

蓝图(lantutechnology)软件开发小组

联系方式: lantutechnology@163.com xRedRobe@163.com

博客园 闪存 首页 新随笔 联系 管理 订阅 XML

sqlite3使用简介

转自: <http://blog.csdn.net/kfqcome/article/details/6570495>

一. 使用流程

要使用sqlite, 需要从sqlite官网下载到三个文件, 分别为sqlite3.lib, sqlite3.dll, sqlite3.h, 然后再在自己的工程中配置好头文件和库文件, 同时将dll文件放到当前目录下, 就完成配置可以使用sqlite了。

使用的过程根据使用的函数大致分为如下几个过程:

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

这几个过程是概念上的说法, 而不完全是程序运行的过程, 如sqlite3_column()表示的是对查询获得一行里面的数据的列的各个操作统称, 实际上在sqlite中并不存在这个函数。

1. sqlite3_open(): 打开数据库

在操作数据库之前, 首先要打开数据库。这个函数打开一个sqlite数据库文件的连接并且返回一个数据库连接对象。这个操作同时程序中的第一个调用的sqlite函数, 同时也是其他sqlite api的先决条件。许多的sqlite接口函数都需要一个数据库连接对象的指针作为它们的第一个参数。

函数定义

```
1 int sqlite3_open(  
2  
3     const char *filename,      /* Database filename (UTF-8) */  
4  
5     sqlite3 **ppDb            /* OUT: SQLite db handle */  
6  
7 );  
8  
9 int sqlite3_open16(  
10  
11     const void *filename,      /* Database filename (UTF-16) */  
12  
13     sqlite3 **ppDb            /* OUT: SQLite db handle */  
14  
15 );  
16  
17 int sqlite3_open_v2(  
18  
19     const char *filename,      /* Database filename (UTF-8) */  
20  
21     sqlite3 **ppDb,            /* OUT: SQLite db handle */  
22  
23     int flags,                  /* Flags */  
24  
25     const char *zVfs            /* Name of VFS module to use */  
26  
27 );
```

<		2011年7月						>		
日	一	二	三	四	五	六				
26	27	28	29	30	1	2				
3	4	5	6	7	8	9				
10	11	12	13	14	15	16				
17	18	19	20	21	22	23				
24	25	26	27	28	29	30				
31	1	2	3	4	5	6				

搜索

找我看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

我的标签

iphone(2)
sqlite(1)
安装包(1)
海量数据(1)
数据库(1)
.net(1)
app(1)
C#(1)
excel(1)
ipa(1)
更多

随笔档案

2014年5月 (1)
2012年1月 (1)
2011年11月 (1)
2011年9月 (1)
2011年8月 (1)
2011年7月 (3)
2010年12月 (1)
2009年2月 (1)
2007年12月 (1)
2007年11月 (1)
2007年10月 (1)
2007年8月 (1)
2007年4月 (1)
2007年3月 (1)
2007年2月 (3)

资料链接

ASP.NET开发实践系列
C++ 博客-小明思考
Web 服务工厂
支持asp.net2.0和Sql server及Access的

说明:

[免费空间](#)

[支持asp.net2.0免费空间说](#)

假如这个要被打开的数据文件不存在, 则一个同名的数据库文件将被创建。如果使用sqlite3_open和sqlite3_open_v2的话, 数据库将采用UTF-8的编码方式, sqlite3_open16采用UTF-16的编码方式

返回值:

如果sqlite数据库被成功打开(或创建), 将会返回SQLITE_OK, 否则将会返回错误码。Sqlite3_errmsg()或者sqlite3_errmsg16可以用于获得数据库打开错误码的英文描述, 这两个函数定义为:

```
1 const char *sqlite3_errmsg(sqlite3*);
2
3 const void *sqlite3_errmsg16(sqlite3*);
```

参数说明:

filename: 需要被打开的数据库文件的文件名, 在sqlite3_open和sqlite3_open_v2中这个参数采用UTF-8编码, 而在sqlite3_open16中则采用UTF-16编码

ppDb: 一个数据库连接句柄被返回到这个参数, 即使发生错误。唯一的一场是如果sqlite不能分配内存来存放sqlite对象, ppDb将会被返回一个NULL值。

flags: 作为数据库连接的额外控制的参数, 可以是SQLITE_OPEN_READONLY, SQLITE_OPEN_READWRITE和SQLITE_OPEN_READWRITE|SQLITE_OPEN_CREATE中的一个, 用于控制数据库的打开方式, 可以和SQLITE_OPEN_NOMUTEX, SQLITE_OPEN_FULLMUTEX, SQLITE_OPEN_SHARED_CACHE, 以及SQLITE_OPEN_PRIVATECACHE结合使用, 具体的详细情况可以查阅文档

2. Sqlite3_prepare()

这个函数将sql文本转换成一个准备语句(prepared statement)对象, 同时返回这个对象的指针。这个接口需要一个数据库连接指针以及一个要准备的包含SQL语句的文本。它实际上并不执行(evaluate)这个SQL语句, 它仅仅为执行准备这个sql语句

函数定义(仅列出UTF-8的)

```
1 int sqlite3_prepare(
2
3     sqlite3 *db,           /* Database handle */
4
5     const char *zSql,      /* SQL statement, UTF-8 encoded */
6
7     int nByte,             /* Maximum length of zSql in bytes. */
8
9     sqlite3_stmt **ppStmt, /* OUT: Statement handle */
10
11     const char **pzTail    /* OUT: Pointer to unused portion of zSql */
12 );
13
14
15 int sqlite3_prepare_v2(
16
17     sqlite3 *db,           /* Database handle */
18
19     const char *zSql,      /* SQL statement, UTF-8 encoded */
20
21     int nByte,             /* Maximum length of zSql in bytes. */
22
23     sqlite3_stmt **ppStmt, /* OUT: Statement handle */
24
25     const char **pzTail    /* OUT: Pointer to unused portion of zSql */
26 );
27
```

参数:

db: 数据指针

zSql: sql语句, 使用UTF-8编码

nByte: 如果nByte小于0, 则函数取出zSql中从开始到第一个0终止符的内容; 如果nByte不是负的, 那么它就是这个函数能从zSql中读取的字节数的最大值。如果nBytes非负, zSql在第一次遇见'/000'或'u000'的时候终止

pzTail: 上面提到zSql在遇见终止符或者是达到设定的nByte之后结束, 假如zSql还有剩余的内容, 那么这些剩余的内容被存放到pzTail中, 不包括终止符

ppStmt: 能够使用sqlite3_step()执行的编译好的准备语句的指针, 如果错误发生, 它被置为NULL, 假如如输入的文本不包括sql语句。调用过程必须负责在编译好的sql语句完成使用后使用sqlite3_finalize()删除它。

说明

如果执行成功，则返回SQLITE_OK，否则返回一个错误码。推荐在现在任何的程序中都使用sqlite3_prepare_v2这个函数，sqlite3_prepare只是用于前向兼容

备注

<1>准备语句（prepared statement）对象

```
typedef struct sqlite3_stmt sqlite3_stmt;
```

准备语句（prepared statement）对象一个代表一个简单SQL语句对象的实例，这个对象通常被称为“准备语句”或者“编译好的SQL语句”或者就直接称为“语句”。

语句对象的生命周期经历这样的过程：

- l 使用sqlite3_prepare_v2或相关的函数创建这个对象
- l 使用sqlite3_bind_*()给宿主参数（host parameters）绑定值
- l 通过调用sqlite3_step一次或多次来执行这个sql
- l 使用sqlite3——reset()重置这个语句，然后回到第2步，这个过程做0次或多次
- l 使用sqlite3_finalize()销毁这个对象

在sqlite中并没有定义sqlite3_stmt这个结构的具体内容，它只是一个抽象类型，在使用过程中一般以它的指针进行操作，而sqlite3_stmt类型的指针在实际上是一个指向Vdbe的结构体得指针

<2>宿主参数(host parameters)

在传给sqlite3_prepare_v2()的sql的语句文本或者它的变量中，满足如下模板的文字将被替换成一个参数：

- l ?
- l ?NNN，NNN代表数字
- l :VVV，VVV代表字符
- l @VVV
- l \$VVV

在上面这些模板中，NNN代表一个数字，VVV代表一个字母数字标记符（例如:222表示名称为222的标记符），sql语句中的参数（变量）通过上面的几个模板来指定，如

“select ? from ?”这个语句中指定了两个参数，sqlite语句中的第一个参数的索引值是1，这就知道这个语句中的两个参数的索引分别为1和2，使用“?”的话会被自动给予索引值，而使用“?NNN”则可以自己指定参数的索引值，它表示这个参数的索引值为NNN。“:VVV”表示一个名为“VVV”的参数，它也有一个索引值，被自动指定。

可以使用sqlite3_bind_*()来给这些参数绑定值

3. sqlite3_setp()

这个过程用于执行有前面sqlite3_prepare创建的准备语句。这个语句执行到结果的第一行可用的位置。继续前进到结果的第二行的话，只需再次调用sqlite3_setp()。继续调用sqlite3_setp()知道这个语句完成，那些不返回结果的语句（如：INSERT，UPDATE，或DELETE），sqlite3_step()只执行一次就返回

函数定义

```
int sqlite3_step(sqlite3_stmt*);
```

返回值

函数的返回值基于创建sqlite3_stmt参数所使用的函数，假如是使用老版本的接口sqlite3_prepare()和sqlite3_prepare16()，返回值会是 SQLITE_BUSY，SQLITE_DONE，SQLITE_ROW，SQLITE_ERROR 或 SQLITE_MISUSE，而v2版本的接口sqlite3_prepare_v2()和sqlite3_prepare16_v2()则会同时返回这些结果码和扩展结果码。

对所有V3.6.23.1以及其前面的所有版本，需要在sqlite3_step()之后调用sqlite3_reset()，在后续的sqlite3_step之前。如果调用sqlite3_reset重置准备语句失败，将会导致sqlite3_step返回SQLITE_MISUSE，但是在V3.6.23.1以后，sqlite3_step()将会自动调用sqlite3_reset。

```
int sqlite3_reset(sqlite3_stmt *pStmt);
```


sqlite3_reset用于重置一个准备语句对象到它的初始状态，然后准备被重新执行。所有sql语句变量使用sqlite3_bind*绑定值，使用sqlite3_clear_bindings重设这些绑定。Sqlite3_reset接口重置准备语句到它代码开始的时候。sqlite3_reset并不改变在准备语句上的任何绑定值，那么这里猜测，可能是语句在被执行的过程中发生了其他的改变，然后这个语句将它重置到绑定值的时候的那个状态。

4. sqlite3_column()

这个过程从执行sqlite3_step()执行一个准备语句得到的结果集的当前行中返回一个列。每次sqlite3_step得到一个结果集的列停下后，这个过程就可以被多次调用去查询这个行的各列的值。对列操作是有多个函数，均以sqlite3_column为前缀

```
1 const void *sqlite3_column_blob(sqlite3_stmt*, int iCol);  
2
```

```
3 int sqlite3_column_bytes(sqlite3_stmt*, int iCol);
4
5 int sqlite3_column_bytes16(sqlite3_stmt*, int iCol);
6
7 double sqlite3_column_double(sqlite3_stmt*, int iCol);
8
9 int sqlite3_column_int(sqlite3_stmt*, int iCol);
10
11 sqlite3_int64 sqlite3_column_int64(sqlite3_stmt*, int iCol);
12
13 const unsigned char *sqlite3_column_text(sqlite3_stmt*, int iCol);
14
15 const void *sqlite3_column_text16(sqlite3_stmt*, int iCol);
16
17 int sqlite3_column_type(sqlite3_stmt*, int iCol);
18
19 sqlite3_value *sqlite3_column_value(sqlite3_stmt*, int iCol);
```



说明

第一个参数为从sqlite3_prepare返回来的prepared statement对象的指针，第二参数指定这一行中的想要被返回的列的索引。最左边的一列的索引号是0，行的列数可以使用sqlite3_column_count()获得。

这些过程会根据情况去转换数值的类型，sqlite内部使用sqlite3_snprintf()去自动进行这个转换，下面是关于转换的细节表：

内部类型	请求的类型	转换
NULL	INTEGER	结果是0
NULL	FLOAT	结果是0.0
NULL	TEXT	结果是NULL
NULL	BLOB	结果是NULL
INTEGER	FLOAT	从整形转换到浮点型
INTEGER	TEXT	整形的ASCII码显示
INTEGER	BLOB	同上
FLOAT	INTEGER	浮点型转换到整形
FLOAT	TEXT	浮点型的ASCII显示
FLOAT	BLOB	同上
TEXT	INTEGER	使用atoi()
TEXT	FLOAT	使用atof()
TEXT	BLOB	没有转换
BLOB	INTEGER	先到TEXT，然后使用atoi
BLOB	FLOAT	先到TEXT，然后使用atof
BLOB	TEXT	如果需要的话添加0终止符

注：BLOB数据类型是指二进制的数据块，比如要在数据库中存放一张图片，这张图片就会以二进制形式存放，在sqlite中对应的数据类型就是BLOB

int sqlite3_column_bytes(sqlite3_stmt*, int iCol)int sqlite3_column_bytes16(sqlite3_stmt*, int iCol)
两个函数返回对应列的内容的字节数，这个字节数不包括后面类型转换过程中加上的0终止符。

下面是几个最安全和最简单的使用策略

- 先sqlite3_column_text(), 然后 sqlite3_column_bytes()
- 先sqlite3_column_blob(), 然后sqlite3_column_bytes()
- 先sqlite3_column_text16(), 然后sqlite3_column_bytes16()

5. sqlite3_finalize

```
int sqlite3_finalize(sqlite3_stmt *pStmt);
```

这个过程销毁前面被sqlite3_prepare创建的准备语句，每个准备语句都必须使用这个函数去销毁以防止内存泄露。

在空指针上调用这个函数没有什么影响，同时可以准备语句的生命周期的任一时刻调用这个函数：在语句被执行前，一次或多次调用sqlite_reset之后，或者在sqlite3_step任何调用之后不管语句是否完成执行

6. sqlite3_close

这个过程关闭前面使用sqlite3_open打开的数据库连接，任何与这个连接相关的准备语句必须在调用这个关闭函数之前被释放

二. 使用举例



```
1 #include "stdafx.h"
2
3 #include "sqlite3.h"
4
5 static int callback(void *NotUsed, int argc, char **argv, char **azColName)
6
7 {
8
9     int i;
10
11     for(i=0; i<argc; i++){
12
13         printf("%s = %s/n", azColName[i], argv[i] ? argv[i] : "NULL");
14
15     }
16
17     printf("/n");
18
19     return 0;
20 }
21
22
23 #define CHECK_RC(rc,szInfo,szErrMsg,db) if(rc!=SQLITE_OK) /
24
25     {printf("%s error!/n",szInfo);/
26
27     printf("%s/n",szErrMsg);    /
28
29     sqlite3_free(szErrMsg);      /
30
31     sqlite3_close(db);          /
32
33     return 0;}
34
35 int _tmain(int argc, _TCHAR* argv[])
36
37 {
38
39
40
41     sqlite3 *db;
42
43     char *dbPath="f:/test.db";
44
45     char *szErrMsg = 0;
46
47
48
49     int rc= sqlite3_open(dbPath, &db);
50
51     CHECK_RC(rc,"open database",db);
52
53     char *szSql="create table UserInfo(ID int primary key , UserName char, PassWord ch
54 ar);";
55
56     rc=sqlite3_exec(db,szSql,0,0,&szErrMsg);
```

```
56
57     CHECK_RC(rc,"create table",szErrMsg,db);
58
59     rc=sqlite3_exec(db,"insert into UserInfo(ID,UserName,PassWord) values(1,'kfqcome',
60 '123456')",0,0,&szErrMsg);
61
62     CHECK_RC(rc,"insert info",szErrMsg,db);
63
64     rc=sqlite3_exec(db,"insert into UserInfo(ID,UserName,PassWord) values(2,'miss wang
65 ','654321')",0,0,&szErrMsg);
66
67     CHECK_RC(rc,"insert info",szErrMsg,db);
68
69     szSql="select * from UserInfo";
70
71     rc = sqlite3_exec(db,szSql, callback, 0, &szErrMsg);
72
73     CHECK_RC(rc,"query values",szErrMsg,db);
74
75     sqlite3_close(db);
76
77     getchar();
78
79     return 0;
80 }
```

输出的结果:

ID = 1

UserName = kfqcome

PassWord = 123456

ID = 2

UserName = miss wang

PassWord = 654321

这里执行sql语句用的是sqlite3_exec，它是前面几个函数的封装

```
1 int sqlite3_exec(
2
3     sqlite3*,                                /* An open database */
4
5     const char *sql,                          /* SQL to be evaluated */
6
7     int (*callback)(void*,int,char**,char**), /* Callback function */
8
9     void *,                                  /* 1st argument to callback */
10
11     char **errmsg                             /* Error msg written here */
12 );
13 );
```

sqlite3_exec是sqlite3_prepare_v2, sqlite3_step()和sqlite3_finalize()的封装，能让程序多次执行sql语句而不要写许多重复的代码。

Sqlite3_exec接口执行0或多个UTF-8编码的，分号分割的sql语句，传到第二个参数中。如果sqlite3_exec的第三个参数回调函数指针不为空，那么它会为每个来自执行的SQL语句的结果行调用（也就是说回调函数会调用多次，上面例子中会返回2个结果行，因而会被执行2次），第4个参数是传给回调函数的第一个参数，如果回调函数指针为空，那么回调不会发生同时结果行被忽略。

如果在执行sql语句中有错误发生，那么当前的语句的执行被停止，后续的语句也被跳过。第五个参数不为空的时候，它被分配内存并写入了错误信息，所以在sqlite3_exec后面需要调用sqlite3_free去释放这个对象以防止内存泄露

回调函数:

```
int (*callback)(void*,int,char**,char**), /* Callback function */
```

第一个参数通过sqlite3_exec的第4个参数传入的

第二个参数是结果行的列数

第三个参数是行中列数据的指针

第四个参数是行中列名称的指针

END

绿色通道: [好文要顶](#) [关注我](#) [收藏该文](#) [与我联系](#)



[南山放牛](#)

[关注 - 1](#)

[粉丝 - 0](#)

[+加关注](#)

0

0

(请您对文章做出评价)

« 上一篇: [SQL大量数据查询分页存储过程](#)

» 下一篇: [sqlite3中的数据类型](#)

posted @ 2011-07-14 16:23 [南山放牛](#) 阅读(741) 评论(0) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论, 请 [登录](#) 或 [注册](#), [访问网站首页](#)。

[博客园首页](#) [博问](#) [新闻](#) [闪存](#) [程序员招聘](#) [知识库](#)

最新IT新闻:

- [美国十大最酷办公室](#)
- [2015智能手环进化方向: 细分领域、数据分析](#)
- [给亚马逊敲个警钟——贝索斯的7个失败决策](#)
- [这15个免费学习网站, 每一个企业家都应知道](#)
- [杀死你的APP的6个致命错误](#)
- » [更多新闻...](#)

最新知识库文章:

- [“米粉节”背后的故事——小米网抢购系统开发实践](#)
- [什么是用户体验, 什么不是?](#)
- [项目初始会议 - 如何在一次会议中达成共识](#)
- [禅意设计: 网络简洁设计的缘起和未来](#)
- [通过一组RESTful API暴露CQRS系统功能](#)
- » [更多知识库文章...](#)

Copyright ©2014 [南山放牛](#)