SQLite3 C 语言 API 入门

下载 SQLite3

我们下载 sqlite 源码包，只需要其中的 sqlite3.c、sqlite.h 即可。

最简单的一个创建表操作

```
#include <stdio.h>
#include "sqlite3.h"

int main(int argc,char *argv[]){
   const char *sql_create_table="create table t(id int primary key,msg varchar(128))";
   char *errmsg = 0;
   int ret = 0;

   sqlite3 *db = 0;
   ret = sqlite3_open("./sqlite3-demo.db",&db);
   if(ret != SQLITE_OK){
      fprintf(stderr,"Cannot open db: %s\n",sqlite3_errmsg(db));
      return 1;
   }
   printf("Open database\n");

   ret = sqlite3_exec(db,sql_create_table,NULL,NULL,&errmsg);
   if(ret != SQLITE_OK){
      fprintf(stderr,"create table fail: %s\n",errmsg);
   }
   sqlite3_free(errmsg);
   sqlite3_close(db);

   printf("Close database\n");

   return 0;
}
```

在这个操作中我们执行了如下操作：

- 打开数据库
- 执行 SQL 语句
- 关闭数据库

当然这中间会有一些状态的判断以及内存指针的释放等。
打开数据库的 API 如下：

```
int sqlite3_open(
  const char *filename,   /* Database filename (UTF-8) */
  sqlite3 **ppDb          /* OUT: SQLite db handle */
);
```

这里会引入一个非常复杂的 sqlite3 的数据结构。这个根据需要以后酌情了解些。

打开数据库除了这种形式意外，还有 sqlite3_open、sqlite3_open16、sqlite3_open_v2 几种形式，基本上类似。
大部分 sql 操作都可以通过 sqlite3_exec 来完成，它的 API 形式如下：

```
int sqlite3_exec(
```

```
  sqlite3*,                                  /* An open database */
  const char *sql,                           /* SQL to be evaluated */
  int (*callback)(void*,int,char**,char**),  /* Callback function */
  void *,                                    /* 1st argument to callback */
  char **errmsg                              /* Error msg written here */
);
```

各个参数的意义为：

- sqlite3 描述的是数据库句柄
- sql 要执行的 SQL 语句
- callback 回调函数
- void *回调函数的第一个参数
- errmsg 错误信息，如果没有 SQL 问题则值为 NULL

回调函数式一个比较复杂的函数。它的原型是这样的:

```
int callback(void *params,int column_size,char **column_value,char **column_name){
```

每一个参数意义如下：

- params 是 sqlite3_exec 传入的第四个参数
- column_size 是结果字段的个数
- column_value 是返回记录的一位字符数组指针
- column_name 是结果字段的名称

通 常情况下 callback 在 select 操作中会使用到，尤其是处理每一行记录数。返回的结果每一行记录都会调用下“回调函数”。 如果回调函数返回了非 0，那么 sqlite3_exec 将返回 SQLITE_ABORT，并且之后的回调函数也不会执行，同时未执行的子查询也不会继续执 行。
对于更新、删除、插入等不需要回调函数的操作，sqlite3_exec 的第三、第四个参数可以传入 0 或者 NULL。
通常情况下 sqlite3_exec 返回 SQLITE_OK=0 的结果，非 0 结果可以通过 errmsg 来获取对应的错误描述。
Windows 下编译：

```
D:\home\dev\c>cl /nologo /TC sqlite3-demo.c sqlite3.c
```

GCC 下编译:

```
$ gcc -o sqlite3-demo.bin sqlite3-demo.c sqlite3.c
```

删除表操作

为了防止垃圾数据，我们在加载数据库的时候删除表操作。
简单的删除操作可以直接使用 sqlite3_exec 即可。这里不需要回调函数以及回调函数的参数。 当然需要可以关注 sqlite3_exec 返回的结果是否为 SQLITE_OK 的值。

```
    const char *sql_drop_table="drop table if exists t";
   const char *sql_create_table="create table t(id int primary key,msg varchar(128))";
... ...
   sqlite3_exec(db,sql_drop_table,0,0,&errmsg);
   sqlite3_exec(db,sql_create_table,0,0,&errmsg);
```

插入数据

插入第一条数据

```
    ret = sqlite3_exec(db,"insert into t(id,msg) values(1,'Ady Liu')",NULL,NULL,&errmsg);
    printf("Insert a record %s\n",ret == SQLITE_OK ? "OK":"FAIL");
```

返回值 ret 为 SQLITE_OK 即操作成功。

插入多条数据，并删除数据

```
    ret = sqlite3_exec(db,"insert into t(id,msg) values(1,'Ady Liu')",NULL,NULL,&errmsg);
    printf("Insert a record %s\n",ret == SQLITE_OK ? "OK":"FAIL");
    ret = sqlite3_exec(db,"insert into t(id,msg) values(2,'IMXYLZ')",NULL,NULL,&errmsg);
    printf("Insert a record %s\n",ret == SQLITE_OK ? "OK":"FAIL");
    ret = sqlite3_exec(db,"delete from t where id < 3",NULL,NULL,&errmsg);
    printf("Delete records: %s\n",ret == SQLITE_OK ? "OK":"FAIL");
```

插入多条数据，简单的使用 sqlite3_exec 进行 SQL 执行即可。当然这里是完整的 SQL 字符串。

预编译操作

```
    int i = 0;
    sqlite3_stmt *stmt;
    char ca[255];
... ...
    //prepare statement
    sqlite3_prepare_v2(db,"insert into t(id,msg) values(?,?)",-1,&stmt,0);
    for(i=10;i<20;i++){
        sprintf(ca,"HELLO#%i",i);
        sqlite3_bind_int(stmt,1,i);
        sqlite3_bind_text(stmt,2,ca,strlen(ca),NULL);
        sqlite3_step(stmt);
        sqlite3_reset(stmt);
    }
    sqlite3_finalize(stmt);
```

预编译操作比较麻烦的，完整的预编译操作的流程是：

1. 通过 sqlite3_prepare_v2()创建一个 sqlite3_stmt 对象
2. 通过 sqlite3_bind_*()绑定预编译字段的值
3. 通过 sqlite3_step()执行 SQL 语句
4. 通过 sqlite3_reset()重置预编译语句，重复操作 2 多次
5. 通过 sqlite3_finalize()销毁资源

sqlite3_prepare_v2()有个多种类似的形式，完整的 API 语法是：

```
int sqlite3_prepare(
  sqlite3 *db,            /* Database handle */
  const char *zSql,       /* SQL statement, UTF-8 encoded */
  int nByte,              /* Maximum length of zSql in bytes. */
  sqlite3_stmt **ppStmt,  /* OUT: Statement handle */
  const char **pzTail     /* OUT: Pointer to unused portion of zSql */
);
```

各个参数的定义为：

- db 为 sqlite3 的句柄
- zSql 为要执行的 SQL 语句
- nByte 为要执行语句在 zSql 中的最大长度，如果是负数，那么就需要重新自动计算

- ppStmt 为预编译后的句柄
- pzTail 预编译后剩下的字符串（未预编译成功或者多余的）的指针，通常没什么用，传入 0 或者 NULL 即可。

绑定参数 sqlite3_bind_*有多种形式，分别对应不同的数据类型:

```
int sqlite3_bind_blob(sqlite3_stmt*, int, const void*, int n, void(*)(void*));
int sqlite3_bind_double(sqlite3_stmt*, int, double);
int sqlite3_bind_int(sqlite3_stmt*, int, int);
int sqlite3_bind_int64(sqlite3_stmt*, int, sqlite3_int64);
int sqlite3_bind_null(sqlite3_stmt*, int);
int sqlite3_bind_text(sqlite3_stmt*, int, const char*, int n, void(*)(void*));
int sqlite3_bind_text16(sqlite3_stmt*, int, const void*, int, void(*)(void*));
int sqlite3_bind_value(sqlite3_stmt*, int, const sqlite3_value*);
int sqlite3_bind_zeroblob(sqlite3_stmt*, int, int n);
```

预编译 SQL 语句中可以包含如下几种形式：

- ?
- ?NNN
- :VVV
- @VVV
- $VVV

NNN 代表数字，VVV 代表字符串。
如 果是?或者?NNN，那么可以直接 sqlite3_bind_*()进行操作，如果是字符串，还需要通过 sqlite3_bind_parameter_index()获取对应的 index，然后再调用 sqlite3_bind_*()操作。这通常用于构造 不定条件的 SQL 语句（动态 SQL 语句）。

查询操作

回调函数的解释参考最上面的描述。 首先声明一个回调函数。

```
int print_record(void *,int,char **,char **);
```

查询代码

```
    //select data
    ret = sqlite3_exec(db,"select * from t",print_record,NULL,&errmsg);
    if(ret != SQLITE_OK){
        fprintf(stderr,"query SQL error: %s\n",errmsg);
    }
```

现在定义回调函数，只是简单的输出字段值。

```
int print_record(void *params,int n_column,char **column_value,char **column_name){
    int i;
    for(i=0;i<n_column;i++){
        printf("\t%s",column_value[i]);
    }
    printf("\n");
    return 0;
}
```

不使用回调的查询操作

定义使用的变量

```
char **dbresult; int j,nrow,ncolumn,index;
```

查询操作

```
 //select table
ret = sqlite3_get_table(db,"select * from t",&dbresult,&nrow,&ncolumn,&errmsg);
if(ret == SQLITE_OK){
    printf("query %i records.\n",nrow);
    index=ncolumn;
    for(i=0;i<nrow;i++){
        printf("[%2i]",i);
        for(j=0;j<ncolumn;j++){
            printf(" %s",dbresult[index]);
            index++;
        }
        printf("\n");
    }
}
sqlite3_free_table(dbresult);
```

sqlite3_get_table 的 API 语法：

```
int sqlite3_get_table(
  sqlite3 *db,          /* An open database */
  const char *zSql,     /* SQL to be evaluated */
  char ***pazResult,    /* Results of the query */
  int *pnRow,           /* Number of result rows written here */
  int *pnColumn,        /* Number of result columns written here */
  char **pzErrmsg       /* Error msg written here */
);
void sqlite3_free_table(char **result);
```

其中：

- db 是 sqlite3 的句柄
- zSql 是要执行的 sql 语句
- pazResult 是执行查询操作的返回结果集
- pnRow 是记录的行数
- pnColumn 是记录的字段个数
- pzErrmsg 是错误信息

由于 sqlite3_get_table 是 sqlite3_exec 的包装，因此返回的结果和 sqlite3_exec 类似。
pazResult 是一个(pnRow+1)*pnColumn 结果集的字符串数组，其中前 pnColumn 个结果是字段的名称，后 pnRow 行记录是真实的字段值，如果某个字段为空，则对应值为 NULL。
最后需要通过 sqlite3_free_table()释放完整的结果集。

更新操作

```
sqlite3_exec(db,"update t set msg='MESSAGE#10' where id=10",NULL,NULL,&errmsg);
```

当然了，我们也可以使用预编译方法进行更新操作。

受影响的记录数

我们可以使用 sqlite3_change(sqlite3 *)的 API 来统计上一次操作受影响的记录数。

```
 ret = sqlite3_exec(db,"delete from t",NULL,NULL,&errmsg);

if(ret == SQLITE_OK){

printf("delete records: %i\n",sqlite3_changes(db));

}
```

总结

这里我们接触了 SQLITE3 的 13 个 API：

- sqlite3_open()
- sqlite3_exec()
- sqlite3_close()
- sqlite3_prepare_v2
- sqlite3_bind_*()
- sqlite3_bind_parameter_index()
- sqlite3_step()
- sqlite3_reset()
- sqlite3_finalize()
- sqlite3_get_table
- sqlite3_change()
- sqlite3_free()
- sqlite3_free_table()

事实上截止到 SQLITE3.7.14(2012/09/03) 一共提供了 204 个 API 函数（ http://www.sqlite.org/c3ref/funclist.html ）。
但最精简的 API 函数大概有 6 个：

- sqlite3_open()
- sqlite3_prepare()
- sqlite3_step()
- sqlite3_column()
- sqlite3_finalize()
- sqlite3_close()

核心 API 也就 10 个（在精简 API 基础上增加 4 个）：

- sqlite3_exec()
- sqlite3_get_table()
- sqlite3_reset()
- sqlite3_bind()

因此掌握起来还是比较容易的。