SQLite 系统构架及虚拟机分析

党玉春[1]，翟秀云[1]，陈明通[2]
(1.攀枝花学院 机电工程学院，四川 攀枝花 617000；
2.攀枝花学院 材料工程学院，四川 攀枝花 617000)

摘 要：就目前广泛使用的轻量级数据库 SQLite 的构架进行分析，特别是对其中的虚拟数据库引擎(VDBE)做了原理性的剖析，并结合实例，展示了 SQLite 的应用及 SQLite 内部 VDBE 指令程序的运行方式。

关键词：SQLite；构架；VDBE；虚拟机
中图分类号：TP3 **文献标识码**：A **文章编号**：1674-7720(2012)10-0067-04

Configuration of SQLite system and analysis of virtual machine

Dang Yuchun[1], Zhai Xiuyun[1], Chen Mingtong[2]
(1.Electromechanical Engineering School, Panzhihua University, Panzhihua 617000, China;
2.Iron&Steel Research Institute, Panzhihua University, Panzhihua 617000, China)

Abstract: SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. This paper describes the architecture of the source code of SQLite, analyses the relationship among the modules, and focus on the internal mechanism of its virtual database engine(VDBE).

Key words: SQLite; configuration; VDBE; virtual machine

SQLite 是遵守 ACID 的轻量级关系型数据库管理系统，完全免费、开源，无需任何配置也无需任何安装程序[1]。它广泛应用在各种嵌入式系统中，在 iOS 和 Android 等系统中都是集成在各自的库中。

虚拟机是当前比较流行的一种软件构架，特别是在解释性编程语言领域。在安全领域，虚拟机也被用于实现软件的加密，是公认的一种非常高效且实用的技术手段。SQLite 用较小规模的代码用 C 语言实现了一个程序虚拟机，提高了代码的独立性，降低了耦合性，同时保持了很高的效率。

1 SQLite 数据库构架

图 1 所示为 SQLite 系统的总体构架图[2]。整体上 SQLite 可以分为前端和后端：前端负责从用户数据到平台不相关的指令的转换；后端处理数据流，深入到具体数据库数据在磁盘上的操作，这些数据是和平台相关的。SQLite 的平台无关性通过其内部实现的虚拟数据库引擎 VDME(Virtual Database Engine)来完成，总地来说，就是将 SQL 语句先翻译成一种专门设计的语言，然后下层再

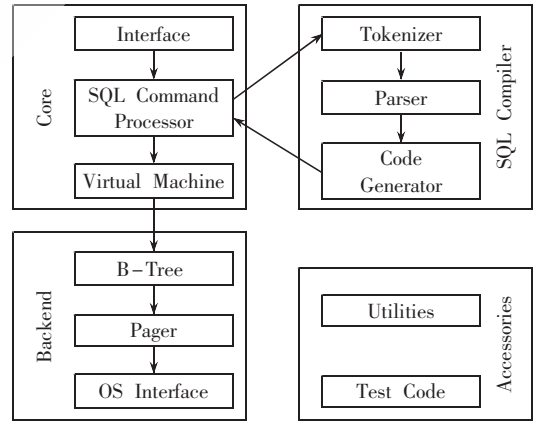

图 1 SQLite 系统的总体构架图

调用平台相关的系统 API 接口，完成相应的功能。

SQLite 的源代码由 96 个 C 语言文件(.c 和.h)组成，在编译之前会由 Makefile 生成一个完整的文件，即为可以在官方网站上下载的 sqlite3.c 和 sqlite3.h 等文件，然后编译形成所需要的库或者可执行文件。

图 1 给出了 SQLite 的主要模块及相互之间的关系，以下将分别介绍各个部分的功能。

(1)接口(Interface)

SQLite 库提供的对外不调用的接口大多数都在 main.c、legacy.c 和 vdbeapi.c 中，其他一些散布在源代码的不同部分。对接口的查询可以在文档中找到详细的介绍。为了避免命名上的冲突，所有外部可以调用的接口都以 sqlite3_ 开头[3]。

(2)SQL 编译器(SQL Compiler)

这是一个比较完整的编译器构架，分别完成词法分析、语法分析和中间代码生成。词法分析器(Tokenizer)由 C 语言实现，包含在 tokenize.c 中；语法分析器(Parser)由 Lmon LALR(1)生成，和 YACC/BISON 类似，不兼容，但是生成的代码是可重入且线程安全的，代码包含在 parse.c 中；代码生成器(Code Generator)生成虚拟机执行的中间代码，包含的文件相对较多，例如 select.c、update.c 等，大多和 SQL 命令同名对应。

(3)虚拟机 VM(Virtual Machine)

代码生成器生成的中间代码会通过 VM 执行。这部分后面会有更详细的分解。

(4)B-Tree(B-树)

数据库在磁盘上的操作都是通过 B-树的，对应于数据库中的每一个表或者索引都会有相应的 B-树。实现和接口分别在 btree.c 和 btree.h 中[4]。

(5)页缓存(Page Cache)

数据的读写都以 Chunk 为单位进行，这样可以提高效率。页缓存负责这部分工作，同时提供了回滚(roll-back)等功能，并对数据库文件进行管理。实现和接口分别在 pager.c 和 pager.h 中。

(6)系统接口(OS Interface)

SQLite 提供了一个系统抽象层，定义在 os.h 中。每个支持的平台有自己对应的实现文件，例如 os_uinx.c 和 os_win.c(及相应的头文件 os_unix.h 和 os_win.h)。

(7)功能和测试(Utility 和 Test Code)

2 VDBE 框架及关键源码分析

虚拟数据库引擎 VDBE(Virtual Database Engine)居于 SQLite 数据库的核心部分。从整个 SQLite 的构架可以看出，它处在整个系统的中间部分：前端代码完成对 SQL 语言的编译，相当于简化版本的一个编译器；后端完成物理上的操作，即利用 B-Tree 和 Pager 对物理硬盘上的数据进行实际的操作。VDBE 完成了这个层次上的抽象链接。

整个虚拟数据库引擎(VDBE)由若干个 C 语言文件组成，主题实现都包含在了 vdbe.c(vdbe.h)中。vdbeInt.h 定义了 VDBE 内部使用的各种结构和函数原型。vdbeaux.c 实现了 VDBE 内部和整个 SQLite 构建 VDBE 程序需要的其他功能性函数代码。vebeaip.c 包含了供外部接口函数(SQLite 库外的应用程序，如 sqlite3_bind 系列函数)使用的一些结构。vdbemen.c 实现了在 vdbe 的存储管理。

对于用户的 SQL 语句，编译器会生成一个虚拟机实例。虚拟机实例在内部和外部是不同的。对内看到的是一个 vdbe 结构的实例，这个结构定义在 vdbeInt.h 中，代码如下：

```
struct Vdbe {
  sqlite3 *db;                    /* 数据库连接 */
  Op *aOp;                        /* 保存虚拟机的空间 */
  …                               /* 其他指令 */
  int nOp;                        /* 生成的指令的条数 */
  char *zSql;                     /* SQL 语句 */
  …                               /* 其他指令 */
  SubProgram *pProgram;           /* 虚拟机使用的其他子程序，
                                     链表 */
};
```

一个虚拟机实例可以有多个子程序，每个子程序可以由多条指令组成。下面是子程序的结构：

```
struct SubProgram {
  VdbeOp *aOp;                    /* 指令 */
  int nOp;                        /* 指令条数 */
  int nMem;                       /* 需要的内部空间 */
  int nCsr;                       /* 需要的游标 */
  void *token;                    /* 循环触发时需要的 id */
  SubProgram *pNext;              /* 链表的下一个 */
};
```

现在的 SQLite 有 142 条操作指令，都定义在 opcodes.h 中，在 vdbe.c 中有相应的源代码，将解析一些指令作为代表，详细的技术文档可以查看官方文档。所有的指令大概可以分为 3 类：

(1)数据操作：包含算术、逻辑运算、字符串操作等；

(2)数据管理：主要关于内存和磁盘的操作。内存上如栈(stack)操作、数据的传送等，磁盘操作主要是 B-Tree 和 Pager 模块，包括打开及操作游标、事务的开始与结束等；

(3)控制流：指令的跳转。

SQL 语句在生成 VDBE 程序后，每条指令包含了一个操作码(opcode)和至多 5 个操作数(operands:P1、P2、P3、P4 和 P5)。其中：

(1)P1、P2、P3 都是 32 bit 的带符号整数，它们通常引用的是寄存器。

(2)P2 在所有的有跳转功能的指令中表示目的地址。例如上面的第 2 条指令将会跳转到第 10 条指令，然后顺序执行。

(3)P4 可以是 32 bit 或者 64 bit 的带符号整型数据、字符串、BLOB 数据(二进制大对象)、函数指针等其他多样的对象。

(4)P5 通常是无符号的字符，充当的是标识位。

在 SQLite 的 VDBE 内部，所有的指令都是 VdbeOp 结构的一个实例(定义在 vdbe.h 中)，结构的定义也主要是这 5 个操作数。

```
struct VdbeOp {
u8 opcode;                          /* 操作码类型 */
…                                   /* 其他数据接口 */
signed char p4type;                 /* p4 的类型 */
u8 p5;                              /* p5 是无符号字符型 */
int p1;                             /* 操作数 1 */
int p2;          /* 操作数 2,通常是跳转指令的目的 */
int p3;                             /* 操作数 3 */
union { /* … */ } p4;               /* p4 是一个联合,
                                    可以有不同的类型 */
…                                   /* 其他数据接口 */
};
```

由代码生成器生成的程序交由 VM 执行。sqlite3_step()会触发内部 vdbe 解释生成的 vdbe 指令。指令的执行在如下的函数中进行 (SQLITE_PRIVATE 即为 static 关键字),此处去掉了烦琐的细节,只展示其中的关键结构和一个指令的执行。

```
SQLITE_PRIVATE int sqlite3VdbeExec(
    Vdbe *p                         /* VDBE 实例 */
) {
    int   pc;                       /* 程序计数器 */
    Op   *aOp = p->aOp;             /* 得到所有的指令 */
    Op   *pOp;                      /* 当前指令 */
    int   rc= SQLITE_OK;            /* 返回值 */
    sqlite3* db = p->db;            /* 数据库连接实例 */
    u8 encoding = ENC(db);          /* UTF-8 编码 */
    …                               /* 其他初始化代码 */
    switch ( pOp->opcode ){         /* 在此之后就是一个
                                    非常大的 case 代码
        case OP_Goto: {
            CHECK_FOR_INTERRUPT;
            pc=pOp->p2-1;   /* 调整程序计数器 */
            break;
        }
        …                           /* 其他的 case 指令 */
    }
    …                               /* 其他指令 */
}
```

这个函数是整个 VDBE 的核心执行函数,虽然重要,但是代码的原理非常简单,就是一系列的 switch-case 语句。在相应的 case 情况下,会执行相应的底层代码,进行数据库的磁盘操作。

3 实验

3.1 数据库编程接口

SQLite 的编程模型比较简单,下面的例子给出了一个基本的框架。

```
#include "sqlite3.h"
#include <stdlib.h>
int main(int argc, char **argv)
{
    char      *file = "./test.db";      /* 数据库文件 */
    sqlite3   *db = NULL;       /* 数据库连接实例 */
    int       rc = 0;                   /* 返回值 */
    sqlite3_initialize();               /* 初始库 */
    rc= sqlite3_open_v2(file, &db,
                SQLITE_OPEN_READWRITE, NULL);
            /* 准备 SQL 语句,生成 VDBE 程序 */
    sqlite3_stmt    *stmt = NULL:
    rc=sqlite3_prepare_v2(db, "SELECT * FROM FILM",
                                -1, &stmt, NULL);
        if (rc != SQLITE_OK) exit(-1);
    while (sqlite3_step(stmt) == SQLITE_ROW) {
        const char *data = (const char*)
                        sqlite3_column_text(stmt, 0);
        printf("%s\n", data?data:"[NULL]");
    }
    sqlite3_finalize(stmt);
    sqlite3_close(db);                  /* 关闭 */
    sqlite3_shutdown();                 /* 释放资源 */
}
```

在上面的例子中,使用了 sqlite3_prepare_v2()和 sqlite3_step()函数,这是和内部的虚拟机联系非常紧密的两个函数,也是了解 SQLite 虚拟机的两个点。sqlite3_prepare_v2()完成的是将 SQL 语句提交给 SQL 编译器,编译成 VDBE 指令程序,sqlite3_step()将驱动 VDBE 执行指令程序。

从应用上来说,这仅仅是最简单的数据库应用框架,更多的接口信息可以查看官方的文档。

3.2 VDBE 程序分析

在官方提供的下载中,有编译好的命令行可执行程序,可以作为完全的 SQLite 数据库管理工具。同时,它也考虑了一些 Debug 和 Test 功能,可以利用它们深入了解 SQLite 的内部机制。可以利用 SQLite 命令行程序中的 explain 命令查看由代码生成器生成的中间代码的形式,这只需要在相应的 SQL 代码前面加上 explain 就可以了。如以搜索的命令行显示(如图 2 所示,箭头表示实际执行顺序):

图 2 中,“addr”列是虚拟机的地址编号,并不是指令执行的顺序,由于跳转指令的存在,用箭头标示出了指令运行的实际顺序,也可以在 SQLite 编译时指定相应的选项,然后利用指令“pragma vdbe_trace=on;”详细地看到指令的运行过程和堆栈的变化情况。

指令 0~指令 12 都是对 SQLite 数据库内部的准备:由指令 1 跳转到指令 10,指令 10(Transaction)开始一个事务,指令 11(VerifyCookie)在执行一个指令前检查数据库模式是否发生了变化,当发生了变化时要重置,指令 12(TableLock)将要读的数据库表锁起来,指令 13(Goto)跳转到指令 2。

从指令 2 开始是实际的对数据库的操作了。指令 2

图 2 SQLite 中间代码示例(命令行)

(OpenRead)会打开一个数据库表的只读游标,P1 作为这个游标的标志,P2 是打开的数据库表的根页(root page),P3==0 表明是主数据库,P4 表明数据库有两列,P5 说明是以 P2 的值作为根页。(OpenRead 指令的各个操作数还可以有其他含义,这里只是针对这条 SQL 语句的解释,请查看技术文档。)指令 3(Rewind)~指令 7(Next)完成了对所有查询数据的遍历。指令 8(Close)关闭游标,指令 9(Halt)结束这个 VDBE 程序。

VDBE 对上层提供的就是这样的接口,而对下层将是调用相应的接口实现相应的功能,并由此完成模块上的解耦合。

由 VDBE 的定义、代码分析及以上的实验,可以总结出 SQLite 的整体构架:

外部调用 SQLite 接口函数 sqlite3_prepare(),SQL 语句通过 SQL 编译器生成对应的 VDBE 指令程序;

内部调用 sqlite3_step()驱动,内部执行 sqlite3VdbeExec(),switch-case 语句执行相应指令。底层通过 B-Tree 和 Pager 实现对磁盘数据库文件的管理,如图 3 所示。

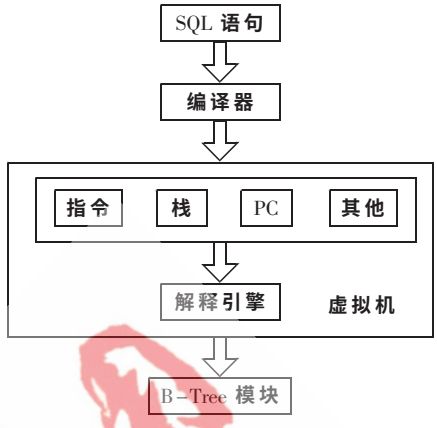

图 3 SQLite 的整体构架

在实际应用中,可以设计一个面向应用的指令集,利用程序虚拟机设计中间抽象层,提高平台通用性。同时程序虚拟机也为语言虚拟机、系统虚拟机及安全沙盒等技术提供了技术基础。

参考文献

[1] OWENS M.The definitive guide to SQLite[M].Apress,2006.

[2] KREIBICH J A.Using SQLite[M].O'Reilly Media,2010.

[3] 李蔚,陈亚峰.嵌入式数据库 SQLite 及其应用研究[J].沿海企业与科技,2010(10):45-47.

[4] 杜国祥,石俊杰.SQLite 嵌入式数据库的应用[J].电脑编程技巧与维护,2010(14):43-46.

(收稿日期:2012-01-20)

作者简介:

党玉春,女,1963 年生,硕士,副教授,主要研究方向:工业工程。