# GDBM编程接口

1. gdbm接口
2. gdbm\_open();
3. gdbm\_strerror();
4. gdbm\_close();
5. gdbm\_sync(); // 调用了<unistd.h> fsync()
6. gdbm\_store();
7. gdbm\_fetch();
8. gdbm\_delete();
9. gdbm\_exists();
10. gdbm\_firstkey();
11. gdbm\_nextkey();
12. 封装后接口

**public:**

1. **dbm\_shell**(**void**);**~dbm\_shell**();
2. **bool** **open**(**const** std::string& path);
3. **void** **close**(**void**);
4. **void** **sync**(**void**);
5. **bool** **insert**(**const** **void**\* key\_buf, size\_t key\_len, **const** **void**\* val\_buf, size\_t val\_len);
6. **bool** **replace**(**const** **void**\* key\_buf,size\_t key\_len, **const** **void**\* val\_buf, size\_t val\_len);
7. **bool** **fetch**(**const** **void**\* key\_buf, size\_t key\_len, base::char\_sptr\* val\_buf, size\_t\* val\_len);
8. **bool** **erase**(**const** **void**\* key\_buf, size\_t key\_len);
9. **bool** **exist**(**const** **void**\* key\_buf, size\_t key\_len);
10. **bool** **first**(base::char\_sptr\* key\_buf, size\_t\* key\_len);
11. **bool** **next**(**const** **void**\* key\_buf, size\_t key\_len, base::char\_sptr\* nkey\_buf, size\_t\* nkey\_len);
12. size\_t **size**(**void**) **const** { **return** \_size; }

返回数据库记录数

1. size\_t **mark\_erasure**(**const** base::char\_sptr& key\_buf, size\_t key\_len);

将待删除数据的键值存储在一个vector容器中

1. size\_t **erase\_marked**(**void**);

调用erase()函数清除vector容器中记录的键值所对应的数据，其中erase()函数调用::gdbm\_delete()

1. std::string **path**(**void**) { **return** \_path; }

**private**:

1. **bool** **store**(**const** **void**\* key\_buf, size\_t key\_len, **const** **void**\* val\_buf, size\_t val\_len, **int** flag);
2. **struct** mark\_unit{ base::char\_sptr key\_buf; size\_t key\_len; };
3. std::vector<mark\_unit> \_mark\_units;
4. base::log\_ptr \_log;
5. GDBM\_FILE \_gdbf;
6. std::string \_path;
7. size\_t \_size;

# SQLite特性

1. 调用少量的C函数memset()、memcpy()、memcmp()、strcmp()、malloc()、free()及realloc()
2. 独立的C函数库：sqlite3.c、sqlite3.h 、sqlite3ext.h及libsqlite3.so
3. 用户接口shell.c，可编译生成命令行处理器
4. 对跨平台的磁盘文件及内存进行操作
5. 无配置文件，只有针对每一个数据库文件的编译选项（PRAGMA）和编程时的编译参数（宏）

# SQLite命令行处理器

Command Line Processor

|  |  |  |
| --- | --- | --- |
| SQLite version 3.7.13 2012-06-11 02:05:22  Enter ".help" for instructions  Enter SQL statements terminated with a ";"  sqlite> .help | | |
| .backup ?DB? FILE | Backup DB (default "main") to FILE | |
| .bail ON|OFF | Stop after hitting an error.  Default OFF | |
| .databases | List names and files of attached databases | |
| .dump ?TABLE? ... | Dump the database in an SQL text format  If TABLE specified, only dump tables matching  LIKE pattern TABLE. | |
| .echo ON|OFF | Turn command echo on or off | |
| .exit | Exit this program | |
| .explain ?ON|OFF? | Turn output mode suitable for EXPLAIN on or off.  With no args, it turns EXPLAIN on. | |
| .header(s) ON|OFF | Turn display of headers on or off | |
| .help | Show this message | |
| .import FILE TABLE | Import data from FILE into TABLE | |
| .indices ?TABLE? | Show names of all indices  If TABLE specified, only show indices for tables matching  LIKE pattern TABLE. | |
| .load FILE ?ENTRY? | Load an extension library | |
| .log FILE|off | Turn logging on or off. FILE can be stderr/stdout | |
| .mode MODE ?TABLE? | Set output mode where MODE is one of: | |
| csv | Comma-separated values |
| column | Left-aligned columns. (See .width) |
| html | HTML <table> code |
| insert | SQL insert statements for TABLE |
| line | One value per line |
| list | Values delimited by .separator string |
| tabs | Tab-separated values |
| tcl | TCL list elements |
| .nullvalue STRING | Print STRING in place of NULL values | |
| .output FILENAME | Send output to FILENAME | |
| .output stdout | Send output to the screen | |
| .prompt MAIN CONTINUE | Replace the standard prompts | |
| .quit | Exit this program | |
| .read FILENAME | Execute SQL in FILENAME | |
| .restore ?DB? FILE | Restore content of DB (default "main") from FILE | |
| .schema ?TABLE? | Show the CREATE statements  If TABLE specified, only show tables matching  LIKE pattern TABLE. | |
| .separator STRING | Change separator used by output mode and .import | |
| .show | Show the current values for various settings | |
| .stats ON|OFF | Turn stats on or off | |
| .tables ?TABLE? | List names of tables  If TABLE specified, only list tables matching  LIKE pattern TABLE. | |
| .timeout MS | Try opening locked tables for MS milliseconds | |
| .trace FILE|off | Output each SQL statement as it is run | |
| .vfsname ?AUX? | Print the name of the VFS stack | |
| .width NUM1 NUM2 ... | Set column widths for "column" mode | |
| .timer ON|OFF | Turn the CPU timer measurement on or off | |
| sqlite> | | |

# SQLite工作模式



# SQLite临时文件

1. Rollback Journals

出现在事务的起始点，消失在事务的结束点

若PRAGMA locking\_mode=EXCLUSIVE; // 默认为normal

则不会在事务结束点删除回滚日志

若pragma journal\_mode=persist; // 默认为delete

也不会在事务结束点删除回滚日志

若pragma journal\_mode=off;

则不会产生回滚日志

存储于磁盘上

1. Master Journal Files

在一个数据库连接使用attach命令接入N个外部数据库的情况下，事务开始点产生的回滚日志

存储于磁盘上

1. Statement Journal Files

针对单个SQL语句的回滚日志

存储于磁盘上

1. Temp Databases

使用PRAGMA synchronous=OFF

使用PRAGMA journal\_mode=PERSIST

1. Materializations Of Views And Subqueries

存储视图、子查询等临时表

1. Transient Indices

瞬时索引，比如

ORDER BY从句

GROUP BY从句

DISTINCT聚合查询关键词

复合SELECT从句joined by UNION, EXCEPT或INTERSECT

1. Transient Database Used By VACUUM

使用VACUUM压缩数据库文件时使用的临时文件

1. P.S.

Temp Databases

Materializations Of Views And Subqueries

Transient Indices

Transient Database Used By VACUUM

这四类临时文件的存储位置受pragma temp\_store和SQLITE\_TEMP\_STORE影响，如下表所示：



但是即使将它们的位置设置为磁盘，SQLite也会将它们存储在内存页中，直到内存页满才写进磁盘，而这个内存页大小由SQLITE\_DEFAULT\_TEMP\_CACHE\_SIZE决定，默认是500页。

# SQLite数据库切换

1. 数据库切换



分为以下三种情况

1. 文件数据库与文件数据库的切换

伪代码示例

dbswitch(**const** **char**\* dbfile\_to, **const** **char**\* dbfile\_from)

{

sqlite3\_open(dbfile\_to, &pto);

sqlite3\_open(dbfile\_from, &pfrom);

pbackup = sqlite3\_backup\_init(pto, "main", pfrom, "main");

**if** (pbackup)

{

sqlite3\_backup\_step(pbackup, -1);

sqlite3\_backup\_finish(pbackup);

}

sqlite3\_close(pto);

sqlite3\_close(pfrom);

}



1. 内存数据库与文件数据库的切换

dbswitch(**sqlite3**\* pmem, **const** **char**\* dbfile\_from)

{

sqlite3\_open(dbfile\_from, &pfrom);

pbackup = sqlite3\_backup\_init(pmem, "main", pfrom, "main");

**if** (pbackup)

{

sqlite3\_backup\_step(pbackup, -1);

sqlite3\_backup\_finish(pbackup);

}

sqlite3\_close(pfrom);

}



1. 内存数据库与内存数据库的切换

dbswitch(**sqlite3**\* pmem\_to, **sqlite3**\* pmem\_from)

{

pbackup = sqlite3\_backup\_init(pmem\_to, "main", pmem\_from, "main");

**if** (pbackup)

{

sqlite3\_backup\_step(pbackup, -1);

sqlite3\_backup\_finish(pbackup);

}

}



# 基于enc::base::jssdb百万级数据库操作

1. 打开数据库
2. 指定数据库模式（磁盘文件、内存驻留）

调用sqlite3\_open()

1. 编译选项设置

调用sqlite3\_exec()执行如下设置

PRAGMA synchronous;（对常规操作的速度影响极大）

PRAGMA synchronous = FULL;(2) // 最佳安全性

PRAGMA synchronous = NORMAL;(1) // 安全性与速度平衡

PRAGMA synchronous = OFF;(0) // 最佳速度

1. 若采用磁盘文件模式（支持最大2TB，同时受操作系统最大文件大小限制，一般支持4GB以内文件大小，对于百万级的记录数，磁盘文件大小的限制不用考虑）
2. 临时缓冲参数设置

连接缓冲区大小

PRAGMA cache\_size; // 一个连接可以占用内存的页数，对当前连接有效

页大小

PRAGMA page\_size; // 二的N次幂字节

临时存储

PRAGMA temp\_store;

1. 关闭数据库

调用sqlite3\_close()

1. 关闭文件数据库

数据保存在本地磁盘

1. 关闭内存数据库

没有dump的话全部数据消失

// 常规操作

1. 打印全部记录（遍历，fetch all）print

调用sqlite3\_get\_table()执行’select \* from *tablename*;’，然后从结果数组中可获取结果

1. 根据key查找并获取记录（fetch key）

调用sqlite3\_get\_table()执行’select \* from *tablename* where *key* = \*\*;’，然后从结果数组中可获取结果

1. 插入一条记录

调用sqlite3\_exec()执行’insert into *tablename*(\*\*) values(\*\*);’

1. 替换一条记录

调用sqlite3\_exec()执行’update *tablename* set \*\* = \*\* where *key* = \*\*;’

1. 删除记录（标注待删除记录，删除已标注记录）

将待删除记录主键值录入某个结构，最后读取该结构的主键值进行删除操作，可调用sqlite3\_exec()执行’delete from *tablename* where *key* = \*\*;’

1. 逐条处理记录（获取第一条记录，处理，获取下一条记录，循环）

综合4-7的操作

// 特殊操作

1. 返回数据库表中的记录数

可以通过一个变量表中的记录数目

或者执行select count(\*) from *tablename*;获取记录数目

1. 数据库同步，默认同步main数据库
2. 内存->文件（dump）// 备份功能
3. 文件->内存（load）
4. 指向文件数据库的指针
5. 指向内存数据库的指针
6. 同步方式一，调用系统函数fsync()
7. 同步方式二，调用SQLite C接口

sqlite3\_backup\_init()

sqlite3\_backup\_step()

sqlite3\_backup\_finish()



# 应用框架中的SQLite类设计

1. 方法一：以操作为接口，将SQL语句隐藏与类成员函数之内

将数据库配置、事务性、锁机制写进成员函数之内



\_sldb：数据库句柄

\_key：主键

\_mode：模式（内存/文件）

\_size：记录条数

1. 方法二：以SQL语句为接口，将SQLite C API封装为类成员函数

接口灵活，但需要调用者考虑数据库配置、事务性和锁机制



\_sldb：数据库句柄

\_key：主键

\_mode：模式（内存/文件）

\_size：记录条数