SQLite 研究

目录

[1. 简介 1](#_Toc5030)

[2. SQLite 基础认知 2](#_Toc1907)

[2.1 数据库文件格式 2](#_Toc12122)

[2.2 SQLite 权限 2](#_Toc24687)

[2.3 支持Sql标准 2](#_Toc31429)

[2.4 数据类型 2](#_Toc27447)

[2.5 内置函数 3](#_Toc17143)

[2.6 Sqllite数据库锁级别 3](#_Toc18978)

[3. Sqllite 的并发： 4](#_Toc11341)

[3.1 并发锁一般解决思路： 4](#_Toc30200)

[3.1.1 避免重复打开数据库可以使用单例模式，使用单例来保证数据库连接对象的唯一性。 4](#_Toc740)

[3.1.2 合并多个数据库. 4](#_Toc12043)

[4. Sqllite返回码 5](#_Toc16325)

[5. 参考网络文档 5](#_Toc25654)

[6. Sqlite知识补充 6](#_Toc28786)

[6.1 死锁的情况 7](#_Toc9924)

[6.2 事务类型的使用原则 7](#_Toc23468)

[6.3 对SQLITE\_BUSY的处理 7](#_Toc14981)

[7. 同一数据库多进程读写的解决方案 9](#_Toc21942)

# 简介

官方网站：https://www.sqlite.org/about.html

# SQLite 基础认知

## 数据库文件格式

数据库文件名 xxx.db 后缀为.db

## SQLite 权限

SQLite 没有用户帐户概念，而是根据文件系统确定所有数据库文件的权限。

## 支持Sql标准

SQLite 支持多数（但不是全部）的 SQL92 标准。不受支持的一些功能包括 触发器，视图，存储过程等。

|  |
| --- |
| ATTACH DATABASE |
| BEGIN TRANSACTION |
| comment |
| COMMIT TRANSACTION |
| COPY |
| CREATE INDEX |
| CREATE TABLE |
| CREATE TRIGGER |
| CREATE VIEW |
| DELETE |
| DETACH DATABASE |
| DROP INDEX |
| DROP TABLE |
| DROP TRIGGER |
| DROP VIEW |
| END TRANSACTION |
| EXPLAIN |
| expression |
| INSERT |
| ON CONFLICT clause |
| PRAGMA |
| REPLACE |
| ROLLBACK TRANSACTION |
| SELECT |
| UPDATE |

## 数据类型

Typelessness(无类型) 对于SQLite来说对字段不指定类型是完全有效的。.但主键Primary Key必须有类型

SQLite支持常见的数据类型：**SQLite 的存储宽度是根据输入来自动调整的**

|  |  |
| --- | --- |
| VARCHAR(10), |  |
| NVARCHAR(15), |  |
| TEXT, | 文本字符串，缺省的编码为utf-8 |
| INTEGER, | 带符号的整数，根据值的大小，自动存储为1,2,3,4,5,8字节6种 |
| FLOAT, |  |
| BOOLEAN, |  |
| CLOB, |  |
| BLOB, | blob数据，不定长 |
| TIMESTAMP, |  |
| NUMERIC(10,5) |  |
| VARYING CHARACTER (24), |  |
| NATIONAL VARYING CHARACTER(16) |  |

## 内置函数

|  |  |
| --- | --- |
| SQLite COUNT 函数 | SQLite COUNT 聚集函数是用来计算一个数据库表中的行数。 |
| SQLite MAX 函数 | SQLite MAX 聚合函数允许我们选择某列的最大值。 |
| SQLite MIN 函数 | SQLite MIN 聚合函数允许我们选择某列的最小值。 |
| SQLite AVG 函数 | SQLite AVG 聚合函数计算某列的平均值。 |
| SQLite SUM 函数 | SQLite SUM 聚合函数允许为一个数值列计算总和。 |
| SQLite RANDOM 函数 | SQLite RANDOM 函数返回一个介于 -9223372036854775808 和 +9223372036854775807 之间的伪随机整数。 |
| SQLite ABS 函数 | SQLite ABS 函数返回数值参数的绝对值。 |
| SQLite UPPER 函数 | SQLite UPPER 函数把字符串转换为大写字母。 |
| SQLite LOWER 函数 | SQLite LOWER 函数把字符串转换为小写字母。 |
| SQLite LENGTH 函数 | SQLite LENGTH 函数返回字符串的长度。 |
| SQLite sqlite\_version 函数 | SQLite sqlite\_version 函数返回 SQLite 库的版本。 |

## Sqllite数据库锁级别

SQLite 只提供数据库级的锁定。不提供表或行级别锁。

以上为sqllite基础知识。其它sqllite 基本使用可参考

<http://www.runoob.com/sqlite/sqlite-tutorial.html>

# Sqllite 的并发：

sqlite同一时间只能进行一个写操作，当同时有两个写操作的时候,后执行的只能先等待,如果等待时间超过5秒,就会产生这种错误database is locked。

## 并发锁一般解决思路：

### 避免重复打开数据库可以使用单例模式，使用单例来保证数据库连接对象的唯一性。

2 读写分离

单库单连接，多连接写多个库。

1. 动态按需创建多个数据库文件。多个进程或多线程将数据写到多个数据库中，数据库文件个数按需动态创建。
2. 规范数据库文件个数。可先连接固定，文件数量可控。

同时读取多个数据库文件。

使用sqllite 的ATTACH DATABASE 同时访问多个数据库文件。可类比dbms 的scheme，第二个sqllite库的mytable查询时用SecondaryDB.mytable

命令使用参考地址：

附加数据库

<http://www.runoob.com/sqlite/sqlite-attach-database.html>

分离数据库

<http://www.runoob.com/sqlite/sqlite-detach-database.html>

### 合并多个数据库.

前提条件： 多个数据库已做 附加ATTACH DATABASE，各库中同表主键不冲突。

1、先连接数据库A

2、再attach数据库B作为第二数据库：   ATTACH "c:\database\_b\places.sqlite" AS SecondaryDB;   （若还要attach其它数据库，则 ATTACH "c:\database\_b\third.sqlite" AS ThirdDB;）

3、(若要将B中的某个表的数据追加到A中的某个表中，则： INSERT OR IGNORE INTO MyTable SELECT \* FROM SecondaryDB.MyTable;  但条件是这2个表的主键没有冲突，若有冲突，则修改B的主键)

4、然后提交  COMMIT;

5、最后分离数据库  DETACH DATABASE SecondaryDB;

# Sqllite返回码

|  |  |
| --- | --- |
| Return result | describe |
| SQLITE\_OK | 成功 | Successful result |
| SQLITE\_ERROR | SQL错误 或 丢失数据库 | SQL error or missing database |
| SQLITE\_INTERNAL | SQLite 内部逻辑错误 | Internal logic error in SQLite |
| SQLITE\_PERM | 拒绝访问 | Access permission denied |
| SQLITE\_ABORT | 回调函数请求取消操作 | Callback routine requested an abort |
| SQLITE\_BUSY | 数据库文件被锁定 | The database file is locked |
| SQLITE\_LOCKED | 数据库中的一个表被锁定 | A table in the database is locked |
| SQLITE\_NOMEM | 某次 malloc() 函数调用失败 | A malloc() failed |
| SQLITE\_READONLY | 尝试写入一个只读数据库 | Attempt to write a readonly database |
| SQLITE\_INTERRUPT | 操作被 sqlite3\_interupt() 函数中断 | Operation terminated by sqlite3\_interrupt() |
| SQLITE\_IOERR | 发生某些磁盘 I/O 错误 | Some kind of disk I/O error occurred |
| SQLITE\_CORRUPT | 数据库磁盘映像不正确 | The database disk image is malformed |
| SQLITE\_NOTFOUND | sqlite3\_file\_control() 中出现未知操作数 | Unknown opcode in sqlite3\_file\_control() |
| SQLITE\_FULL | 因为数据库满导致插入失败 | Insertion failed because database is full |
| SQLITE\_CANTOPEN | 无法打开数据库文件 | Unable to open the database file |
| SQLITE\_PROTOCOL | 数据库锁定协议错误 | Database lock protocol error |
| SQLITE\_EMPTY | 数据库为空 | Database is empty |
| SQLITE\_SCHEMA | 数据结构发生改变 | The database schema changed |
| SQLITE\_TOOBIG | 字符串或二进制数据超过大小限制 | String or BLOB exceeds size limit |
| SQLITE\_CONSTRAINT | 由于约束违例而取消 | Abort due to constraint violation |
| SQLITE\_MISMATCH | 数据类型不匹配 | Data type mismatch |
| SQLITE\_MISUSE | 不正确的库使用 | Library used incorrectly |
| SQLITE\_NOLFS | 使用了操作系统不支持的功能 | Uses OS features not supported on host |
| SQLITE\_AUTH | 授权失败 | Authorization denied |
| SQLITE\_FORMAT | 附加数据库格式错误 | Auxiliary database format error |
| SQLITE\_RANGE | 传递给sqlite3\_bind()的第二个参数超出范围 | 2nd parameter to sqlite3\_bind out of range |
| SQLITE\_NOTADB | 被打开的文件不是一个数据库文件 | File opened that is not a database file |
| SQLITE\_ROW | sqlite3\_step() 已经产生一个行结果 | sqlite3\_step() has another row ready |
| SQLITE\_DONE | sqlite3\_step() 完成执行操作 | sqlite3\_step() has finished executing |

# 参考网络文档

* SQLite 官网
* SQL As Understood By SQLite
* System.Data.SQLite
* 菜鸟教程
* SQL 教程

# Sqlite知识补充

sqlite3总共有三种事务类型：BEGIN [DEFERRED /IMMEDIATE / EXCLUSIVE] TRANSCATION，五种锁，按锁的级别依次是：UNLOCKED /SHARED /RESERVERD /PENDING /EXCLUSIVE。当执行select即读操作时，需要获取到SHARED锁（共享锁），当执行insert/update/delete操作(即内存写操作时)，需要进一步获取到RESERVERD锁（保留锁），当进行commit操作(即磁盘写操作时)，需要进一步获取到EXCLUSIVE锁（排它锁）。

对于RESERVERD锁，sqlite3保证同一时间只有一个连接可以获取到保留锁，也就是同一时间只有一个连接可以写数据库(内存)，但是其它连接仍然可以获取SHARED锁，也就是其它连接仍然可以进行读操作（这里可以认为写操作只是对磁盘数据的一份内存拷贝进行修改，并不影响读操作）。

对于EXCLUSIVE锁，是比保留锁更为严格的一种锁，在需要把修改写入磁盘即commit时需要在保留锁/未决锁的基础上进一步获取到排他锁，顾名思义，排他锁排斥任何其它类型的锁，即使是SHARED锁也不行，所以，在一个连接进行commit时，其它连接是不能做任何操作的（包括读）。

PENDING锁（即未决锁），则是比较特殊的一种锁，它可以允许已获取到SHARED锁的事务继续进行，但不允许其它连接再获取SHARED锁，当已存在的SHARED锁都被释放后（事务执行完成），持有未决锁的事务就可以获得commit的机会了。sqlite3使用这种锁来防止writer starvation（写饿死）。

## ****死锁的情况****

死锁的情况：当两个连接使用begin transaction开始事务时，第一个连接执行了一次select操作（已经获取到SHARED锁），第二个连接执行了一次insert操作（已经获取到了RESERVERD锁），此时第一个连接需要进行一次insert/update/delete（需要获取到RESERVERD锁），第二个连接则希望执行commit（需要获取到EXCLUSIVE锁），由于第二个连接已经获取到了RESERVERD锁，根据RESERVERD锁同一时间只有一个连接可以获取的特性，第一个连接获取RESERVERD锁的操作必定失败，而由于第一个连接已经获取到SHARED锁，第二个连接希望进一步获取到EXCLUSIVE锁的操作也必定失败。就导致了事务死锁。

## ****事务类型的使用原则****

在用”begin transaction”显式开启一个事务时，默认的事务类型为DEFERRED(推迟)，锁的状态为UNLOCKED，即不获取任何锁，如果在使用的数据库没有其它的连接，用begin就可以了。如果有多个连接都需要对数据库进行写操作，那就得使用BEGIN IMMEDIATE/EXCLUSIVE开始事务了。

使用事务的好处是：1.一个事务的所有操作相当于一次原子操作，如果其中某一步失败，可以通过回滚来撤销之前所有的操作，只有当所有操作都成功时，才进行commit，保证了操作的原子特性；2.对于多次的数据库操作，如果我们希望提高数据查询或更新的速度，可以在开始操作前显式开启一个事务，在执行完所有操作后，再通过一次commit来提交所有的修改或结束事务。

## ****对SQLITE\_BUSY的处理****

当有多个连接同时对数据库进行写操作时，根据事务类型的使用原则，我们在每个连接中用BEGIN IMMEDIATE开始事务，即多个连接都尝试取得保留锁的情况，根据保留锁同一时间只有一个连接可以获取到的特性，其它连接都将获取失败，即事务开始失败，这种情况下，sqlite3将返回一个SQLITE\_BUSY的错误，如果我们不希望操作就此失败而返回，就必须处理SQLITE\_BUSY的情况，sqlite3提供了sqlite3\_busy\_handler或sqlite3\_busy\_timeout来处理SQLITE\_BUSY，对于sqlite3\_busy\_handler，我们可以指定一个busy\_handler来处理，并可以指定失败重试的次数。而sqlite3\_busy\_timeout则是由sqlite3自动进行sleep并重试，当sleep的累积时间超过指定的超时时间时，最终返回SQLITE\_BUSY。需要注意的是，这两个函数同时只能使用一个，后面的调用会覆盖掉前次调用。从使用上来说，sqlite3\_busy\_timeout更易用一些，只需要指定一个总的超时时间，然后sqlite自己会决定多久进行重试以及重试的次数，直到达到总的超时时间最终返回SQLITE\_BUSY。并且，这两个函数一经调用，对其后的所有数据库操作都有效，非常方便。

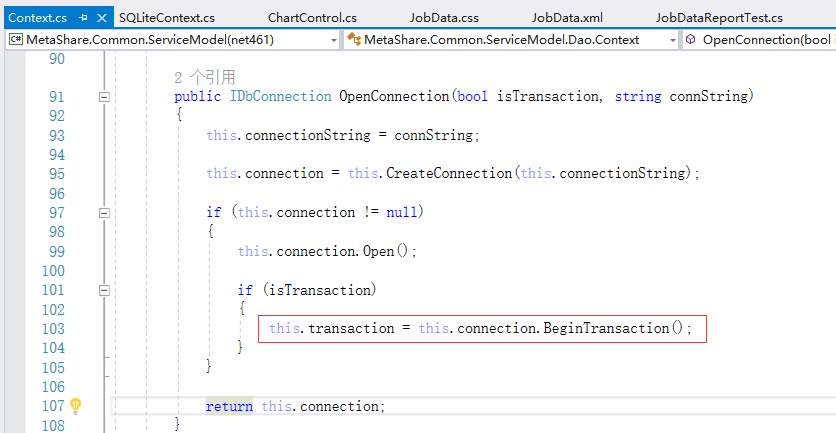
那么为什么我们在每个连接中用BEGIN IMMEDIATE开始事务，因为他防止了死锁的发生，理由如下：

一个deferred事务不获取任何锁，直到它需要锁的时候，而且BEGIN语句本身也不会做什么事情——它开始于UNLOCK状态；默认情况下是这样的。如果仅仅用BEGIN开始一个事务，那么事务就是DEFERRED的，同时它不会获取任何锁，当对数据库进行第一次读操作时，它会获取SHARED LOCK；同样，当进行第一次写操作时，它会获取RESERVED LOCK。  
 由BEGIN开始的Immediate事务会试着获取RESERVED LOCK。如果成功，BEGIN IMMEDIATE保证没有别的连接可以写数据库。但是，别的连接可以对数据库进行读操作，但是RESERVED LOCK会阻止其它的连接BEGIN IMMEDIATE或者BEGIN EXCLUSIVE命令，SQLite会返回SQLITE\_BUSY错误。这时你就可以对数据库进行修改操作，但是你不能提交，当你COMMIT时，会返回SQLITE\_BUSY错误，这意味着还有其它的读事务没有完成，得等它们执行完后才能提交事务。  
 Exclusive事务会试着获取对数据库的EXCLUSIVE锁。这与IMMEDIATE类似，但是一旦成功，EXCLUSIVE事务保证没有其它的连接，所以就可对数据库进行读写操作了。  
上面那个例子的问题在于两个连接最终都想写数据库，但是他们都没有放弃各自原来的锁，最终，shared 锁导致了问题的出现。如果两个连接都以BEGIN IMMEDIATE开始事务，那么死锁就不会发生。在这种情况下，在同一时刻只能有一个连接进入BEGIN IMMEDIATE，其它的连接就得等待。BEGIN IMMEDIATE和BEGIN EXCLUSIVE通常被写事务使用。就像同步机制一样，它防止了死锁的产生。

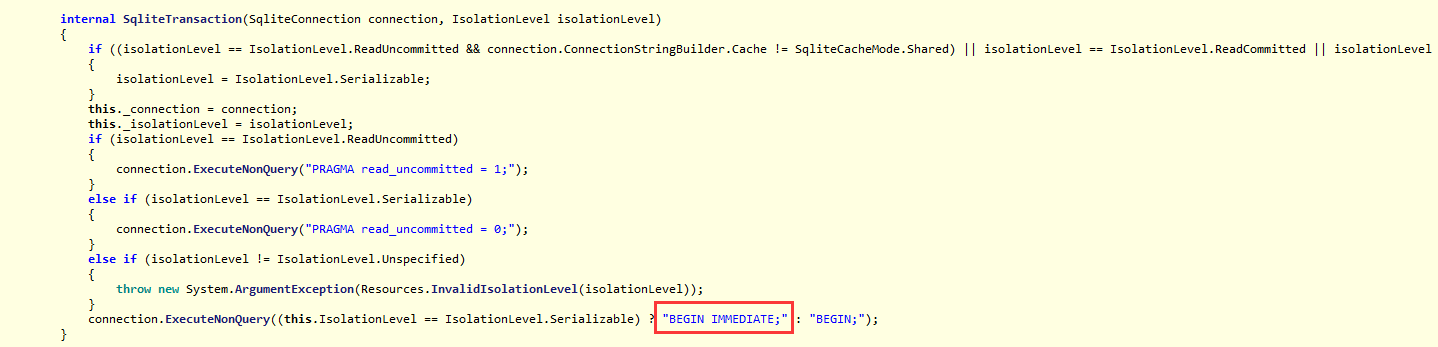
# 同一数据库多进程读写的解决方案

解决方案一：延长等待机制

根据6的阐述，结合我们目前的架构，commonlibrary里context类里的方法，如下



查看了Micosoft.Data.Sqlite的底层代码，发现他的BeginTransaction（）方法用的就是Immediate的事务类型，如下

所以绝对不会出现死锁的情况。于是解决多进程写的方式

方式一：

更改数据库链接的DefaultTimeout时间，单位是秒，原来的默认时间是30s，这样在超时前，其他链接会一直尝试去链接，超时后会提示SQLite Error 5: 'database is locked'.'

方式二：

根据6的阐述可以定义sqlite3\_busy\_handler来处理，但是目前所引程序集Micosoft.Data.Sqlite未找到如何去扩写，反编译查看也尚未找到。

解决方案二：