**Sqlite**

**1.使用**

1).继承SQLiteOpenHelper

public SQLiteOpenHelper(@Nullable Context context, @Nullable String name,

@Nullable CursorFactory factory, int version) {

this(context, name, factory, version, null);

}

参数包含DB的名称和version。

2).获取或者创建SQLiteDatabase对象

private SQLiteDatabase getDatabaseLocked(boolean writable) {

try {

db = SQLiteDatabase.openDatabase(filePath, params);

setFilePermissionsForDb(filePath.getPath());

} catch (SQLException ex) {

}

onConfigure(db);

**final int version = db.getVersion();**

if (version != mNewVersion) {

db.beginTransaction();

try {

if (version == 0) {

onCreate(db);

} else {

if (version > mNewVersion) {

onDowngrade(db, version, mNewVersion);

} else {

onUpgrade(db, version, mNewVersion);

}

}

db.setVersion(mNewVersion);

db.setTransactionSuccessful();

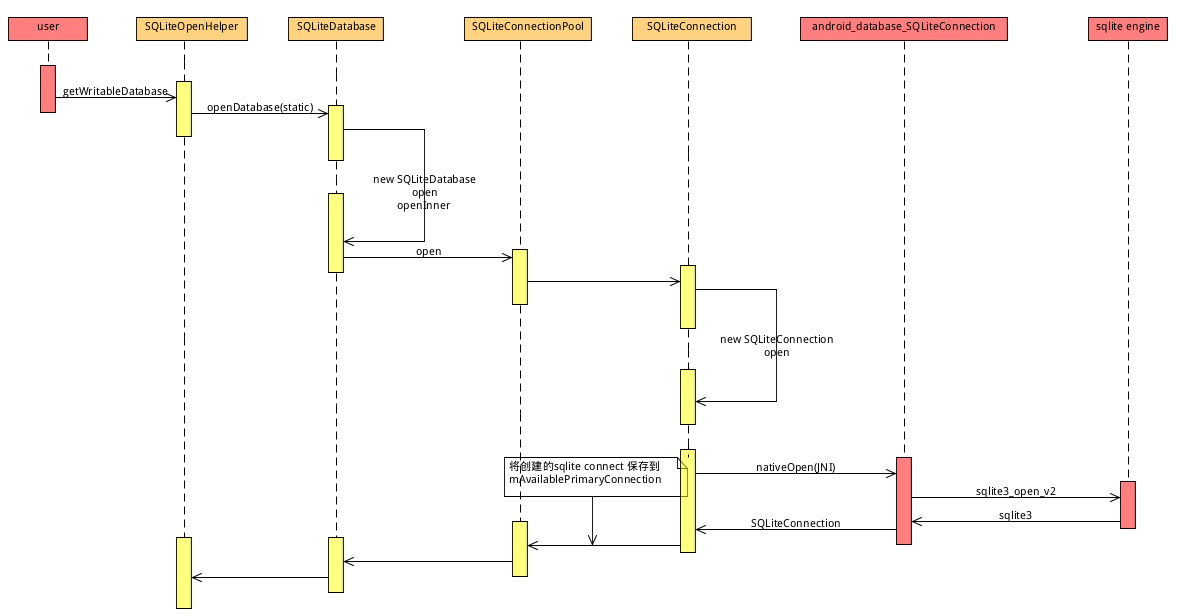
} finally {

db.endTransaction();

}

}

**2.openDB：**



创建ConnectionPool对象：

public static SQLiteConnectionPool open(SQLiteDatabaseConfiguration configuration) {

if (configuration == null) {

throw new IllegalArgumentException("configuration must not be null.");

}

// Create the pool.

**SQLiteConnectionPool pool = new SQLiteConnectionPool(configuration);**

pool.open(); // might throw

return pool;

}

创建PrimaryConnection：

private void open() {

// Open the primary connection.

// This might throw if the database is corrupt.

**mAvailablePrimaryConnection = openConnectionLocked(mConfiguration,**

**true /\*primaryConnection\*/); // might throw**

// Mark it released so it can be closed after idle timeout

synchronized (mLock) {

if (mIdleConnectionHandler != null) {

mIdleConnectionHandler.connectionReleased(mAvailablePrimaryConnection);

}

}

// Mark the pool as being open for business.

mIsOpen = true;

mCloseGuard.open("close");

}

在Native创建一个SQLiteConnection（sqlite3）对象将其ptr保存在java层的SQLiteConnection对象中。

Open成功之后将SQLiteDatabase对象保存在静态全局变量sActiveDatabases中，

该变量记录当前进程open的所有的SQLiteDatabase。

// Stores reference to all databases opened in the current process.

// (The referent Object is not used at this time.)

// INVARIANT: Guarded by sActiveDatabases.

private static WeakHashMap<SQLiteDatabase, Object> sActiveDatabases = new WeakHashMap<>();

sActiveDatabases.put(this, null);

获取DB version:

比较当前version和DB文件中记录的version来调用oncreate，onDowngrade，onUpgrade函数。

**3.增删改:对DB进行写**

execSQL

Update

Delete

Insert

对DB写操作都会创建SQLiteStatement进行execute操作。

public int delete(String table, String whereClause, String[] whereArgs) {

acquireReference();

try {

SQLiteStatement statement = new SQLiteStatement(this, "DELETE FROM " + table +

(!TextUtils.isEmpty(whereClause) ? " WHERE " + whereClause : ""), whereArgs);

try {

return **statement.executeUpdateDelete**();

} finally {

statement.close();

}

} finally {

releaseReference();

}

}

通过ThreadLocal获取当前线程的SQLiteSession 最终使用当前线程的SQLiteSession 进行execute。

public int executeUpdateDelete() {

acquireReference();

try {

return **getSession().executeForChangedRowCount**(

getSql(), getBindArgs(), getConnectionFlags(), null);

} catch (SQLiteDatabaseCorruptException ex) {

onCorruption();

throw ex;

} finally {

releaseReference();

}

}

protected final SQLiteSession getSession() {

return mDatabase.getThreadSession();

}

SQLiteSession getThreadSession() {

return mThreadSession.get(); // initialValue() throws if database closed

}

// Thread-local for database sessions that belong to this database.

// Each thread has its own database session.

// INVARIANT: Immutable.

private final ThreadLocal<SQLiteSession> mThreadSession = ThreadLocal

.withInitial(this::createSession);

public int executeForChangedRowCount(String sql, Object[] bindArgs, int connectionFlags,

CancellationSignal cancellationSignal) {

if (sql == null) {

throw new IllegalArgumentException("sql must not be null.");

}

if (executeSpecial(sql, bindArgs, connectionFlags, cancellationSignal)) {

return 0;

}

**acquireConnection(sql, connectionFlags, cancellationSignal); // might throw**

try {

**return mConnection.executeForChangedRowCount(sql, bindArgs,**

**cancellationSignal); // might throw**

} finally {

**releaseConnection(); // might throw**

}

}

private void acquireConnection(String sql, int connectionFlags,

CancellationSignal cancellationSignal) {

if (**mConnection** == null) {

assert mConnectionUseCount == 0;

**mConnection = mConnectionPool.acquireConnection(sql, connectionFlags,**

**cancellationSignal); // might throw**

mConnectionFlags = connectionFlags;

}

mConnectionUseCount += 1;

}

acquireConnection：

mConnection为null使用ConnectionPool去获取一个可用的SQLiteConnection对象。

SQLiteConnectionPool中的mAvailableNonPrimaryConnections容器存放所有可用的非主连接。mAvailablePrimaryConnection存放主连接。mAcquiredConnections 保存被获取的连接。

// Strong references to all available connections.

private final ArrayList<SQLiteConnection> **mAvailableNonPrimaryConnections** =

new ArrayList<SQLiteConnection>();

private SQLiteConnection **mAvailablePrimaryConnection**;

WeakHashMap<SQLiteConnection, AcquiredConnectionStatus> **mAcquiredConnections** =

new WeakHashMap<SQLiteConnection, AcquiredConnectionStatus>();

public SQLiteConnection acquireConnection(String sql, int connectionFlags,

CancellationSignal cancellationSignal) {

SQLiteConnection con = **waitForConnection(sql, connectionFlags, cancellationSignal);**

synchronized (mLock) {

if (mIdleConnectionHandler != null) {

mIdleConnectionHandler.connectionAcquired(con);

}

}

return con;

}

private SQLiteConnection waitForConnection(String sql, int connectionFlags,

CancellationSignal cancellationSignal) {

**synchronized (mLock) {**

// Try to acquire a connection.

SQLiteConnection connection = null;

if (!wantPrimaryConnection) {

**connection = tryAcquireNonPrimaryConnectionLocked(**

**sql, connectionFlags); // might throw**

}

if (connection == null) {

//这个函数中主连接不存在的时候，获取重新创建主连接。

**connection = tryAcquirePrimaryConnectionLocked(connectionFlags); // might throw**

}

if (connection != null) {

return connection;

}

tryAcquireNonPrimaryConnectionLocked

从mAvailableNonPrimaryConnections容器中获取一个非主连接，获取成功之后从mAvailableNonPrimaryConnections容器中删除该连接，同时将该连接加入到mAcquiredConnections容器中。非主连接容器获取失败使用mAvailablePrimaryConnection主连接，主连接获取仍然失败使用openConnectionLocked重新创建一个SQLiteConnection对象。

最终调用SQLiteConnection进行execute，在native层就是sqlite3\_step进行写操作。

**releaseConnection：**

public void releaseConnection(SQLiteConnection connection) {

synchronized (mLock) {

if (mIdleConnectionHandler != null) {

mIdleConnectionHandler.connectionReleased(connection);

}

**AcquiredConnectionStatus status = mAcquiredConnections.remove(connection);**

if (status == null) {

throw new IllegalStateException("Cannot perform this operation "

+ "because the specified connection was not acquired "

+ "from this pool or has already been released.");

}

if (!mIsOpen) {

closeConnectionAndLogExceptionsLocked(connection);

} else if (connection.isPrimaryConnection()) {

if (recycleConnectionLocked(connection, status)) {

assert mAvailablePrimaryConnection == null;

mAvailablePrimaryConnection = connection;

}

wakeConnectionWaitersLocked();

} else if (mAvailableNonPrimaryConnections.size() >= mMaxConnectionPoolSize - 1) {

closeConnectionAndLogExceptionsLocked(connection);

} else {

if (recycleConnectionLocked(connection, status)) {

**mAvailableNonPrimaryConnections.add(connection);**

}

wakeConnectionWaitersLocked();

}

}

}

db.beginTransaction();

try {

...

db.setTransactionSuccessful();

} finally {

db.endTransaction();

}

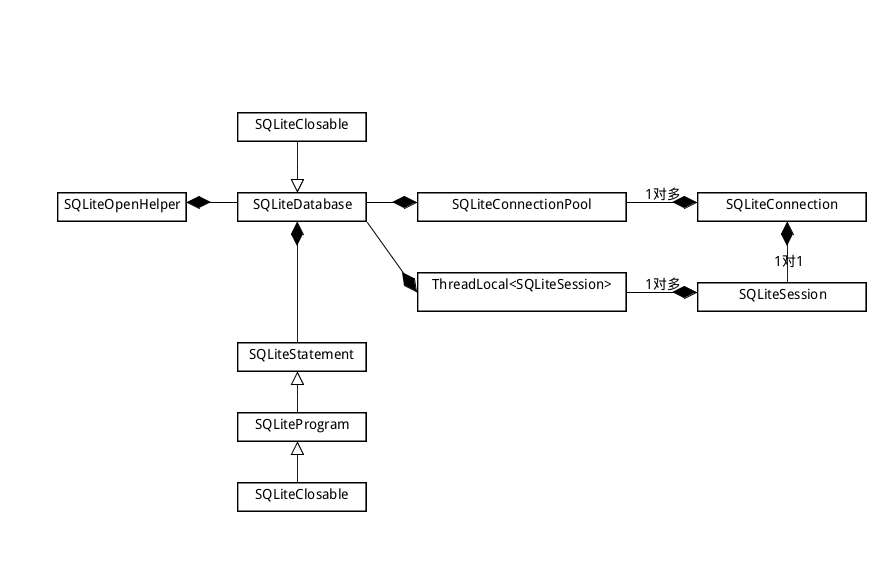
beginTransaction也会去acquireConnection增加mConnectionUseCount当没有调用的时候就会导致获取的连接一直不会释放到连接池。

当不去releaseConnection会导致获取acquireConnectionblock。

**Dump SQLiteConnectionPool：**

dumpsys dbinfo

实现：AMS 添加DbBinder binder服务，通过该服务实现所有应用的Database dump功能。

****

**4.对DB进行查询（读）**

dbh = new DbHelper(this);

SQLiteDatabase db = dbh.getWritableDatabase();

Cursor c = db.query("TableName", new String[]{"ColumnName"}

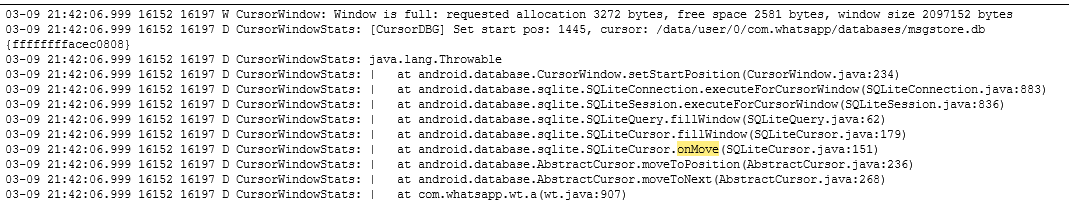
, "ColumnName LIKE ?" ,new String[]{\_data+"%"}, null, null, null);

while(c.moveToNext())

{

// your calculation goes here

}

****

**moveToNext-------moveToPosition-------getCount()**

**因为**