

个人简介
专业打杂程序员
联系方式
新浪微博 腾讯微博

IT新闻:
苹果新Retina MacBook Pro (2014年中) 开箱图+SSD简单测试 6分钟前
网吧里玩出的世界冠军 打场游戏赚了400万 9分钟前
Twitter收购深度学习创业公司Madbits 33分钟前
昵称: YY哥
园龄: 7年2个月
粉丝: 342
关注: 2
+加关注

< 2009年2月 >						
日	一	二	三	四	五	六
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
1	2	3	4	5	6	7

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

随笔分类

c/c++(9)
Linux相关(24)
MySQL(11)
Others(2)
Web技术(12)
数据结构与算法(15)
数据库技术(30)
系统相关(3)
云计算与虚拟化(3)

随笔档案

2014年7月 (4)

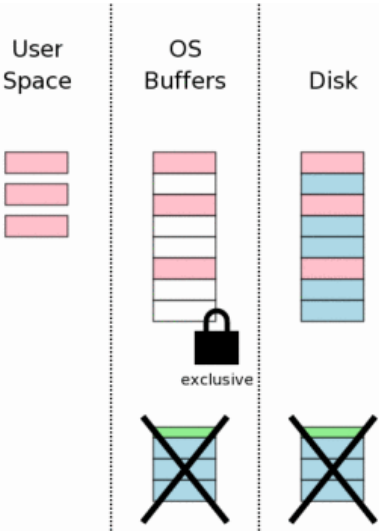
SQLite入门与分析(四)---Page Cache之事务处理(3)

写在前面：由于内容较多，所以断续没有写完的内容。

11、删除日志文件(Deleting The Rollback Journal)

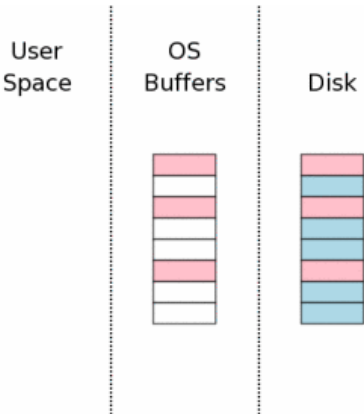
一旦更改写入设备，日志文件将会被删除，这是事务真正提交的时刻。如果在这之前系统发生崩溃，就会进行恢复处理，使得数据库和没发生改变一样；如果在这之后系统发生崩溃，表明所有的更改都已经写入磁盘。SQLite就是根据日志存在情况决定是否对数据库进行恢复处理。

删除文件本质上不是一个原子操作，但是从用户进程的角度来看是一个原子操作，所以一个事务看起来是一个原子操作。在许多系统中，删除文件也是一个高代价的操作。作为优化，SQLite可以配置成把日志文件的长度截为0或者把日志文件头清零。



12、释放锁(Releasing The Lock)

作为原子提交的最后一步，释放排斥锁使得其它进程可以开始访问数据库了。下图中，我们指明了当锁被释放的时候用户空间所拥有的信息已经被清空了.对于老版本的SQLite你可这么认为。但最新的SQLite会保存些用户空间的缓存不会被清空——万一下一个事务开始的时候，这些数据刚好可以用上呢。重新利用这些内存要比再次从操作系统磁盘缓存或者硬盘中读取要来得轻松与快捷得多，何乐而不为呢？在再次使用这些数据之前，我们必须先取得一个共享锁，同时我们还不得不去检查一下，保证还没有其他进程在我们拥有共享锁之前对数据库文件进行了修改。数据库文件的第一页中有一个计数器，数据库文件每做一次修改，这个计数器就会增长一下。我们可以通过检查这个计数器就可得知是否有其他进程修改过数据库文件。如果数据库文件已经被修改过了，那么用户内存空间的缓存就不得不清空，并重新读入。大多数情况下，这种情况不大会发生，因此用户空间的内存缓存将是有效的，这对于性能提高来说作用是显著的。



以上两步是在sqlite3BtreeCommit()---btree.c函数中实现的。

代码如下：

```

//提交事务,至此一个事务完成.主要做两件事:
//删除日志文件,释放数据库文件的写锁
int sqlite3BtreeCommit(Btree *p){
    BtShared *pBt = p->pBt;
}
```

- 2014年3月 (1)
- 2013年9月 (1)
- 2013年8月 (1)
- 2013年2月 (1)
- 2012年11月 (4)
- 2012年1月 (1)
- 2011年12月 (1)
- 2011年10月 (1)
- 2011年3月 (1)
- 2010年9月 (1)
- 2010年8月 (1)
- 2010年7月 (3)
- 2010年6月 (2)
- 2010年5月 (7)
- 2010年4月 (1)
- 2010年3月 (1)
- 2010年1月 (1)
- 2009年12月 (2)
- 2009年10月 (2)
- 2009年9月 (14)
- 2009年8月 (4)
- 2009年6月 (14)
- 2009年5月 (3)
- 2009年4月 (1)
- 2009年3月 (3)
- 2009年2月 (11)
- 2008年10月 (7)
- 2008年8月 (5)
- 2008年7月 (1)
- 2008年6月 (2)
- 2008年5月 (2)
- 2008年4月 (5)

kernel

kernel中文社区
LDN
The Linux Document Project
The Linux Kernel Archives

manual

cppreference
gcc manual
mysql manual

sites

Database Journal
Fedora镜像
highscalability
KFUPM ePrints
Linux docs
Linux Journal
NoSQL
SQLite

技术社区

apache
CSDN
IBM-developerworks
lucene中国
nutch中国
oldlinux
oracle's forum

最新评论

1. Re:理解MySQL——架构与概念
我试验了下.数据 5 9 10 13 18begin;select * from asf_execution where num> 5 and num 5 and INSTANCE_ID_<18 lock in share mode;会有 1.行锁 2.间隙锁 [5 18)插

```
btreeIntegrity(p);

/* If the handle has a write-transaction open, commit the shared-btrees
** transaction and set the shared state to TRANS_READ.
*/
if( p->inTrans==TRANS_WRITE ){
    int rc;
    assert( pBt->inTransaction==TRANS_WRITE );
    assert( pBt->nTransaction>0 );

    //调用pager, 提交事务
    rc = sqlite3pager_commit(pBt->pPager);
    if( rc!=SQLITE_OK ){
        return rc;
    }
    pBt->inTransaction = TRANS_READ;
    pBt->inStmt = 0;
}
unlockAllTables(p);

/* If the handle has any kind of transaction open, decrement the transaction
** count of the shared btree. If the transaction count reaches 0, set
** the shared state to TRANS_NONE. The unlockBtreeIfUnused() call below
** will unlock the pager.
*/
if( p->inTrans!=TRANS_NONE ){
    pBt->nTransaction--;
    if( 0==pBt->nTransaction ){
        pBt->inTransaction = TRANS_NONE;
    }
}
}

//提交事务, 主要调用pager_unwritelock()函数
int sqlite3pager_commit(Pager *pPager){
    int rc;
    PgHdr *pPg;

    if( pPager->errCode ){
        return pPager->errCode;
    }
    if( pPager->state<PAGER_RESERVED ){
        return SQLITE_ERROR;
    }
    TRACE2("COMMIT %d\n", PAGERID(pPager));
    if( MEMDB ){
        pPg = pager_get_all_dirty_pages(pPager);
        while( pPg ){
            clearHistory(PGHDR_TO_HIST(pPg, pPager));
            pPg->dirty = 0;
            pPg->inJournal = 0;
            pPg->inStmt = 0;
            pPg->needSync = 0;
            pPg->pPrevStmt = pPg->pNextStmt = 0;
            pPg = pPg->pDirty;
        }
        pPager->pDirty = 0;
#ifdef NDEBUG
        for(pPg=pPager->pAll; pPg; pPg=pPg->pNextAll){
            PgHistory *pHist = PGHDR_TO_HIST(pPg, pPager);
            assert( !pPg->alwaysRollback );
            assert( !pHist->pOrig );
            assert( !pHist->pStmt );
        }
#endif
        pPager->pStmt = 0;
        pPager->state = PAGER_SHARED;
        return SQLITE_OK;
    }
    if( pPager->dirtyCache==0 ){
        /* Exit early (without doing the time-consuming sqlite3OsSync() calls)
        ** if there have been no changes to the database file. */
        assert( pPager->needSync==0 );
        rc = pager_unwritelock(pPager);
        pPager->dbSize = -1;
        return rc;
    }
}
```

入INSERT I.....

--麒麟飞
2. Re:理解MySQL——架构与概念
例1-5
insert into t(i) values(1);
这句话应该是可以插入的。
不会被阻塞

--麒麟飞
3. Re:理解MySQL——架构与概念
注：SELECT ... FOR UPDATE仅在自动提交关闭(即手动提交)时才会对元组加锁，而在自动提交时，符合条件的元组不会被加锁。

这个是错误的.自动提交的,也会尝试获取排它锁。
你可以试验下。

--麒麟飞
4. Re:浅谈mysql的两阶段提交协议
YY哥 偶像啊!细腻文笔 配有说服力的代码和图 我崇拜你 !!!
之前sqlite的深入分析帮了我大忙..
现在做mysql相关 有来你的博客找东西 哈哈哈哈哈!

--hark.perfe
5. Re:(i++)+(i++)与(++i)+(++i)
@arrowcat
这类语句本身没什么意义，但是楼主思考的角度让我豁然开朗。

--HJWAJ

阅读排行榜

1. 理解MySQL——索引与优化(77627)

2. SQLite入门与分析(一)---简介(48610)

3. 理解MySQL——复制(Replication)(26209)

4. libevent源码分析(19048)

5. SQLite入门与分析(二)---设计与概念(16977)

评论排行榜

1. (i++)+(i++)与(++i)+(++i)(40)

2. SQLite入门与分析(一)---简介(30)

3. 浅谈SQLite——实现与应用(20)

4. 一道算法题,求更好的解法(18)

5. 理解MySQL——索引与优化(16)

推荐排行榜

1. SQLite入门与分析(一)---简介(12)

2. 理解MySQL——索引与优化(12)

3. 浅谈SQLite——查询处理及优化(10)

4. 乱谈服务器编程(9)

5. libevent源码分析(6)

```
assert( pPager->journalOpen );
rc = sqlite3pager_sync(pPager, 0, 0);

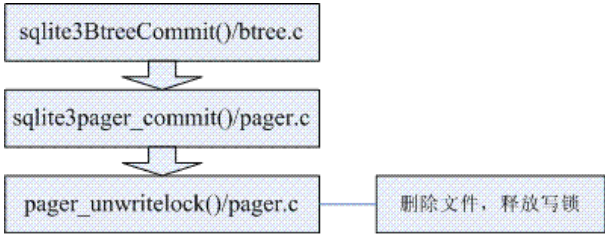
//删除文件,释放写锁
if( rc==SQLITE_OK ){
    rc = pager_unwritelock(pPager);
    pPager->dbSize = -1;
}
return rc;
}

//对数据库加read lock, 删除日志文件
static int pager_unwritelock(Pager *pPager){
    PgHdr *pPg;
    int rc;
    assert( !MEMDB );
    if( pPager->state<PAGER_RESERVED ){
        return SQLITE_OK;
    }
    sqlite3pager_stmt_commit(pPager);
    if( pPager->stmtOpen ){
        sqlite3sClose(&pPager->stfd);
        pPager->stmtOpen = 0;
    }
    if( pPager->journalOpen ){

        //关闭日志文件
        sqlite3sClose(&pPager->jfd);
        pPager->journalOpen = 0;
        //删除日志文件
        sqlite3Delete(pPager->zJournal);
        sqliteFree( pPager->aInJournal );
        pPager->aInJournal = 0;
        for(pPg=pPager->pAll; pPg; pPg=pPg->pNextAll){
            pPg->inJournal = 0;
            pPg->dirty = 0;
            pPg->needSync = 0;
#ifdef SQLITE_CHECK_PAGES
            pPg->pageHash = pager_pagehash(pPg);
#endif
        }
        pPager->pDirty = 0;
        pPager->dirtyCache = 0;
        pPager->nRec = 0;
    }else{
        assert( pPager->aInJournal==0 );
        assert( pPager->dirtyCache==0 || pPager->useJournal==0 );
    }

    //释放写锁, 加读锁
    rc = sqlite3sUnlock(pPager->fd, SHARED_LOCK);
    pPager->state = PAGER_SHARED;
    pPager->origDbSize = 0;
    pPager->setMaster = 0;
    pPager->needSync = 0;
    pPager->pFirstSynced = pPager->pFirst;
    return rc;
}
```

下图可进一步描述该过程:



最后来看看sqlite3BtreeSync()和sqlite3BtreeCommit()是如何被调用的。
一般来说，事务提交方式为自动提交的话，在虚拟机中的OP_Halt指令实现提交事务，相关代码如下：

```

//虚拟机停机指令
case OP_Halt: { /* no-push */
    p->pTos = pTos;
    p->rc = pOp->p1;
    p->pc = pc;
    p->errorAction = pOp->p2;
    if( pOp->p3 ){
        sqlite3SetString(&p->zErrMsg, pOp->p3, (char*)0);
    }
    //设置虚拟机状态SQLITE_MAGIC_RUN 为 SQLITE_MAGIC_HALT,
    //并提交事务
    rc = sqlite3VdbeHalt(p);
    assert( rc==SQLITE_BUSY || rc==SQLITE_OK );
    if( rc==SQLITE_BUSY ){
        p->rc = SQLITE_BUSY;
        return SQLITE_BUSY;
    }
    return p->rc ? SQLITE_ERROR : SQLITE_DONE;
}

//当虚拟机要停机时,调用该函数,如果VDBE改变了数据库且为自动
//提交模式,则提交这些改变
int sqlite3VdbeHalt(Vdbe *p){
    sqlite3 *db = p->db;
    int i;
    int (*xFunc)(Btree *pBt) = 0; /* Function to call on each btree backend */
    int isSpecialError; /* Set to true if SQLITE_NOMEM or IOERR */

    /* This function contains the logic that determines if a statement or
    ** transaction will be committed or rolled back as a result of the
    ** execution of this virtual machine.
    **
    ** Special errors:
    **
    ** If an SQLITE_NOMEM error has occurred in a statement that writes to
    ** the database, then either a statement or transaction must be rolled
    ** back to ensure the tree-structures are in a consistent state. A
    ** statement transaction is rolled back if one is open, otherwise the
    ** entire transaction must be rolled back.
    **
    ** If an SQLITE_IOERR error has occurred in a statement that writes to
    ** the database, then the entire transaction must be rolled back. The
    ** I/O error may have caused garbage to be written to the journal
    ** file. Were the transaction to continue and eventually be rolled
    ** back that garbage might end up in the database file.
    **
    ** In both of the above cases, the Vdbe.errorAction variable is
    ** ignored. If the sqlite3.autoCommit flag is false and a transaction
    ** is rolled back, it will be set to true.
    **
    ** Other errors:
    **
    ** No error:
    **
    */

    if( sqlite3MallocFailed() ){
        p->rc = SQLITE_NOMEM;
    }
    if( p->magic!=VDBE_MAGIC_RUN ){
        /* Already halted. Nothing to do. */
        assert( p->magic==VDBE_MAGIC_HALT );
        return SQLITE_OK;
    }
    //释放虚拟机中所有的游标
    closeAllCursors(p);
    checkActiveVdbeCnt(db);

    /* No commit or rollback needed if the program never started */
    if( p->pc>=0 ){

        /* Check for one of the special errors - SQLITE_NOMEM or SQLITE_IOERR */
        isSpecialError = ((p->rc==SQLITE_NOMEM || p->rc==SQLITE_IOERR)?1:0);
        if( isSpecialError ){
            /* This loop does static analysis of the query to see which of the

```

```

** following three categories it falls into:
**
**     Read-only
**     Query with statement journal
**     Query without statement journal
**
** We could do something more elegant than this static analysis (i.e.
** store the type of query as part of the compilation phase), but
** handling malloc() or IO failure is a fairly obscure edge case so
** this is probably easier. Todo: Might be an opportunity to reduce
** code size a very small amount though...
*/
int isReadOnly = 1;
int isStatement = 0;
assert(p->aOp || p->nOp==0);
for(i=0; i<p->nOp; i++){
    switch( p->aOp[i].opcode ){
        case OP_Transaction:
            isReadOnly = 0;
            break;
        case OP_Statement:
            isStatement = 1;
            break;
    }
}

/* If the query was read-only, we need do no rollback at all. Otherwise,
** proceed with the special handling.
*/
if( !isReadOnly ){
    if( p->rc==SQLITE_NOMEM && isStatement ){
        xFunc = sqlite3BtreeRollbackStmt;
    }else{
        /* We are forced to roll back the active transaction. Before doing
        ** so, abort any other statements this handle currently has active.
        */
        sqlite3AbortOtherActiveVdbes(db, p);
        sqlite3RollbackAll(db);
        db->autoCommit = 1;
    }
}

/* If the auto-commit flag is set and this is the only active vdbe, then
** we do either a commit or rollback of the current transaction.
**
** Note: This block also runs if one of the special errors handled
** above has occurred.
*/
//如果自动提交事务，则提交事务
if( db->autoCommit && db->activeVdbeCnt==1 ){
    if( p->rc==SQLITE_OK || (p->errorAction==OE_Fail && !isSpecialError) ){
        /* The auto-commit flag is true, and the vdbe program was
        ** successful or hit an 'OR FAIL' constraint. This means a commit
        ** is required.
        */
        //提交事务
        int rc = vdbeCommit(db);
        if( rc==SQLITE_BUSY ){
            return SQLITE_BUSY;
        }else if( rc!=SQLITE_OK ){
            p->rc = rc;
            sqlite3RollbackAll(db);
        }else{
            sqlite3CommitInternalChanges(db);
        }
    }else{
        sqlite3RollbackAll(db);
    }
}else if( !xFunc ){
    if( p->rc==SQLITE_OK || p->errorAction==OE_Fail ){
        xFunc = sqlite3BtreeCommitStmt;
    }else if( p->errorAction==OE_Abort ){
        xFunc = sqlite3BtreeRollbackStmt;
    }else{
        sqlite3AbortOtherActiveVdbes(db, p);
    }
}

```

```

        sqlite3RollbackAll(db);
        db->autoCommit = 1;
    }
}

/* If xFunc is not NULL, then it is one of sqlite3BtreeRollbackStmt or
** sqlite3BtreeCommitStmt. Call it once on each backend. If an error occurs
** and the return code is still SQLITE_OK, set the return code to the new
** error value.
*/
assert(!xFunc ||
       xFunc==sqlite3BtreeCommitStmt ||
       xFunc==sqlite3BtreeRollbackStmt
);
for(i=0; xFunc && i<db->nDb; i++){
    int rc;
    Btree *pBt = db->aDb[i].pBt;
    if( pBt ){
        rc = xFunc(pBt);
        if( rc && (p->rc==SQLITE_OK || p->rc==SQLITE_CONSTRAINT) ){
            p->rc = rc;
            sqlite3SetString(&p->zErrMsg, 0);
        }
    }
}

/* If this was an INSERT, UPDATE or DELETE and the statement was committed,
** set the change counter.
*/
if( p->changeCntOn && p->pc>=0 ){
    if( !xFunc || xFunc==sqlite3BtreeCommitStmt ){
        sqlite3VdbeSetChanges(db, p->nChange);
    }else{
        sqlite3VdbeSetChanges(db, 0);
    }
    p->nChange = 0;
}

/* Rollback or commit any schema changes that occurred. */
if( p->rc!=SQLITE_OK && db->flags&SQLITE_InternChanges ){
    sqlite3ResetInternalSchema(db, 0);
    db->flags = (db->flags | SQLITE_InternChanges);
}
}

/* We have successfully halted and closed the VM. Record this fact. */
if( p->pc>=0 ){
    db->activeVdbeCnt--;
}
p->magic = VDBE_MAGIC_HALT;
checkActiveVdbeCnt(db);

return SQLITE_OK;
}

//提交事务,主要调用:
//sqlite3BtreeSync()---同步btree, sqlite3BtreeCommit()---提交事务
static int vdbeCommit(sqlite3 *db){
    int i;
    int nTrans = 0; /* Number of databases with an active write-transaction */
    int rc = SQLITE_OK;
    int needXcommit = 0;

    for(i=0; i<db->nDb; i++){
        Btree *pBt = db->aDb[i].pBt;
        if( pBt && sqlite3BtreeIsInTrans(pBt) ){
            needXcommit = 1;
            if( i!=1 ) nTrans++;
        }
    }
}

/* If there are any write-transactions at all, invoke the commit hook */
if( needXcommit && db->xCommitCallback ){
    sqlite3SafetyOff(db);
    rc = db->xCommitCallback(db->pCommitArg);
    sqlite3SafetyOn(db);
    if( rc ){

```

```

        return SQLITE_CONSTRAINT;
    }
}

/* The simple case - no more than one database file (not counting the
** TEMP database) has a transaction active.  There is no need for the
** master-journal.
**
** If the return value of sqlite3BtreeGetFilename() is a zero length
** string, it means the main database is :memory:.  In that case we do
** not support atomic multi-file commits, so use the simple case then
** too.
**/
//简单的情况,只有一个数据库文件,不需要master-journal
if( 0==strlen(sqlite3BtreeGetFilename(db->aDb[0].pBt)) || nTrans<=1 ){
    for(i=0; rc==SQLITE_OK && i<db->nDb; i++){
        Btree *pBt = db->aDb[i].pBt;
        if( pBt ){
            //同步btree
            rc = sqlite3BtreeSync(pBt, 0);
        }
    }

    /* Do the commit only if all databases successfully synced */
    //commite事务
    if( rc==SQLITE_OK ){
        for(i=0; i<db->nDb; i++){
            Btree *pBt = db->aDb[i].pBt;
            if( pBt ){
                sqlite3BtreeCommit(pBt);
            }
        }
    }
}

/* The complex case - There is a multi-file write-transaction active.
** This requires a master journal file to ensure the transaction is
** committed atomicly.
**/
#ifdef SQLITE_OMIT_DISKIO
else{
    int needSync = 0;
    char *zMaster = 0; /* File-name for the master journal */
    char const *zMainFile = sqlite3BtreeGetFilename(db->aDb[0].pBt);
    OsFile *master = 0;

    /* Select a master journal file name */
    do {
        u32 random;
        sqliteFree(zMaster);
        sqlite3Randomness(sizeof(random), &random);
        zMaster = sqlite3MPrintf("%s-mj%08X", zMainFile, random&0x7fffffff);
        if( !zMaster ){
            return SQLITE_NOMEM;
        }
    }while( sqlite3OsFileExists(zMaster) );

    /* Open the master journal. */
    rc = sqlite3OsOpenExclusive(zMaster, &master, 0);
    if( rc!=SQLITE_OK ){
        sqliteFree(zMaster);
        return rc;
    }

    /* Write the name of each database file in the transaction into the new
    ** master journal file. If an error occurs at this point close
    ** and delete the master journal file. All the individual journal files
    ** still have 'null' as the master journal pointer, so they will roll
    ** back independently if a failure occurs.
    **/
    for(i=0; i<db->nDb; i++){
        Btree *pBt = db->aDb[i].pBt;
        if( i==1 ) continue; /* Ignore the TEMP database */
        if( pBt && sqlite3BtreeIsInTrans(pBt) ){
            char const *zFile = sqlite3BtreeGetJournalname(pBt);
            if( zFile[0]==0 ) continue; /* Ignore :memory: databases */
            if( !needSync && !sqlite3BtreeSyncDisabled(pBt) ){

```

```

        needSync = 1;
    }
    rc = sqlite3OsWrite(master, zFile, strlen(zFile)+1);
    if( rc!=SQLITE_OK ){
        sqlite3OsClose(&master);
        sqlite3OsDelete(zMaster);
        sqliteFree(zMaster);
        return rc;
    }
}

/* Sync the master journal file. Before doing this, open the directory
** the master journal file is store in so that it gets synced too.
*/
zMainFile = sqlite3BtreeGetDirname(db->aDb[0].pBt);
rc = sqlite3OsOpenDirectory(master, zMainFile);
if( rc!=SQLITE_OK ||
    (needSync && (rc=sqlite3OsSync(master,0))!=SQLITE_OK) ){
    sqlite3OsClose(&master);
    sqlite3OsDelete(zMaster);
    sqliteFree(zMaster);
    return rc;
}

/* Sync all the db files involved in the transaction. The same call
** sets the master journal pointer in each individual journal. If
** an error occurs here, do not delete the master journal file.
**
** If the error occurs during the first call to sqlite3BtreeSync(),
** then there is a chance that the master journal file will be
** orphaned. But we cannot delete it, in case the master journal
** file name was written into the journal file before the failure
** occurred.
*/
for(i=0; i<db->nDb; i++){
    Btree *pBt = db->aDb[i].pBt;
    if( pBt && sqlite3BtreeIsInTrans(pBt) ){
        rc = sqlite3BtreeSync(pBt, zMaster);
        if( rc!=SQLITE_OK ){
            sqlite3OsClose(&master);
            sqliteFree(zMaster);
            return rc;
        }
    }
}
sqlite3OsClose(&master);

/* Delete the master journal file. This commits the transaction. After
** doing this the directory is synced again before any individual
** transaction files are deleted.
*/
rc = sqlite3OsDelete(zMaster);
assert( rc==SQLITE_OK );
sqliteFree(zMaster);
zMaster = 0;
rc = sqlite3OsSyncDirectory(zMainFile);
if( rc!=SQLITE_OK ){
    /* This is not good. The master journal file has been deleted, but
    ** the directory sync failed. There is no completely safe course of
    ** action from here. The individual journals contain the name of the
    ** master journal file, but there is no way of knowing if that
    ** master journal exists now or if it will exist after the operating
    ** system crash that may follow the fsync() failure.
    */
    return rc;
}

/* All files and directories have already been synced, so the following
** calls to sqlite3BtreeCommit() are only closing files and deleting
** journals. If something goes wrong while this is happening we don't
** really care. The integrity of the transaction is already guaranteed,
** but some stray 'cold' journals may be lying around. Returning an
** error code won't help matters.
*/
for(i=0; i<db->nDb; i++){

```




```
Btree *pBt = db->adb[i].pBt;
if( pBt ){
    sqlite3BtreeCommit(pBt);
}
}
}
#endif

return rc;
}
```

分类: 数据库技术

绿色通道: [好文要顶](#) [关注我](#) [收藏该文](#) [与我联系](#) 

YY哥
关注 - 2
粉丝 - 342
[+加关注](#)

0 0

(请您对文章做出评价)

« 上一篇: [SQLite入门与分析\(四\)---Page Cache之事务处理\(2\)](#)
» 下一篇: [SQLite Version3.3.6源代码文件结构](#)

posted @ 2009-02-26 14:53 YY哥 阅读(4793) 评论(2) 编辑 收藏

评论列表

#1楼 2009-03-06 22:28 Soli

Not bad.
But please don't fold the code. Thank you.

支持(0) 反对(0)

#2楼 2012-01-11 16:42 遗忘海岸

^_^,....

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

[博客园首页](#) [博问](#) [新闻](#) [闪存](#) [程序员招聘](#) [知识库](#)



阿里云 云服务器

¥0 / 半年
原价: ¥280 / 半年

免费抢

- 最新IT新闻:
- Twitter收购深度学习创业公司Madbits
 - 这两个前亚马逊员工要把亚马逊赶出印度
 - Twitter财报中你不能错过的6个数据
 - 甲骨文对CEO拉里森每年股票奖励削减过半
 - Facebook关闭Gifts礼品商店：探索电商新路
- » 更多新闻...

- 最新知识库文章:
- 如何在网页中使用留白
 - SQL/NoSQL两大阵营激辩：谁更适合大数据
 - 如何获取（GET）一杯咖啡——星巴克REST案例分析
 - 为什么程序员的工作效率跟他们的工资不成比例
 - 我眼里的DBA
- » 更多知识库文章...