Oracle 数据文件结构
（节选）

作者　Tomcoding

2017 年 6 月 27 日

前言

Oracle 数据库在物理上是由一系列文件组成的，我们考虑数据是存储在文件上的这种方式来研究它的结构，其他存储在裸设备和 ASM 文件系统上的数据也有相同的块（block）结构格式。

Oracle 的逻辑存储结构是由表空间（tablespace），段（segment），区（extent），块（block）组成的。表空间是由一个或多个数据文件（datafile）组成；段在表空间之下，是为每一个存储单元（表，表分区，cluster 等）分配的空间，段也可以跨数据文件；段下面是区，段由多个区组成，区是由一组连续的块组成的，必须在一个文件内；块是数据存储的最小单位，是由一个或多个操作系统块组成的。

我们这篇文档是为了导出 Oracle 数据库的表数据准备基础知识，所以只分析表的数据块结构，同时还要分析表是怎样构成的，通过什么方式找到一个表所有的数据块，还要分析数据字典从哪儿来的。有了这些知识，就可以在数据库不启动的状态下，把一个或多个表的数据都导出来，Oracle 的 DUL 工具就是这样工作的。当我们完成这些工作后，也尝试着写一个简单的 DUL 工具。

基础知识

在开始分析数据文件结构之前，还要把一些概念搞清楚，这样才能更好的理解文档。网络上有很多介绍 Oracle 概念的文章，如果后面遇到什么不理解的概念，而我们在这儿也没有涉及，上网查阅一下就能解决了。

文件号

一个 Oracle 数据文件会有两个文件编号，一个叫绝对文件号，另一个叫相对文件号，这里的相对是参照表空间来说的。绝对文件号在整个数据库中是唯一的，按照数据文件添加到数据库中的顺序编号。相对文件号，在一个表空间内是唯一的，最大是 1023。相对文件号是 Oracle 为了突破原来版本数据库最多只能有 1023 个数据文件的限制而引入的，当数据库中数据文件个数没有超过 1023

时，绝对文件号和相对文件号是一致的，当创建大文件表空间（bigfile tablespace）时，其中的相对文件号永远是 1024。在 Oracle 的数据字典中，经常会看到绝对文件号和相对文件号，搞清楚它们的区别，可以更好的理解文件结构。

数据字典

数据字典是用来解释存储数据的规则，比如表的结构，对象的名称，字段的名称和顺序等等。Oracle 的数据字典是存储在一些系统表中的，这些叫做基础表（Base Table），同时为了方便查询，Oracle 在这些基础表上建立了多个视图，比如经常用到的 DBA_OBJECTS，DBA_TABLES 等。这些基础表全部存储在 SYSTEM 表空间中，所以我们如果想得到数据库的数据字典，就要从系统表空间的数据文件中解析这些数据。

RDBA

RDBA 是 Relative Data Block Address 的首字母缩写，是由 10 位的相对文件号和 22 位的块号组成的。比如有一个 RDBA=0x01400f87，转换成二进制看就是 0000 0001 0100 0000 0000 1111 1000 0111，那么前 10 位是 0000 0001 01 等于 0x05，后 22 位是 00 0000 0000 1111 1000 0111 等于 0x0f87=3975，表明块地址在相对文件号是 5 的文件，第 3975 块上。在 Oracle 数据块的 DUMP 文件中可以看到多种 DBA，最常看到的是 rdba，还有 bdba，它们的表示方式和算法都是一样的，只是根据不同的含义叫了不同的名字。

我们看到 RDBA 中用 10 位来表示相对文件号，如果到最大是 0x3ff=1023，所以一个表空间内最多有 1023 个数据文件，看看上面介绍**文件号**的内容，现在就能明白限制在哪儿了。同样 22 位表示块的个数，如果块大小是 8K 的话，最多就是 0x3fffff*8192/1024/1024=32767M=32G，一般默认数据块大小为 8K 的时候，数据文件最大只能有 32G 大小。

大文件表空间

一般来说，一个表空间中会有多个数据文件，可以分散 I/O，但如果数据量

大的时候，由于单个文件的大小限制，所以就需要很多文件来满足存储要求，创建和管理数据文件比较麻烦，表空间的管理也变复杂，为应对这种情况，Oracle 从 10g 开始引入了大文件表空间。这个概念很好理解，就是把上面看到的 RDBA 中的相对文件号的 10 位也用作块个数，这样就用 32 位表示块号，最大能到 4G 个块，如果采用默认为 8K 的块大小，一个数据文件就能有 32T 字节，如果采用最大块大小 32K，则一个数据文件就能有 128T 字节。由于在 RDBA 中已经不能表达相对文件号，所以大文件表空间中只能有一个数据文件，固定相对文件号为 1024，其实由于进位的关系，相当于 0 了，所以在组合 RDBA 时，就没有相对文件号了。

限制条件

Oracle 数据库的版本很多，不同版本的文件格式也有细微的差别，我们把研究的版本限定在目前最常用的 11g，具体的版本号是 11.2.0.1。其他版本在有差异的地方会提到，没有提到的如果遇到版本格式不一致的，可以用下面的方法研究得到。

研究方法

数据文件的块从 0 开始编号，第 0 块和第 1 块的格式与其他块不同。第 0 块是系统的基本信息，第 1 块是文件头信息，可以用下面的语句 ALTER SESSION SET EVENTS ’IMMEDIATE TRACE NAME FILE_HDRS LEVEL 10’;把文件头信息 dump 到 trace 文件中，然后与文件的二进制格式对照研究。其他的块根据类型不同，格式也不相同，可以用 ALTER SYSTEM DUMP DATAFILE fno BLOCK blk;把块信息 dump 到 trace 文件中，与这个块的二进制格式对照研究，其中 fno 是绝对文件号，blk 是块号，这儿的 fno 和 blk 从 DBA_SEGMENTS 中查到，SEGMENT_NAME 就是对象的名称。遇到不确定的信息项可以写段程序，把特定项修改成好辨认的值，然后用上面的方法对照研究，不过要记得把原来的值恢复，否则可能造成数据库下次不能启动。

另一种方法就是通过 bbed 来查看特定块的结构，与上面的方法对照，一起配合分析块的结构。

结构相同的部分

cache layer

Oracle 数据文件的每一个块头都有一个相同的结构叫做 cache layer,这是一个固定大小的结构，总共 20（0x14）个字节，通过 bbed 看到这个结构名称是 kcbh，各个字段的名称和长度如下：

```
struct kcbh, 20 bytes
ub1 type_kcbh;   /* Block type * /
ub1 frmt_kcbh;
ub1 spare1_kcbh;
ub1 spare2_kcbh;
ub4 rdba_kcbh;   /* relative DBA /
ub4 bas_kcbh;    /* base of SCN */
ub2 wrp_kcbh;    /* wrap of SCN */
ub1 seq_kcbh;    /* sequence # of changes at same scn */
ub1 flg_kcbh;
ub2 chkval_kcbh;
ub2 spare3_kcbh;
```

为了看清字段位置，下面的格式标明了它们的偏移（offset）。

```
0x0000-0x0000  block type
0x0001-0x0001  format
0x0002-0x0002  spare1_kcbh
0x0003-0x0003  spare2_kcbh
0x0004-0x0007  rdba
0x0008-0x000B  SCN base
0x000C-0x000D  SCN wrap
0x000E-0x000E  seqence number
0x000F-0x000F  flag
0x0010-0x0011  check value
0x0012-0x0013  spare3_kcbh
```

下面解释一下各个字段的含义。

type_kcbh	块的类型，我们看到最多的就是 0x06 事务数据块，其他的参考下面的列表
frmt_kcbh	版本格式，8i 之前是 0x01,8i 之后是 0x02。10g 之后前半个字节用来表示块的大小，2k: 0x62 4k: 0x82 8k: 0xA2 16k: 0xC2
spare1_kcbh	备用字段，兼容之前的版本，不再使用，始终为 0
spare2_kcbh	备用字段，兼容之前的版本，不再使用，始终为 0
rdba_kcbh	相对数据块地址，本块的 DBA，参考前面基础知识中 RDBA 的介绍
bas_kcbh	系统改变号（System Change Number）的基础部分。这个块变化时的 SCN。
wrp_kcbh	SCN 的 wrap 部分。SCN 由 wrap+base 组成
seq_kcbh	同一个 SCN 多次改变块内容时，每改变一次 sequence 加 1
flg_kcbh	标志位，定义如下： flag:Flag(as defined in kcbh.h) #define KCBHFNEW 0x01 /* new block - zeroed data area */ #define KCBHFDLC 0x02 /* Delayed Logging Change advance SCN/seq */ #define KCBHFCKV 0x04 /* ChecK Value saved-block xor's to zero */ #define KCBHFTMP 0x08 /* Temporary block */
chkval_kcbh	检查值，这个块中，除去这两个字节，从开头以两个字节为一个整数，与后面的值异或操作得到的值，用于检查一个块的完整性。
spare3_kcbh	备用字段

块类型列表：

0x01 - KTU UNDO HEADER (undo segment header block)
0x02 - KTU UNDO BLOCK (undo data block)
0x03 - KTT SAVE UNDO HEADER
0x04 - KTT SAVE UNDO BLOCK
0x05 - DATA SEGMENT HEADER
0x06 - trans data
0x07 - Unknown
0x08 - Unknown
0x09 - Unknown

```
0x0A - DATA SEGMENT FREE LIST BLOCK
0x0B - Unknown = data file header (block 1)
0x0C - DATA SEGMENT HEADER WITH FREE LIST BLOCKS
0x0D - Compatibility segment
0x0E - KTU UNDO HEADER W/UNLIMITED EXTENTS
0x0F - KTT SAVE UNDO HEADER W/UNLIMITED EXTENTS
0x10 - DATA SEGMENT HEADER - UNLIMITED
0x11 - DATA SEGMENT HEADER WITH FREE LIST BLKS - UNLIMITED
0x12 - EXTENT MAP BLOCK
0x13 - Unknown = rman file header block (block 1)
0x14 - Unknown = rman file directory block (block 2)
0x15 - Unknown = control file header block (block 1)
0x16 - 22 DATA SEGMENT FREE LIST BLOCK WITH FREE BLOCK COUNT
0x17 - BITMAPPED DATA SEGMENT HEADER
0x18 - BITMAPPED DATA SEGMENT FREELIST
0x19 - BITMAP INDEX BLOCK
0x1A - BITMAP BLOCK
0x1B - LOB BLOCK
0x1C - KTU BITMAP UNDO HEADER - LIMITED EXTENTS
0x1D - KTFB Bitmapped File Space Header
0x1E - KTFB Bitmapped File Space Bitmap
0x1F - TEMP INDEX BLOCK
0x20 - FIRST LEVEL BITMAP BLOCK
0x21 - SECOND LEVEL BITMAP BLOCK
0x22 - THIRD LEVEL BITMAP BLOCK
0x23 - PAGETABLE SEGMENT HEADER (BMB for ASSM)
0x24 - PAGETABLE EXTENT MAP BLOCK
0x25 - EXTENT MAP BLOCK OF SYSTEM MANAGED UNDO SEGMENT
0x26 - KTU SMU HEADER BLOCK
```

```
0x27 - Unknown
0x28 - PAGETABLE MANAGED LOB BLOCK
0x29 - Unknown
0x2A - Unknown
0x2B - Unknown
0x2C - Unknown
0x2D - Unknown
0x2E - Unknown
0x2F - Unknown
```

块尾

在一个 dump 出来的块信息中，总会看到一项叫做 tail。一个真实的信息：

```
Block dump from disk:
buffer tsn: 6 rdba: 0x01400f87 (5/3975)
scn: 0x0000.00313140 seq: 0x01 flg: 0x06 tail: 0x31400601
frmt: 0x02 chkval: 0x2014 type: 0x06=trans data
```

块尾的信息有四个字节，也是为了校验块的完整性的。观察一下 dump 出的同一行上其他信息，能看到块尾的组成了吗？开始两个字节是 SCN base 的低位 2 字节，后面一个字节是标志 flg，再后一个字节是序号 seq。

BLOCK 0

每个数据文件的第 0 块包含一些基本信息，内容不多。这部分信息在 Oracle 10g 之前和之后是不一样的。在 10g 之前，第 0 块没有加上面提到的 cache layer 信息，所以信息从偏移 0x04 开始，前面四个字节是填充信息。在 10g 和 10g 之后的版本，偏移就从 cache layer 之后（0x14）开始了。主要有三部分信息：

```
ub4  block0_size;
ub4  blocks_in_file;
ub1  platform_id[4];
```

block0_size	第 0 块的大小，可能与其他块的大小不同，block1 中的块大小才是整

	个文件中使用的块大小
blocks_in_file	数据文件中包含块的个数，不包含 block0，这个参数乘以 block1 中块的大小，再加上 block0 的大小就是整个数据文件的大小
platform_id	Oracle 服务器平台的标识，在 10g 之前，UNIX 平台是 5A5B5C5D 这四个数字，在 Windows 平台是 6A6B6C6D。在 10g 和 10g 之后是 7A7B7C7D，不再区分是 UNIX 还是 Windows 平台。这个标识还是用来区分平台的大小字节序的，如果是 0x7A7B7C7D 顺序，就是大字节序（big endian），如果是 0x7D7C7B7A，就是小字节序（little endian）。

BLOCK 1

每个数据文件的第 1 块中包含了一些数据库相关的重要信息。可以用前面提到的 ALTER SESSION SET EVENTS 'IMMEDIATE TRACE NAME FILE_HDRS LEVEL 10'; 把文件头信息 dump 到跟踪文件中，通过与 block 1 的二进制对照研究其中的内容。如果精确一些，可以用 bbed 查看文件头信息，进入 bbed（怎样使用 bbed 可以在网络上找一下，有很多介绍，这里就省略了），输入 map 命令：

```
BBED> map
 File: /oracle11/product/11.2.0/dbs/road_ts01.dbf (5)
 Block: 1                                          Dba:0x01400001
------------------------------------------------------------
 Data File Header

 struct kcvfh, 860 bytes                    @0

 ub4 tailchk                                @8188
```

从这里看到 BLOCK 1 开始是一个 860 字节的结构，叫做 kcvfh，查看详细的结构信息可以用 print 命令：

```
BBED> p kcvfh
struct kcvfh, 860 bytes                    @0
   struct kcvfhbfh, 20 bytes               @0
      ub1 type_kcbh                        @0         0x0b
      ub1 frmt_kcbh                        @1         0xa2
      ub1 spare1_kcbh                      @2         0x00
```

```
      ub1 spare2_kcbh                         @3        0x00
      ub4 rdba_kcbh                           @4        0x01400001
      ub4 bas_kcbh                            @8        0x00000000
      ub2 wrp_kcbh                            @12       0x0000
      ub1 seq_kcbh                            @14       0x01
      ub1 flg_kcbh                            @15       0x04 (KCBHFCKV)
      ub2 chkval_kcbh                         @16       0x5c38
      ub2 spare3_kcbh                         @18       0x0000
   struct kcvfhhdr, 76 bytes                  @20
      ub4 kccfhswv                            @20       0x00000000
      ub4 kccfhcvn                            @24       0x0b200000
      ub4 kccfhdbi                            @28       0xff31537b
      text kccfhdbn[0]                        @32       O
      text kccfhdbn[1]                        @33       R
      text kccfhdbn[2]                        @34       A
      text kccfhdbn[3]                        @35       1
      text kccfhdbn[4]                        @36       1
      text kccfhdbn[5]                        @37       G
      text kccfhdbn[6]                        @38
      text kccfhdbn[7]                        @39
      ... ...
```

这样对照 block 1 的二进制数据，能更详细的看到数据格式。下面是 Oracle 11.2.0.1 的数据文件 block 1 的各个字段对应的位置，其他版本的（比如 10g）的格式有些不一样。

```
0x0000-0x0013  cache layer(每个块都有的结构，20 bytes)
0x0014-0x0017  db version
0x0018-0x001B  compatible version
0x001C-0x001F  db id
0x0020-0x0027  db name
0x0028-0x002B  control sequence
0x002C-0x002F  blocks count
0x0030-0x0033  block size
0x0034-0x0035  file number
0x0036-0x0037  file type
0x0038-0x003B  activation id
0x003C-0x003F  cks (kccfhcks)
0x0040-0x005F  tag
0x0060-0x0063  root dba
0x0064-0x0067  creation checkpoint SCN base
0x0068-0x0069  creation checkpoint SCN wrap
0x006A-0x006B  padding bytes[2]
0x006C-0x006F  creation checkpoint SCN time
0x0070-0x0073  reset logs count
```

0x0074-0x0077 reset logs SCN base
0x0078-0x0079 reset logs SCN wrap
0x007A-0x007B padding bytes[2]
0x007C-0x007F begin backup time
0x0080-0x0083 begin backup SCN base
0x0084-0x0085 begin backup SCN wrap
0x0086-0x0087 padding bytes[2]
0x0088-0x0089 begin backup thread#
0x008A-0x008B file status
0x008C-0x008F checkpoint count
0x0090-0x0093 recovered time
0x0094-0x0097 control count
0x0098-0x009B backup checkpoint SCN base
0x009C-0x009D backup checkpoint SCN wrap
0x009E-0x009F padding bytes[2]
0x00A0-0x00A3 backup checkpoint time
0x00A4-0x00A5 backup thread#
0x00A6-0x00A7 padding bytes[2]
0x00A8-0x00AB backup rba.sequence#
0x00AC-0x00AF backup rba.block#
0x00B0-0x00B1 backup rba.offset
0x00B2-0x00B3 padding bytes[2]
0x00B4-0x00BB enabled threads，重复
0x00BC-0x0137 unknown
0x0138-0x013B bhz (kcvfhbhz)
0x013C-0x013F xcd[0] (space_kcvmxcd)
0x0140-0x0143 xcd[1]
0x0144-0x0147 xcd[2]
0x0148-0x014B xcd[3]
0x014C-0x014F tablespace number
0x0150-0x0151 tablespace name length
0x0152-0x016F tablespace name 30 bytes
0x0170-0x0173 relative file number
0x0174-0x0177 recovery fuzzy SCN base
0x0178-0x0179 recovery fuzzy SCN wrap
0x017A-0x017B padding bytes[2]
0x017C-0x017F recovery fuzzy time
0x0180-0x0183 absolute fuzzy SCN base
0x0184-0x0185 absolute fuzzy SCN wrap
0x0186-0x0187 padding bytes[2]
0x0188-0x018B bbc (kcvfhbbc)
0x018C-0x018F ncb (kcvfhncb)
0x0190-0x0193 mcb (kcvfhmcb)
0x0194-0x0197 lcb (kcvfhlcb)

```
0x0198-0x019B  bcs (kcvfhbcs)
0x019C-0x019D  ofb (kcvfhofb)
0x019E-0x019F  nfb (kcvfhnfb)
0x01A0-0x01A3  prev reset logs count
0x01A4-0x01A7  prev reset logs SCN base
0x01A8-0x01A9  prev reset logs SCN wrap
0x01AA-0x01AB  padding bytes[2]
0x01AC-0x01AF  prfs (kcvfhprfs) SCN base
0x01B0-0x01B1  prfs (kcvfhprfs) SCN wrap
0x01B2-0x01BB  unknown
0x01BC-0x01BF  trt (kcvfhtrt)
0x01C0-0x01E3  unknown
0x01E4-0x01E7  checkpoint SCN base
0x01E8-0x01E9  checkpoint SCN wrap
0x01EA-0x01EB  padding bytes[2]
0x01EC-0x01EF  checkpoint time
0x01F0-0x01F1  checkpoint thread#
0x01F2-0x01F3  padding bytes[2]
0x01F4-0x01F7  checkpoint rba.sequence
0x01F8-0x01FB  checkpoint rba.block
0x01FC-0x01FD  checkpoint rba.offset
0x01FE-0x01FF  padding bytes[2]
0x0200-0x0207  etb[8] (kcvcpetb) enabled threads
```

BLOCK 1 中的信息比较多，好多涉及创建，备份，恢复，reset logs 等信息，这些只针对特定情况，可以先忽略它们。在这里只把与数据存储相关的字段列出来，说明它们的含义。

compatible version @（0x0018-0x001B）	兼容版本号
db id @（0x001C-0x001F）	数据库标识号，唯一，可以判断一个文件是否某个数据库
db name @（0x0020-0x0027）	数据库名称
blocks count @（0x002C-0x002F）	文件中块的个数，不包含 block 0
blocks size @（0x0030-0x0033）	文件中块的大小（字节数），也不包含 block 0
file number @（0x0034-0x0035）	绝对文件号
file type @（0x0036-0x0037）	文件类型，3 是数据文件，2 是日志文件
root dba	这个字段只在系统表空间的 1 号文件中有值，其他的都是 0。这个地

@（0x0060-0x0063）	址是 Oracle 用来找到数据字典加载项的，指向 bootstrap$表。
ts number @（0x014C-0x014F）	表空间编号，对应基表 ts$中的 ts#，标识这个文件属于哪个表空间
ts name length @（0x0150-0x0151）	表空间名称长度，表空间名称最多由 30 个字符组成
ts name @（0x0152-0x016F）	表空间名称
relative fno @（0x0170-0x0173）	相对文件号，在一个表空间内唯一

数据文件结构

数据文件是由数据块组成的，不同的块有不同的类型，不同类型的块结构也不相同。最终表的数据存储都在对应的段（segment）的区（extent）所在的数据块(block)上。这些块分为普通表的数据块和簇(cluster)数据块。段(segment)头的块，包含了段的 extents 信息，有了这些信息，我们就知道一个普通表的数据块都有哪些。

普通表的数据块结构

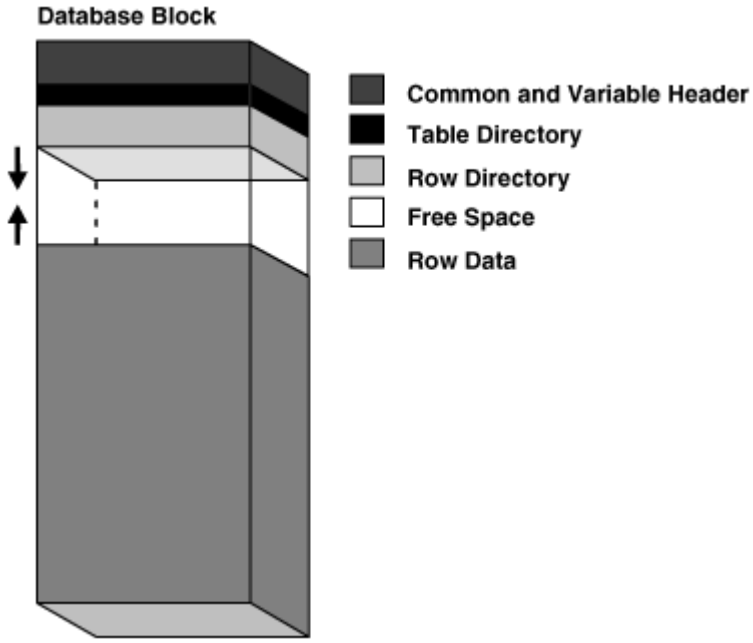
创建一个普通表，比如叫做 student，create table student (id number, name char(20), school varchar2(100));在表中插入几条数据，用下面的语句查到数据所在的块号：

```
select
 dbms_rowid.rowid_relative_fno(rowid) rfn,
 dbms_rowid.rowid_block_number(rowid) block#,
 dbms_rowid.rowid_row_number(rowid) row#,
 id,
 name
from student;
```

根据相对文件号和块号把块数据 dump 到 trace 文件中，然后结合本块的二

进制数据来研读其中的内容。

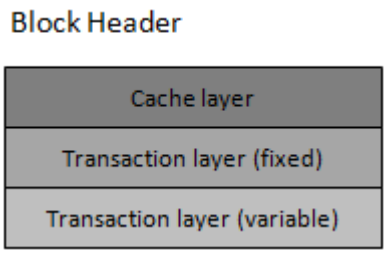
在开始研究数据块结构之前，先了解一下数据块都有哪些部分组成，Oracle 的官方文档中给出了一个大概的结构图，如下：



从上面的图中我们看到，数据块开始有一个头结构，这部分由通用的固定长度和可变长度的结构组成，接着是表的目录结构和表记录行的目录结构，后面是表的行数据，中间夹着的是空闲的空间，可以看到，Oracle 数据是从块尾往上增长的，表的记录行（row directory）是从上往下增长，当剩余空间达到一个比例（pctfree）时，就不能在这个块中插入数据了。

BLOCK HEADER

对数据块有了一个基本概念后，我们就来看看这些不同区域的详细信息。先来看看数据块头的内容。数据块头又分为几个部分，cache 层和事务层，事务层又分为固定大小的部分和可变的部分，如下图：



我们来逐一看一下都是些什么内容。

```
0x0000-0x0013  cache layer
0x0014-0x0014  type
0x0015-0x0017  padding bytes[3]
0x0018-0x001B  dataobj#
0x001C-0x001F  csc_base
0x0020-0x0021  csc_wrap
0x0022-0x0023  padding
0x0024-0x0025  itc
0x0026-0x0026  flg
0x0027-0x0027  fsl
0x0028-0x002B  fnx
```

这部分是 common header 和 transaction header 的固定部分，24 个字节。

cache layer	每个块头都有的部分，20 字节参考前面的解释
type	数据块的类型，01-data, 02-index
dataobj#	seg/obj，在段中的 object number
csc_base	Cleanout SCN base
csc_wrap	Cleanout SCN wrap
itc	itl 的个数，itl 结构的大小是固定的，itc 决定了事务层可变部分的大小。由于 itc<255, 所以取到值后应该屏蔽掉高位 itc&0xFF
flg	标志，ver 和 object id on block? 都通过这个标志得到
fsl	Free space lock
fnx	A pointer to the next block on the free list chain

下一部分是事务层的可变部分，也就是 ITL 部分，由上面的 itc 决定有几个 ITL slot，在 9i 之后，默认有 2 个 ITL。ITL(Interested Transaction List)是 Oracle 数据块内部的一个组成部分，用来记录该块所有发生的事务，一个 itl 可以看作是一个记录，也叫 itl 槽位（slot）。ITL 格式如下：

```
0x002C-0x002D  xid.usn
0x002E-0x002F  xid.slt
0x0030-0x0033  xid.sqn
0x0034-0x0037  uba.dba
```

0x0038-0x0039 uba.seq
0x003A-0x003A uba.rec
0x003B-0x003B padding
0x003C-0x003D flag
0x003E-0x003F fsc/scn wrap
0x0040-0x0043 scn base

下表是对 ITL 各个字段的解释，每个 ITL 占用 24 字节。

xid	事务 ID，标识一个事务
uba	Undo 段的块地址，每个事务开始都会分配一个 uba
flag	跟 lock 共享，flag 占半个字节，lock 占其余的半个字节。 flag 有 4 位 C--- = transaction has been committed and locks cleaned out -B-- = this undo record contains the undo for this ITL entry --U- = transaction committed (maybe long ago); SCN is an upper bound ---T = transaction was still active at block cleanout SCN lock 是本块在这个事务中行级锁的个数
fsc/scn	作为 SCN 时是事务的 commit SCN 作为 fsc 是这个事务释放出的字节数

后面是相同结构的 ITL 记录，由 itc 控制有几条。

从 bbed 中看到整个事务层部分，包括固定的和可变的，都叫做 ktbbh，ITL 还有单独的名字叫做 ktbbhitl。事务层固定部分占 24 字节，每个 ITL 占 24 字节。

…… 部分信息省略……

Data Header

接下来就是数据层（Data Layer）部分，图中看到的 Table Directory，Row Directory，Free Space，Row Data 都属于数据层。数据层有一个头结构（Data Header），从 bbed 中看到这个结构叫 kdbh，有 14 个字节。我们假设 ITL 有默认的 2 条记录，表空间采用 ASSM 管理，那么数据头的地址就从 0x64=100 开始。

0x0064-0x0064 flag

0x0065-0x0065 ntab
0x0066-0x0067 nrow
0x0068-0x0069 frre
0x006A-0x006B fsbo
0x006C-0x006D fseo
0x006E-0x006F avsp
0x0070-0x0071 tosp

下表是对数据层头结构的各个字段的解释。

flag	N=pctfree hit (clusters) F=don't put on free list K=flushable cluster keys
ntab	本块含有的表的个数。普通表的存储，一个块只存储一个表的数据，所以 ntab 一直是 1,当存储的是 cluster 表时，几个表的数据存储在同一个块中，ntab 就是 cluster 表的个数，会大于 1
nrow	本块中包含了多少条数据
frre	First free row index entry 这是第一个空闲行的索引入口，如果不为-1，那么说明下面的行目录中有空闲行，那么这就是那个索引。
fsbo	空闲的空间开始的偏移，一般在 row directory 之后，作为下一个 row offset 开始的地方
fseo	空闲的空间结束的偏移，一般在最后一条记录结束的地方，由于数据是从块底部往上增加的，所以 fseo 作为下一条记录往上开始的位置
avsp	可用空间大小，avsp=fseo-fsbo
tosp	总共可用空间（total space），这儿等于 avsp，如果有删除的行，或者有的字段 update 后变小，tosp 就会比 avsp 大

Table Directory

下面紧接着的就是表目录的信息，在 dump 文件中叫 pti，是一个结构数组，在 bbed 中这个结构叫 kdbt。在网上看到很多研究块结构的帖子对这部分都没有

明确解释，在这儿我们详细说明一下。

…… 部分信息省略……

Row Directory

行目录的结构比较简单，其实就是一个 2 字节的数组，在 bbed 中是 sb2 kdbr[]，每个数组元素中存放着一条记录的偏移量，叫做 row offset，这个偏移量有一个相对地址，这个相对地址是数据头 kdbh 的基地址，这一点一定要搞清楚，因为很容易会把这个偏移量看做是相对于块头的地址。

Row Data

行数据（row data）就是表中实际存储的数据了，上面介绍的所有信息都是为了怎样定位到一条条的行数据，块中的数据记录是根据 Row Directory 中的偏移量来定位的，这个偏移量又是以数据头（Data Header）为基地址的，所以第一条记录的数据在 kdbh 加上第一个偏移量的位置，第二条记录的数据在 kdbh 加上第二个偏移量的位置，依次类推。

…… 部分信息省略……

Cluster 表的数据块结构

…… 部分信息省略……

段头的存储结构

通过上面的文字，我们已经介绍了数据块的结构，现在就可以把一个块中的数据导出（export）来了，到了这里，你可能有一个疑问，那么一个表到底有哪些数据块呢？如果要把一个表的数据都导出来，从哪个块开始呢？如果你想到了

上面的问题，说明你是一个善于思考的学习者。Oracle 对段的管理结构能回答你的这些问题，第二个问题很简单，在 Oracle 给表分配段空间的时候，就把段头的地址（DBA）记录到数据字典基表 tab$中了，字段 file#和 block#记录了段头的相对文件号和开始的块号。找到了段头的块，就能找到段中包含了哪些区（extents），也就能找到表中包含了哪些块了。

Oracle 对段（segment）空间的管理有自动段管理（ASSM）和手工段管理（MSSM）两种方式。为了解决并发效率，避免段头争用，在版本 9.2 时引入了自动段管理，ASSM 用位图来跟踪和管理每个分配到对象的块。传统的管理是 MSSM，使用 FREELIST 来管理段中的空闲数据块。现在的 Oracle 版本默认都是采用 ASSM 方式。ASSM 和 MSSM 段头块的结构并不相同，我们分别介绍。

ASSM 管理的段头块

…… 部分信息省略……

MSSM 管理的段头块

…… 部分信息省略……

B*Tree 索引存储结构

…… 部分信息省略……

LOB 存储结构

…… 部分信息省略……

LOB 概念

…… 部分信息省略……

LOB Index

…… 部分信息省略……

LOB Segment

…… 部分信息省略……

LOB Locator

…… 部分信息省略……

LOB Chunk

…… 部分信息省略……

LOB In Row

…… 部分信息省略……

LOB Out Row

…… 部分信息省略……

Empty LOB

…… 部分信息省略……

LOB Locator 结构

…… 部分信息省略……

LOB INDEX

…… 部分信息省略……

LOB SEGMENT

…… 部分信息省略……

Secure File

…… 部分信息省略……

In line LOB

…… 部分信息省略……

Out line LOB

…… 部分信息省略……

Out Row

…… 部分信息省略……

Segment

…… 部分信息省略……

LOB 头块

…… 部分信息省略……

DBA0

…… 部分信息省略……

Itree

…… 部分信息省略……