Sqlite 源码分析报告

Btree 模块

专　业：　计算机应用技术

By：LYleonard

2015 年 12 月

目录

第1章 B-树概述

B-树模块主要分为四个文件如下表 1.1

Btree 模块		
btree.h	248 行	头文件，定义了 Btree 提供的操作接口。
btree.c	9636 行	Btree 部分的主要实现
btreeInt.h	702 行	定义以下数据结构： Btree--Btree handler BtCursor--使用的游标 BtLock--btree 锁结构 BtShared--包含了一个打开的数据库的所有信息 MemPage--加载到内存的文件是该数据结构 CellInfo--结构的实例用来保存单元头信息
btmutex.c	325 行	该文件的代码是用来在 B 树对象上实现互斥机制。

表 1.1

B-树的实现代码主要位于源文件 btree.c 中，文件格式的细节被记录在 btree.c 开头的备注里。源文件 btmutex.c 包含的代码用来在 B 树对象上实现互斥机制，这个文件中的代码是属于 btree.c 文件。由于 btree.c 文件代码量太大，所以将其分出成成为独立的一部分。头文件 btree.h，定义了 Btree 提供的操作接口。对每个接口具体做什么，进行了而详细的描述。btreeInt.c 定义了一些重要的数据结构包括定 Btree、BtCursor、BtLock、BtShared、MemPage、CellInfo 等。

根据 SQLite 的架构图 1.1，B 树模块是后端引擎的一部分，位于虚拟机和页缓存之间，它向上连接虚拟机(VM)，向下连接页面调度程序(pager)。虚拟机负责执行程序，负责操纵数据库文件。数据库的存储用 B 树实现。B 树向磁盘请求信息，以固定大小的块。块大小一般在 512 字节到 65536 字节之间，系统默认的块的大小是 1024 字节。B 树从页缓存中请求页，当修改页或者提交或者回滚操作时，通知页缓存。修改页时，如果使用传统的回滚日志，pager 首先将原始页复制到日志文件。同样，page 在 Btree 完成写操作时收到通知，并基于所处的事务状态决定如何处理。Btree 和

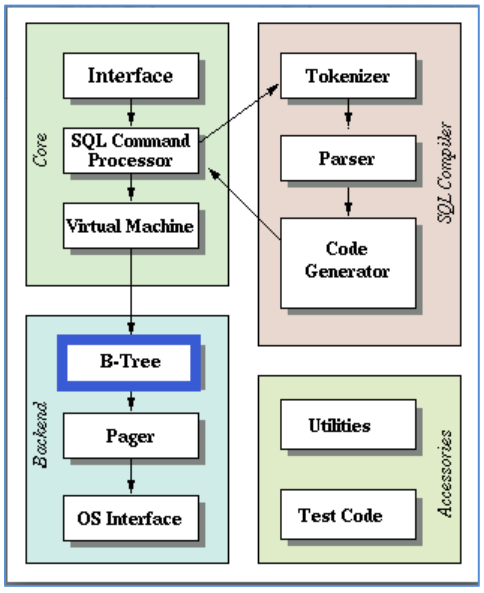


图 1.1

pager 使用的数据都存放在数据库页当中。数据库页当中存储着大量信息（包括记录，字段和索引等）。Btree 负责将页组织为树状结构，数据以 B 树数据结构的形式存储在磁盘上，方便搜索。Pager 根据 Btree 的要求对磁盘进行读写操作。每一个 Btree 都对应一个表或者索引。数据库中的每个表和索引使用一棵单独的 B-树，所有的 B-树存放在同一个磁盘文件中。

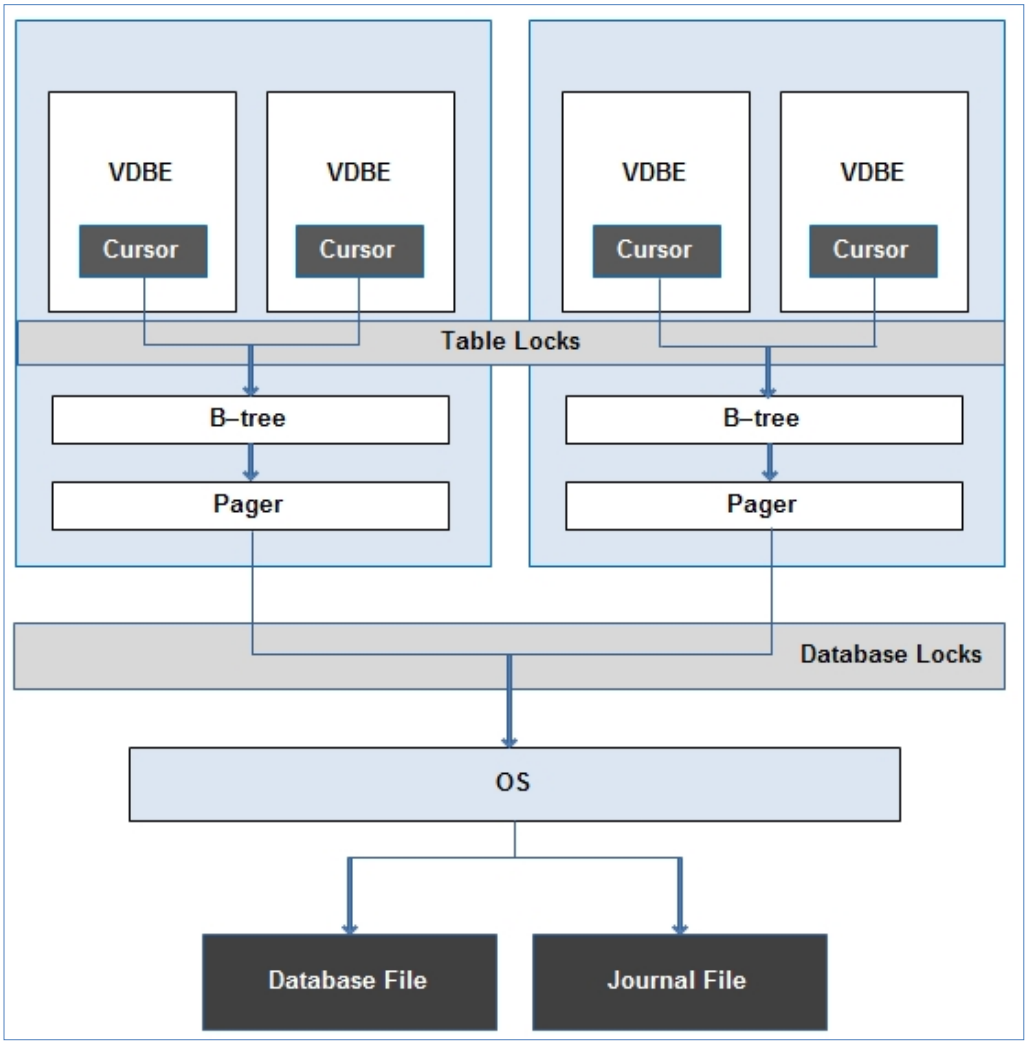


图 1.2

一个 connection 可以有多个 database 对象(一个主要的数据库以及附加的数据库)如上图 1.2，每一个数据库对象有一个 Btree 对象，一个 Btree 有一个 pager 对象。语句最终都是通过 connection 的 Btree 和 pager 从数据库读或者写数据， Btree 通过游标(cursor)遍历存储在页面(page)中的记录。游标在访问页面之前要把数所从 disk 加载到内存，而这就是 pager 的任务。任何时候，如果 Btree 需要页面，它都会请求 pager 从 disk 读取数据，然后把页面(page)加载到页面缓冲区(page cache)，之后，

Btree 和与之关联的游标就可以访问位于 page 中的记录了。如果 cursor 改变了 page，为了防止事务回滚，pager 必须采取特殊的方式保存原来的 page。总的来说，pager 负责读写数据库，管理内存缓存和页面（page），以及管理事务，锁和崩溃恢复(这些在事务一节会详细介绍)。

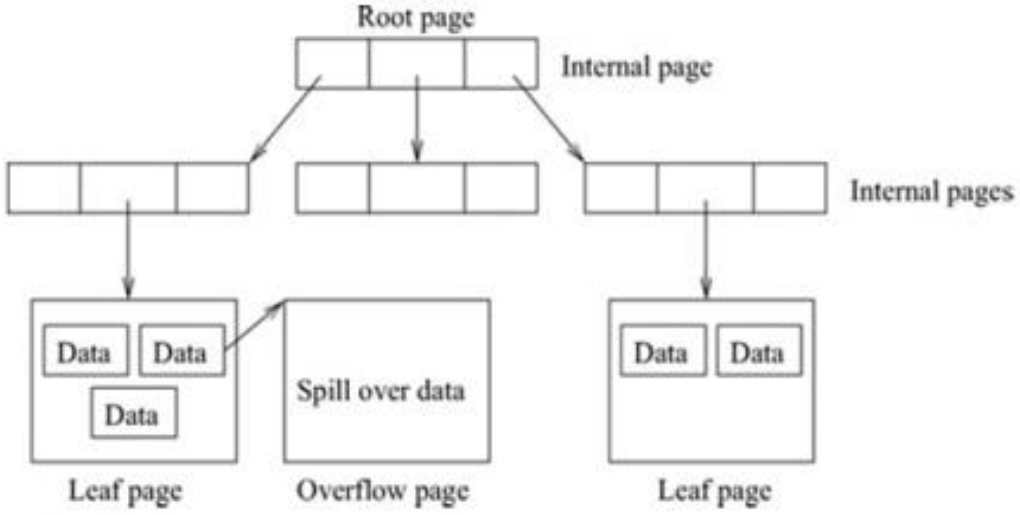


图 1.3 sqlite 中 B 树组织结构

Sqlite 中的 B 树结构如图 1.3 所示，Btree 记录按键值顺序存储。所有的键值在一个 B-tree 中必须唯一（由于键值对应于 rowid 主键，主键具有唯一性）。表使用 B+tree 定义在内部页中，不包含表数据（数据库记录）。图 1.4 为 B+tree 表示一个表的实例。本文没有区分 Btree 和 B+tree，以下统称 Btree。

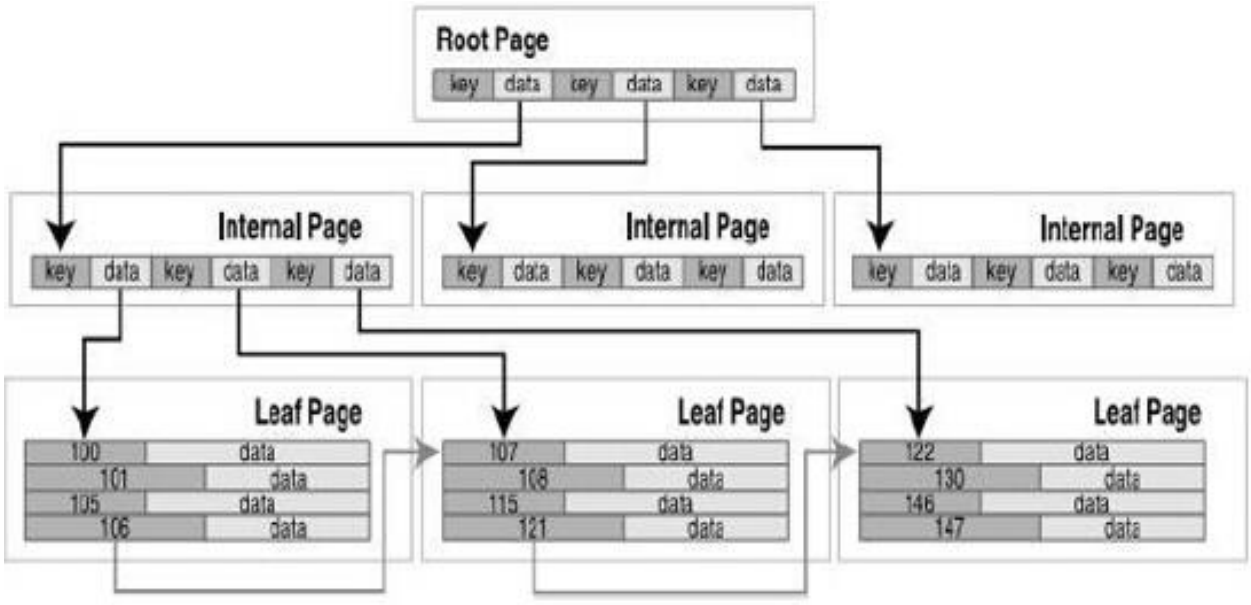


图 1.4 B+tree 组织结构

第 2 章 B 树模块物理存储结构和外部接口函数

2.1 B 树模块物理存储结构

2.1.1 节点的数据结构

SQLite 存储在外部的数据库是以 Btree 来组织的，在 btreeInt.c 中对 B 树模块的头结构进行了详细的描述，具体细节如下。基本思想是文件的每个页都包含 N 个数据库项(即关键字)和 N+1 个指向子页的指针如下表 2.1。数据库中的数据是分为很多的页面存储的，而 SQLite 又将页以 B 树的形式存储在磁盘上。

Ptr(0)	Key(0)	Ptr(1)	Key(1)	...	Key(N-1)	Ptr(N)

表 2.1 节点的数据结构

Ptr(0)指向的页上的所有的 key 的值都小于 Key(0)。所有 Ptr(1)指向的页和子页的所有的 key 的值都大于 Key(0)，小于 Key(1)。所有 Ptr(N)指向的页和子页的 key 的值都大于 Key(N-1)，等等。为了找到一个特定的 key，需要从磁盘上以 O(long(M))来读取，其中 M 是树的阶数。内存中找不到，就发生缺页中断。

在 SQLite 的实现中，一个文件可以含有 1 个或的过独立的 BTree。每一个 BTree 由它的根页的索引来标识。所有项的 key 和数据组成了有效负荷(payload)。数据库的一个页有一个固定的有效负荷总数。如果负荷大于了预先设定的值，那么剩余的字节就会被存储在溢出页上。一个入口的有效负荷再加上前向指针（the preceding pointer）构成了一单元(cell)。每一页都有一个小头部，包含了 Ptr(N)指针和其它一些信息，例如 key 和数据的大小。

2.1.2 数据库首部

页面(page)分四种类型：叶子页面(leaf)，内部页面(internal)，溢出页面(overflow)和空闲页面(free)。Sqlite 通过 B+tree 模型来管理所有的页面。内部页面包含查询时的导航信息，叶子页面存储数据，例如元组。如果一个元组的数据太大，一个页面容纳不下，则一些数据存储在 B 树的页面中，余下的存储在溢出页面中。一个文件分成了多个页。第一页叫做页 1，第二页叫做页 2，以此类推。页的个数为 0 表示没有页。每一页或者是一个 btree 页，或者是一个 freelist 页，或者是一个溢出页。

第一页一定是一个 btree 页。第一页的前面 100 个字节包含了一个特殊的首部(文件头)，它是这个文件的描述，文件头的格式如下表 2.2。

偏移量	大小	说明
0	16	首部字符串,如果不改源程序,此字符串永远是"Sqlite format 3"
16	2	页大小(以字节为单位).
18	1	文件格式版本(写).如果该值大于 1,表示文件为只读.
19	1	文件格式版本(读).如果该值大于 1,SQLite 认为文件格式错,拒绝打开此文件.
20	1	每页尾部保留空间的大小.(留作它用,默认为 0.)
21	1	Btree 内部页中一个单元最多能够使用的空间.255 意味着 100%,默认值为 0x40,即 64(25%),这保证了一个结点(页)至少有 4 个单元.
22	1	Btree 内部页中一个单元使用空间的最小值.默认值为 0x20,即 32(12.5%).
23	1	Btree 叶子页中一个单元使用空间的最小值.默认值为 0x20,即 32(12.5%).
24	4	文件修改计数,通常被事务使用,由事务增加其值.SQLite 用此域的值验证内存缓冲区中数据的有效性.
28	4	留作以后使用.
32	4	空闲页列表首指针.
36	4	文件内空闲页的数量.
40	60	15 个 4 字节的元数据变量.
40	4	Schema 版本：每次 schema 改变(创建或删除表)、索引、视图或触发器等对象,造成 sqlite_master 表被修改)时,此值+1.
44	4	模式层的文件格式.当前允许值为 1~4,超过此范围,将被认为是文件格式错.
48	4	文件缓存大小
52	4	对于 auto-vacuum 数据库,此域为数据库中根页编号的最大值,非 0.对于非 auto-vacuum 数据库,此域值为 0.
56	4	1=UTF-8 2=UTF16le 3=UTF16be
60	4	此域值供用户应用程序自由存取,其含义也由用户定义.
64	4	对于 auto-vacuum 数据库,如果是 Incremental vacuum 模式,此域值为 1.否则,此域值为 0.
68	4	未使用
72	4	未使用
76	4	未使用

表 2.2 文件头信息

所有的整数都是大端的。每次修改文件时，文件变化计数器都会增加。这个计数器可以让其他进程知道何时文件被修改了，其 cache 是否需要清理。最大嵌入有效负荷段是一页的所有可用空间，被标准 Btree（非叶数据）表的单独的一个所能使用的总量。值 255 代表 100%。默认情况下，一个单元(cell)的最大数目是被限制的，至少有 4 个单元才能填满一页。因此，默认的最大嵌入负荷段是 64。如果一页的有效负荷大于了最大有效负荷，那么剩下的数据就要被存储到溢出页。一旦分配了一

个溢出页，有可能会有许多数据也被转移到这个溢出页，但是不会让单元 cell 的大小小于最小嵌入有效负荷段。最小页有效负荷段与最小嵌入有效负荷段类似，但是它是应用于 LEAFDATA tree 中的叶节点。一个 LEAFDATA 的最大有效负荷段为 100%（或者是值 255），它不用再首部指定。

当应用程序调用 API sqlite3_open 打开数据库文件时，SQLite 就会读取文件头进行数据库的初始化。

```
int sqlite3BtreeOpen(        //打开数据库文件并返回B树对象
   sqlite3_vfs *pVfs,        //VFS使用B树
   const char *zFilename,    //包含B树数据库文件的名字
   sqlite3 *db,              //相关数据库句柄
   Btree **ppBtree,          //指向在此被写的新的B树对象
   int flags,                //选项标签
   int vfsFlags              //通过sqlite3_vfs.xOpen()标记
){
   ......
      rc = sqlite3PagerOpen(pVfs, &pBt->pPager, zFilename,EXTRA_SIZE, flags, vfsFlags, pageReinit);//
分配并且初始化一个新页面对象
      if( rc==SQLITE_OK ){
         rc = sqlite3PagerReadFileheader(pBt->pPager,sizeof(zDbHeader),zDbHeader);//读取文件头
      }
      ......
}
```

2.1.3 页面结构

Btree 的每个页被分为三部分：首部、单元（cell）指针数组、单元 cell 的内容和未分配区域。只有页 1 在页首部有 100 字节的文件头，如图 2.1.

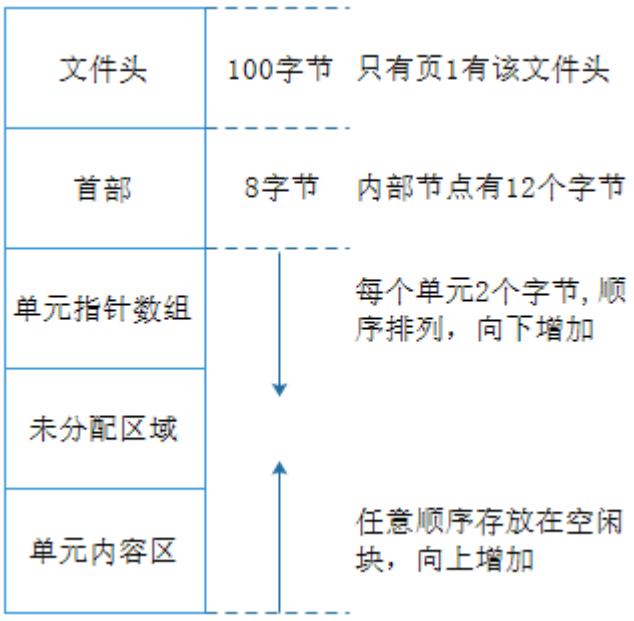


图 2.1 页结构

页首部如下表 2.3 所示：

偏移量	大小	说明
0	1	页标志位：1: intkey, 2: zerodata, 4: leafdata, 8: leaf
1	2	第 1 个自由块的偏移量字节
3	2	本页的单元数
5	2	单元内容区的起始地址（字节）
7	1	碎片的字节数
8	4	最右孩子的页号(the Ptr(n) value).仅内部页有此域.

表 2.3 页首部信息

标志位定义了这个 BTree 页的格式。leaf 标志意味着这个页没有孩子 children。zerodata0 数据表示只含有 key，没有数据；intkey 标志意味着 key 是一个整数，而且是被存储在单元 cell 首部的 key 大小处，而不是在有效负荷区域。单元指针数组从页首部开始。单元指针数组包含 0 个或多余 2 个字节的数字，这个数字代表单元内容区域中的单元内容从文件起始位置的偏移量。单元指针是有序的。系统保证空闲空间位于最后一个单元指针之后，这样可以保证新的单元可以很快的添加，而不用重新整理(defragment)该页。单元内容存储在页的末尾，且是向文件的起始方向增长。

在单元内容区域中的未使用的空间被链接到到链表空闲块(freeblocks)上。每一个 freeblock 至少有 4 个字节。第一个 freeblock 的偏移地址在页首部给出，freeblock 也是递增有序的。因为一个 freeblock 至少有 4 个字节，所有在单元内容区域的 3 个或少于 3 个字节的未用空间不能存放在 freeblock 链表上。这些 3 个或少于 3 个的空闲空间被称为碎片。所有碎片的总个数被记录,存储于页首部的偏移地址为 7 的位置。

2.1.4 单元的格式

单元是变长的字节串。一个单元存储一个有效载荷(payload)，它的结构如下：

大小	说明
4	左子的页码，如果有叶标志则省略。
var(1–9)	数据的字节数，若有 zerodata 标志则省略。
var(1–9)	关键字的字节数，若有 intkey 标志则是关键字本身。
*****	有效载荷，存储数据库中某个表的一条记录。
4	溢出页链表中第 1 个溢出页的页号。如果没有溢出页，无此域。

表 2.4 单元格式

对于内部页面，每个单元包含 4 个字节的左孩子页面指针；对于叶子页面，单元不需要孩子指针。接下来是数据的字节数、关键字的长度，单元格式：(a)一个单元的格式 (b)有效载荷的结构(图 2.3 中的 payload 不一定连续存放)。

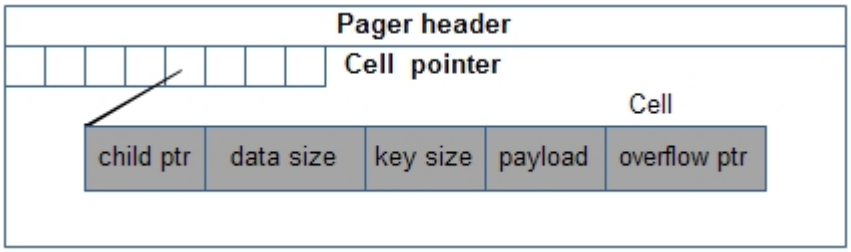


图 2.2 单元数据结构

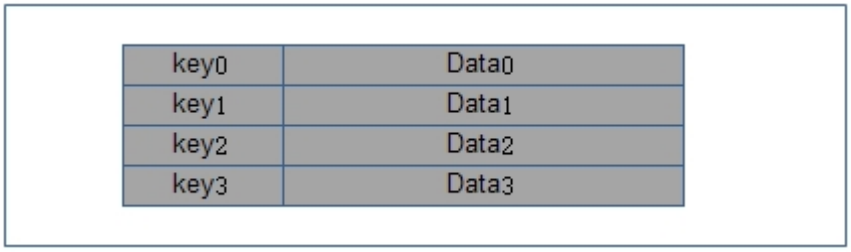


图 2.3 payload 结构

单元是可变长度的。单元被存储于页末尾的单元内容区域。指向单元的 cell 指针数组紧跟在页首部的后面。单元不必是连续或者有序的，但是单元指针是连续和有序的。单元内容充分利用了可变长度整数。可变长度整数是从 1 到 9 个字节，每个字节的低 7 位被使用。整个整数由 8 位的字节组成，其中第一个字节的第 8 位被清零。整数最重要的字节出现在第一个。可变长度整数一般不多于 9 个字节。作为一种特殊情况，第九个字节的所有 8 个位都会被认为是数据。这就允许了 64 位整数变编码为 9 个字节，如下表 2.4.

0x00	0x00000000
0x7f	0x0000007f
0x81 0x00	0x00000080
0x82 0x00	0x00000100
0x80 0x7f	0x0000007f
0x8a 0x91 0xd1 0xac 0x78	0x12345678
0x81 0x81 0x81 0x81 0x01	0x10204081

表 2.5 可变长整数的转换

2.1.5 空闲页面

空闲页面链表（freelist),在文件头偏移 32 的 4 个字节记录着空闲页面链的第一个页面,偏移 36 处的 4 个字节为空闲页面的数量。空闲页面链表的组织形式如下：

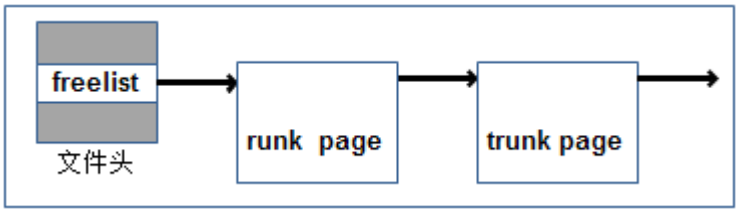


图 2.4 空闲页面

空闲页面分为两种页面：trunk pages（主页面）和 leaf pages(叶子页面)。文件头的指针指向空闲链表的第一个 trunk page，每个 trunk page 指向多个叶子页面。trunk page 的格式如下，从页面的起始处开始：(1)4 个字节，指向下一个 trunk page 的页面号；(2)4 个字节，该页面的叶子页面指针的数量；(3)指向叶子页面的页面号，每项 4 个字节。当一个页面不再使用时，SQLite 把它加入空闲页面链表，并不从本地文件系统中释放掉。当添加新的数据到数据库时，SQLite 就从空闲链表上取出空闲页面用来在存储数据。当空闲链表为空时，SQLite 就通过本地文件系统增加新的页面，添加到数据库文件的末尾。

2.1.6 溢出页面

小的元组能够存储在一个页面中，但是一个大的元组可能要扩展到溢出页面，一个单元的溢出页面形成一个单独的链表。每一个溢出页面(除了最后一个页面)全部填充数据(除了最开始处的 4 个字节),开始处的 4 个字节存储下一个溢出页面的页面号。最后一个页面甚至可以只有一个字节的数据，但是一个溢出页面绝不会存储两个单元的数据。溢出页面形成一个链表。除了最后一个页面，每个页面都已填满 4 字节大小的数据。最后一个页面可能少于 4 字节。溢出页面格式如下图 2.5。

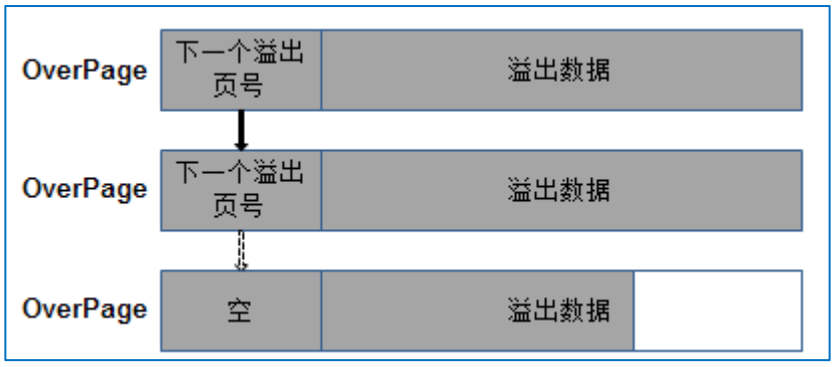


图 2.5 溢出页面链表

2.2 B 树文件子系统调用接口

游标函数	功能
sqlite3BtreeCursor	创建一个指向特定 B 树的游标.可以是读或写游标，但读游标和写游标不能同时在同一 B 树中
sqlite3BtreeTripAllCursors	遍历所有游标
sqlite3BtreeCursorSize	返回 BtCursor 对象的字节大小
sqlite3BtreeCursorZero	初始化将被转换成一个 BtCursor 对象的存储器
sqlite3BtreeMovetoUnpacked	游标指向一个 Key/pIdxKey 相对应的条目
sqlite3BtreeCursorHasMoved	游标是否移动,若出错返回错误代码
sqlite3BtreeFirst	游标移至 B 树第一条记录
sqlite3BtreeLast	游标移至 B 树最后一条记录
sqlite3BtreeNext	移动游标至当前游标所指记录的下一条
sqlite3BtreePrevious	移动游标至当前游标所指记录的前一条
sqlite3BtreeCloseCursor	关闭 Btree 游标
sqlite3BtreeClearCursor	清除当前游标位置
sqlite3BtreeSetCachedRowid	设置相同数据库文件中中每个游标的 cache 行号
sqlite3BtreeGetCachedRowid	返回游标的缓存的 rowid

表函数	功能
sqlite3BtreeCreateTable	在数据库中创建一个空 B 树,采用图格式(B+树)或索引格式(B 树)
sqlite3BtreeDropTable	删除数据库中的一个 B 树
sqlite3BtreeClearTable	删除 B 树中所有数据，但保持 B 树结构完整
sqlite3BtreeLockTable	获得表的根页 iTab 上的锁

记录函数	功能
sqlite3BtreeDelete	删除游标所指记录
sqlite3BtreeInsert	在 B 树的适当位置插入一条记录
sqlite3BtreeKeyFetch	用于快速访问 key
sqlite3BtreeDataFetch	用于快速访问 data
sqlite3BtreeDataSize	返回当前游标锁所指记录的数据字长度
sqlite3BtreeData	返回当前游标锁所指记录的数据
sqlite3BtreePutData	修改数据内容
sqlite3BtreeKeySize	返回当前游标锁时记录的关键字长度
sqlite3BtreeKey	返回当前游标锁时记录的关键字

页函数	功能
sqlite3BtreeSetPageSize	设置数据库页大小
sqlite3BtreeGetPageSize	返回数据库页大小
sqlite3BtreeMaxPageCount	设置数据库的最大页数
sqlite3BtreeLastPage	返回 B 树的最后一个页的大小
sqlite3BtreeGetReserve	页中未被使用的字节数

事务函数	功能
sqlite3BtreeBeginTrans	开始一个新事务
sqlite3BtreeCommitPhaseOne	两阶段提交的第一阶段
sqlite3BtreeCommitPhaseTwo	两阶段提交的第二阶段
sqlite3BtreeCommit	提交当前事务
sqlite3BtreeRollback	回滚当前进程中的事务
sqlite3BtreeBeginStmt	开始一个语句子事务
sqlite3BtreeIsrans	是否在事务中
sqlite3BtreeIsInReadTrans	在读或写事务中
sqlite3BtreeIsInBackup	回滚事务
sqlite3BtreeGetFilename	返回底层数据库文件中完整的路径名
sqlite3BtreeGetJournalname	返回数据库中日志文件的路径名
sqlite3BtreeSync	同步 btree 对应的数据库文件

配置管理函数	功能
sqlite3BtreeOpen	打开数据库文件并返回 B 树对象
sqlite3BtreeClose	关闭数据库并使所有游标无效
sqlite3BtreeSetCacheSize	控制页缓存大小
sqlite3BtreeSyncDisabled	如果在磁盘上没有 sync()同步函数，则不能同步
sqlite3BtreeSecureDelete	设置 BTS_SECURE_DELETE 标志
sqlite3BtreeSetAutoVacuum	设置数据库自动清理空闲页属性
sqlite3BtreeGetAutoVacuum	获取数据库是否是自动清理页
sqlite3BtreeegrityCheck	对 BTree 文件做一个完整性的检查
sqlite3BtreeegrityCheck	返回与 B 树相关的页.该函数仅用来测试和调试.
sqlite3BtreeCacheOverflow	此函数在游标上设置一个溢出页缓存标志
sqlite3BtreeSetVersion	在数据库头部设置"读版本"和"写版本"域
sqlite3BtreeSetSafetyLevel	改变磁盘数据的访问方式，以增加或减少数据库抵御操作系统崩溃或电源故障等损害的能力

第3章 B 树模块中函数调用流程

3.1 B 树模块中的数据结构

在 btreeInt.c 中定义了 Btree、BtCursor、BtShared、MemPage、CellInfo 等数据结构，B 树模块内部数据结构的设计都是按图 1.1 的体系结构完成的，层次非常清晰。SQLite 中有数不清的结构，B 树模块内部的数据结构关系如图 3.1。图中的箭头代表包含关系。

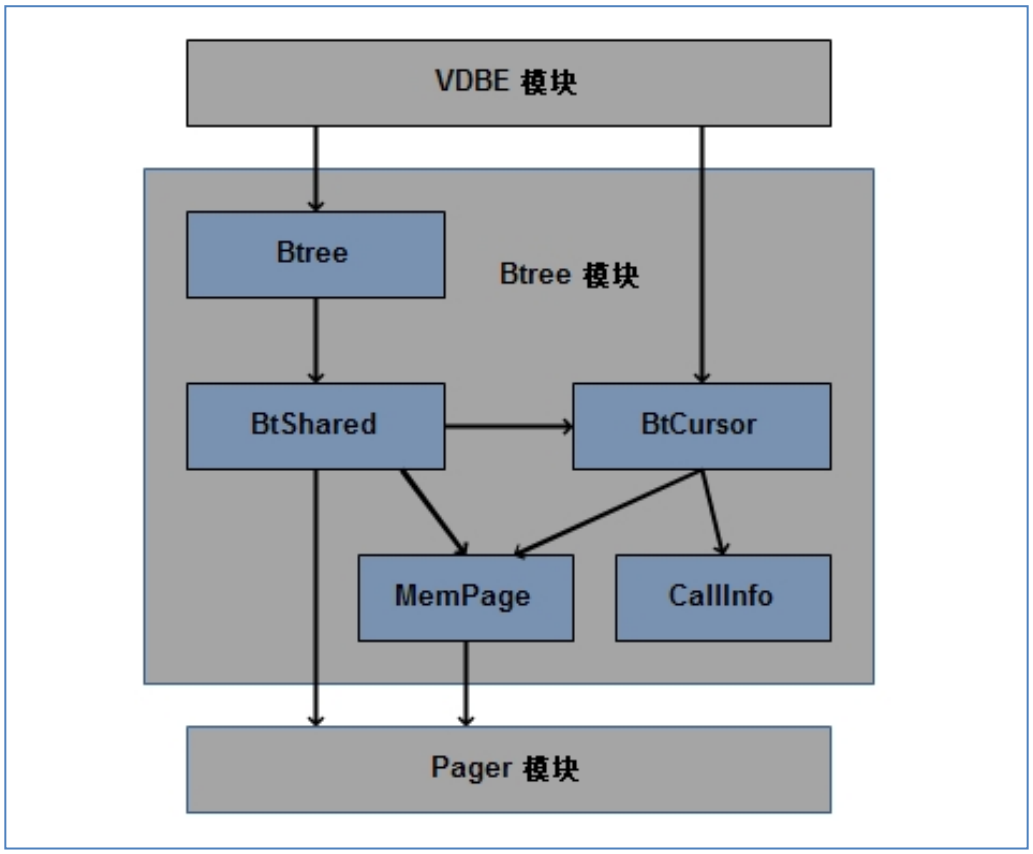


图 3.1

数据库文件包括 B 树模块中的 Btree 结构,VDBE 模块中的 VdbeCursor 结构包含 BtCursor 结构。BTShared 结构包含 Pager 模块中的 Pager 结构,MemPage 结构包含 Pager 模块中的 PgHdr 结构。这些结构存储关于页的信息，这些信息从原始的文件页中解码得到的。每当文件的一个页加载到内存，下面结构的一个实例也会增加并被初始化为 0。

1.MemPage 结构

每当文件的一个页加载到内存，下面结构的一个实例也会增加并被初始化为0。这个结构存储关于页的信息，这些信息从原始的文件页中解码得到的。

```
struct MemPage {
   u8 isInit;              //如果预先初始化则为真
   u8 nOverflow;           //在aCell[]中溢出单元体的数目
   u8 intKey;              // 如果intkey标志设置则为True
   u8 leaf;                //如果是叶子，则为True
   u8 hasData;             //页面已存数据则为真
   u8 hdrOffset;           //对page1为100其他为0
   u8 childPtrSize;        //如果是叶子则为0，如果不是叶子则为4
   u8 max1bytePayload;     /* min(maxLocal,127) */
   u16 maxLocal;           // BtShared.maxLocal或BtShared.maxLeaf的副本
   u16 minLocal;           //BtShared.minLocal或BtShared.minLeaf的副本
   u16 cellOffset;         //单元指针数组的偏移量，aData中第1个单元的指针
   u16 nFree;              //页上的可使用空间的总和（字节数）
   u16 nCell;              //本页的单元数, local and ovfl
   u16 maskPage;           //页偏移量的标记
   u16 aiOvfl[5];          //在aiOvfl-th的non-overflow 单元的前面插入i-th溢出单元
   u8 *apOvfl[5];          //指向溢出单元的指针
   BtShared *pBt;          //指向BtShared的指针，该页是BtShared的一部分
   u8 *aData;              //指向页数据的磁盘映像的指针
   u8 *aDataEnd;           //可用数据的最后的一个字节
   u8 *aCellIdx;           //单元的指针域
   DbPage *pDbPage;        //Pager的页句柄
   Pgno pgno;              //本页的编号
};
#define EXTRA_SIZE sizeof(MemPage) /*附加信息大小*/
```

2. BtLock结构

BtLock链表结构被存储于BtShared.pLock中。当游标在有根页面BtShared.iTable得表上被打开时，锁被添加(或从READ_LOCK升级到WRITE_LOCK)。当事务提交或回滚或者当btree句柄被关闭时，锁从这个列表上移除。

```
struct BtLock {                 //btree锁结构
   Btree *pBtree;               //btree句柄持有锁
   Pgno iTable;                 //表的根页
   u8 eLock;                    //读锁或者写锁
   BtLock *pNext;               //BtShared.pLock列表的后继
};
#define READ_LOCK       1
#define WRITE_LOCK      2
/*定义读锁为1，写锁为2 */
```

3. Btree结构

在数据库连接中，为每一个打开的数据库文件保持一个指向本对象实例的指针。这个结构对数据库连接是透明的。数据库连接不能看到此结构的内部，只能通过指针来操作此结构。有些数据库文件，其缓冲区可能被多个连接所共享。在这种情况下，每个连接都单独保持到此对象的指针。但此对象的每个实例指向相同的BtShared对象。数据库缓冲区和schema都包含在BtShared对象中。

sqlite3.mutex下可以访问这个结构的所有字段。当在引用的BtShared中存在游标时，pBt指针本身可能不会改变.它指向这Btree，因为那些游标必须通过btree来寻找BtShared结构和通常在没有持有sqlite3.mutex的情况下来这样处理。

```
struct Btree {
  sqlite3 *db;                  //拥有此btree的数据库连接
  BtShared *pBt;                //此btree的可共享内容
  u8 inTrans;                   //事务类型
  u8 sharable;                  //如果共享pBt给数据库连接db则返回true
  u8 locked;                    //当前数据库连接锁定了pBt则返回true
  int wantToLock;               //嵌套调用sqlite3BtreeEnter()的数量
  int nBackup;                  //读这btree备份操作的数量
  Btree *pNext;                 //相同数据库连接的其他可共享B树列表
  Btree *pPrev;                   //相同列表的返回指针
#ifndef SQLITE_OMIT_SHARED_CACHE
  BtLock lock;                    //对象用于锁第1页
#endif
};
```

如果启用了共享数据扩展,可能有多个用户来使用Btree结构。最多的可能是打开写事务,但任意数值都可能激活读事务。

```
#define TRANS_NONE  0
#define TRANS_READ  1
#define TRANS_WRITE 2
```

4. BtShared结构

BtShared对象的一个实例描述一个单独的数据库文件。一个数据库文件同时可以被多个数据库连接使用。当多个数据库连接共享相同的数据库文件时，每个连接有它自己的文件Btree，这些Btree指向同一个本BtShared对象。BtShared.nRef是共享此数据库文件的连接的个数。在BtShared互斥锁下可以访问这个结构的字段，除了nRef和pNext，他们在全局变量SQLITE_MUTEX_STATIC_MASTER互斥下访问。一

旦最初设置只要nRef>0，pPager字段不得被修改。 BtShared下pSchema字段可能设置一次并且只要nRef>0则不变。

```
struct BtShared { //该对象实例代表一个数据库文件，它包含了一个打开的数据库的所有信息
    Pager *pPager;                  //页缓冲区
    sqlite3 *db;                    //当前正在使用B树的数据库连接
    BtCursor *pCursor;              //包含当前打开的所有游标的列表
    MemPage *pPage1;                //数据库的第一个页
    u8 openFlags;                   //sqlite3BtreeOpen()的标签
#ifndef SQLITE_OMIT_AUTOVACUUM
    u8 autoVacuum;                  //auto-vacuum数据库可用返回true
    u8 incrVacuum;                  //incr-vacuum数据库可用返回true
#endif
  u8 inTransaction;                 //事务状态
  u8 max1bytePayload;               //对于一个1-byte的有效载荷，其单元的第一个字节的最大值
  u16 btsFlags;                     //布尔型参数。参阅下面的BTS_*宏
  u16 maxLocal;                     //在non-LEAFDATA表中本地有效载荷的最大值
  u16 minLocal;                     //在non-LEAFDATA表中本地有效载荷的最小值
  u16 maxLeaf;                      //在LEAFDATA表中本地有效载荷的最大值
  u16 minLeaf;                      //在LEAFDATA表中本地有效载荷的最小值
  u32 pageSize;                     //每页的字节数
  u32 usableSize;   //每页可用字节数.pageSize-每页尾部保留空间的大小,在文件头偏移20处。
  int nTransaction;                 //开放事务（读+写）的数量
  u32 nPage;                        //在数据库中页的数量
  void *pSchema;                    //指向由sqlite3BtreeSchema()所申请空间的指针
  void (*xFreeSchema)(void*);       //BtShared.pSchema的析构函数
  sqlite3_mutex *mutex;             //非递归互斥锁需要访问这个对象
  Bitvec *pHasContent;              //这个事务移动页面集合到空闲列表
#ifndef SQLITE_OMIT_SHARED_CACHE
  int nRef;                         //共享此数据库文件的连接的个数
  BtShared *pNext;                  //在可共享BtShared结构上的后继
  BtLock *pLock;                    //在shared-btree结构上持有的锁列表
  Btree *pWriter;                   //B树带有当前开放性写事务
#endif
  u8 *pTmpSpace;                    //BtShared.pageSize临时使用的空字节数
};
```

BtShared结构中btsFlags标签的宏定义.

```
#define BTS_READ_ONLY          0x0001   //优先文件是只读的
#define BTS_PAGESIZE_FIXED     0x0002   //固定页面大小
#define BTS_SECURE_DELETE      0x0004   //编译指令secure_delete启用
#define BTS_INITIALLY_EMPTY    0x0008   //在事务的开始数据库是空
#define BTS_NO_WAL             0x0010   //不打开write-ahead-log文件
#define BTS_EXCLUSIVE          0x0020   //pWrite独占锁
#define BTS_PENDING            0x0040   //等待读锁清除
```

5. CellInfo结构

CellInfo结构的实例用来保存单元头信息。parseCellPtr()负责根据从原始磁盘页中取得的信息填写此结构。

```
typedef struct CellInfo CellInfo;
```

```
struct CellInfo {   //该结构的实例用来保存单元头信息
    i64 nKey;              //关键字的字节数。如果intkey标志被设置，此域即为关键字本身。
    u8 *pCell;             //指向单元内容的指针
    u32 nData;             //数据的字节数
    u32 nPayload;          //有效载荷的总量
    u16 nHeader;           //记录头的字节数
    u16 nLocal;            //有效载荷局部持有的量
    u16 iOverflow;         //溢出页链表中第1个溢出页的页号。如果没有溢出页，无此域。
    u16 nSize;             //单元数据的大小（不包括溢出页上的内容）
 };
```

Sqlite的B树结构的最大深度，任何一个比此深度更大的B树都是被视为无效的，这个值的计算方法是基于一个最大2^31页的数据库大小，其根节点有2分支、所有其他内部节点3个分支。

```
#define BTCURSOR_MAX_DEPTH 20
```

6.BtCursor结构

游标是指向一个特定条目的指针，它是在一个数据库文件的特定Btree中。条目由MemPage和MemPage.aCell[]的下标确定。单个数据库文件可被多个数据库连接共享，但游标不能被共享。每个游标与特定的数据库连接相关联来确定是否为BtCursor.pBtree.db.

```
struct BtCursor {                  //B树上的游标，游标是指向一个特定条目的指针
   Btree *pBtree;                  //属于这个B树的游标
   BtShared *pBt;                  //该游标指向BtShared
   BtCursor *pNext, *pPrev;    //形成一个所有游标的链表
   struct KeyInfo *pKeyInfo; //参数传递给比较函数
#ifndef SQLITE_OMIT_INCRBLOB
   Pgno *aOverflow;                //缓存溢出的页面位置
#endif
   Pgno pgnoRoot;                  //此Btree的根页页号
   sqlite3_int64 cachedRowid; //下一个rowid的缓存，0表示无效
   CellInfo info;                  //当前指向的单元（cell）的解析结果
   i64 nKey;                       //pKey的大小或最后的整数键值
   void *pKey;                     //游标最后已知的位置的键值
   int skipNext;                   //如果为负Prev()无操作，如果为正Next()无操作
   u8 wrFlag;                      //写标签，如果可写为真
   u8 atLast;                      //指针指向最后条目
   u8 validNKey;                   //如果info.nKey有效为真
   u8 eState;                      //CURSOR_XXX常量之一
#ifndef SQLITE_OMIT_INCRBLOB
   u8 isIncrblobHandle;            //如果游标是一个incr.io句柄则为真
#endif
   u8 hints;                       //游标的掩码位，通过CursorSetHints()设置
   i16 iPage;                      //当前页在apPage中的索引
   u16 aiIdx[BTCURSOR_MAX_DEPTH];     //apPage[i]中的当前索引。
                                      注：单元指针数组中的当前下标。
   MemPage *apPage[BTCURSOR_MAX_DEPTH];       //从根页到本页的所有页
};
```

BtCursor.eState(即是BtCursor结构的游标状态)的可能值如下：

1.CURSOR_VALID:游标指向一个有效条目。

2.CURSOR_INVALID:游标指向一个无效条目。因为表为空或BtreeCursorFirst()没有被调用。

3.CURSOR_REQUIRESEEK: 游标打开的表仍然存在,但游标最后一次使用后已被修改。游标位置保存在BtCursor的变量pKey和nKey中。当一个游标在这个状态时,restoreCursorPosition()可以尝试寻找保存的游标位置。

4. CURSOR_FAULT:一个不可恢复的错误(一个I/O错误或malloc失败)发生在一个不同的连接,这种连接共享带有这个游标的BtShared缓存。错误导致缓存处于不一致的状态。其他都不会拥有这个游标。任何试图使用该游标的都应该返回错误并存

储在BtCursor.skip中.

```
#define CURSOR_INVALID          0
#define CURSOR_VALID            1
#define CURSOR_REQUIRESEEK      2
#define CURSOR_FAULT            3
```

如果数据库页是PENDING_BYTE 崩溃，则该页不会在被使用。

```
# define PENDING_BYTE_PAGE(pBt) PAGER_MJ_PGNO(pBt)
```

以下的宏定义了一个数据库页的pointer-map入口位置。第一个参数为每个数据库页上可用的字节数(通常是1024年)。第二个是在指针位图中的页码。PTRMAP_PAGENO返回数据库pointer-map页面的页码,存储所需的指针。PTRMAP_PTROFFSET返所要的映射条目的偏移量。如果使用pgno参数传递给PTRMAP_PAGENO是pointer-map页面，则返回pgno。(pgno == PTRMAP_PAGENO(pgsz pgno))可以用来测试pgno是否是pointer-map页面。PTRMAP_ISPAGE实现这个测试。

```
#define PTRMAP_PAGENO(pBt, pgno) ptrmapPageno(pBt, pgno)
#define PTRMAP_PTROFFSET(pgptrmap, pgno) (5*(pgno-pgptrmap-1))
#define PTRMAP_ISPAGE(pBt, pgno) (PTRMAP_PAGENO((pBt),(pgno))==(pgno))
```

指针位图的目的是容易从一个文件中的位置移动页面到另一个位置来作为autovacuum的一部分。移动一个页面时,在父节点中的指针必须更新。指针映位图用于快速查找父页面。

1.PTRMAP_ROOTPAGE: 数据库是一个根页页。

2.PTRMAP_FREEPAGE: 数据库的页面是一个未使用的(空闲)页面。

3.PTRMAP_OVERFLOW1: 数据库溢出页面列表的第1页，页码标识包含指向溢出页的单元。

4.PTRMAP_OVERFLOW2: 数据库溢出页面列表的第2页，页码标识溢出页中指向该页的前驱页面。

5.PTRMAP_BTREE: 数据库是一个非根btree页面页。页码标识btree父页面。

```
#define PTRMAP_ROOTPAGE 1
#define PTRMAP_FREEPAGE 2
#define PTRMAP_OVERFLOW1 3
#define PTRMAP_OVERFLOW2 4
#define PTRMAP_BTREE 5
```

宏 ISAUTOVACUUM 用在 balance_nonroot() 中，用于确定数据库是否支持 auto-vacuum。

```
#ifndef SQLITE_OMIT_AUTOVACUUM
#define ISAUTOVACUUM (pBt->autoVacuum)
#else
#define ISAUTOVACUUM 0
#endif
```

7. IntegrityCk结构

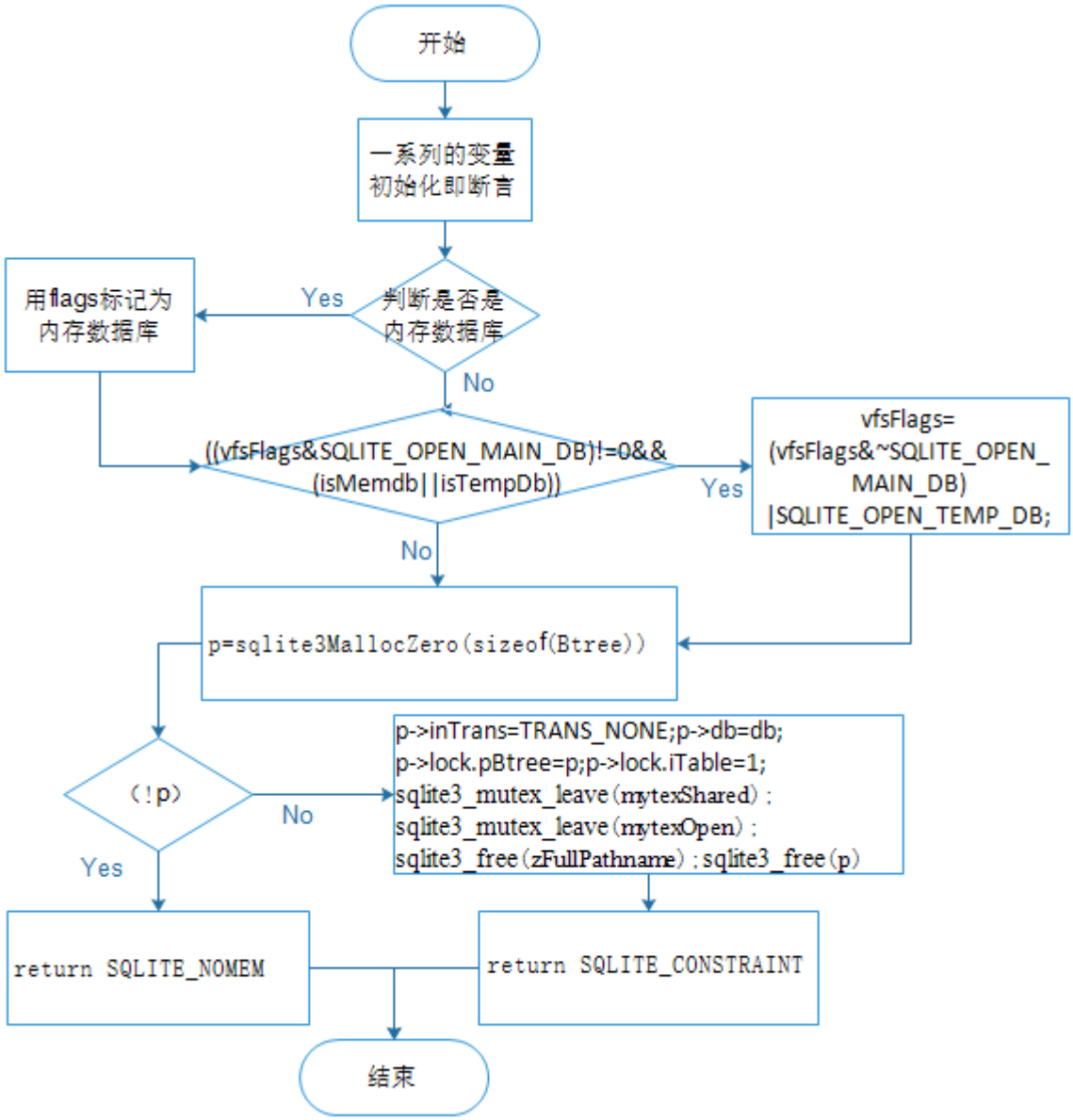
IntegrityCk结构是通过所有的完整性检查例程传递跟踪一些全局状态的信息。aRef[]数组位数据库的每个页分配1位。它是完整性检查的程序，数据库中使用的每个页面都设置了相应的位。这允许完整性检查检测使用两次和孤立的页面 (这两个表明无效)。

```
typedef struct IntegrityCk IntegrityCk;
struct IntegrityCk {
    BtShared *pBt;              //B树正在检查数据完整性
    Pager *pPager;              //相关页面调度程序，也可以通过pBt->pPager访问
    u8 *aPgRef;                 //在db中每页1位
    Pgno nPage;                 //在数据库中页的数量
    int mxErr;                  //当这个变量达到零的时候，停止积累错误
    int nErr;                   //当前已经写到zErrMsg中的信息数量
    int mallocFailed;           //一个内存分配发生错误
    StrAccum errMsg;            //收集错误信息文本
};
```

3.2 B 树模块中部分函数的流程图

3.2.1 sqlite3BtreeOpen 函数

sqlite3BtreeOpen 函数功能是打开数据库文件并返回 B 树对象，其流程图如下：



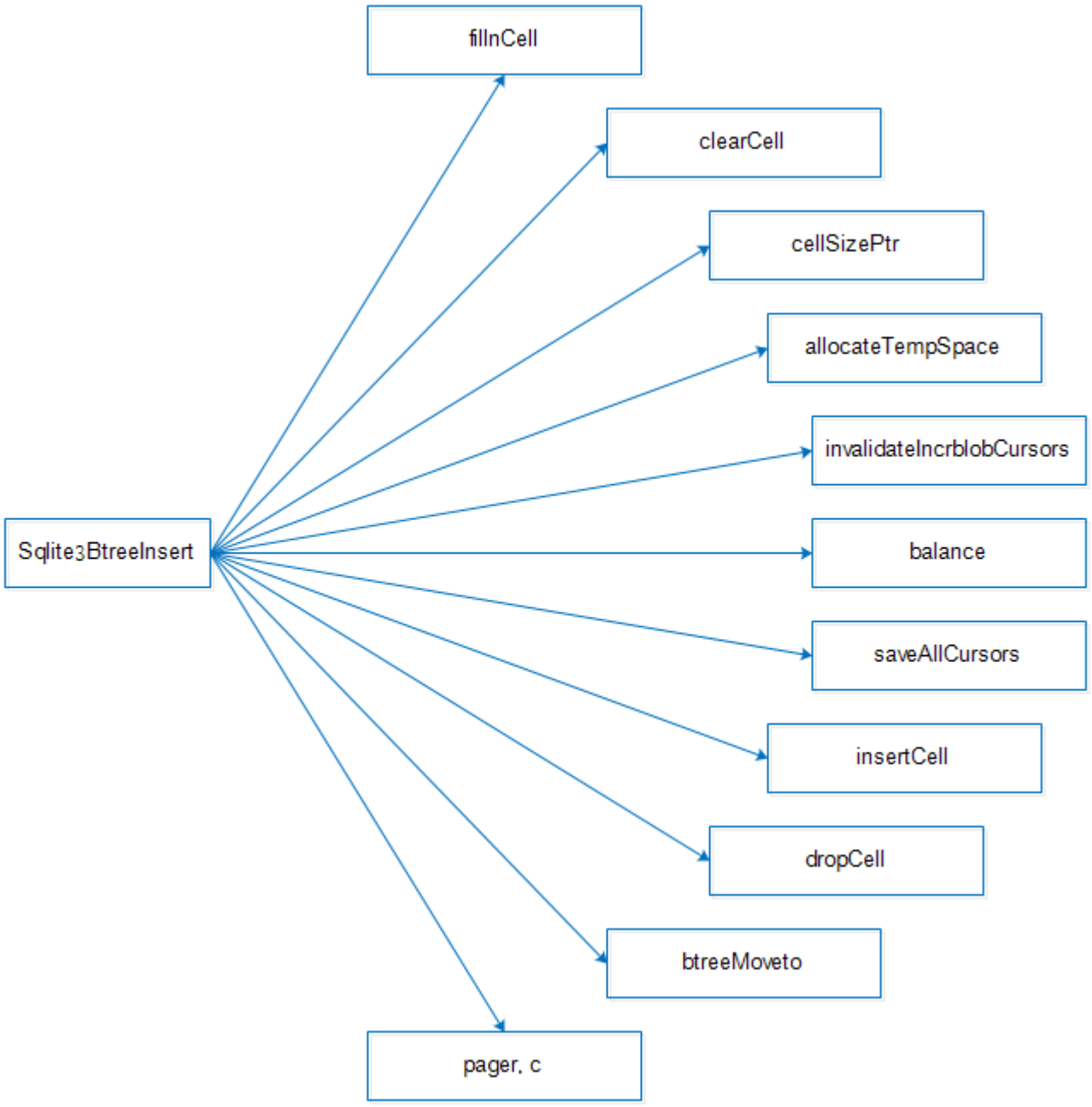
zFilename 是这个数据库文件的名字.如果 zFilename 为空,则将创建一个临时数据库.这个临时数据库在内存中唯一的,或用了基于磁盘的内存缓存，无论哪种方式当 sqlite3BtreeClose()被调用的时候,这个临时数据库将自动删除.如果 zFilename 是 ":memory:"那么内存数据库将会创建并且关闭时自动销毁. “flags”参数是一个可能包

含的位掩码位 BTREE_OMIT_JOURNAL、BTREE_MEMORY. 如果数据库已经在相同的数据库连接中打开了并且在共享缓存模式下,然后用一个打开将会失败返回 SQLITE_CONSTRAINT 错误.在同一数据库连接中我们不能允许两个或多个 BtShared 对象,因为这样做会导致锁问题.

3.2.2 sqlite3BtreeInsert 函数

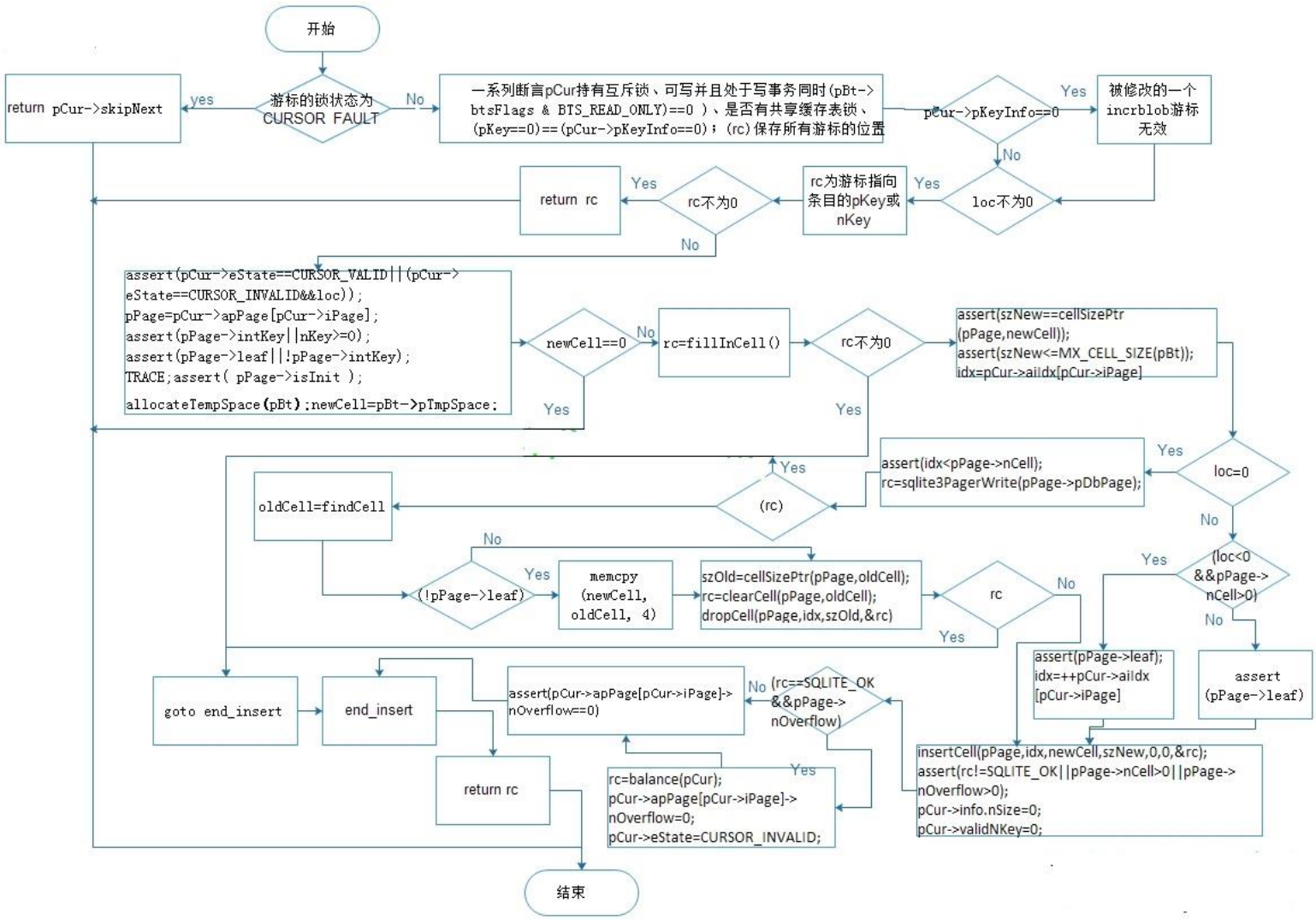
sqlite3BtreeInsert 函数的功能是插入新记录到 B 树，sqlite3BtreeInsert 函数中调用的相关函数如下图。

对于 sqlite3BtreeInsert()这样的修改数据库的函数，在修改数据库之前必须开始



写事务.如果没有写事务则下面的函数都不会有任何作用.sqlite3BtreeCreateTable(); sqlite3BtreeCreateIndex();sqlite3BtreeClearTable();sqlite3BtreeDropTable(); sqlite3BtreeInsert();sqlite3BtreeDelete();sqlite3BtreeUpdateMeta().

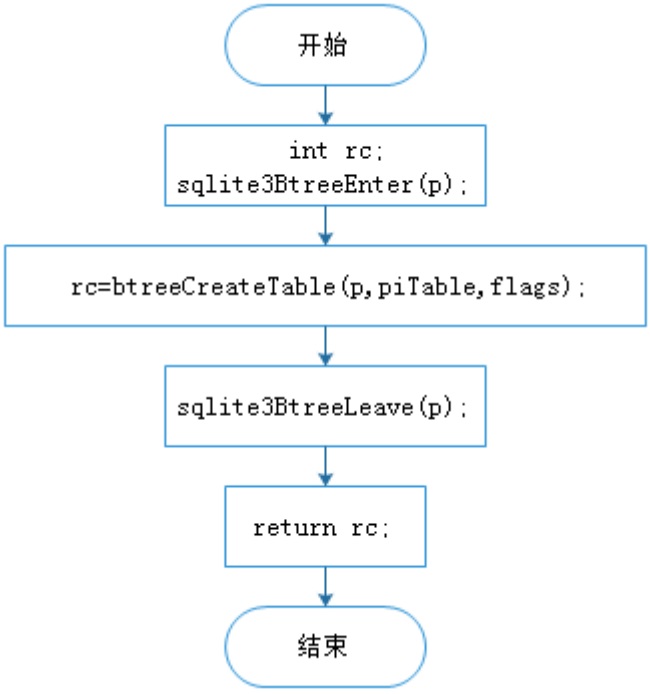
插入一条新记录到 B 树.关键字由(pKey,nKey)给出,数据域有(pData,nData)给出. 游标仅仅被用作定义记录应该插入到什么表中,其他游标指向任意位置. 如果 seekResult 参数非零,那么一个成功的调用 MovetoUnpacked()寻找(pKey,nKey)已经被执行的游标 pCur. seekResult 是搜索返回的结果(如果 pCur 指向一个比(pKey,nKey)还小的条目则是一个负值,或如果 pCur 指向一个比(pKey,nKey)更大的值则该值为正值.). 如果 seekResult 参数是非零,那么调用者保证游标 pCur 向一行被复写的现有副本.如果 seekResult 参数是 0,那么游标 pCur 可能指向任何条目或不指向任何条目,所以在插入键值之前这个函数必须寻找游标.



insertCell()函数是在 pPage 的单元索引 i 处插入一个新单元.pCell 指向单元的内容.这时，如果没有错误发生,且 pPage 有溢出单元,调用 balance()重新分布树内的单元.因为 balance()可能移动游标,所以清零 BtCursor.info.nSize 和 BtCursor.validNKey 两个变量. 当 balance()用于使 BtCursor 内容无效时,以前 SQLite 的版本调用 moveToRoot()来移动游标回到根页面用来使 BtCursor.apPage[]和 BtCursor.aiIdx[]的内容无效.相反,将游标状态设置为“invalid”.这使得常见的插入操作更快. 这里有一个微小但重要的优化.当用一个游标插入多个记录到一个 intkeyB 树时使(在处理一个"INSERT INTO ... SELECT"语句),如果可能的话,使游标指向表中最后一个条目这是有利的.如果游标指向表中最后一个条目,下一行插入有一个比已存在的最后键值要大的整数键值,它可以在没有游标的的行插入.这可以极大地提高性能.

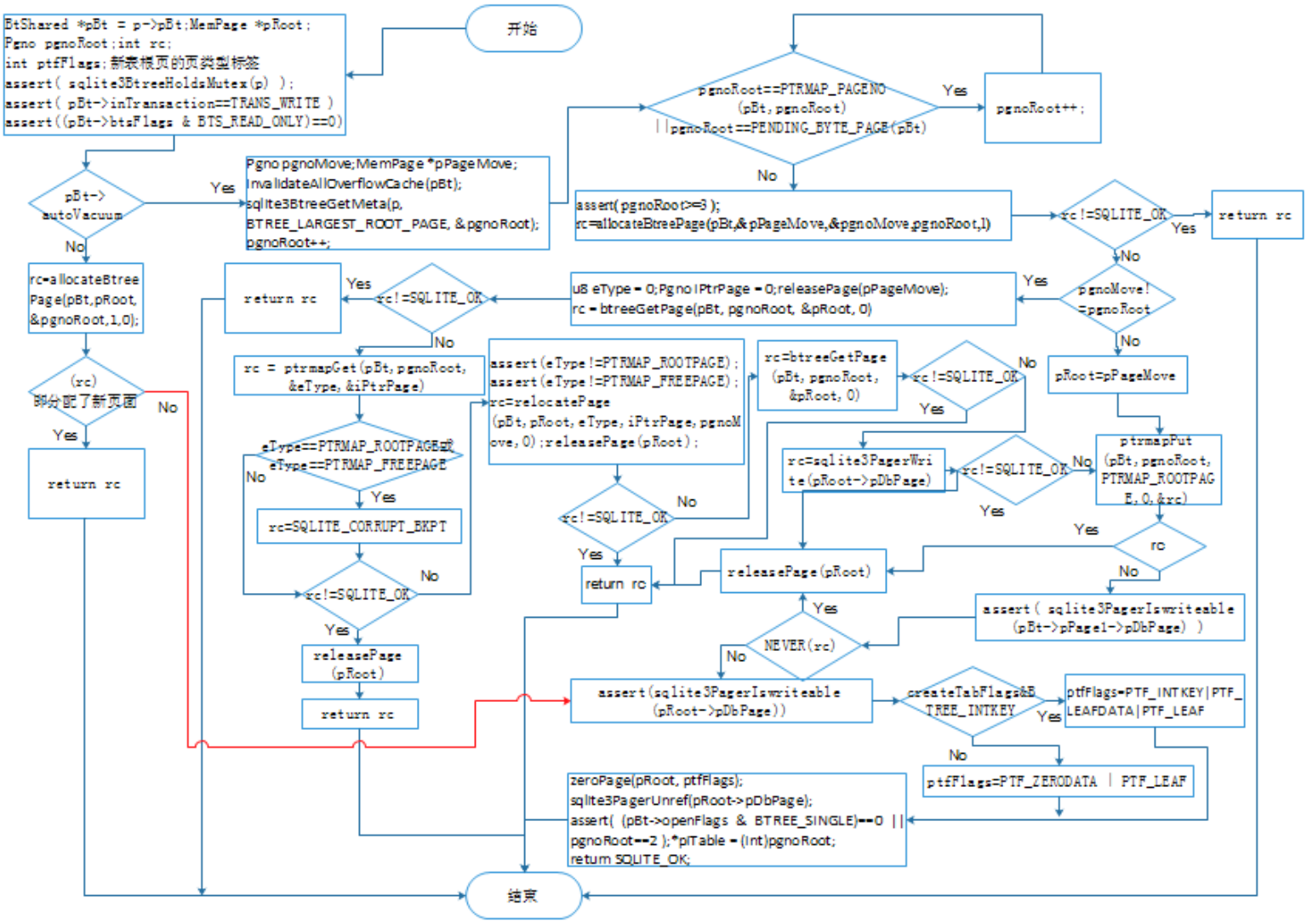
3.2.3 sqlite3BtreeCreateTable 函数

sqlite3BtreeCreateTable 函数功能是在数据库中创建一个空 B 树,采用图格式(B+树)或索引格式(B 树),创建的流程如下.



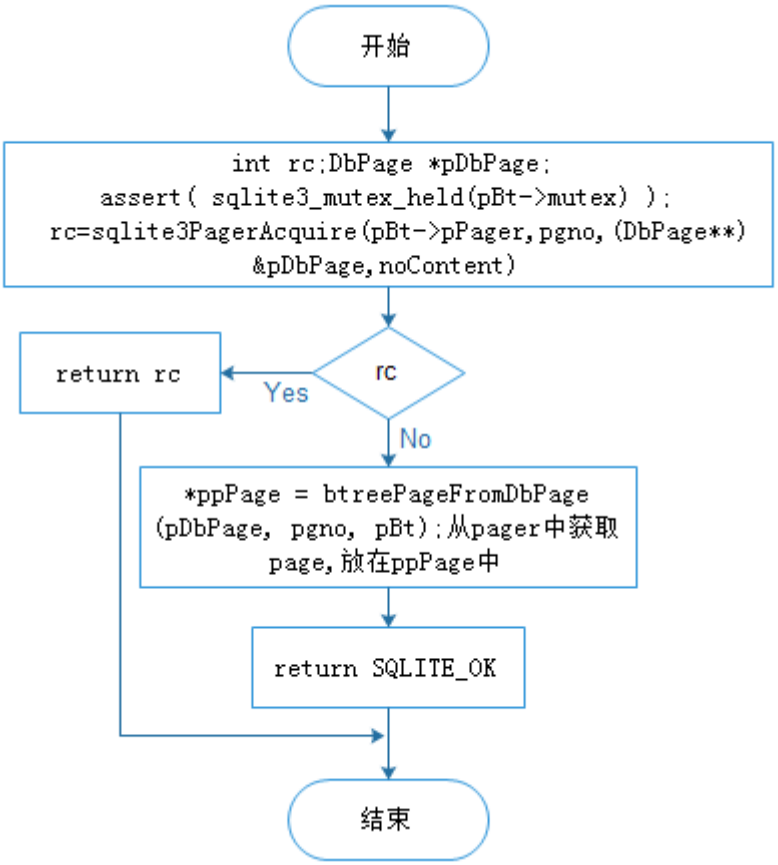
在创建 B 树过程中要调用 btreeCreateTable()函数。btreeCreateTable()函数功能是创建新的 B 树表. 移动现有数据库为新表的根页面腾出空间用新根页更新映射寄存器和 metadata. 新表根页页码写到*piTable 中, 类型由标志参数决定,标志参数只有以下的可用.其他标志可能没有作用. 标签 BTREE_INTKEY|BTREE_LEAFDAT 用于带

有列 id 键值的 SQL 表. 标签 BTREE_ZERODATA 用于 SQL 索引。btreeCreateTable() 函数流程图分别如下。

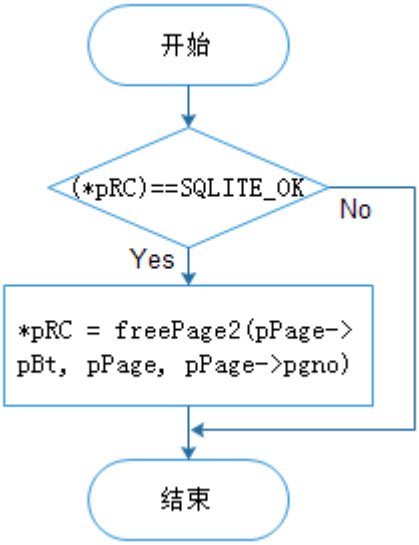


3.2.4 btreeGetPage 函数

btreeGetPage()函数的功能是从页管理器得到一个页。如果需要,则初始化 MemPage.pBt 和 MemPage.aData 的元素. 如果无内容标签设定了,那意味着我们将不关心此时的页面内容.所以不要去磁盘获取内容.只需在内容中填写使用零即可.如果以后我们在这个页面上调用 sqlite3PagerWrite(),这意味着我们已经开始关注内容,并应出现在该点的磁盘读取.

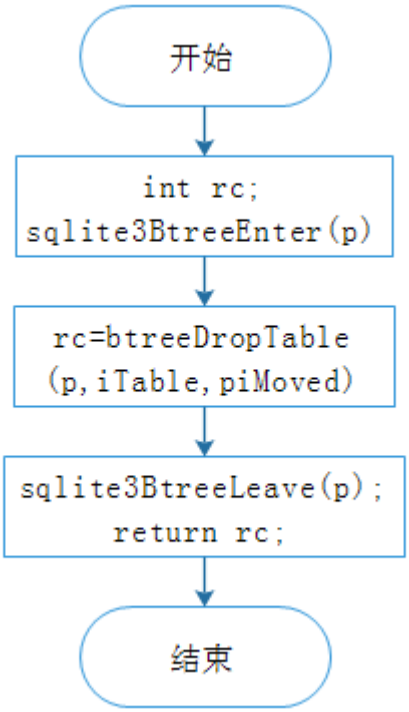


freePage()功能释放页。



3.2.5 sqlite3BtreeDropTable 函数

sqlite3BtreeDropTable()函数功能是删除数据库中的一个 B 树，流程如下.



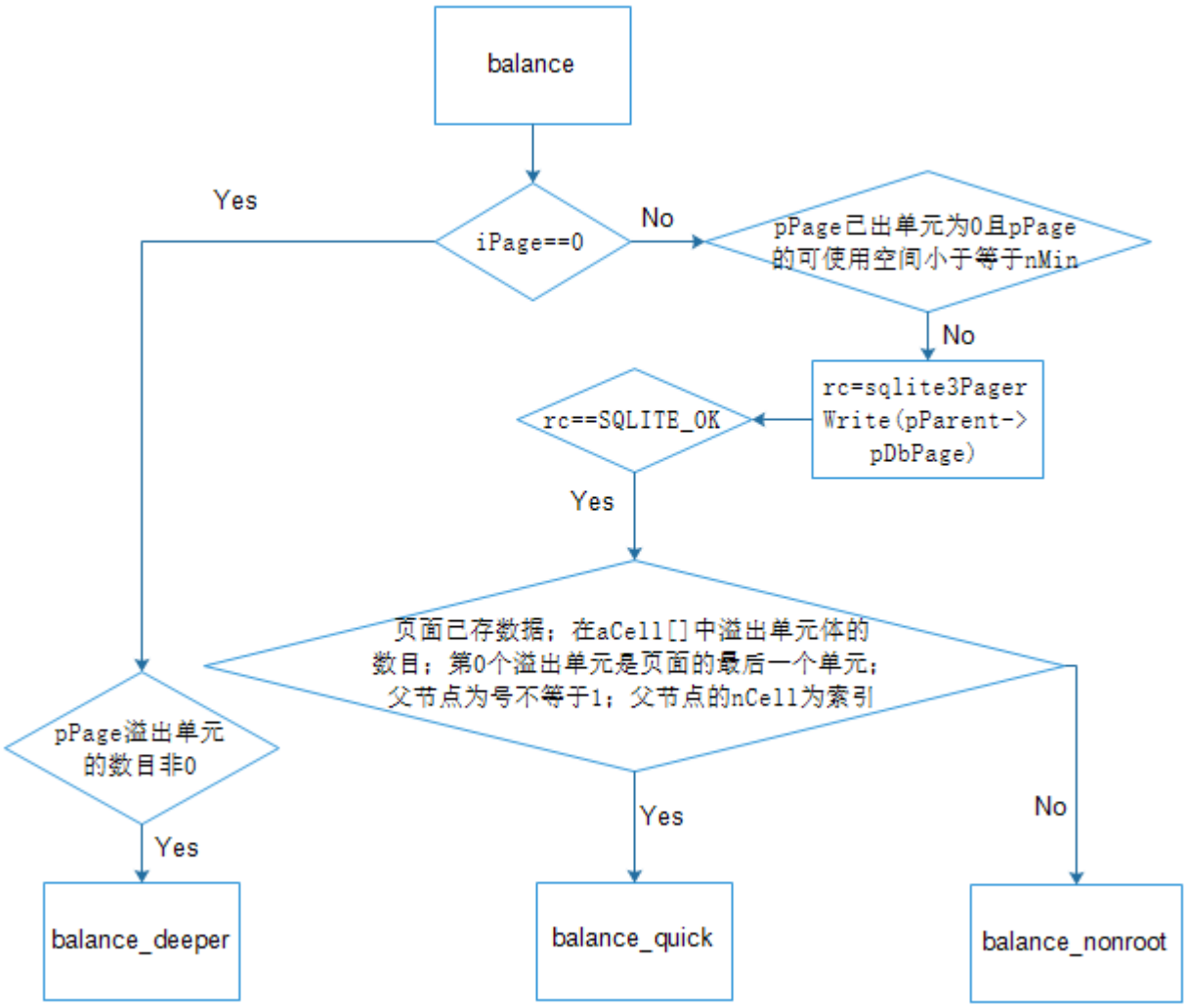
其中流程中 btreeDropTable()功能是清除表上的所有信息并且添加标的根到空闲列表.除此之外,在页 1 上的表的根从不加入到空闲列表. 在表上如果有任何开放性游标，那么这个函数失败返回 SQLITE_LOCKED. 如果 AUTOVACUUM 可用并且 iTable 上的页不是数据库文件的最后根页,那么数据库文件中最后的根页将被移动到被 iTable 占用的位置并且上次被根页占用的加到空闲列表而不是 iTbale.也就是说,所有的根页数据库文件的开始,这对于 AUTOVACUUM 正常工作是有必要的.*piMoved 被设置为移动之前文件中是最后根页的页码.如果没有页要移动,则*piMoved 设为 0.最后的根页是记录在 meta[3]中并且 meta[3]的值在这个过程中被更新.

3.2.6. balance 函数

balance()中的以下参数确定在一个平衡操作中有多少相邻页面参与进来.NN 是参与平衡操作的页面相邻页面的数量.NB 的所涉及的页面总数,包括目标页面和 NN 的相邻页面.NN 的最小值是 1.增加 NN 使之大于 1(2 或 3)能够改善 SELECT 和 DELETE 性能,以换取更大的插入和更新性能的退化.NN 值似乎给了最好的结果.下面的参数确定在平衡操作里面涉及多少相邻的页面,数量记为 NN.NB 是参与的页的

总数量.NN 的最小值是 1.使 NN 为 2 或 3, 能够改善 SELECT 和 DELETE 性能.

```
#define NN 1              //pPage 两侧相邻的页数
#define NB (NN*2+1)       //在平衡中涉及的总页数
```



以上流程图中 iPage 为当前页在从根页到本页的所有页中的索引.

balance_quick()这个函数处理处理一些特殊情况,一个新条目被插入到树的最右端.换句话说,当新条目将成为树中最大的条目.而不是试图平衡最右边的 3 个叶页面,添加一个新页面的右边,放一个新条目在这个页面中.这使得树的右边不平衡的.但奇怪的是,我们将插入新的条目到最后,所以很快将空页面添满. pPage 是叶子页面,它是树上最右边的页面.pParent 是它的父节点.pPage 必须有单独的溢出条目也页面上最右边的条目. pSpace 缓冲区用于存储将插入 pParent 的临时副本的单元.这样一个单元包含在一个可变长度的整数后的 4 字节页码组成. 换句话说,最多 13 字节.因此,pSpace 缓冲区必须要至少 13 个字节大小.

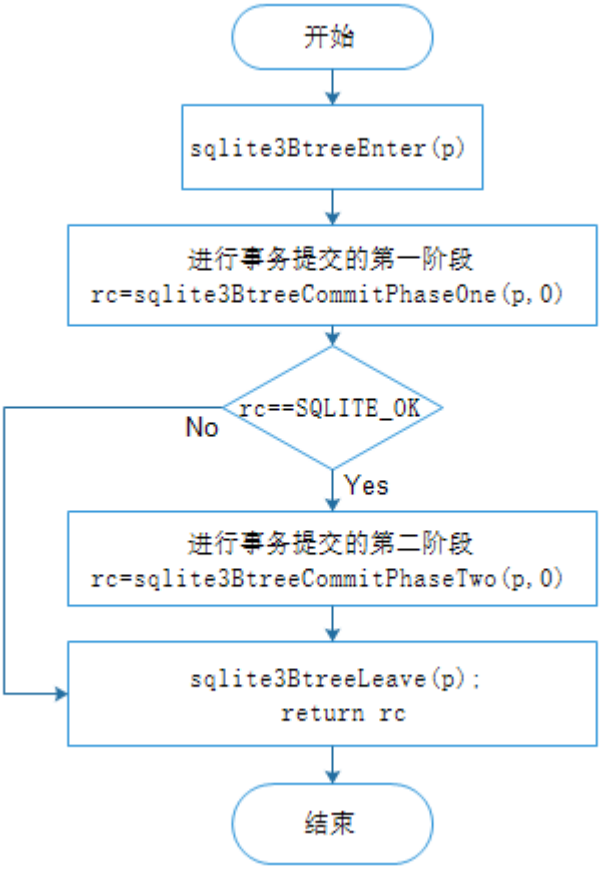
balance_nonroot 函数在 pParent（一个 B 树内部(非叶)页，以下简称“页面”）的第 iParentIdx 孩子上重新分配单元和达到 2 个兄弟节点,这样对所有页面都有相同数量的自由空间.通常在页面两侧的一个兄弟节点是平衡的,如果页面的父节点是第一个或最后一个孩子则兄弟节点可能来自一侧.如果页面已经少于 2 兄弟 (如果页面是一个根或根的子页面,有些移动可能发生)那么所有可用的兄弟姐妹参与平衡.页面的兄弟的数量可能会增加或减少一个或两个,尽量保持页面几乎填满但不完全为满.注意,当调用这个函数,在页面上的一些单元可能不完全是存储在 MemPage.aData[].如果页面过满可能会发生这种情况.这个函数分配给页面的所有单元及返回之前，其兄弟会写入 MemPage.aData[].在平衡页面和其兄弟过程中,单元可能从父页面(pParent) 插入或删除.插入可能导致父页面过度满.如果这发生了,调用函数负责调用正确的平衡函数来解决这个问题(见 balance()函数).如果这个函数失败,它可能使数据库在一个崩溃的状态.因此如果这个函数失败,数据库应该进入事务的回滚.balance_nonroot()的第三个参数 aOvflSpace 是一个指针,指向一个足够存放页的缓冲区.如果单元正插入父页面(pParent),该父页面变得过度满,那么这个缓冲区用于存储父页面的溢出单元.因为这个函数最多插入四个独立的单元进入父页面,并且存储在一个内部节点中的单元的最大值总是小于 1/4 的页面大小,这个 aOvflSpace[]缓冲区是保证溢出单元足够大.如果 aOvflSpace 没有设定指针,则函数返回 SQLITE_NOMEM.

balance_deeper 函数当 B 树结构的根页过度满(有一个或多个溢出页)时,用该函数进行调整平衡.一个新的孩子页将被分配并且当前根页的内容包括溢出单元将被拷贝到新的孩子节点.根页被重写使之为空页,最右孩子指针指向该新的孩子页.在返回之前,所有 pointer-map 条目对应新子页面包含指针的页面被更新.条目对应的根页的新右子结点指针根也更新.如果成功,*ppChild 将包含一个对孩子页的引用并返回 SQLITE_OK.在这种情况下,调用者需要在*ppChild 上对 releasePage()调用恰好一次.如果出现错误,返回一个错误代码并且 ppChild 设置为 0.

3.2.7 sqlite3BtreeCommit 函数

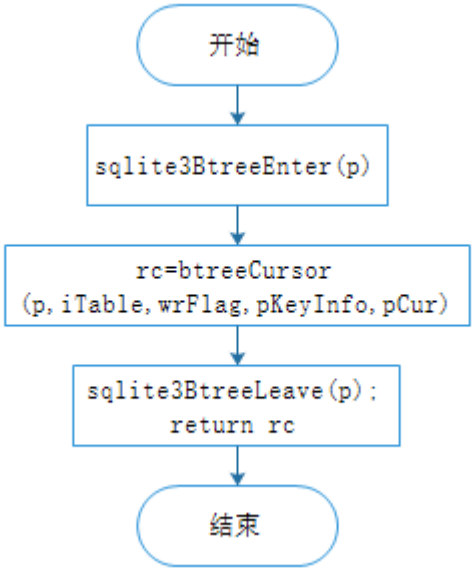
sqlite3BtreeCommit()函数的功能是事务的提交。提交阶段分为 2 部分,第一部分为 sqlite3BtreeCommitPhaseOne，第二部分是 sqlite3BtreeCommitPhaseTwo()。第一阶段调用后才能调用第二阶段.第一阶段完成写信息到磁盘.第二阶段释放写锁,若无活动游标,释放读锁.提交分为 2 个阶段是为了保证所有节点在进行事务提交时保持一

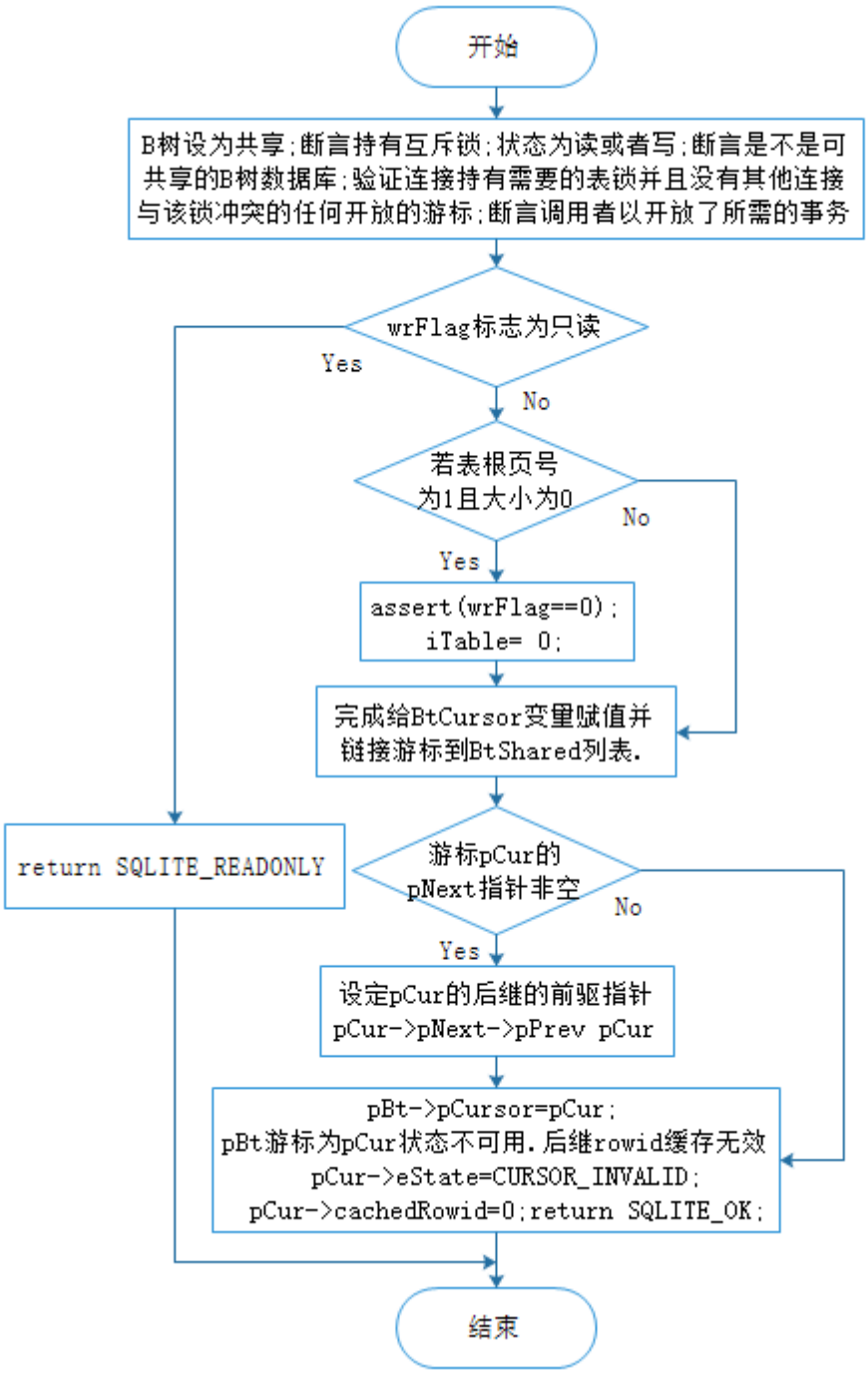
致性.在分布式系统中,每个节点虽然可以知晓自己的操作时成功或者失败,却无法知道其他节点的操作的成功或失败.当一个事务跨越多个节点时,为了保持事务的 ACID 特性,需要引入一个作为协调者的组件来统一掌控所有节点(称作参与者)的操作结果并最终指示这些节点是否要把操作结果进行真正的提交(比如将更新后的数据写入磁盘等等).因此,二阶段提交的算法思路可以概括为： 参与者将操作成败通知协调者,再由协调者根据所有参与者的反馈情报决定各参与者是否要提交操作还是中止操作。函数 sqlite3BtreeCommit()的流程如右图。



3.2.8 sqlite3BtreeCursor 函数

sqlite3BtreeCursor() 创建一个指向特定 Btree 的游标.游标可以是读游标,也可以是写游标,但是读游标和写游标不能同时在同一个 Btree 中存在.其中 btreeCursor()函数是为 Btree 创建一个新的游标，其流程如右图。





如果请求创建一个只读游标,那么数据库上至少打开一个只读事务.如果被请求的是写游标,必须有一个打开的写事务.如 wrFlag== 0,则游标仅能用于读取，如果 wrFlag== 1 并且其他条件也满足,则游标可用于读或者写.允许写操作要有以下条件：

1.wrFlag==1 的游标必须已被打开;2.共享相同的页缓存，不是 READ_UNCOMMITTED 状态,wrFlag==0 时,游标可能不是打开状态;3.数据库必须是可写的(而不是只读介质);4.必须有一个活动的事务。假设在调用这个程序之前,sqlite3BtreeCursorZero()已经被调用,并用 pCur 初始化了内存空间。

第 4 章 总 结

通过对 Sqlite 数据库源码的 B 树模块的分析，进一步理解了 Sqlite 的 B 树模块是如何实现的以及如何使用数据结构—B 树实现数据在磁盘上的存储，等底层的实现机制。从总体上来说，Sqlite 设计数据结构时将 Btree 的实现分为两个层次 Btree 结构和 BtrShared 结构。在 Sqlite 中每个数据库连接都有一个独立的 Btree 结构，多个数据库连接共享一个 BtrShared 结构。B 树结构中常用的数据结构还有 BtCursor、MemPage、CellInfo、BTLock、IntegrityCk 等结构。

从内部的存储细节上看，Btree 模块将 sqlite 数据库中的数据存储到页中，通过 Pager 管理页的各种操作，而数据库中的数据是分为很多页面进行存储的，而 SQLite 又将页以 B 树的形式存储在磁盘上。具体的存储结构包括 B 树中的节点结构、数据库首部信息、页面结构、单元的结构、空闲页面结构、溢出页面结构等。B 树模块中还包含了大量的文件子系统调用接口，这些调用接口函数的功能涉及表函数、记录函数、页面函数、游标函数、事务函数、配置管理函数等。在 btree.c 文件中包含了大量的函数，用于实现 B 树的相关功能的函数，包括实现创建 B 树、删除 B 树、创建表、插入记录、删除记录、平衡 B 树结构、游标移动、事务处理等的函数。这些函数最复杂的应该是 balance 函数。它又包含 balance_quick、balance_nonroot、balance_deeper 等三个函数，在这三个函数中最复杂的是 balance_nonroot 函数，balance_nonroot 函数包含大约 600 行代码，实现比较复杂，它是调整 B 树平衡的最主要的函数。其他还有比较重要的函数如游标类的函数，读写操作都要涉及到游标而且在共享的数据库连接中读写操作还会涉及锁的互斥问题。

经过对 Sqlite 源码的 btree.c 的分析，体会到开发一个数据库系统的难度之大。虽然进行了几个月的源码分析，但是还是对其中的代码实现不太了解，需要我们一直不断地深入分析和学习，必须具备一定的专业知识素养和专业知识在能充分理解其源码。遇到问题一定要坚持不懈、不放弃，通过更重途径解决难题，而且在分析过程中要多思考，多查阅资料。