  value=33
 Bind#21~ Bind#23  value=0
 Bind#24
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=04d73518  bln=22  avl=02  flg=05
  value=8
 Frames pfr 04D713C0 siz=7188 efr 04D71338 siz=7172
 Cursor frame dump
  enxt: 4.0x00000404  enxt: 3.0x000009b8  enxt: 2.0x000001f4  enxt: 1.0x00000c54
  pnxt: 2.0x00000004  pnxt: 1.0x0000000c
 kxscphp  04D70064 siz=2016 inu=0 nps=624
 kxscbhp  04D66510 siz=2016 inu=0 nps=1432
----------------------------------------
Cursor#1(04D60C24) state=BOUND curiob=04D74B38
 curflg=4c fl2=400 par=00000000 ses=34352EC4
 child cursor: 3
 sqltxt(04AAFC04)=drop tablespace enmo
```

```
  hash=00000000000000000000000000000000
  parent=04AA2870 maxchild=01 plk=04AAFD24 ppn=y
cursor instantiation=04D74B38 used=1281340689
 child#0(04AA2264) pcs=04AA2684
  clk=04AA2334 ci=04AA2010 pn=04AA23A4 ctx=04AA1814
 kgsccflg=0 llk[04D74B3C,04D74B3C] idx=0
 xscflg=c0802276 fl2=c000400 fl3=2202008 fl4=100
 Frames pfr 04D74AFC siz=240 efr 04D74AAC siz=232
 Cursor frame dump
  enxt: 1.0x000000e8
  pnxt: 1.0x00000008
 kxscphp  04D66020 siz=1000 inu=0 nps=140
 kxscefhp 04D704C0 siz=4072 inu=0 nps=244
----------------------------------------
```

而如果此时我们尝试去重新创建 ENMO 表空间，则会出现 25013 / 25015 错误，客户的情况完全得以重演。这里的 25015 错误跟随着几个参数，其中 8 指表空间号，新创建的 ENMO 表空间被重新赋予了 8 号表空间，第二个参数 6 是指表空间记录号，5 是指数据文件号：

```
SQL> create tablespace enmo datafile size 10M;
create tablespace enmo datafile size 10M
*
第 1 行出现错误:
ORA-00603: ORACLE server session terminated by fatal error
ORA-00600: internal error code, arguments: [25015], [8], [6], [5], [], [], [], []
ORA-00600: internal error code, arguments: [25015], [8], [6], [5], [], [], [], []
ORA-00600: internal error code, arguments: [25013], [0], [8], [ENMO], [ENMO], [5], [6], []
进程 ID: 0
会话 ID: 159 序列号: 14
```

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。
```

至此，还有一件事情没有重现，就是表空间和文件名的对应问题，客户现场出现了表空间和裸设备名称对应的混乱。

2.5.5 一致性损坏的显示错误

继续进行我们的测试，以下过程我们连续创建了两个表空间，并通过手工删除一个表空间 ENMO 的 ts$信息，通过命令 DROP EYGLE 表空间：

```
Mon Aug 09 18:30:11 2010
create tablespace eygle
Mon Aug 09 18:30:13 2010
Completed: create tablespace eygle
Mon Aug 09 18:30:29 2010
create tablespace enmo
Mon Aug 09 18:30:31 2010
Completed: create tablespace enmo
Mon Aug 09 18:30:55 2010
drop tablespace eygle
Mon Aug 09 18:30:55 2010
ORA-959 signalled during: drop tablespace eygle...
Mon Aug 09 18:31:04 2010
drop tablespace enmo
Deleted Oracle managed file D:\ORACLE\ORADATA\ENMO\ENMO\DATAFILE\O1_MF_ENMO_65ZP65JJ_.DBF
Completed: drop tablespace enmo
```

在接下来重新创建 EYGLE 表空间的过程中，系统再次出现错误：

```
Mon Aug 09 18:39:47 2010
create tablespace eygle datafile size 10M
Mon Aug 09 18:39:48 2010
```

```
Errors in file d:\oracle\admin\enmo\udump\enmo_ora_4512.trc:
ORA-00600: 内部错误代码，参数: [25013]，[0]，[6]，[EYGLE]，[EYGLE]，[2]，[2]，[]

Mon Aug 09 18:39:48 2010
ORA-600 signalled during: create tablespace eygle datafile size 10M...
Mon Aug 09 18:39:48 2010
Errors in file d:\oracle\admin\enmo\udump\enmo_ora_4512.trc:
ORA-00600: 内部错误代码，参数: [25015]，[6]，[2]，[2]，[]，[]，[]，[]
ORA-00600: 内部错误代码，参数: [25013]，[0]，[6]，[EYGLE]，[EYGLE]，[2]，[2]，[]
```

数据库再次启动之后，提示存在一个活动事务：

```
ORACLE Instance enmo (pid = 8) - Error 600 encountered while recovering transaction (14, 40).
```

此时当尝试创建任意表空间是都出现错误：

```
Mon Aug 09 18:41:31 2010
create tablespace eyglee datafile size 10M
Mon Aug 09 18:41:31 2010
Errors in file d:\oracle\admin\enmo\udump\enmo_ora_2116.trc:
ORA-00600: 内部错误代码，参数: [25013]，[0]，[7]，[EYGLEE]，[EYGLe]，[6]，[7]，[]

Mon Aug 09 18:41:32 2010
ORA-600 signalled during: create tablespace eyglee datafile size 10M...
Mon Aug 09 18:41:32 2010
Errors in file d:\oracle\admin\enmo\udump\enmo_ora_2116.trc:
ORA-00600: 内部错误代码，参数: [25015]，[7]，[7]，[6]，[]，[]，[]，[]
ORA-00600: 内部错误代码，参数: [25013]，[0]，[7]，[EYGLEE]，[EYGLe]，[6]，[7]，[]
```

并进而出现另外一个未决事务：

```
ORACLE Instance enmo (pid = 8) - Error 600 encountered while recovering transaction (20, 6).
```

此时数据库存在两个死事务，无法回滚：

```
SQL> select distinct KTUXECFL,count(*) from x$ktuxe group by KTUXECFL;

KTUXECFL                  COUNT(*)
------------------------ ----------
DEAD                              2
NONE                            576
```

设置两个回滚段参数，阻止这两个事务的回滚，再次强制恢复数据库：

```
Mon Aug 09 18:47:48 2010
ALTER SYSTEM SET _offline_rollback_segments='_SYSSMU14$','_SYSSMU20$' SCOPE=SPFILE;
Mon Aug 09 18:48:03 2010
ALTER SYSTEM SET _corrupted_rollback_segments='_SYSSMU20$','_SYSSMU14$' SCOPE=SPFILE;
ALTER SYSTEM SET _allow_resetlogs_corruption=TRUE SCOPE=SPFILE;
```

注意，以上的 25013 和 25015 错误事实上是在执行 CREATE TABLESPACE 时，内部的递归 SQL 出现了问题（读者可以尝试研究一下 CREATE TABLESPACE 的内部操作，据此理解失败事务的停滞之处），并且未能成功回滚，由于我们阻止了回滚，实际上字典中的部分相关信息已经存在了。

屏蔽回滚段后启动数据库，此时再次尝试创建一个 TEST 表空间，可以预期又出现如下一系列错误：

```
SQL> create tablespace test datafile size 10M;
create tablespace test datafile size 10M
*
第 1 行出现错误:
ORA-00603: ORACLE server session terminated by fatal error
ORA-00600: internal error code, arguments: [25015], [8], [8], [7], [], [], [],[]
ORA-00600: internal error code, arguments: [25015], [8], [8], [7], [], [], [],[]
ORA-00600: internal error code, arguments: [25013], [0], [8], [TEST], [ENMO],[7], [8], []
进程 ID: 0
会话 ID: 159 序列号: 3
```

告警日志记录如下信息：

```
Mon Aug 09 18:49:10 2010
create tablespace test datafile size 10M
Mon Aug 09 18:49:10 2010
Errors in file d:\oracle\admin\enmo\udump\enmo_ora_4512.trc:
ORA-00600: 内部错误代码, 参数: [25013], [0], [8], [TEST], [ENMO], [7], [8], []

Mon Aug 09 18:49:11 2010
ORA-600 signalled during: create tablespace test datafile size 10M...
Mon Aug 09 18:49:11 2010
Errors in file d:\oracle\admin\enmo\udump\enmo_ora_4512.trc:
ORA-00600: 内部错误代码, 参数: [25015], [8], [8], [7], [], [], [], []
ORA-00600: 内部错误代码, 参数: [25013], [0], [8], [TEST], [ENMO], [7], [8], []

Mon Aug 09 18:49:11 2010
Errors in file d:\oracle\admin\enmo\udump\enmo_ora_4512.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], [8], [7], [], [], [], []
ORA-00600: 内部错误代码, 参数: [25013], [0], [8], [TEST], [ENMO], [7], [8], []
```

启动数据库，一个新的活动事务出现：

```
Mon Aug 09 18:50:36 2010
ORACLE Instance enmo (pid = 8) - Error 600 encountered while recovering transaction (11, 46).
```

```
SQL> select distinct KTUXECFL,count(*) from x$ktuxe group by KTUXECFL;

KTUXECFL                  COUNT(*)
------------------------ ----------
DEAD                              1
NONE                            481
```

类似的进行处理，再次设置隐含参数，将相应回滚段屏蔽：

```
Mon Aug 09 18:51:03 2010
ALTER   SYSTEM   SET   _offline_rollback_segments='_SYSSMU14$','_SYSSMU20$','_SYSSMU11$'
SCOPE=SPFILE;
Mon Aug 09 18:51:16 2010
ALTER   SYSTEM   SET   _corrupted_rollback_segments='_SYSSMU20$','_SYSSMU14$','_SYSSMU11$'
SCOPE=SPFILE;
```

数据库可以再次成功启动，而此时，数据库内部的信息变成了如下，TEST 表空间对应了 ENMO 的数据文件，EYGLEE 对应了 EYGLE 的文件名。

这就是因为事务的完整性被破坏的缘故，与客户最后的故障现象完全相符合：

```
SQL> select file_name,tablespace_name from dba_data_files;
FILE_NAME                                                                TABLESPACE_NAME
------------------------------------------------------------------------ ---------------
D:\ORACLE\ORADATA\ENMO\USERS01.DBF                                       USERS
D:\ORACLE\ORADATA\ENMO\SYSAUX01.DBF                                      SYSAUX
D:\ORACLE\ORADATA\ENMO\ENMO\DATAFILE\O1_MF_EYGLE_65ZP5MPW_.DBF           EYGLE
D:\ORACLE\ORADATA\ENMO\SYSTEM01.DBF                                      SYSTEM
D:\ORACLE\ORADATA\ENMO\ENMO\DATAFILE\O1_MF_UNDOTBS2_65ZN4G7P_.DBF        UNDOTBS2
D:\ORACLE\ORADATA\ENMO\ENMO\DATAFILE\O1_MF_ENMO_65ZPGB95_.DBF            TEST
D:\ORACLE\ORADATA\ENMO\ENMO\DATAFILE\O1_MF_EYGLE_65ZOT6TT_.DBF           EYGLEE
D:\ORACLE\ORADATA\ENMO\ENMO\DATAFILE\O1_MF_UNDOTBS1_65ZQGJCH_.DBF        UNDOTBS1
```

虽然出了一系列的异常，但是这个表空间文件除了命名上的问题，完全可以正常被使用，以下是简单的测试尝试：

```
SQL> create user test identified by test default tablespace test;

用户已创建。

SQL> grant connect,resource,dba to test;
```

```
授权成功。

SQL> connect test/test
已连接。
SQL> create table test as select * from dba_users;

表已创建。
```

那么总结一下，实际上在遇到了 25013 / 25015 错误后，如果细致冷静，认真判断，是可以较快速、简单的解决问题的，最忌讳的是进行大量错误的尝试，这可能会使情况变得更糟糕。

2.5.6 实际的处理过程

在实际解决这个问题时，我们首先通过隐含参数打开数据库，然后尝试处理异常的两个文件。

当我了解到主要是 2 个数据文件：184，187 号文件存在问题，文件上又没有业务数据，我选择了直接从 file$中 DELETE 了这两条记录。删除文件之后再次启动数据库时，一件恐怖的事情发生了，Oracle 经过字典检查，认为自 184 号文件之后的所有文件都不存在，从控制文件中将这些文件依次删除了。

以下是一段日志的摘要：

```
Sun Jun 27 23:04:48 2010
SMON: enabling cache recovery
Sun Jun 27 23:04:48 2010
Dictionary check beginning
File #185 in the controlfile not found in data dictionary.
Removing file from controlfile.
data file 185: '/dev/vg02/rlvdata225_8G'
File #186 in the controlfile not found in data dictionary.
Removing file from controlfile.
data file 186: '/dev/vg02/rlvdata226_8G'
File #188 in the controlfile not found in data dictionary.
Removing file from controlfile.
data file 188: '/dev/vg02/rlvdata228_8G'
File #189 in the controlfile not found in data dictionary.
Removing file from controlfile.
data file 189: '/dev/vg02/rlvdata229_8G'
File #190 in the controlfile not found in data dictionary.
Removing file from controlfile.
data file 190: '/dev/vg02/rlvdata231_8G'
。。。。。。。。。。。。
```

```
File #232 in the controlfile not found in data dictionary.
Removing file from controlfile.
data file 232: '/dev/vg02/rlvdata087_16G'
Dictionary check complete
```

注意这里的句读断句：File #186 in the controlfile，not found in data dictionary.

也就是说，文件 186 在控制文件中有，但是在数据字典中没有，所以数据库认为 186 号文件非法，直接从控制文件中删除了这个文件的信息（Removing file from controlfile） 。

可是文件 186 等在数据库中都是存在的？为什么数据库会认为不存在呢？

通过前面的分析测试，我们可以知道，Oracle 为了保证数据文件号不被浪费，在数据库删除文件时，并不会删除 file$中的数据，这样可以保证文件号可以被最大限度的重用，文件号对于数据库来说，是个稀缺资源。

由于对于文件号来说，不存在跳号的问题，所以数据库在启动时只要从 1 号文件递归开始校验即可，当遇到一个空的文件号时，数据库认为一致性遭到破坏，最简单的数据字典的查询显示都可能出现异常，所以 Oracle 放弃后面的所有文件，选择清除控制文件中的内容，这就是 Oracle 数据库表空间和数据文件的维护机制。

了解了这个过程之后，我将删除的两条记录还原回 file$中去，更改状态为 INVALID（即将 status$更新为 2），重建控制文件，数据库就彻底恢复了正常。

2.5.7 字典检查何时发生？

数据字典的检查在正常情况下并不会发生，通常在执行了 Resetlogs 操作后，Oracle 数据库必须执行字典检查以确认一致性等内容。

跟踪数据库启动时进行字典检查的文件，可以清晰地看到递归文件号的字典检查过程：

```
====================
PARSING IN CURSOR #4 len=122 dep=1 uid=0 oct=3 lid=0 tim=222660383007 hv=1330125001
ad='55bcbed8'
select blocks,NVL(ts#,-1),status$,NVL(relfile#,0),maxextend,inc,
 crscnwrp,crscnbas,NVL(spare1,0) from file$ where file#=:1  ----〉字典表读取
END OF STMT
PARSE #4:c=0,e=611,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=4,tim=222660383004
BINDS #4:
kkscoacd
 Bind#0
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=9fffffffbf368c90  bln=22  avl=02  flg=05
  value=1                                          ----〉传入绑定变量文件号 1
EXEC #4:c=0,e=760,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=4,tim=222660383882
WAIT #4: nam='db file sequential read' ela= 898 file#=1 block#=258 blocks=1 obj#=-1
tim=222660384821
```

```
FETCH #4:c=0,e=972,p=1,cr=2,cu=0,mis=0,r=1,dep=1,og=4,tim=222660384880
=====================
PARSING IN CURSOR #2 len=122 dep=1 uid=0 oct=3 lid=0 tim=222660402928 hv=1330125001
ad='55bcbed8'
select blocks,NVL(ts#,-1),status$,NVL(relfile#,0),maxextend,inc,
 crscnwrp,crscnbas,NVL(spare1,0) from file$ where file#=:1
END OF STMT
PARSE #2:c=0,e=22,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=222660402926
BINDS #2:
kkscoacd
 Bind#0
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=9fffffffbf368c90  bln=22  avl=02  flg=05
  value=2                                          ----〉传入绑定变量文件号 2
EXEC #2:c=0,e=94,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=222660403124
FETCH #2:c=0,e=11,p=0,cr=2,cu=0,mis=0,r=1,dep=1,og=4,tim=222660403155
STAT #2 id=1 cnt=1 pid=0 pos=1 obj=17 op='TABLE ACCESS BY INDEX ROWID FILE$ (cr=2 pr=0 pw=0
time=10 us)'
STAT #2 id=2 cnt=1 pid=1 pos=1 obj=41 op='INDEX UNIQUE SCAN I_FILE1 (cr=1 pr=0 pw=0 time=5
us)'
=====================
PARSING IN CURSOR #5 len=122 dep=1 uid=0 oct=3 lid=0 tim=222660418094 hv=1330125001
ad='55bcbed8'
select blocks,NVL(ts#,-1),status$,NVL(relfile#,0),maxextend,inc,
 crscnwrp,crscnbas,NVL(spare1,0) from file$ where file#=:1
END OF STMT
PARSE #5:c=0,e=13,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=222660418092
BINDS #5:
kkscoacd
 Bind#0
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
  kxsbbbfp=9fffffffbf368c90  bln=22  avl=02  flg=05
  value=3                                          ----〉传入绑定变量文件号 3
EXEC #5:c=0,e=92,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=222660418286
FETCH #5:c=0,e=11,p=0,cr=2,cu=0,mis=0,r=1,dep=1,og=4,tim=222660418317
STAT #5 id=1 cnt=1 pid=0 pos=1 obj=17 op='TABLE ACCESS BY INDEX ROWID FILE$ (cr=2 pr=0 pw=0
time=8 us)'
```

```
STAT #5 id=2 cnt=1 pid=1 pos=1 obj=41 op='INDEX UNIQUE SCAN I_FILE1 (cr=1 pr=0 pw=0 time=5
us)'
====================
```

这就是字典检查时发生在内部的递归 SQL 操作。

参考文献：

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

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

ITPUB 论坛 http://www.itpub.net/199099.html

IxOra 网站 http://www.ixora.com.au/tips/creation/bsq.htm