， 事 务 回 滚 
． 事 筹 复 （ 实 例 复 ， 利 用 回 滚 来 恢 复 未 提 交 的 数 据 ） 
． 读 一 致 性 （ 构 造 CR ） 

undo 可以存放事务（undo 段头slot）

实例恢复redo中有已提交和未提交的数据，未提交数据需要undo来回滚

CR=>select

flashback

undo 碰到的问题：

1 undo表空间非常大？

2 ora-01555

3 undo segment header 等待

4 ora-600 ==>4193,4194,4xxx

v$rollname 
dba_rollback_segs; 
dba undo extents 
v$undostat 
v$rollstat 

1 表空间与回滚段的关系

回滚段初始10个，后期随着事务量增加而不断增加！！！

oracle算法为在足够空间的话，每个事务一个回滚段

SQL> select \* from v$rollname;   --找到所有的回滚段！

       USN NAME                               CON\_ID

---------- ------------------------------ ----------

         0 SYSTEM                                  0           ---在system表空间中，ddl的回滚

         1 \_SYSSMU1\_1607784644$                    0

         2 \_SYSSMU2\_3197760948$                    0

         3 \_SYSSMU3\_2165878102$                    0

         4 \_SYSSMU4\_910366693$                     0

         5 \_SYSSMU5\_3484600714$                    0

         6 \_SYSSMU6\_3822221795$                    0

         7 \_SYSSMU7\_233526786$                     0

         8 \_SYSSMU8\_153572397$                     0

         9 \_SYSSMU9\_889291946$                     0

        10 \_SYSSMU10\_3901898544$                   0

        11 \_SYSSMU11\_1962061695$                   0

12 rows selected.

update test\_transaction set salary=6000 where id=1;

SQL> select addr,xidusn,xidslot,xidsqn from v$transaction;

ADDR                 XIDUSN    XIDSLOT     XIDSQN

---------------- ---------- ---------- ----------

00007FFFA48573C8          8         28       2006

2 回滚段

回滚段的段头：xid，tx table  （slot）：事务信息

回滚块：uba， undo数据修改前的值

回滚段本质也是表，是由区（extents）组成：在段的使用过程中，区有各种状态（dba\_undo\_extents）

Active：活动状态，事务没有提交，绝不允许被覆盖重用

Unexpired：非过期状态，事务提交，但在undo\_retention保护范围内，一般不会优先被覆盖使用，

                  又因为undo\_retention默认非强制，所以在undo表空间不够用的情况下会被覆盖重用

Expired：过期状态，提交且在undo\_retention保护范围外，优先覆盖使用

Free： 没有使用的状态，优先使用

SQL> select owner,EXTENT\_ID,BYTES,STATUS from dba\_undo\_extents where segment\_name='\_SYSSMU7\_4222772309$';

OWN  EXTENT\_ID      BYTES STATUS

--- ---------- ---------- ---------

SYS          0      65536 EXPIRED

SYS          1      65536 UNEXPIRED

SYS          2    1048576 ACTIVE

rollback！

区的状态由参数undo\_retention确定！！也和表空间大小有很大关系，

SQL> show parameter undo

NAME                     TYPE     VALUE

------------------------------------ ----------- ------------------------------

undo\_management              string     AUTO

undo\_retention                 integer     900

undo\_tablespace              string     UNDOTBS1

======================================

Data segment 
Data 
Rollback segment 
TX table Slot 
undo 

参考事务知识！！

事物开始

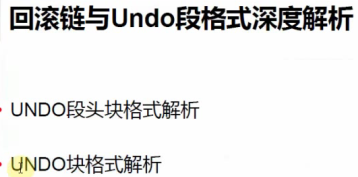
先找到一个回滚段头（rollback segment  TX table slot），把事务存储在回滚段的断头事务表里的slot。把undo record记录到undo块

然后去修改数据块的数据。在数据库块段头ITL槽存储有xid和uba，和事务关联起来！

最后把修改放到data row

ROLLBACK使用XID，CR使用UBA

=====================================



UNDO段头块深入解析 <http://www.itpub.net/thread-1841245-1-1.html>

配置实验环境，创建一个很小的undo表空间

create undo tablespace undotbs2 datafile '/u02/app/[oracle](http://www.itpub.net/tree/index_1/)/undotbs02.dbf' size 192k;

alter system set undo\_tablespace=undotbs2;

查看有一个回滚段

select \* from v$rollname;

       USN NAME

---------- ------------------------------

     0 SYSTEM

    11 \_SYSSMU11\_391121102$

select extent\_id,file\_id,block\_id,blocks,bytes from dba\_extents where segment\_name='\_SYSSMU11\_391121102$';

EXTENT\_ID    FILE\_ID    BLOCK\_ID     BLOCKS     BYTES

---------- ---------- ---------- ---------- ----------

     0        6           8      8     65536

     1        6          16      8     65536

select header\_file,header\_block from dba\_segments where segment\_name='\_SYSSMU11\_391121102$';

HEADER\_FILE HEADER\_BLOCK

----------- ------------

      6           8

产生一个事务

update test\_transaction set salary=6000 where id=1;

找到事务所在的回滚段

select addr,xidusn,xidslot,xidsqn from v$transaction;

dump回滚段的段头

alter session set tracefile\_identifier='undo\_header';

alter system dump undo header '\_SYSSMU11\_391121102$';

select \* from v$diag\_info;

分析UDNO段头块的日志/u02/app/oracle/diag/rdbms/orcl/orcl/trace/orcl\_ora\_14770\_undo\_header.trc

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Undo Segment:  \_SYSSMU11\_391121102$ (11)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  Extent Control Header

  -----------------------------------------------------------------

  Extent Header:: spare1: 0      spare2: 0      #extents: 2      #blocks: 15

                  last map  0x00000000  #maps: 0      offset: 4080

      Highwater::  0x01800011  ext#: 1      blk#: 1      ext size: 8

  #blocks in seg. hdr's freelists: 0

  #blocks below: 0

  mapblk  0x00000000  offset: 1

                   Unlocked

     Map Header:: next  0x00000000  #extents: 2    obj#: 0      flag: 0x40000000

  Extent Map

  -----------------------------------------------------------------

   0x01800009  length: 7

   0x01800010  length: 8

Retention Table

  -----------------------------------------------------------

Extent Number:0  Commit Time: 1500685651

Extent Number:1  Commit Time: 0

#extents: 2  表示17号UNDO段有两个区

#blocks: 15   表示17号UNDO回滚段两个区中有15个UNDO BLOCK可用。（为什么不是16个UNDO BLOCK块呢，去掉一个UNDO段头块）

ext#: 1       表示这个事务发生在第2个区（从0开始）

blk#: 1       表示这个事务发生在第1个区的第1个块上

ext size:8     表示1个区上有8个UNDO BLOCK可用

对比上面查询的dba\_extents信息

 Extent Map

11号UNDO回滚段的区地图一共有两个区：   第一个区对应的是6号文件7个UNDO BLOCK，第二个区对应的是6号文件8个UNDO BLOCK

Retention Table

区的提交时间戳

  TRN CTL:: seq: 0x0002 chd: 0x0019 ctl: 0x0018 inc: 0x00000000 nfb: 0x0001

            mgc: 0xb000 xts: 0x0068 flg: 0x0001 opt: 2147483646 (0x7ffffffe)

            uba: 0x01800011.0002.04 scn: 0x0000.00000000

  uba: 0x01800011.0002.04 表示新事务的第一条UNDO记录（由三部分组成undo块的地址、UNDO块被重用的次数、在UNDO块的第几条记录）

      undo块的地址：        0x01800011 即6号文件的17号块

      UNDO块被重用的次数：  0002       即UNDO块被覆盖了2次

      在UNDO块的第几条记录  04         即在UNDO块的第4条 scn 表示17号UNDO段头块中最小的提交的SCN。实际上这个SCN就是事务表中最小的SCN所对应的事务槽上的SCN

Version: 0x01

  FREE BLOCK POOL::

    uba: 0x01800011.0002.06 ext: 0x1  spc: 0x1ca6

    uba: 0x00000000.0000.00 ext: 0x0  spc: 0x0

    uba: 0x00000000.0000.00 ext: 0x0  spc: 0x0

UNDO块的空闲池，当事务做了提交会把此事务所在的UNDO块加入空闲池中。

uba:  由三部分组成undo块的地址、UNDO块被重用的次数、在UNDO块的第几条记录，当undo块的地址为0说明UNDO块不是空闲的，即0x00000000

ext:  UNDO块是在哪个区（extent）

spc:  UNDO块中多少空闲空间，单位字节  从上面的UNDO空闲池中看，没有空闲的UNDO块。

  TRN TBL::

  index  state cflags  wrap#    uel         scn            dba            parent-xid    nub     stmt\_num    cmt

  ------------------------------------------------------------------------------------------------

   0x00    9    0x00  0x0003  0x0001  0x0000.00189857  0x0180000a  0x0000.000.00000000  0x00000001   0x00000000  1500685410

   ......

   0x06    9    0x00  0x0003  0x0007  0x0000.00189877  0x0180000c  0x0000.000.00000000  0x00000001   0x00000000  1500685470

   0x07    9    0x00  0x0003  0x0009  0x0000.00189879  0x0180000c  0x0000.000.00000000  0x00000001   0x00000000  1500685470

   0x08   10    0x80  0x0003  0x0000  0x0000.001897e3  0x0180000c  0x0000.000.00000000  0x00000001   0x00000000  0

   0x09    9    0x00  0x0003  0x000a  0x0000.00189897  0x0180000d  0x0000.000.00000000  0x00000001   0x00000000  1500685530

   ......

  0x0a    9    0x00  0x0003  0x000b  0x0000.0018989f  0x00000000  0x0000.000.00000000  0x00000000   0x00000000  1500685530

   ......

   0x0f    9    0x00  0x0003  0x0010  0x0000.001898d9  0x0180000f  0x0000.000.00000000  0x00000001   0x00000000  1500685590

   0x21    9    0x00  0x0002  0x0000  0x0000.00000000  0x00000000  0x0000.000.00000000  0x00000000   0x00000000  0

TRN TBL::（事务表）是UNDO段头块最重要的。我们一一来解释每个字段的意思：

index 表示事务表中槽号，只是一个序列而已，从0x00开始（中间0x0a-0x0f）到0x21结束,11g的版本有34个槽。

state 表示事务状态：9代表事务不活动，10代表事务正在活动，从这里我们看出16进制第0x08号槽上的事务正在活动。

wrap# 表示事务表上的事务槽被重用的次数，它是XID的一部分

uel   表示当前活动事务所在事务槽的下一个事务槽的指针（即如果又发生一个新的事务，此时就会用到UEL指向的事务槽上的index)

scn   表示务事启动、提交、回滚的SCN

dba   表示uba:第一部分的undo块地址，这个DBA是(rollback)回滚的起始点，也就是说是记录事务修改的最后一条记录所在UNDO块的地址

nub   表示当前事务所用到的UNDO块的个数。

cmt   表示最接近当前的提交时间戳，是从1970年1月1号零晨开始的（以秒为单位记录）。0表示事务正在活动。

========================================

UNDO块格式解析 <http://blog.csdn.net/guoyjoe/article/details/14088663>

=================================

DM 
INSERT 
DELETE 

dml产生undo数据量大小

insert into test\_undo values (1,'aaa');

commit

update test\_undo set name='bbb' where name='aaa';

commit;

delete from test\_undo where id=1;

commit;

事务1 insert   ==》生成的undo最少==》记录rowid，==》rollback的话就做delete where rowid=

事务2 updae  ==》生成的undo中间==》记录rowid，修改前的值，修改前的值对应的事务(即事务1)==》rollback 做update改回去

事务3 delete  ==》生成的undo最多==》记录rowid,存放删除的整行记录 ，这行记录对应的以前的事务（既事务2）==》rollback做insert

实验：找到transaction对应的undo

create table test\_undo (id int,name varchar2(22));

select userenv('sid') from dual;

USERENV('SID')

--------------

        61

insert into test\_undo values (1,'aaa');

让这个事务先不提交，方便我们找到这个事务

首先我得找到sql对应的transaction(v$transaction.addr = v$session.Taddr )

select userenv('sid') from dual;

select t.addr,s.taddr,s.sid from v$transaction t,v$session s where t.addr = s.Taddr and sid=56;

ADDR         TADDR             SID

---------------- ---------------- ----------

000000006CD06030 000000006CD06030      56

然后找到这个事务相关信息

SQL> select addr,xidusn,xidslot,xidsqn,ubafil,ubablk,ubasqn,ubarec,status,xid from v$transaction where addr='000000006CD06030';

ADDR             XIDUSN    XIDSLOT       XIDSQN     UBAFIL     UBABLK     UBASQN     UBAREC STATUS           XID

---------------- ---------- ---------- ---------- ---------- ---------- ---------- ---------- ---------------- ----------------

000000006CD06030      1        27         2334       3      22567        467        9 ACTIVE           01001B001E090000

那么我们可以看到这条语句的undo在UBAFIL 3号文件UBABLK 22567号块上

我们先刷新一下buffer cache，再dump

alter system flush buffer\_cache;

dump undo block info（转储undo数据块信息）

alter session set tracefile\_identifier='undo\_dump';

alter system dump datafile 3 block 22567;

select \* from v$diag\_info

######################################################

理解UBA

UBA=dba.sequence#.record#

file number,block number=v$transaction  UBAFIL UBABLK

uba seqence#(重用次数) =v$transaction UBASQN

uba record#（这个undo块的第几条undo record,也就是第几条scn）=v$transaction UBAREC

理解XID

xid: 0x0001.01b.0000091e

transaction undo segment number=v$trasaction  XIDUSN

transaction table slot(槽号)= v$transaction XIDSLOT

scn wrap（被覆盖几次） =v$transaction XIDSQN

序号（同一个事务可能具有多个 SCN ,实际上每一个 DML 操作都有一个SCN）

理解rowid

对象号，文件号，块号，行号

SQL> select rowid,dbms\_rowid.ROWID\_OBJECT('AAATy4AAGAAAADWAAA') objectid,dbms\_rowid.ROWID\_RELATIVE\_FNO('AAATy4AAGAAAADWAAA') rfile,dbms\_rowid.ROWID\_BLOCK\_NUMBER('AAATy4AAGAAAADWAAA') block,dbms\_rowid.ROWID\_ROW\_NUMBER('AAATy4AAGAAAADWAAA') row# from test.test\_undo;

ROWID             OBJECTID       RFILE      BLOCK      ROW#

------------------ ---------- ---------- ---------- ----------

AAATy4AAGAAAADWAAA    81080           6    214         0

######################################################

trace file内容：

\*\*\* ACTION NAME:() 2015-12-04 19:23:34.686

Start dump data blocks tsn: 2 file#:3 minblk 22567 maxblk 22567

Block dump from cache:

Dump of buffer cache at level 4 for tsn=2 rdba=12605479

BH (0x613df5a8) file#: 3 rdba: 0x00c05827 (3/22567) class: 18 ba: 0x610c8000

UNDO BLK:

xid: 0x0001.01b.0000091e  seq: 0x1d3 cnt: 0x9   irb: 0x9   icl: 0x0   flg: 0x0000

 Rec Offset      Rec Offset      Rec Offset      Rec Offset      Rec Offset

---------------------------------------------------------------------------

0x01 0x1f08     0x02 0x1e28     0x03 0x1d48     0x04 0x1c68     0x05 0x1b88

0x06 0x1a8c     0x07 0x1a24     0x08 0x19cc     0x09 0x1944

然后我们找到第9条undo record

txn start scn: scn: 0x0000.004ce24e logon user: 0

prev brb: 20971993 prev bcl: 0

xid：事务编号

seq：被覆盖了几次=UBASQN

cnt：undo block上一共多少条record

irb： 做rollback的的起始点（rollback正好是倒着来的）

icl： rollback结束点

rec： record 0x01 ==我们 insert 的记录

 0x09 0x1944  就是指第9条undo record和他的偏移量

\*-----------------------------

\* Rec #0x9  slt: 0x1b  objn: 81080(0x00013cb8)  objd: 81080  tblspc: 4(0x00000004)

\*       Layer:  11 (Row)   opc: 1   rci 0x00

Undo type:  Regular undo    Begin trans    Last buffer split:  No

Temp Object:  No

Tablespace Undo:  No

rdba: 0x00000000Ext idx: 0

flg2: 0

\*-----------------------------

uba: 0x00c05827.01d3.06 ctl max scn: 0x0000.002ad2ae prv tx scn: 0x0000.002ad2c4

txn start scn: scn: 0x0000.002ad74c logon user: 96

 prev brb: 12605476 prev bcl: 0

KDO undo record:

KTB Redo

op: 0x03  ver: 0x01

compat bit: 4 (post-11) padding: 1

op: Z

KDO Op code: DRP row dependencies Disabled

  xtype: XA flags: 0x00000000  bdba: 0x018000d6  hdba: 0x018000d2

itli: 1  ispac: 0  maxfr: 4858

tabn: 0 slot: 0(0x0)

End dump data blocks tsn: 2 file#: 3 minblk 22567 maxblk 22567

objd： object id  对象好

bdba: 0x018000d6：文件号块号

slot: 0(0x0) ：第一行

以上3个就是存储的rowid

hdba： 数据块所在区的第一个块

KDO Op code: DRP  ：rollback做delete row piece

思考： update和delete操作，dump看一下！！

可以直接dump数据块的信息来看。dump数据块》uba》dump undo块》cnt irb找到undo record

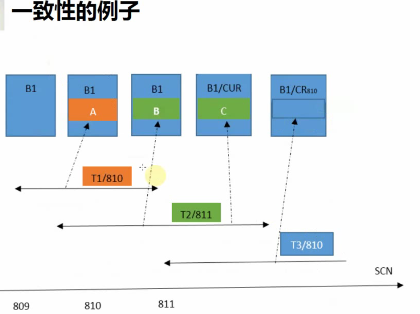
深 入 理 解 Oracle 的 回 滚 操 作 
． 回 滚 的 始 点 
． 回 滚 链 格 式 的 解 析 
Transaction 
KTLJXC 
undo b 《 
undo block 

确定rollback的起点==》事务表==》uba==>undo record

先找到undo segment header block的transaction table 》通过transaction table 中dba等信息找到事务对应的undo block》实现rollback

Provides the read committed isolation level for 
transactions 
• Two possible levels for read consistency: 
— Statement: The data returned by a query is consistent 
with respect to the start of the query (snap_scN = 
current scN at the time the query started its execute 
phase). 
— Transaction: Data returned by a query is consistent 
with respect to the beginning of the transaction 
(snap_SCN = Current SCN at the time the 
transaction started). 

事务ACID 的 C！！



8:09： t3开始select

8:10：t1 修改了数据为A

8:11：t2 修改了数据为B

8:12：t2 修改了数据为C

8:13：t3 读到了B1数据块，应该读出来的是那个数据呢？

读的数据是scn<8:10的数据 既A之前的那个数据

III III 
III III III III 
III III III 
nı III III 

情况1》A

情况2》A

情况3》A

情况4》A

select 跨事务读取数据就会有一致性读！！一致性读需要考虑以下条件

1 未commit的数据不会读

2 select scn《块的scn

实验： 一致性读！！！

session1 test

update test\_undo set name='kkk' where name='aaa';

session2 sys

grant select on v\_$mystat to test;

grant select on v\_$statname to test;

select a.name,b.value from v$statname a,v$mystat b where a.statistic#=b.statistic# and a.name like '%CR%';

select \* from test.test\_undo;

select a.name,b.value from v$statname a,v$mystat b where a.statistic#=b.statistic# and a.name like '%CR%';

此语句也可以统计本会话中的其他统计信息

我们会发现CR blocks created在增加！！

NAME      VALUE

---------------------------------------------------------------- ----------

CR blocks created 10

Read Consistency:Key Terms 
• Current block: The most recent version of a block 
• CR block: A consistent version of a block that 
contains only committed changes 
• snap SCN: The SCN used to construct a snapshot 
of a block (this is the consistency point of 
reference) 
• Snap_UBA: The undo block address (UBA) in the 
snapshot block 
• Env SCN: The SCN at the current time 
• Env UBA.• The UBA of the current transaction 

Read Consistency Algorithm 
k usa 

我们只简单看一下过程

1 逻辑读buffer cache

           ==》如果找到数据就读出来==》判断是否copy current enough（就是块的scn是否小于select scn）==>

否的就构造CR块（物理读undo表空间）==》CR块是否可用==》可用就直接3使用，不可用就4clone

          ==》没有找到数据库就进行2物理读disk

2 物理读disk

3 使用

4 clone数据块==》是否有行锁==》块清除

CLONE 
Transaction Internals: Read Consistency 
Data row n 
Clone the 
block 
Roll back 
uncomrnitted 
row 
changes 
D aa row old 

读数据块==》发现是脏块没有提交 data now(new)==>clone==》

此时原来的块变成CR块，clone块变成crrent block

crrent block如果scn>select scn 则做rollback，读取前映像

Block Cleanout 
Active Tx 
Upper bound 
SCN 
snap SCN 
ITL I 
Block i 
Cache layer 
TX Qable 
scanning 
Undo segment 
header 
Cache layer 
2 
Extent control 
@Finding 
Extent map 
if Tx is active or 
Undo Tx control 
committed 
ree blocks ool 
Transactions tabl 
ndex wra 
I 
I 
Cleanout 
info area 
Application 
Redo 
Log 

块清除，是清除行锁

发生物理读的时候，oracle会根据块上的rtl槽，找到uba，确认是否commit。如果确认了，则去掉行锁。



*XEORA-OI 555CEi* 
• The rollback information itself has been overwritten 
• The transaction slot in the undo segment header (in 
the transaction table) has been overwritten, and the 
undo to roll it back sufficiently is also overwritten 



• Increase size of rollback segments 
Reduce the number Of commit operations 
• Avoid using full table scans or long running queries 
Avoid fetch across commit operations 
With nested loops, avoid having the outer SELECT 
command visit the same block at different times 
• Add additional rollback segments. 
• Introduce dummy Sorts, so that the block is 
revisited from temporary segments 
• Easiest solution: convert to AUM, enlarge the undo 
tablespace as necessary 

为 什 么 我 的 UNDO 不 够 了 一 Undo 段 内 部 
重 用 规 则 
· 1 、 首 先 检 貢 本 回 滚 段 当 前 区 是 否 有 空 间 块 ， 如 果 有 直 接 使 用 · 
． 2 、 如 果 没 有 ， 事 务 伸 展 到 下 一 个 区 ， 这 被 称 为 一 次 Wrap ， v$rollstat 中 的 WRAPS 
列 加 1 · 如 果 本 回 滚 段 有 多 个 事 务 ， 第 一 个 伸 展 到 下 一 个 区 的 事 务 会 引 起 WRAPS 
列 加 1 ， 其 他 事 务 冉 伸 展 至 一 下 区 ， 此 列 不 冉 增 加 。 
． 3 、 如 果 本 回 滚 段 中 没 有 可 用 的 区 ， 从 UN DO 表 空 间 中 请 求 区 。 这 被 称 为 一 次 
Extend 讠 己 Äv$rollstat 的 EXTENTS 列 。 
． 4 、 UNDO 轰 空 间 中 没 有 Free 区 ， 从 其 他 回 滚 段 Stea 《 过 期 的 区 · Steal 的 单 位 一定 
是 以 区 为 单 位 。 无 论 Stea | 是 否 成 功 ， VWNDOSTAT 的 EXPSTEALCNT 列 都 会 加 1 
此 列 统 计 的 是 次 数 。 EXPBLKRELCNT 列 是 成 功 Steal 的 块数。 
· 5 、 如 果 stea 起 期 区 不 成 功 ， 试 图 扩 展 数 据 文 件 。 
． 6 、 如 果 无 法 扩 展 文 件 ， 在 本 回 藻 段 中 重 用 未 过 期 的 区 ， 
V$IJN DOSTAT. LJNXPBLKREUCNT 列 增 加 （ 增 加 值 是 区 中 块 的 个 数 ） 
． 7 、 如 本 回 滚 段 无 过 期 的 块 ， 则 Stea 《 其 他 回 滚 段 中 未 过 期 的 区 · 每 Steal— 次 ， 无 
论 是 否 成 功 ， 都 会 记 入 VSUNDOSTAT.UNXPSTEALCNT 列 。 Steal 的 块数 记 
． 入 UNXPBLKREUCNT 列 。 
． 8 、 如 还 不 行 ， 报 ORA ． 30036 错 误 · 

extentea 
select a.waits, a.shrinks, a.wraps, a.status, 
b.segment_name, b.tablespace_name 
from v$rollstat a, b 
where a.usn = b.segment_id 
order by b.segment_name; 
v$rollstat.waits ? 
v$rollstat.shrinks 
v$rollstat.wraps 

UNDO 表 空 间 的 
小 亻 
以 及 创 
估 算 UNPO 的 大 小 
． 2 、 创 建 UNDO 表 空 间 
， 3 、 切 换 UNDO 表 空 间 
， 4 、 监 控 UNDO 段 是 否 在 线 

desc v$undostat;

