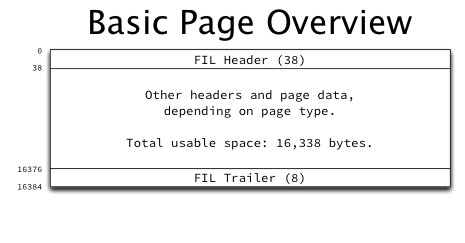
# innodb存储说明

进行innodb的存储是以页尾单位进行存储的，默认的页大小为16k，除非特殊情况，下文都以16k的页作为说明。可以通过修改innodb\_page\_size进行页大小的设置（在5.6之后的版本）。

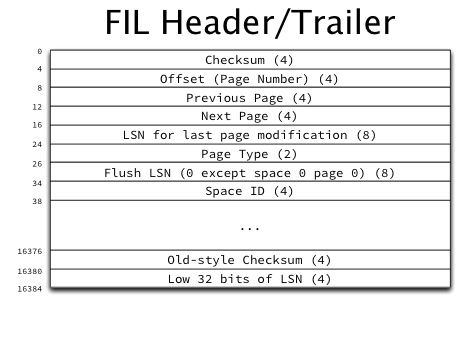
## innodb的页组织情况



fil是file的缩写。其中fil header是固定的38字节，fil trailer是固定的8字节

fil header/trailer是用来记录页相关信息的，在后面将会一一介绍其中的内容，所存储的页相关信息。

如下图，是页头和页尾的信息



### header说明

checksum：页面的校验码，用于确保页面没有被篡改

offset：页面的编号，0-based，如第一页就是0，第二页是1，参看下面的实例

previous/next page：前/后页

LSN for last …：记录LSN数值

page type：页面类型。有如下页面类型FIL\_PAGE\_INDEX/ FIL\_PAGE\_UNDO\_LOG/ FIL\_PAGE\_INODE等等。如下面所举的表文件，FIL\_PAGE\_INODE的值是17855，十六进制为0x45BF,第四页为表索引页。

flush LSN：

space id：存储空间的id

### trailer说明：

old-style checksum：早期的页面校验码，不推荐使用。现在偶尔也会在其他情况使用

low 32 bits LSN：

### 实际例子

以下是有131072条记录innobase的表文件(idb文件)的1-6页的前48个字节，查看fil head的内容。

红色为页所在位置的编号，绿色为页类型

第一页



第二页



第三页



第四页



第五页



第六页



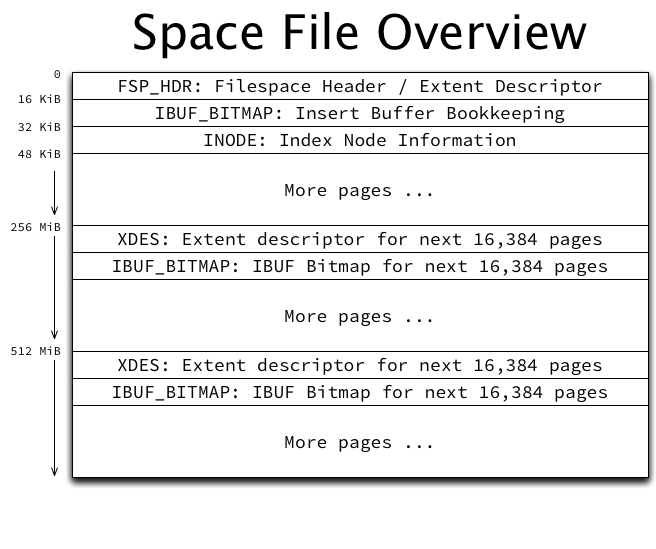
## 文件空间

innobase的数据文件(.ibd)由多个页组成（最多2的32次方个页），页面分为控制页和数据页。

控制页：用于存储控制信息，通过这些控制信息，innodb对数据页进行管理。

数据页：用于存储数据信息，数据页又分为保存B+数中间节点的internal node页和保存实际数据的leaf node。数据页以64个页为一个extend进行管理。

如下是一个数据文件的空间使用情况，在后面对各个页面进行说明



### 控制页

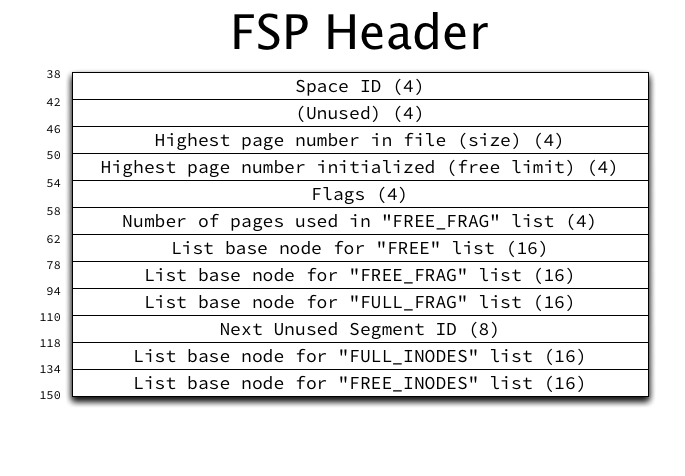
控制页就是用来管理ibd文件中所有页的页，关于ibd文件中所有页的使用情况将被保存在这些控制页中。

#### FSP\_HDR

每个ibd文件的第一个页，在原文件中的类型定义为FIL\_PAGE\_TYPE\_FSP\_HDR。一个ibd文件中，只有一个页具有这样的类型。

该页保存了FSP header信息和extend的管理信息（后面将介绍extend的管理）。完成了两个职能，一个是file级别的信息管理，一个是extend级别的信息管理。

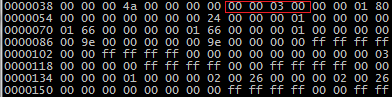
下图是FSP header的内容



space id：每个文件是一个space，每个文件被指定一个space id作为innobase的space空间的唯一标识。

highest pagenumber in file：总共分配了多少页

参考如下图t1\_innobase.ibd，12582912是768页，红框内300是16进制，10进制也是768，由此得出对应的关系





highest page number initialized：在扩展文件后，并不是所有的页面都被初始化，该部分记录最高被初始化的页面（fil header被初始化），该值永远小于等于“highest pagenumber in file”的数值

flags：与该存储空间相关的标志信息

Next Unused Segment ID：下一个可使用的段id

list base node for "free" list：空闲的xdes entry的链表

list base node for "free\_frag" list：部分使用的xdes entry的链表，该链表中的xdes entry所管理的页用于保存B+树中的叶子结点的数据。

list base node for "full\_frag" list：所有页面都被使用的xdes entry的链表，该链表中的xdes entry所管理的页用于保存B+树中的叶子结点的数据。

list base node for "full\_inodes" list：所有页面都被使用的xdes entry的链表，该链表中的xdes entry所管理的页用于保存B+树中的中间结点的数据。

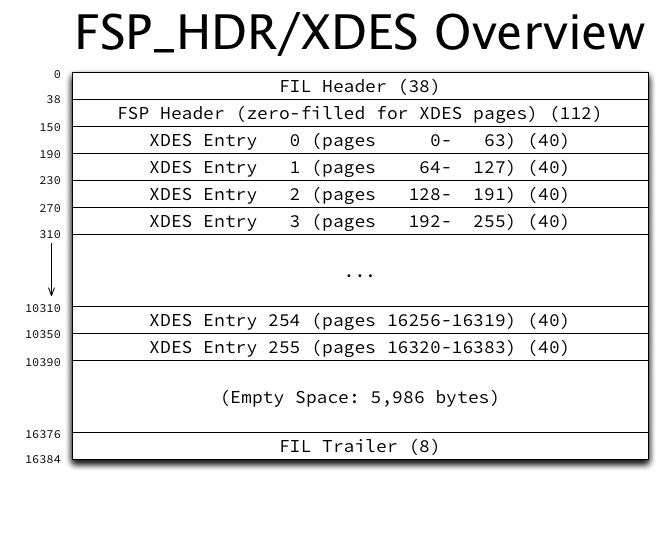
list base node for "free\_inodes" list：部分使用的xdes entry的链表，该链表中的xdes entry所管理的页用于保存B+树中的中间结点的数据。

作为extend descriptor的功能，将在下面讲解

#### Extend descriptor

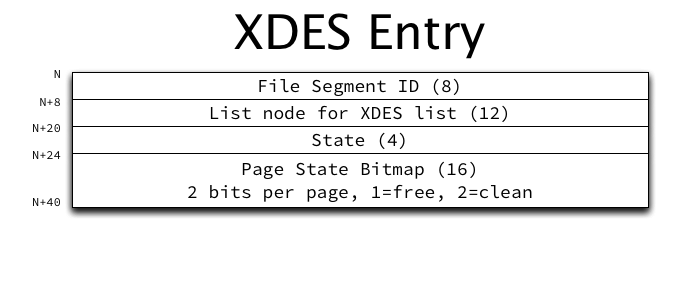
该控制页面用于管理extend，extend用于管理64个page。

如下是extend descriptor页面的结构



extend dexcriptor的信息固定从第150个字节便宜开始，每个页面保存256个entry信息，每个entry对应一个extend。也就是一个exetnd descriptor页面管理256\*64个页面。

下面是entry的结构



file segment id：当前extend所从属的segment

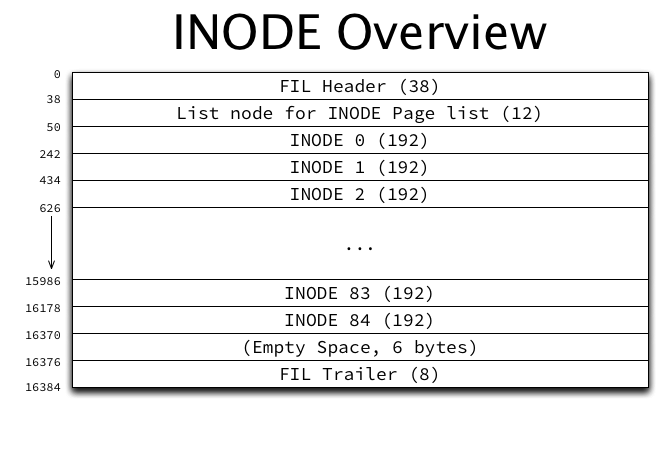
List node for XDES list：extend entry的链表节点，为将各个entry连接起来使用

state：说明当前的extend的状态，分别是所有页未使用（free），部分页未使用（free frag），全部页已使用。

page state bitmap：用于说明当前extend的页面使用情况，2个bit说明使用情况，64个页面，所以占用16字节。只有第一个bit位用来说明使用情况，第二个bit位作为保留使用。

#### inode page

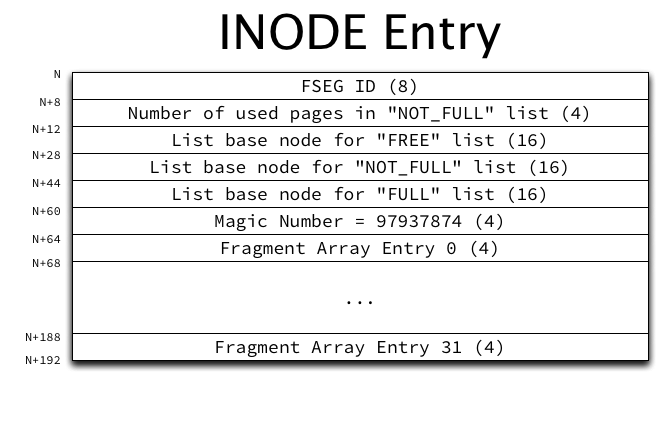
inode page记录inode entry。如下是inode page的结构



list node for INODE page list：INODE page的链接节点。

INODE 0-84：INODE entry

如下是inode entry的结构



FSEG ID：0表示该entry尚未使用，否则存储FSEG ID。

list base node for "free" list：属于该INODE的free extend entry的链表

list base node for "not full" list：属于该INODE的not full extend entry的链表

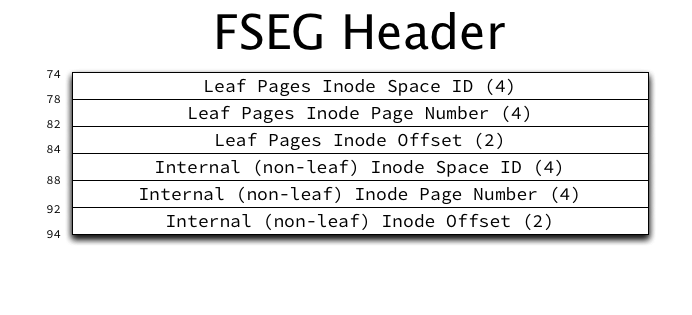
list base node for "full" list：属于该INODE的full extend entry的链表

magic number：魔数，用于程序运行时检查

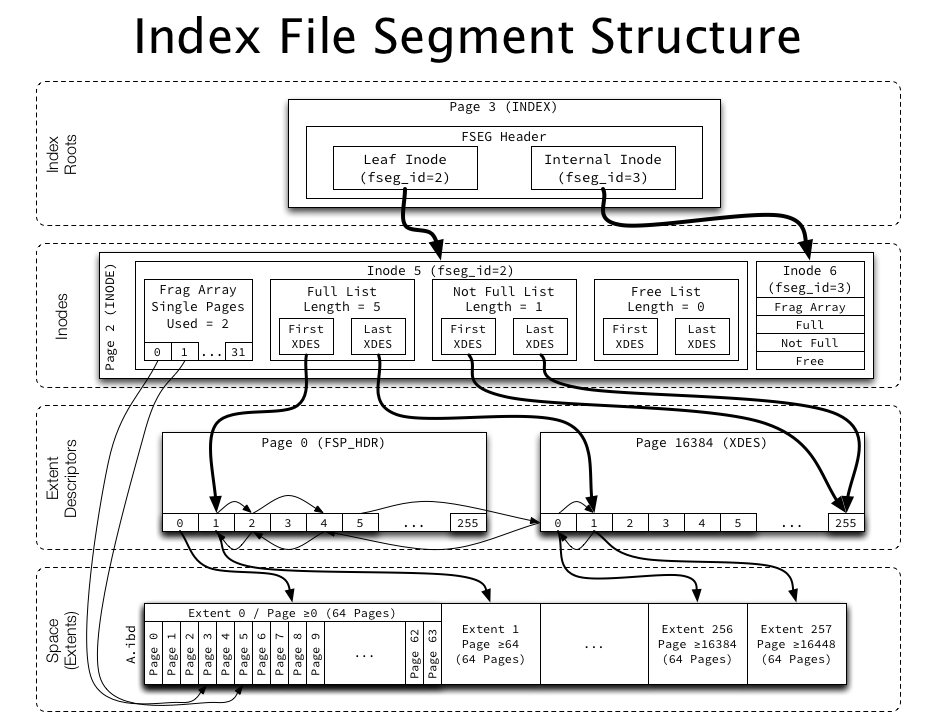
fragment array entry：用于记录当前使用的页

innobase的存储核心是索引，每个表创建伊始就创建了一个cluster索引，这个索引和其他后续在该表上创建的索引并无不同。每个索引使用2个inode entry，用于存储页节点，一个inode entry用于保存叶节点数据，一个inode entry用于保存中间节点的数据。

下图所列的是一个索引的root信息



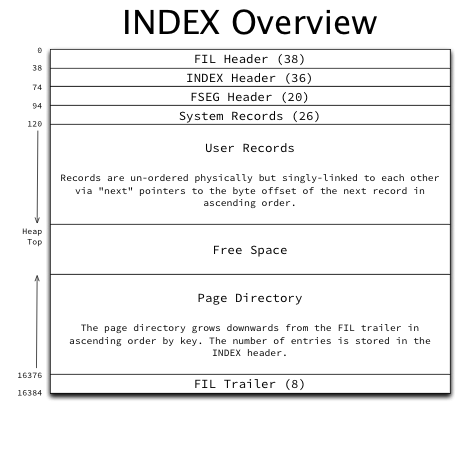
下图说明一个索引是如何组织的



### 数据页

在innobase中，表中的数据都是通过索引存储的。创建表可以看作是完成两件事情，一是定义了表结构，二是创建一个cluster index。后续创建的索引，都将依赖于最开始创建的cluster index。

索引是通过数据页组成的，根据在B+树中的作用，数据页又分为leaf page和internal page。前者存储的是插入的数据，后者存储的到leaf page的索引信息。不过这两种页面的结构类似，下面说明数据页的结构



fil header/trailer：页头/尾，参考前面所讲的页结构

index header：索引页头，用于保存该索引页的信息，后面将会讲到

FSEG header：FSEG头

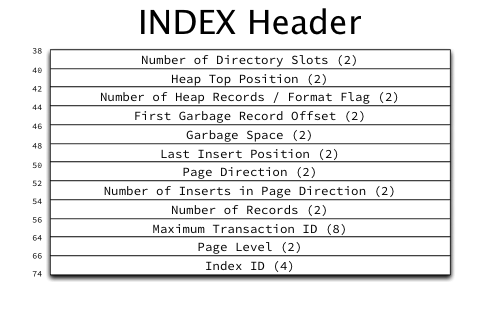
sytem records：系统记录，也称虚记录

user records：用户数据，对于internal page来说是索引记录，对于leaf page来说是插入记录

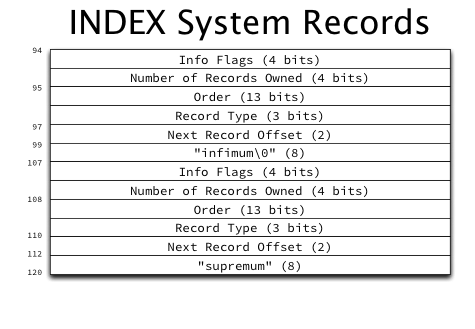
free space：可用空间

page directory：页中记录的偏移

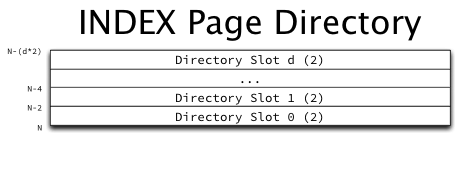
如下是index header的结构



如下是系统记录的结构



如下是page directory的结构



## 总结

从两个方面看innobase的存储组织。一个是围绕extend的页面管理，这是存储空间上的管理；一个是围绕B+树实现的，数据存储。