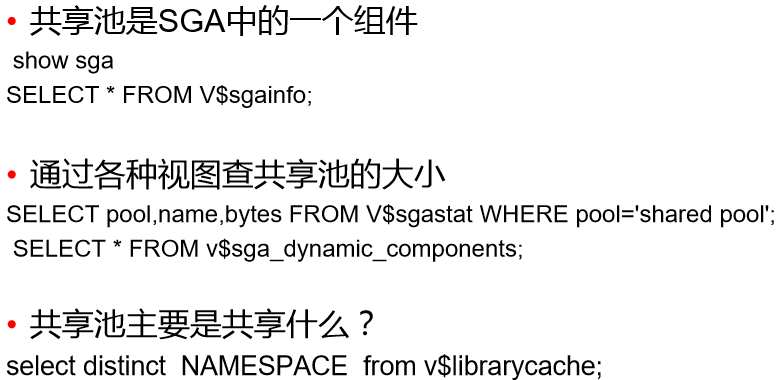
**》观察组件**



Fixed Size              2227376 bytes   --存放sga源数据

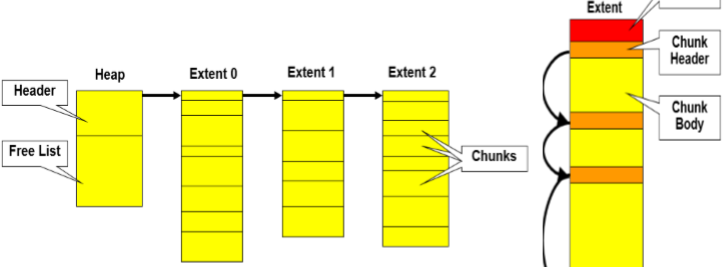
Variable Size            163578704 bytes  --各种pool的大小

v$librarvcache(库缓存) v$rowcache（数据字典缓存）都是描述的内存结构！！！

**》组成结构**



空间组成



总堆 heap由extent组成，extent由大小不一的chunk组成！chunk是最小的内存单元

通过视图x$ksmsp 可以查询总共多少个chunk

SQL> select count(\*) from x$ksmsp;

  COUNT(\*)

----------

     33222

空间使用



大小不一的chunk按照大小相同挂载带不同的chunk list上，如大小为32byte的chunk都挂载带第一个chunk lists上

当我需要存储sql时，把sql做hash 计算出使用大小，然后找到不同的bucket来使用chunk

dump shared pool 转储共享池

alter session set events 'immediate trace name heapdump level 2';

KGH Latch Directory Information

ldir state: 2  last allocated slot: 97

Slot [  1] Latch: 0x6000a018  Index: 1  Flags:  3  State: 2  next:  (nil)    --latch信息

Slot [  2] Latch: 0x6e24fc20  Index: 1  Flags:  3  State: 2  next:  (nil)

......

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

HEAP DUMP heap name="sga heap(1,0)"  desc=0x60053950                       ---sga heap 总堆

extent sz=0xfe0 alt=248 het=32767 rec=9 flg=-126 opc=0                        --extent sz区大小

parent=(nil) owner=(nil) nex=(nil) xsz=0x400000 heap=(nil)

fl2=0x20, nex=(nil), dsxvers=1, dsxflg=0x0

dsx first ext=0x6f000000

latch set 1 of 1

durations enabled for this heap

reserved granules for root 3 (granule size 4194304)

EXTENT 0 addr=0x6a800000

  Chunk        06a800058 sz=       48  R-freeable  "reserved stoppe"                       --区下面的chunk

  Chunk        06a800088 sz=   212808  R-free      "               "

  Chunk        06a833fd0 sz=       48  R-freeable  "reserved stoppe"

  Chunk        06a834000 sz=  3981312    perm      "perm           "  alo=2239064

......

FREE LISTS:

Bucket 0 size=32

  Chunk        06f000078 sz=        0    kghdsx

Bucket 1 size=40

Bucket 2 size=48

Bucket 3 size=56

  Chunk        06dffffc8 sz=       56    free      "               "

Bucket 4 size=64

Bucket 5 size=72

Bucket 6 size=80

Bucket 7 size=88

Bucket 8 size=96

  Chunk        06c3fffa0 sz=       96    free      "               "

  Chunk        06a83fd0 sz=       96    free      "               "

  Chunk        06a83a00 sz=       96    free      "               "

Bucket 9 size=104

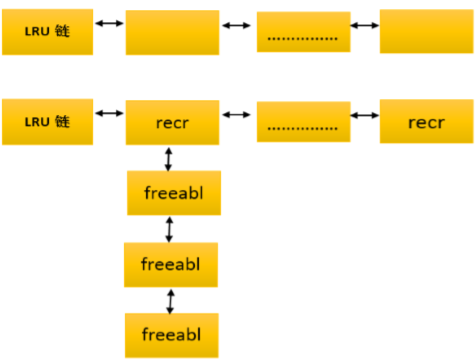
Bucket 10 size=112

Bucket 11 size=120

以上可以看出bucket下的chunk大小相同，但地址不同，很难吧他们合并。如果一个sql需要97这么大，就找bucket=96的

shared pool LRU链表，管理内存的使用





oracle 把recr的chunk挂载到lru链表上，每个recr（recreate）下又挂载freeabl，这样减少了lru长度，减少了搜索的时间。

oracle 先找free list上可用的，没有再找lru上的recr chunk，如果用到2个chunk大小，则使用第一个recr和下面第一个freeabl，剩下的两个freeabl则会被放到free list上！

dump的shared poo 文件中可以找到lru

UNPINNED RECREATABLE CHUNKS (lru first):

  Chunk        06ef71e98 sz=       56    recreate  "fixed allocatio"  latch=0x6001fb28     --挂载lru列表上的chunk都是recreate类型

  Chunk        06ed3a3c0 sz=       56    recreate  "fixed allocatio"  latch=0x6def6e90

  Chunk        06ed07250 sz=       56    recreate  "fixed allocatio"  latch=0x6def6e90

  Chunk        06ed03e30 sz=      528    recreate  "KGLHD          "  latch=(nil)

  Chunk        06ed033e0 sz=      528    recreate  "KGLHD          "  latch=(nil)

可以查看x$ksmsp 来看所有chunk类型

SQL> desc x$ksmsp;

Name                            Null?    Type

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

ADDR                                  RAW(8)                           --chunk在共享池中的地址

INDX                                  NUMBER

INST\_ID                             NUMBER                        --实例id

KSMCHIDX                             NUMBER

KSMCHDUR                             NUMBER

KSMCHCOM                             VARCHAR2(16)            --注释

KSMCHPTR                             RAW(8)

KSMCHSIZ                             NUMBER                     --chunk大小

KSMCHCLS                             VARCHAR2(8)           --chunk类型

KSMCHTYP                             NUMBER

KSMCHPAR                             RAW(8)                     -父游标heap0地址

SQL> select distinct KSMCHCLS from x$ksmsp;

KSMCHCLS

--------

recr               --lru上的  可重用的

freeabl      --可释放的

R-freea              --带R是保留池的 recover pool

perm           --free list上永久的不能被释放的

R-free

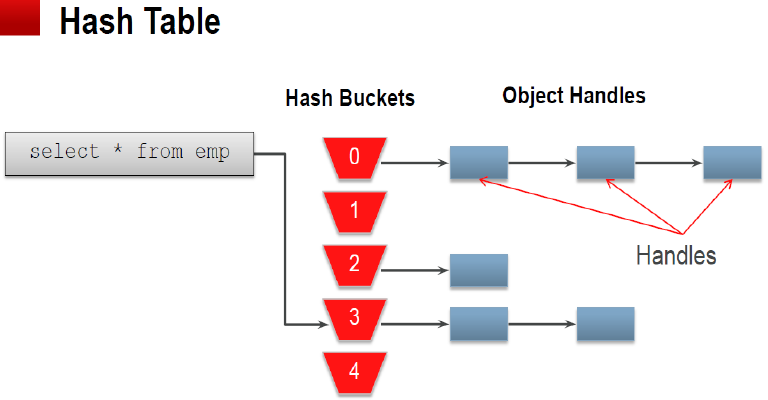
R-perm

free             --free  list上的 可使用

R-recr

》硬解析软解析过程

相关的术语



hash table 描述library cache的

如果执行一个sql，需要对sql做解析 ，参考parse

parse

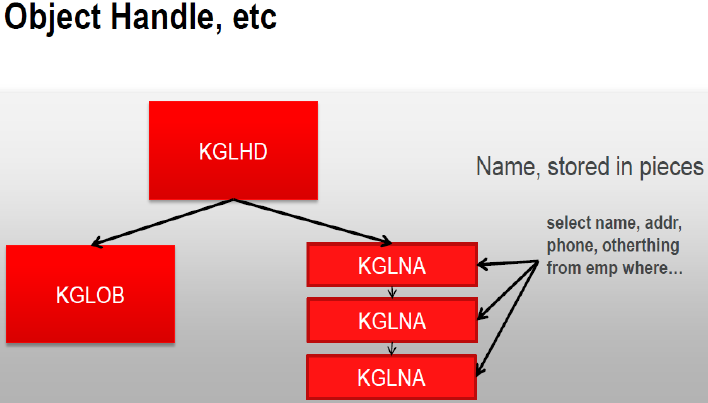
    语法：select ..from..where...  语法正确

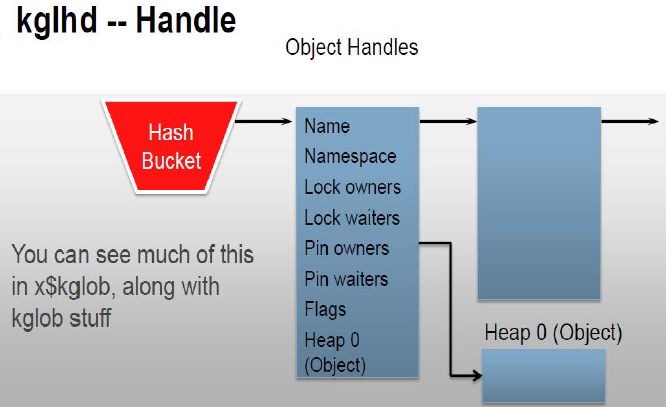
    语义：对象的数据字典是否存在

    权限：用户权限

    转化字符为ascii 然后根据内部hash函数做计算，计算出sql放在在哪个hash bucket上

    在hash bucket上挂载的是chunk=object handles（对象句柄）也就是保存sql语句父游标





kglhd（kernnel general library handle）内核通用库句柄=object handle

kglob（kernel general library object）就是sql语句

kdlna：存储了sql语句文本

可以查询视图x$kglob 来查看object handle，上图可以看出object handles里保存的信息。比如说 name，namespace。。。heap0地址

select salary from hr.employees;    --这个语句有没有被缓存呢？

SQL> desc x$kglob;

Name                            Null?    Type

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

ADDR                                  RAW(8)

INDX                                  NUMBER

INST\_ID                             NUMBER

KGLHDADR                             RAW(8)    --handle的句柄   子游标

KGLHDPAR                             RAW(8)     --handle的父句柄     父游标

KGLHDCLT                             NUMBER

KGLNAOWN                             VARCHAR2(64)

KGLNAOBJ                             VARCHAR2(1000)    --存放sql语句文本

通过x$kglob查看缓存的sql

SQL> select KGLHDADR,KGLHDPAR,KGLNAOBJ from x$kglob where KGLNAOBJ like 'select salary from hr.employees%';

no rows selected

SQL> select salary from hr.employees where employee\_id=198;

    SALARY

----------

      2600

SQL> select KGLHDADR,KGLHDPAR,KGLNAOBJ from x$kglob where KGLNAOBJ like 'select salary from hr.employees%';

KGLHDADR     KGLHDPAR

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

KGLNAOBJ

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

0000000067681480 0000000069A26448

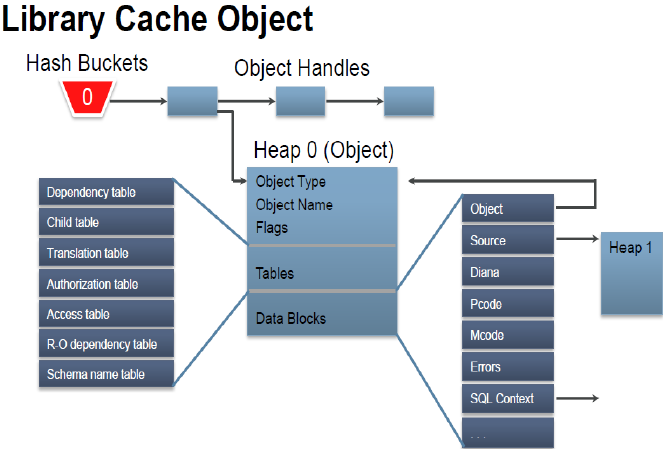
select salary from hr.employees where employee\_id=198

0000000069A26448 0000000069A26448

select salary from hr.employees where employee\_id=198

缓存了两个sql？？？？？？，父游标和子游标

KGLHDADR=KGLHDPAR 父游标等于子游标就是他自身，也就是父游标上的sql



library cache 结构，结合此结构我们再来看一下解析

sql hash分析后

==》hash buckets 0

==》查找object handle=父游标句柄。 如果没有找到父游标句柄就需要硬解析，到freelist上找chunk然后挂载到bucket上，如果找到，则做以下的软解析

==》找到了object handle=父游标（记录了sql文本及多个子游标地址）

       ==》  heap0=子游标句柄，不同的执行计划

什么情况下同一个sql有好几个执行计划？

如果是不同的用户执行同一个没有绑定变量的sql 会发现同一个父游标下游多个子游标

如果使用绑定变量呢，会只有一个zi游标吗？？？实验一下！

我们还可以用一下方式来探究library cache

dump library cache 转储库缓存

alter session set events 'immediate trace name library\_cache level 4';

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

Library Cache Dump

Bucket: #=103349 Mutex=0x6b9cd478(0, 28, 0, 6)

  LibraryHandle:  Address=0x69a26448 Hash=a73193b5 LockMode=0 PinMode=0 LoadLockMode=0 Status=VALD

    ObjectName:  Name=select salary from hr.employees where employee\_id=198

      FullHashValue=3bac7f183b212714fb8dccbfa73193b5 Namespace=SQL AREA(00) Type=CURSOR(00) Identifier=2805044149 OwnerIdn=0

......

LibraryObject:  Address=0x691391a8 HeapMask=0000-0001-0001-0000 Flags=EXS[0000] Flags2=[0000] PublicFlags=[0000]

      ChildTable:  size='16'

        Child:  id='0' Table=0x6913a058 Reference=0x69139aa8 Handle=0x67681480

    NamespaceDump:

      Parent Cursor:  sql\_id=gr3fcrymm34xp parent=0x69139248 maxchild=1 plk=n ppn=n

针对sql，这里我们有一个视图v$sql描述

SQL> select sql\_id,hash\_value,address,child\_address,sql\_text from v$sql where sql\_text like '%where employee\_id=198%';

SQL\_ID           HASH\_VALUE ADDRESS       CHILD\_ADDRESS

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

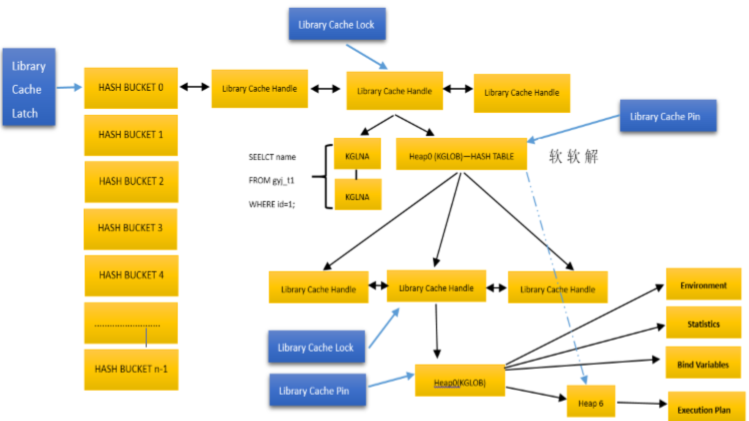
SQL\_TEXT

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

gr3fcrymm34xp 2805044149 0000000069A26448 0000000067681480

select salary from hr.employees where employee\_id=198

**》shared pool latch与libaraycache cache**



看上图，类似cbc latch 和buffer pin lock，对应的锁闩保护对应的对象

在bucket上有library cache latch

在父游标句柄上有library cache lock，

子游标句柄上有library cache pin

也就是说硬解析分配shared pool的时候需要加 library cache latch，软解析时需要加library cache lock/pin

11g  library cache lock/pin变为library cache: mutex X

硬解析时    latch: shared pool

软解析时    library cache: mutex X

begin

  for i in 1..1000000

   loop

     execute immediate 'insert into test\_parse values ('||i||')';

   end loop;

   commit;

end;

/

select event,count(1) from v$session where wait\_class#<>6 group by event;

select sid,event,total\_waits,TIME\_WAITED from v$session\_event where wait\_class<>'idle' and event like '%library%';

减少硬解析

1 修改sql语句，来避免shared pool latch

2 不修改sql的时候可以修改参数（尽量不用）

cursor\_sharing                    string     EXACT   ==》force

减少软解析，

实现软软解析，来避免library cache: mutex X

我们可以通过增加这些操作来观察等待事件！！



11g library cache pin转为cursor pin mutex

软软解析 ：pga缓存相关的sql语句和执行计划句柄，指向heap6的执行计划，所以不需要通过hash bucket 父游标 子游标这些解析过程。

但是软软解析还是对应一个等待事件 cursor pin X。。

如果要解决这个问题，需要用到无解析！oracle自己无法解决，需要应用程序，把执行计划缓存在应用程

实验无解析避免软软解析遇到的cursor pin等待事件

》ORA-04031

$ oerr ora 4031

04031, 00000, "unable to allocate %s bytes of shared memory (\"%s\",\"%s\",\"%s\",\"%s\")"

// \*Cause:  More shared memory is needed than was allocated in the shared

//          pool or Streams pool.

// \*Action: If the shared pool is out of memory, either use the

//          DBMS\_SHARED\_POOL package to pin large packages,

//          reduce your use of shared memory, or increase the amount of

//          available shared memory by increasing the value of the

//          initialization parameters SHARED\_POOL\_RESERVED\_SIZE and

//          SHARED\_POOL\_SIZE.

//          If the large pool is out of memory, increase the initialization

//          parameter LARGE\_POOL\_SIZE.

//          If the error is issued from an Oracle Streams or XStream process,

//          increase the initialization parameter STREAMS\_POOL\_SIZE or increase

//          the capture or apply parameter MAX\_SGA\_SIZE.

原因：

当尝试在共享池分配大块的连续内存失败(很多时候是由于碎片过多,而并非真是内存不足)时,Oracle首先清除共享池中当前没使用的所有对象,使空闲内存块合并.如果仍然没有足够大的单块内存可以满足需要,就会产生ORA-04031错误.

大量SQL没有使用绑定变量，产生过多的硬解析

Shared pool设的太小

open\_cursors 设的过大

在共享池找不到一块连续的空间使用

比如一个chunk是4096

一个sql是5000

找到一个chunk，还需要4byte大小，这个时候，Oracle不是找4byte，而是需要找一个同等大小的chunk，即4096byte

1、刷共享池

  alter system flush shared\_pool;

不过这个办法只是暂时的，治标不治本。

2、保留池的设置

参数shared\_pool\_reserved\_size,shared\_pool\_reserved\_min\_alloc

select REQUEST\_MISSES from v$shared\_pool\_reserved;

3、缓存大对象

   select \* from v$db\_object\_cache where sharable\_mem > 50000

         and (type = 'PACKAGE' or type='PACKAGE BODY' or type = 'FUNCTION' or type='PROCEDURE')

                   and kept = 'NO';

   执行dbms\_shared\_pool.keep('对象名');

4、共享SQL:应用绑定变量或修改参数cursor\_sharing

      a.使用绑定变量

      b.cursor\_sharing参数

         如何找出不能共享cursor的sql

         在v$sql查找执行次数较小的sql语句，观察这些sql语句是否是经常执行的。

         select SQL\_FULLTEXT from v$sql where EXECUTIONS=1 and sql\_text like '%from test.t1%';

         select SQL\_FULLTEXT from v$sql where EXECUTIONS=1 order by sql\_text;

         alter system set cursor\_sharing=SIMILAR  | FORCE;

5、增加shared pool空间

     select sum(bytes/1024/1024) from v$sgastat where pool='shared pool';

     select COMPONENT,CURRENT\_SIZE from V$SGA\_DYNAMIC\_COMPONENTS;

     show parameter sga\_target

     show parameter sga\_max\_size

     alter system set shared\_pool\_size=300M scope=both;

参数open\_cursors、session\_cached\_cursors的设置是否可以解决ORA-04031？为什么？

<http://blog.csdn.net/guoyjoe/article/details/8567087>