第二十讲：InnoDB 存储引擎数据文件和分页机制

知春路遇上八里桥

<2024-08-01 Thu>

1. InnoDB 存储物理结构
2. InnoDB 存储逻辑结构
3. FSP_HDR 页
4. INODE 页
5. INDEX 页

1

InnoDB 存储物理结构

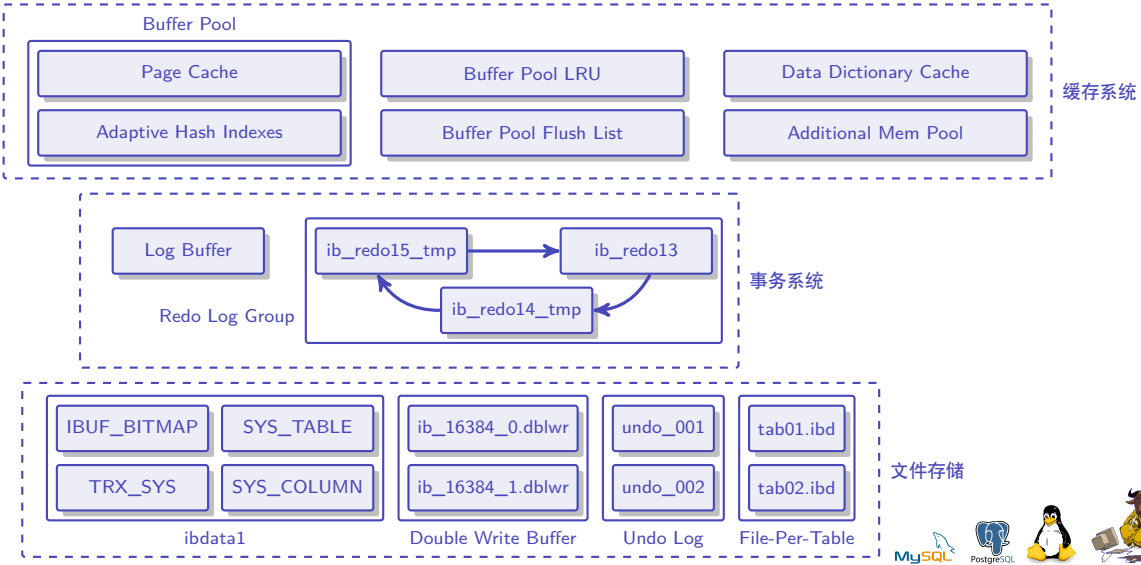
IBD 文件格式

- InnoDB 存储引擎的数据默认使用 IBD 文件进行存储
  - IBD (innobase data) 文件是二进制文件，使用 InnoDB 的自定义格式
  - IBD 的管理数据的基本单位是页 (page)，默认页面大小 (`innodb_page_size`) 为 16K (16384)
  - IBD 默认采用大端字节序 (Big-Endian)
- IBD 文件通常以 `.ibd` 结尾，文件用于存储 InnoDB 表的数据和索引，具体包含：
  - 表的行数据，即用户实际存储的数据记录
  - 表的主键索引和辅助索引
  - 用于支持数据库的事务隔离级别的 MVCC 数据
- InnoDB 的数据组织的顶层结构是表空间 (Tablespace)[1]，常见表空间有：
  - ibdata1 是共享表空间，ibtmp1 是临时表空间

```
mysql> show variables like '%data_file%';
+--------------------------+------------------------+
| Variable_name            | Value                  |
+--------------------------+------------------------+
| innodb_data_file_path    | ibdata1:12M:autoextend |
| innodb_temp_data_file_path | ibtmp1:12M:autoextend  |
+--------------------------+------------------------+
```

  - InnoDB 存储引擎默认开启 `innodb_file_per_table` 选项
    1. 此时每个表会独立创建一个 `table_name.ibd` 独立表空间文件
    2. 如果是 MySQL 5.7 的，还有一个 `.frm` 文件，用于记录表的格式定义

存储物理架构


Buffer Pool
Page Cache
Adaptive Hash Indexes
Buffer Pool LRU
Buffer Pool Flush List
Data Dictionary Cache
Additional Mem Pool
缓存系统
Log Buffer
Redo Log Group
ib_redo15_tmp
ib_redo13
ib_redo14_tmp
事务系统
IBUF_BITMAP
SYS_TABLE
TRX_SYS
SYS_COLUMN
ibdata1
ib_16384_0.dblwr
ib_16384_1.dblwr
Double Write Buffer
undo_001
undo_002
Undo Log
tab01.ibd
tab02.ibd
File-Per-Table
文件存储

数据目录示例

- MySQL 数据目录 (/opt/mysql/data) 中的部分文件

```
-rw-r----- 1 rtc rtc  384 Jul 29 11:10  binlog.index
-rw-r----- 1 rtc rtc 7.3K Jul 29 11:10  binlog.000109
drwxr-x--- 2 rtc rtc 4.0K May  8 21:32  employees
-rw-r----- 1 rtc rtc  12M Jul 29 11:10  ibdata1
-rw-r----- 1 rtc rtc  12M Jul 27 20:37  ibtmp1
-rw-r----- 1 rtc rtc  17K Jul 27 19:51  ib_buffer_pool
drwxr-x--- 2 rtc rtc 4.0K May  8 21:31  mysql
-rw-r----- 1 rtc rtc  26M Jul 27 22:05  mysql.ibd
drwxr-x--- 2 rtc rtc 4.0K May  8 21:31  performance_schema
-rw-r----- 1 rtc rtc  16M Jul 27 22:05  undo_001
-rw-r----- 1 rtc rtc  16M Jul 29 09:00  undo_002
-rw-r----- 1 rtc rtc 192K Jul 29 11:10 '#ib_16384_0.dblwr'
-rw-r----- 1 rtc rtc 8.2M May  8 21:30 '#ib_16384_1.dblwr'
drwxr-x--- 2 rtc rtc 4.0K Jul 27 20:37 '#innodb_redo'
```

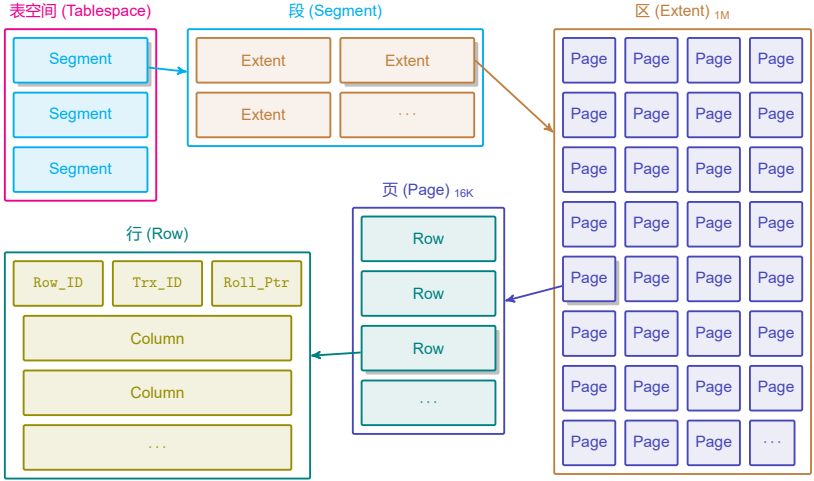
- employees 示例数据库的 file-per-table 表空间示例

```
data/employees $ ll
total 179M
-rw-r----- 1 rtc rtc 128K May  8 21:32 departments.ibd
-rw-r----- 1 rtc rtc  25M May  8 21:35 dept_emp.ibd
-rw-r----- 1 rtc rtc 128K May  8 21:35 dept_manager.ibd
-rw-r----- 1 rtc rtc  22M May  8 21:33 employees.ibd
-rw-r----- 1 rtc rtc 104M May  8 21:47 salaries.ibd
-rw-r----- 1 rtc rtc  27M May  8 21:37 titles.ibd
```

2

InnoDB 存储逻辑结构

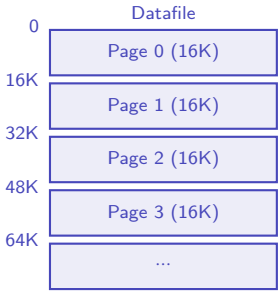
逻辑结构层次



- 逻辑组织结构: 表空间 ⇒ 段 ⇒ 区 ⇒ 页 ⇒ 行

数据文件和页

- 数据文件 (datafile) 是存储在操作系统上的文件
  - 这里重点关注 `.ibd` 数据文件
- 页 (page) 是 InnoDB 引擎操作数据的基本单位
- 页默认大小为 16K, 通常不会修改页面大小
- 历史因素, 页也被称做块 (block), 一般按照习惯
  - 16K 的数据存储在磁盘上称作块
  - 16K 的数据读取到内存中称做页
- 使用 hexdump 工具可以查看内容
  - `-s, --skip <offset>` 跳转若干字节
  - `-n, --length <length>` 打印的字节总数
  - `-C, --canonical` 显示 hex+ASCII 内容



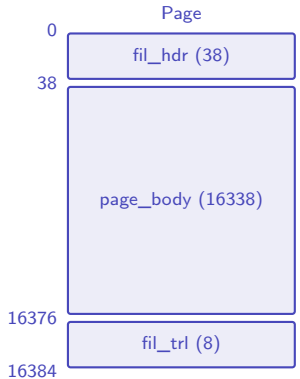

```
data/employees $ hexdump -n 32 departments.ibd
0000000 9e05 fbd9 0000 0000 0100 a538 0000 0100
0000010 0000 0000 3601 b685 0800 0000 0000 0000
0000020
data/employees $ hexdump -C -s 0x10000 -n 0x20 departments.ibd
00010000  a9 1c 8b b0 00 00 00 04  ff ff ff ff ff ff ff ff  |................|
00010010  00 00 00 00 01 38 49 dd  45 bf 00 00 00 00 00 00  |.....8I.E.......|
00010020
```

页的基本结构

- 页分成三个部分，其偏移量的定义位于
  - ☞ storage/innobase/include/fil0types.h

```
110  /** start of the data on the page */
111  constexpr uint32_t FIL_PAGE_DATA = 38;
118  /** size of the page trailer */
119  constexpr uint32_t FIL_PAGE_DATA_END = 8;
```

- 主体部分介绍如下
  - fil_hdr 文件头信息 (File Header)
    1. 记录文件校验参数
    2. 记录一些必要信息
  - fil_trl 文件尾信息 (File Trailer)
    1. 记录校验信息
  - page_body 页数据体
    1. 里面存储实际的数据
    2. 总共有 $16384 - 38 - 8 = 16338$ 字节数可用



页头 fil_hdr

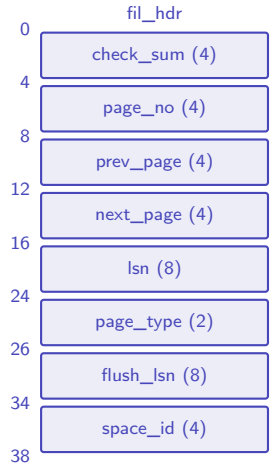
- 源码见 ☞ storage/innobase/include/fil0types.h

```
41  /** MySQL-4.0.14 space id the page belongs to (== 0) but in later
42  versions the 'new' checksum of the page */
43  constexpr uint32_t FIL_PAGE_SPACE_OR_CHKSUM = 0;
44
45  /** page offset inside space */
46  constexpr uint32_t FIL_PAGE_OFFSET = 4;
```

- space_id 是当前表空间的 ID
- page_no 是页码，也叫 offset，是表空间的页面偏移
- page_type 是当前页面的类型 ☞ …/fil0fil.h

```
1181  using page_type_t = uint16_t;
1182
1183  /** File page types (values of FIL_PAGE_TYPE) @{ */
1184  /** B-tree node */
1185  constexpr page_type_t FIL_PAGE_INDEX = 17855;
```

  - 常见类型：FSP_HDR(8)，INDEX(17855) …
- prev_page / next_page 页面双向链表
  - page_no=0 时，prev_page ⇒ server_version
  - page_no=0 时，next_page ⇒ space_version
- 其他的字段后面再聊

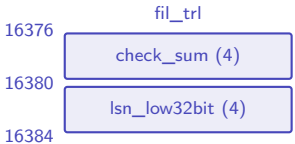


页尾 fil_trl

- 源码见 ☞ … /fil0types.h

```
113  /** File page trailer */
114  /** the low 4 bytes of this are used to store the page checksum, the
115  last 4 bytes should be identical to the last 4 bytes of FIL_PAGE_LSN */
116  constexpr uint32_t FIL_PAGE_END_LSN_OLD_CHKSUM = 8;
117
118  /** size of the page trailer */
119  constexpr uint32_t FIL_PAGE_DATA_END = 8;
```

- check_sum 是校验码
  - 与页头中 fil_hdr.check_sum 相等
- lsn_low32bit 是 LSN 低 32 位
  - 与页头中 fil_hdr.lsn 的低 32 位相等




```
data/employees » hexdump -C -s 16376 -n 8 departments.ibd
00003ff8  05 9e d9 fb 01 36 85 b6       |.....6..|
00004000

fil_trl: FilePageTrailer {
    addr: 0x3ff8@(16376),
    check_sum: 0x059ed9fb(94296571),
    lsn_low32bit: 0x013685b6(20350390),
},
```

3

FSP_HDR 页

FSP_HDR 页 - 表空间描述首部

- `fil_hdr.page_type=FSP_HDR(8)` 是 FSP_HDR 页
  - 一般 FSP_HDR 页位于 datafile 的第 0 页
  - 它里面包含 tablespace 的基本描述信息
  - 8.0 后还包含 `server_version` 和 `space_version`
- FSP_HDR 页用于存储 INODE 和 Extent 的元信息
- `fsp_hdr` 表空间文件首部 (File Space Header)
- `xdes_entry` 区描述符 (File Extent Descriptor)
  - 连续的 256 个都是区描述符
- 源码见 ☞ storage/innobase/include/fsp0fsp.h
- 8.0 后空闲区域存储着 SDI 页的元信息
  - `sdi_version` 和 `sdi_page_no`
  - 具体地址计算方式如下

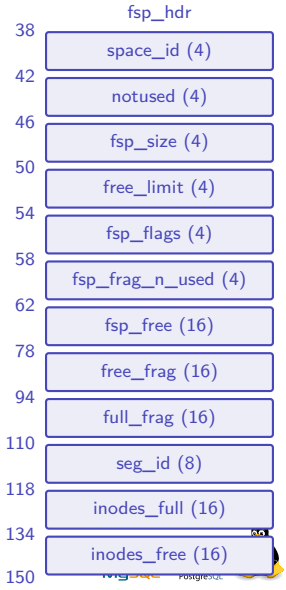
```
// INFO_SIZE = 3 + 4 + 32*2 + 36 + 4 = 111
static constexpr size_t INFO_SIZE =
    (MAGIC_SIZE + sizeof(uint32) + (KEY_LEN * 2) + SERVER_UUID_LEN +
     sizeof(uint32));
// INFO_MAX_SIZE = 111 + 4 = 115
static constexpr size_t INFO_MAX_SIZE = INFO_SIZE + sizeof(uint32);
// sdi_addr = 10505
sdi_addr = 10390 + INFO_MAX_SIZE;
```

fsp_hdr - FSP_HDR 页的首部

- 源码见 ☞ storage/innobase/include/fsp0fsp.h

```
100  /** In-memory representation of the fsp_header_t file structure. */
101  struct fsp_header_mem_t {
102    fsp_header_mem_t(const fsp_header_t *header, mtr_t *mtr);
103
104    ulint m_space_id;
105    ulint m_notused;
106    ulint m_fsp_size;
107    ulint m_free_limit;
108    ulint m_flags;
109    ulint m_fsp_frag_n_used;
110    flst_bnode_t m_fsp_free;
111    flst_bnode_t m_free_frag;
112    flst_bnode_t m_full_frag;
113    ib_id_t m_segid;
114    flst_bnode_t m_inodes_full;
115    flst_bnode_t m_inodes_free;
116
117    std::ostream &print(std::ostream &out) const;
118  };
```

- 跳过 fil_hdr 直接从 38 字节开始偏移
- fsp_size 当前表空间包含的页面数量
- seg_id 下一个分配的段 ID

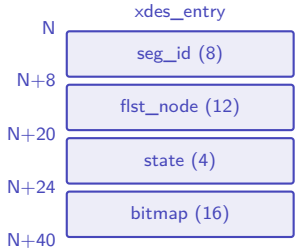


XDES Entry 区描述符项

- 源码见 ☞ storage/innobase/include/fsp0fsp.h

```
267  /** The identifier of the segment to which this extent belongs */
268  constexpr uint32_t XDES_ID = 0;
269  /** The list node data structure for the descriptors */
270  constexpr uint32_t XDES_FLST_NODE = 8;
271  /** contains state information of the extent */
272  constexpr uint32_t XDES_STATE = FLST_NODE_SIZE + 8;
273  /** Descriptor bitmap of the pages in the extent */
274  constexpr uint32_t XDES_BITMAP = FLST_NODE_SIZE + 12;
```

- state 是区的状态变量，参考 xdes_state_t 枚举
  - XDES_NOT_INITED = 0
  - XDES_FREE = 1
  - XDES_FREE_FRAG = 2
  - XDES_FULL_FRAG = 3
  - XDES_FSEG = 4
  - XDES_FSEG_FRAG = 5
- bitmap 用于标记当前区中页面的状态
  - 每页使用 2 位，总共可标记 $16 \times 8 \div 2 = 64$ 页
  - 2 位分别标记：free/clean
    1. XDES_FREE_BIT = 0
    2. XDES_CLEAN_BIT = 1



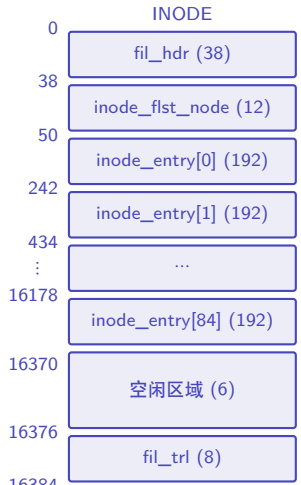
4

INODE 页

INODE 页

- `fil_hdr.page_type=INODE(3)` 是 INODE 页
- INODE 页用于管理表空间中的段 (segment)
- `inode_flst_node` 邻接 INODE 页的指针
  - `prev` 和 `next` 指向前后 INODE 页地址
  - InnoDB 支持 64 个二级索引, 2 个段对应一个索引 (非叶子节点和叶子节点)
  - 共 $64 \times 2 \times 192 = 24576$，超过单页上限 (16K)
- `inode_entry` 是 INODE 项
  - 最多的 85 个都是 INODE 项, 内容后面说明
- 源码见 ☞ storage/innobase/include/fsp0fsp.h

```
196   constexpr uint32_t FSEG_INODE_PAGE_NODE = FSEG_PAGE_DATA;
197   /* the list node for linking
198   segment inode pages */
199
200   constexpr uint32_t FSEG_ARR_OFFSET = FSEG_PAGE_DATA + FLST_NODE_SIZE;
```

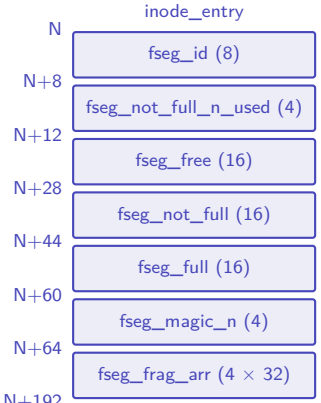


inode_entry INODE 数据项

- inode_entry 管理一个段 (segment)
- 源码见 ☞ storage/innobase/include/fsp0fsp.h

```
203  constexpr uint32_t FSEG_ID = 0;
204  /** number of used segment pages in the FSEG_NOT_FULL list */
205  constexpr uint32_t FSEG_NOT_FULL_N_USED = 8;
206  /** list of free extents of this segment */
207  constexpr uint32_t FSEG_FREE = 12;
```

- fseg_id=0 表示改 inode_entry 未使用
- 三个区的链表
  - fseg_free 未使用的区
  - fseg_not_full 部分使用的区
  - fseg_full 全部使用完的区
- fseg_not_full_n_used 记录 fseg_not_full 的数量
- fseg_magic_n 魔数，等于 97937874
- fseg_frag_arr 碎片区
  - segment 初始扩容时，首先分配这些碎片 page
  - 直到分配完毕才会进入 extent 分配



5

INDEX 页

INDEX 页

- fil_hdr.page_type=INDEX(17855) 是 INDEX 页
- INDEX 页专门用于存放索引信息的页，索引即数据
- idx_hdr 是索引首部，fseg_hdr 是 FSEG 首部
- infimum 最小记录和 supremum 最大记录

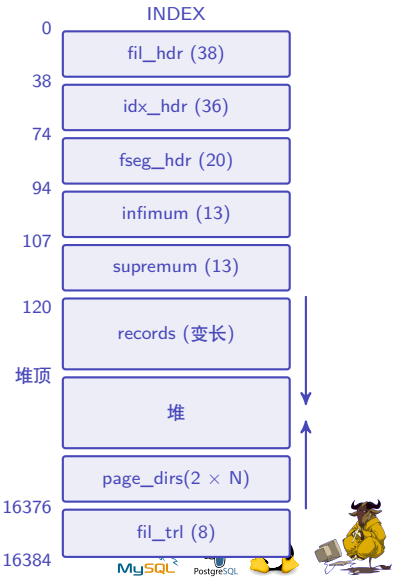
  ☞ storage/innobase/include/page0page.h

```
79  /** Extra bytes of an infimum record */
80  static const byte infimum_extra[] = {
81      0x01,           /* info_bits=0, n_owned=1 */
82      0x00, 0x02      /* heap_no=0, status=2 */
83      /* ?, ?     */ /* next=(first user rec, or supremum) */
84  };
85  /** Data bytes of an infimum record */
86  static const byte infimum_data[] = {
87      0x69, 0x6e, 0x66, 0x69, 0x6d, 0x75, 0x6d, 0x00 /* "infimum\0" */
88  };
```

- records 是索引的记录项，往高地址增长
- page_dirs 页目录每个 2 字节，往低地址增长

  ☞ …/page0page.h

```
63  /* We define a slot in the page directory as two bytes */
64  constexpr uint32_t PAGE_DIR_SLOT_SIZE = 2;
```



idx_hdr 索引页首部

- page_format 是行记录格式，主要有
  - COMPACT 紧凑格式，默认格式
  - REDUNDANT 格式，比较占空间，早期版本使用
- 代码见 ☞ storage/innobase/include/page0types.h

```
56  /** number of slots in page directory */
57  constexpr uint32_t PAGE_N_DIR_SLOTS = 0;
58  /** pointer to record heap top */
59  constexpr uint32_t PAGE_HEAP_TOP = 2;
60  /** number of records in the heap, bit 15=flag: new-style compact page format */
61  constexpr uint32_t PAGE_N_HEAP = 4;
62  /** pointer to start of page free record list */
63  constexpr uint32_t PAGE_FREE = 6;
64  /** number of bytes in deleted records */
65  constexpr uint32_t PAGE_GARBAGE = 8;
66  /** pointer to the last inserted record, or NULL if this info has been reset by
67   a delete, for example */
68  constexpr uint32_t PAGE_LAST_INSERT = 10;
69  /** last insert direction: PAGE_LEFT, ... */
70  constexpr uint32_t PAGE_DIRECTION = 12;
71  /** number of consecutive inserts to the same direction */
72  constexpr uint32_t PAGE_N_DIRECTION = 14;
73  /** number of user records on the page */
74  constexpr uint32_t PAGE_N_RECS = 16;
```

idx_hdr 字段含义

- page_n_dir_slots 页目录槽的数量
- page_heap_top 第一条 Record 位置
- page_n_heap 堆中的 Record 数
  - 高 1 位被用作 page_format
    REC_FORMAT_REDUNDANT = 0,
    REC_FORMAT_COMPACT = 1,
- page_free 空闲 Record 位置
- page_garbage 被删除的 Record 位置
- page_last_insert 最新插入的 Record
- page_direction 最新插入的 Record 方向
- page_n_direction 相同方向连续插入 Record 数量
- page_n_recs Record 数量
- page_max_trx_id 最大事务 ID
  - 二级索引和 insert buffer 用的
- page_level 在 B+ 树的深度
- page_index_id 索引 ID

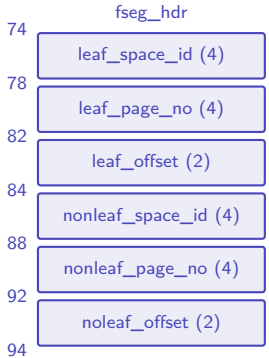
```
idx_hdr: IndexHeader {
    addr: 0x0026@(38),
    page_n_dir_slots: 7,
    page_heap_top: 912,
    page_format: COMPACT,
    page_n_heap: 32794,
    page_free: 0,
    page_garbage: 0,
    page_last_insert: 885,
    page_direction: PAGE_RIGHT,
    page_n_direction: 23,
    page_n_recs: 24,
    page_max_trx_id: 0,
    page_level: 0,
    page_index_id: 157,
},
```

fseg_hdr

- 代码见 ☞ storage/innobase/include/page0types.h

```
90  constexpr uint32_t PAGE_BTR_SEG_LEAF = 36;
91  constexpr uint32_t PAGE_BTR_IBUF_FREE_LIST = PAGE_BTR_SEG_LEAF;
92  constexpr uint32_t PAGE_BTR_IBUF_FREE_LIST_NODE = PAGE_BTR_SEG_LEAF;
93  /* in the place of PAGE_BTR_SEG_LEAF and _TOP
94  there is a free list base node if the page is
95  the root page of an ibuf tree, and at the same
96  place is the free list node if the page is in
97  a free list */
98  constexpr uint32_t PAGE_BTR_SEG_TOP = 36 + FSEG_HEADER_SIZE;
```

- fseg_hdr 主要包含段首部信息 FSEG_HEADER
  - PAGE_BTR_SEG_LEAF 叶子节点的 FSEG
  - PAGE_BTR_SEG_TOP 非叶子节点的 FSEG
  - FSEG_HEADER_SIZE=10



结束