|  |
| --- |
| **SQLite入门与分析(一)** |
|  |
|  |
|  |
|  |
| |  | | --- | | 写在前面：出于项目的需要,最近打算对SQLite的内核进行一个完整的剖析,在此希望和对SQLite有兴趣的一起交流。我知道，这是一个漫长的过程，就像曾经去读Linux内核一样，这个过程也将是辛苦的，但我相信结果一定是美好的... ...接下来是第一章。  　　1、SQLite介绍  　　自几十年前出现的商业应用程序以来，数据库就成为软件应用程序的主要组成部分。正与数据库管理系统非常关键一样，它们也变得非常庞大，并占用了相当多的系统资源，增加了管理的复杂性。随着软件应用程序逐渐模块模块化，一种新型数据库会比大型复杂的传统数据库管理系统更适应。**嵌入式数据库直接在应用程序进程中运行，提供了零配置（zero-configuration）运行模式，并且资源占用非常少**。  　　SQLite是一个**开源的嵌入式关系数据库**，它在2000年由D. Richard Hipp发布，它减少应用程序管理数据的开销，SQLite可移植性好，很容易使用，很小，高效而且可靠。  　　SQLite嵌入到使用它的应用程序进程中，它们共用相同的进程空间，而不是单独的一个进程。从外部看，它并不像一个RDBMS，但在进程内部，它却是完整的，自包含的数据库引擎。  　　嵌入式数据库的一大好处就是在你的程序内部不需要网络配置，也不需要管理。因为客户端和服务器在同一进程空间运行。**SQLite 的数据库权限只依赖于文件系统，没有用户帐户的概念**。SQLite 有数据库级锁定，没有网络服务器。它需要的内存，其它开销很小，适合用于嵌入式设备。你需要做的仅仅是把它正确的编译到你的程序。  　　2、架构(architecture)  　　SQLite采用了模块的设计，它由三个子系统，包括8个独立的模块构成。    http://www.sqlite.com.cn/Upfiles/images/4c258ba0g6d9ae301b80b&690.jpeg    　　2.1、接口(Interface)  　　接口由**SQLite C API**组成，也就是说不管是程序、脚本语言还是库文件，最终都是通过它与SQLite交互的(我们通常用得较多的ODBC/JDBC最后也会转化为相应C API的调用)。  　　2.2、编译器(Compiler)  　　在编译器中，分词器（Tokenizer）和分析器(Parser)对SQL进行语法检查，然后把它转化为底层能更方便处理的分层的数据结构---语法树，然后把语法树传给代码生成器(code generator)进行处理。而代码生成器根据它生成一种针对SQLite的汇编代码，最后由虚拟机(Virtual Machine)执行。  　　2.3、虚拟机(Virtual Machine)  　　架构中最核心的部分是虚拟机，或者叫做**虚拟数据库引擎**(Virtual Database Engine,VDBE)。它和Java虚拟机相似，解释执行字节代码。VDBE的字节代码由128个操作码(opcodes)构成，它们主要集中在数据库操作。它的每一条指令都用来完成特定的数据库操作(比如打开一个表的游标)或者为这些操作栈空间的准备(比如压入参数)。总之，所有的这些指令都是为了满足SQL命令的要求(关于VM，后面会做详细介绍)。  　　2.4、后端(Back-End)  　　后端由B-树(B-tree)，页缓存(page cache，pager)和操作系统接口(即系统调用)构成。B-tree和page cache共同对数据进行管理。B-tree的主要功能就是索引，它维护着各个页面之间的复杂的关系，便于快速找到所需数据。而pager的主要作用就是通过OS接口在B-tree和Disk之间传递页面。  　　3、SQLite的特点(SQLite’s Features and Philosophy)  　　3.1、**零配置**(Zero Configuration)  　　3.2、**可移植**(Portability)：  　　它是运行在Windows,Linux,BSD,Mac OS X和一些商用Unix系统，比如Sun的Solaris,IBM的AIX，同样，它也可以工作在许多嵌入式操作系统下，比如QNX,VxWorks,Palm OS, Symbin和Windows CE。  　　3.3、**Compactness**：  　　SQLite是被设计成轻量级，自包含的。**one header file, one library**, and you’re relational, no external database server required  　　3.4、**简单**(Simplicity)  　　3.5、**灵活**(Flexibility)  　　3.6、**可靠**(Reliability)：  　　SQLite的核心大约有3万行标准C代码，这些代码都是模块化的，很容易阅读。 | |

|  |
| --- |
| **SQLite入门与分析(二)---设计与概念** |
|  |
|  |
|  |
|  |
| |  | | --- | | 我原打算直接从VDBE入手的，因为它起着承上启下的作用，是整个SQLite的核心，并分析源码，但考虑到这是一个系列的文章，我希望能把问题说全，所以还是从基本概念入手，对于初学者，如果没有这些概念，是很继续下去的。好了，下面开始第二章，由于这一章内容很多，我将分两部分讨论，下面开始第一部分。  　　1、API  　　由两部分组成: **核心API(core API) 和扩展API（extension API）**  　　核心API的函数实现基本的数据库操作：连接数据库，处理SQL，遍历结果集。它也包括一些实用函数，比如字符串转换，操作控制，调试和错误处理。  　　扩展API通过创建你自定义的SQL函数去扩展SQLite。  　　1.1、SQLite Version 3的一些新特点：  　　(1)SQLite的API全部重新设计，由第二版的15个函数增加到88个函数。这些函数包括支持UTF-8和UTF-16编码的功能函数。  　　(2)改进并发性能。加锁子系统引进一种锁升级模型(lock escalation model)，解决了第二版的写进程饿死的问题(该问题是任何一个DBMS必须面对的问题)。这种模型保证写进程按照先来先服务的算法得到排斥锁(Exclusive Lock)。甚至，写进程通过把结果写入临时缓冲区(Temporary Buffer)，可以在得到排斥锁之前就能开始工作。这对于写要求较高的应用，性能可提高400%（引自参考文献）。  　　(3)改进的B-树。对于表采用B+树，大大提高查询效率。      (4)SQLite 3最重要的改变是它的存储模型。由第二版只支持文本模型，扩展到**支持5种本地数据类型**。  　　总之，SQLite Version 3与SQLite Vertion 2有很大的不同，在灵活性，特点和性能方面有很大的改进。  　　1.2、主要的数据结构(The Principal Data Structures)  　　SQLite由很多部分组成－parser,tokenize,virtual machine等等。但是从程序员的角度，最需要知道的是:connection, statements, B-tree和pager。它们之间的关系如下：  http://www.sqlite.com.cn/Upfiles/images/4c258ba0g6d9ae1843ea5&690.jpeg      　　上图告诉我们在编程需要知道的三个主要方面：API,事务(Transaction)和锁(Locks)。从技术上来说，B-tree和pager不是API的一部分。但是它们却在事务和锁上起着关键作用（稍后将讨论）。  　　1.3、Connections和Statements  　　Connection和statement是执行SQL命令涉及的两个主要数据结构，几乎所有通过API进行的操作都要用到它们。一个连接(Connection)代表在一个独立的事务环境下的一个连接A (connection represents a single connection to a database as well as a single transaction context)。每一个statement都和一个connection关联，它通常表示一个编译过的SQL语句，在内部，它以VDBE字节码表示。Statement包括执行一个命令所需要一切，包括保存VDBE程序执行状态所需的资源，指向硬盘记录的B-树游标，以及参数等等。  　　1.4、B-tree和pager  　　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，每一个数据库对象有一个B-tree对象，一个B-tree有一个pager对象(这里的对象不是面向对象的“对象”，只是为了说清楚问题)。  　　Statement最终都是通过connection的B-tree和pager从数据库读或者写数据，通过B-tree的游标(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B-tree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，B-tree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)。  　　总之，关于connection和transaction，你必须知道两件事：  　　(1)对数据库的任何操作，一个连接存在于一个事务下。  　　(2)一个连接决不会同时存在多个事务下。  　　whenever a connection does anything with a database, it always operates under exactly one  　　transaction, no more, no less.  　　1.5、核心API  　　核心API 主要与执行SQL命令有关，本质上有两种方法执行SQL语句：prepared query 和wrapped query。Prepared query由三个阶段构成：preparation，execution和finalization。其实wrapped query只是对prepared query的三个过程包装而已，最终也会转化为prepared query的执行。  　　1.5.1、连接的生命周期(The Connection Lifecycle)  　　和大多数据库连接相同，由三个过程构成：  　　（1）连接数据库(Connect to the database)：  **每一个SQLite数据库都存储在单独的操作系统文件中**，连接，打开数据库的C API为：sqlite3\_open()，它的实现位于main.c文件中，如下：  int sqlite3\_open(const char \*zFilename, sqlite3 \*\*ppDb) { 　return openDatabase(zFilename, ppDb, SQLITE\_OPEN\_READWRITE | SQLITE\_OPEN\_CREATE, 0); }  **当连接一个在磁盘上的数据库，如果数据库文件存在，SQLite打开一个文件；如果不存在，SQLite会假定你想创建一个新的数据库。在这种情况下，SQLite不会立即在磁盘上创建一个文件，只有当你向数据库写入数据时才会创建文件，比如：创建表、视图或者其它数据库对象。**如果你打开一个数据，不做任何事，然后关闭它，SQLite会创建一个文件，只是一个空文件而已。  　　另外一个不立即创建一个新文件的原因是，一些数据库的参数，比如：编码，页面大小等，只在在数据库创建前设置。默认情况下，页面大小为1024字节，但是你可以选择512-32768字节之间为 2幂数的数字。有些时候，较大的页面能更有效的处理大量的数据。  　　（2）执行事务(Perform transactions)：  　　all commands are executed within transactions。默认情况下，事务自动提交，也就是每一个SQL语句都在一个独立的事务下运行。当然也可以通过使用BEGIN..COMMIT手动提交事务。  　　（3）断开连接(Disconnect from the database)：  　　主要是关闭数据库的文件。  　　1.5.2、执行Prepared Query  　　前面提到，预处理查询(Prepared Query)是SQLite执行所有SQL命令的方式，包括以下三个过程：  　　(1)Prepared Query：  　　分析器（parser），分词器(tokenizer)和代码生成器(code generator)把SQL Statement编译成VDBE字节码，编译器会创建一个statement句柄(sqlite3\_stmt)，它包括字节码以及其它执行命令和遍历结果集的所有资源。  　　相应的C API为sqlite3\_prepare()，位于prepare.c文件中，如下：  int sqlite3\_prepare( 　sqlite3 \*db,　　　　　　　 　const char \*zSql,　　　　 　int nBytes,　　　　　　　 　sqlite3\_stmt \*\*ppStmt,　　 　const char \*\*pzTail　　　)  { 　int rc; 　rc = sqlite3LockAndPrepare(db,zSql,nBytes,0,ppStmt,pzTail); 　assert( rc==SQLITE\_OK || ppStmt==0 || \*ppStmt==0 );　 　return rc; }  (2)Execution：  　　虚拟机执行字节码，执行过程是一个步进(stepwise)的过程，每一步(step)由sqlite3\_step()启动，并由VDBE执行一段字节码。由sqlite3\_prepare编译字节代码，并由sqlite3\_step()启动虚拟机执行。在遍历结果集的过程中，它返回SQLITE\_ROW，当到达结果末尾时，返回SQLITE\_DONE。  　　(3)Finalization：  　　VDBE关闭statement，释放资源。相应的C API为sqlite3\_finalize()。  　　通过下图可以更容易理解该过程：    http://www.sqlite.com.cn/Upfiles/images/4c258ba0g6d9ae664255b&690.jpeg    最后以一个具体的例子结束本节，下节讨论事务。 #include"sqlite3.h" int main(int argc,char\*\* argv) { 　　int rc,i,ncols; 　　**sqlite3**\* db; 　　**sqlite3\_stmt**\* stmt; 　　char\* sql; 　　const char\* tail; 　　//打开数据 　　rc = **sqlite3\_open**("foods.db",&db); 　　if(rc){ 　　　　fprintf(stderr,"Can'topendatabase:%sn",**sqlite3\_errmsg**(db)); 　　　　**sqlite3\_close**(db); 　　　　exit(1); 　　} 　　 　　sql="select \* from episodes"; 　　//预处理 　　rc=**sqlite3\_prepare**(db, sql, (int)strlen(sql), &stmt, &tail); 　　if(rc!=SQLITE\_OK){ 　　　　fprintf(stderr,"SQLerror:%sn",**sqlite3\_errmsg**(db)); 　　} 　　 　　rc = **sqlite3\_step**(stmt); 　　ncols = **sqlite3\_column\_count**(stmt); 　　while(rc==SQLITE\_ROW){ 　　　　 　　　　for(i=0;i<ncols;i++){ 　　　　　　fprintf(stderr,"'%s'", **sqlite3\_column\_text**(stmt, i)); 　　　　} 　　　　fprintf(stderr,"n"); 　　　　rc = **sqlite3\_step**(stmt); 　　} 　　//释放statement 　　**sqlite3\_finalize**(stmt); 　　//关闭数据库 　　**sqlite3\_close**(db); 　　return0;　　 } | |