

第三章 数据库模块

3.1 数据库介绍

上一章我们封装了 Server 框架和消息处理，实现了一个简单的消息：“登录”，登录账号密码信息是在代码里写死的，实际项目中，大量的客户账号肯定不能写死在代码里吧，于是我们需要一种技术来存储用户的信息。我们可以用 txt 文本或 xml 来存储。没错，它可以工作的不错，但本章我要讲解的是数据库。

数据库（Database）是按照数据结构来组织、存储和管理数据的仓库。比如用户账号，游戏角色都要存储在硬盘上持久保存。你也可以用 xml 或 txt 或自定义格式来保存用户数据。但我这里要介绍另一种方法，即用数据库来保存用户数据。

目前数据库主流的就几种：

1. MSSQL 微软的，很多有历史的程序数据库都用的它，但是他无法跨平台，而且安装笨重
2. MYSQL 甲骨文的，近些年来最流行的数据库了，小巧强大轻便，小型中型甚至大型系统都用它，适用范围广，跨平台，国内著名的公司大多用它
3. SQLite 是一个开源的嵌入式关系数据库,实现自包容、零配置、支持事务的 SQL 数据库引擎。其特点是高度便携、使用方便、结构紧凑、高效、可靠。这几年才火起来的，开源小巧可嵌入是它的优点，很多小型数据库需求都用它就够了。

3.2 SQL 语言

数据库涉及到的操作就是查询、操纵、定义和控制。于是有了 SQL(Structured Query Language)语言，即结构化查询语言。SQL 是一门 ANSI 的标准计算机语言，用来访问和操作数据库系统。SQL 语句用于取回和更新数据库中的数据，例如：`select * from users` 就是一条查询语句，表示查询 users 表的所有数据。

虽然 SQL 有一个规范，但是大部分数据库在遵从规范基础上添加了一些自己的扩展。

SQL 基本数据类型

SQL 基本数据类型主要有几种，数值型数据类型，字符型数据类型，临时数据类型，混合型数据类型。

● 数值数据类型

| 数据类型 | 详细说明 |
|--------------|--|
| INTEGER | 表示能够用 4 个字节保存的整数值，范围包括-2,147,483,648(-2~31)至 2,147,483,647 (2~31-1)。INT 是 INTEGER 的缩写形式。 |
| SMALLINT | 表示能够用 2 个字节保存的整数值，范围从-32768(-2~16)至 32767 (2~16-1)。 |
| TINYINT | 表示能够用 1 个字节保存的，零以上的整数值，范围为 0 至 255 |
| BIGINT | 表示能够用 8 个字节保存的整数值。范围为-2~63 至 2~63-1 |
| DECIMAL(p,s) | 描述定点直。p(精确度)指定数字个数和假定的小数点 s(刻度)数字。DECIMAL 值可以用 5-17 个字节保存，由 p 值决定字节个数。DEC 为 DECIMAL 的缩写。 |
| NUMERIC(p,s) | 和 DECIMAL 相同。 |
| REAL | 适用于浮点值。包括正数值和负数值。正数值范围大约为 2.23E-308 至 1.79E+308,负数值为-1.18E-38 至-1.18E+38。 |
| FLOAT[p] | 表示浮点值，如 REAL。p 定义精确度，即 p<25 表示单精度(4 个字节)，p >=25 为双倍精密度（8 个字节）。 |
| MONEY | 表示币值。MONEY 值和 8 字节的 DECIMAL 值一样。都是小数点后 4 个数字。 |
| SMALLMONEY | 和 MONEY 数据类型一样，只是用 4 个字节保存。 |

INTEGER，SMALLINT，TINYINT，BIGINT，DECIMAL，NUMERIC，REAL，FLOAT，MONEY，SMALLMONEY 他们要么是整数要么是小数，相同类型的区别在于表示数值的范围不同，比如 INTEGER 表示能够用 4 个字节保存的整数值，范围包括 -2,147,483,648(-2~31)至 2,147,483,647 (2~31-1)。INT 是 INTEGER 的缩写形式。TINYINT 表示能够用 1 个字节保存的，零以上的整数值，范围为 0 至 255。

● 字符型数据类型

| 数据类型 | 说明 |
|---------------|---|
| CHAR[(n)] | 表示单字节，有固定长度的字符创，其中 n 为字符串内的字符个数。n 的最大值为 8000。如果省略了 n，那么字符串的长度就假定为 1。 |
| VARCHAR[(n)] | 表示单字节的字符可变长度的字符串(0<n<=8000)。和 CHAR 数据类型相反，VARCHAR 数据值能够在它的实际长度中保存。 |
| NCHAR[(n)] | 保存固定长度的 Unicode 字符。CHAR 和 NCHAR 数据类型之间最主要的区别为 NCHAR 数据类型的每个字符都可以用 2 个字节保存。而 CHAR 数据类型的每个字符只用 1 个字节的存储空间。因此，NCHAR 列中字符的个数最多维 4000。 |
| NVARCHAR[(n)] | 保存具有可变长度的 Unicode 字符串。VARCHAR 和 NVARCHAR 数据类型之间的最主要区别为每个 NVARCHAR 字符都能用 2 个字节保存。而 VARCHAR 字符只用 1 个字节的存储空间。NVARCHAR 列中 |

| |
|----------------|
| 的字符个数最多为 4000。 |
|----------------|

CHAR, VARCHAR 的区别在于 CHAR 是定长的, 如果一个 CHAR(n) 存储的字符串比 n 个字符要短, 那么其余的字符串就会用空格来填充。VARCHAR 是变长的, 则不会浪费空间。

● 临时数据类型

DATETIME、SMALLDATETIME、DATE、TIME、DATETIME2 和 DETETIMEOFFSET。

DATETIME 和 SMALLDATETIME 数据类型指定日期和时间, 每个值都为整数并分别用 4 个字节或 2 个字节保存。DATETIME 和 SMALLDATETIME 的值是作为两个单独的数值保存的。DATETIME 日期值的范围为 01/01/1900 至 12/31/9999。SMALLDATETIME 相应的值为 01/01/1900 至 06/06/2079。时间部分能够再用 4 个字节或 2 个字节保存。它表示午夜过后的一秒的三百分之一(DATETIME)或分钟(SMALLDATETIME)。

● 混合型数据类型

Binary, 大对象数据, CURSOR, 等等。

比如存储图片到数据库, 则用 Binary 或大对象数据。

下面开始学习 Sqlite 数据库的相关操作。

3.4 Sqlite 安装

官网 <http://www.sqlite.org/> <http://www.sqlite.org/download.html> 可以下载到源码 **Source Code** 我下的是 sqlite-amalgamation-3080500.zip

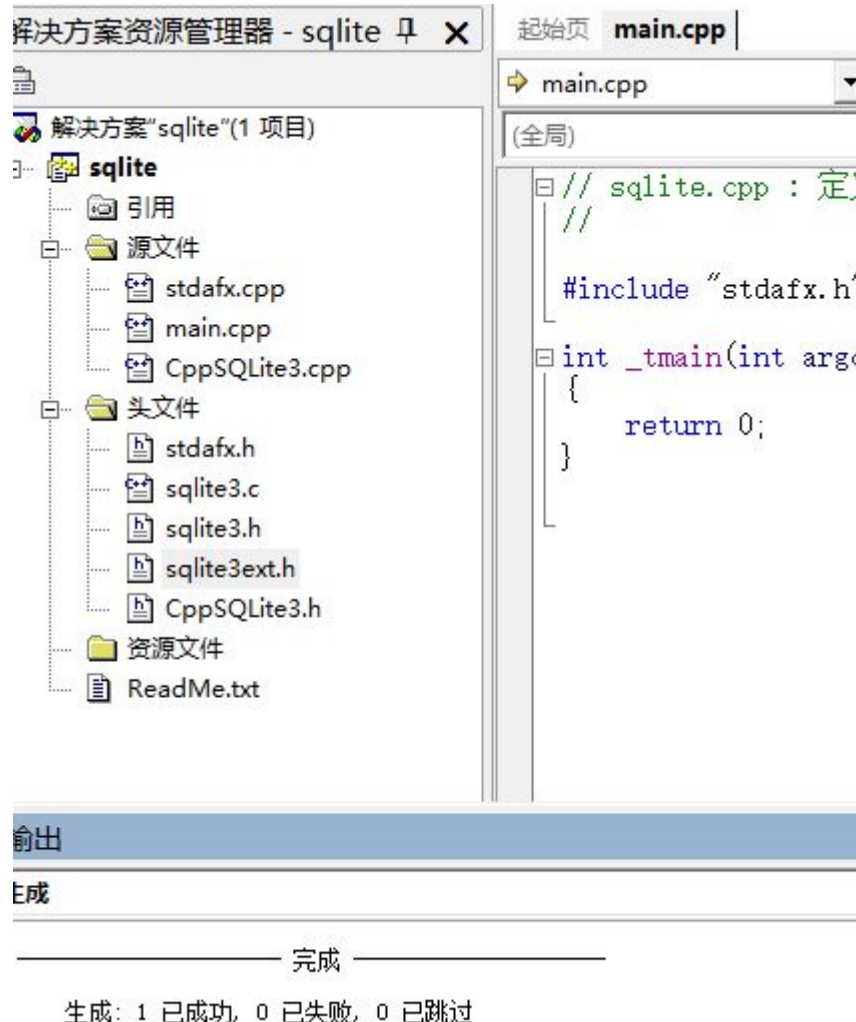
Sqlite 其实有两个版本, sqlite2, sqlite3。目前已经普遍使用 sqlite3 的比较多了。本文使用的版本也是 sqlite3。

解压后如图

| 名称 | 修改日期 | 类型 | 大小 |
|--|----------------|----------|----------|
|  shell.c | 2014/6/4 21:21 | C Source | 122 KB |
|  sqlite3.c | 2014/6/4 21:21 | C Source | 5,117 KB |
|  sqlite3.h | 2014/6/4 21:21 | H 文件 | 352 KB |
|  sqlite3ext.h | 2014/6/4 21:21 | H 文件 | 26 KB |

可以看见 sqlite 源码文件就几个, 所以很方便的嵌入到程序里。

新建一个 c++ 控制台工程，把除了 shell.c 文件以外，其余 3 个文件都复制进新工程，再把 CppSQLite3.h，CppSQLite3.cpp 添加进工程。这两个类是 sqlite 的面向对象的封装，使用会很方便，直接用 sqlite api 会很麻烦。如果顺利，编译一下即可成功。说明我们 sqlite 代码没有编译错误。下一节开始介绍数据库表的相关操作。



3.4 Sqlite 新建数据库

一个数据库首先要有一个库，库通常包含一个或多个表。数据库当然要存储在硬盘来持久化保存，当然现在还有内存数据库，即所有数据全部在内存，所以读写操作会非常的快，但是内存数据也会定期保存一下，不然一旦服务器宕机，数据就完蛋了。Sqlite 默认是文件数据库，但也支持内存数据库。

新建数据库很简单，我们用代码执行即可。
在上一节的 sqlite 工程里写入下面代码

```

#include "sqlite3.h"

#include "CppSQLite3.h"

CppSQLite3DB db;

std::string ExecuteSQL(const char* sql )
{
    std::string ret;
    try
    {
        db.execDML( sql);

        printf("OnExecuteSQL sql[%s] 成功 \n", sql );
        ret="exec ";
        ret+=sql;
        ret+= " 成功";
    }
    catch (CppSQLite3Exception & e)
    {
        printf("OnExecuteSQL sql[%s] 异常[%s] \n", sql, e.errorMessage() );
        ret=e.errorMessage();
    }

    return ret;
}

int _tmain(int argc, _TCHAR* argv[])
{
    //打开d.db
    db.open( "d.db");

    return 0;
}

```

运行后，查看工程目录，发现 d.db 的数据库文件被建立了。由于数据库写入没有任何东西，所以数据为 0 字节

| | | | |
|--------------|-----------------|----------------|------|
| CppSQLite3.h | 2014/9/21 0:02 | 头文件 | 0 KB |
| d.db | 2015/9/11 16:48 | Data Base File | 0 KB |
| main.cpp | 2015/9/11 16:49 | CPP 文件 | 1 KB |

3.5 SQL 数据库表的操作

数据库表由一个名字标识，即表名，比如 `users`(角色)表，`accounts`(账号)表，表里包含一个或多个列，比如账号表一般要有的列有：`id`，账号，密码，注册时间。

● 建表

建表 `Sql`格式如下

```
CREATE TABLE 表名 (  
    列名 1 类型 约束 ,  
    列名 2 类型 约束 ,  
    ...  
)
```

对应建表的 `SQL` 语句为

```
CREATE TABLE users ( id int NOT NULL,  
    username varchar(30) NOT NULL,  
    pwd varchar(30) NOT NULL,  
    regtime datetime,  
    PRIMARY KEY ('id') );
```

`id int` 表示定义了一个叫 `id` 的列，类型为 `int`(`INTEGER` 的简写)，`NOT NULL` 表示不能为空，`username varchar(30)` 定义了一个可变长度的字符串，最大长度为 30，用于保存用户名，`pwd` 和 `username` 一样类型。`regtime datetime` 表示定义了一个 `regtime` 的日期时间，用于记录注册账号时的日期时间。

`PRIMARY KEY ('id')` 表示申明 `id` 为主键。每个表最好定义一个主键，主键会为表建立索引，增快查询速度。当然也可以写成 `id int PRIMARY KEY`，效果是一样的。

如果创建的表名已存在，那么运行会失败，`sql` 会报错。

● 删表

有建表 就有删表。删除表语句格式如下

`drop table` 表名称

例如删除用户表 则是 `drop table users`

如果删除不存在的表，那么运行会失败，`sql` 会报错。可以使用 `drop table if exists users` 来 `if` 判断一下，存在则删除，不存在则什么也不做。

● 插入记录

有了表，我们就可以插入数据保存在表里。
插入语句格式如下

INSERT INTO 表名称 VALUES (值 1, 值 2,...)

我们也可以指定所要插入数据的列：

INSERT INTO 表名称(列 1, 列 2,...) VALUES (值 1, 值 2,...)

向 users 表插入一行记录，id 为 1, username 为 zhangsan,pwd 为 123456, regtime 为当前时间，语句如下

INSERT INTO users VALUES (1, 'zhangsan', '123456', datetime('now'))

也可以指定列名的方式

INSERT INTO users (id,username, pwd , regtime)VALUES (1, 'zhangsan', '123456', datetime('now'))

如果是数值类型的，一般都不用引号括起来，字符类型的一般都要引号括起来。

下面给出代码实际运行例子。

接着上一个项目中写入如下代码

```
//删表
//ExecuteSQL("drop table users");
ExecuteSQL("drop table if exists users");
//建表
char* sql1="CREATE TABLE users (id int NOT NULL,username varchar(30)
NOT NULL,pwd varchar(30) NOT NULL,regtime datetime,PRIMARY KEY
('id'))";

ExecuteSQL(sql1);

//插入一条数据
ExecuteSQL(" INSERT INTO users VALUES ( 1, 'zhangsan', '123456',
datetime('now'))");
```

运行结果如图，

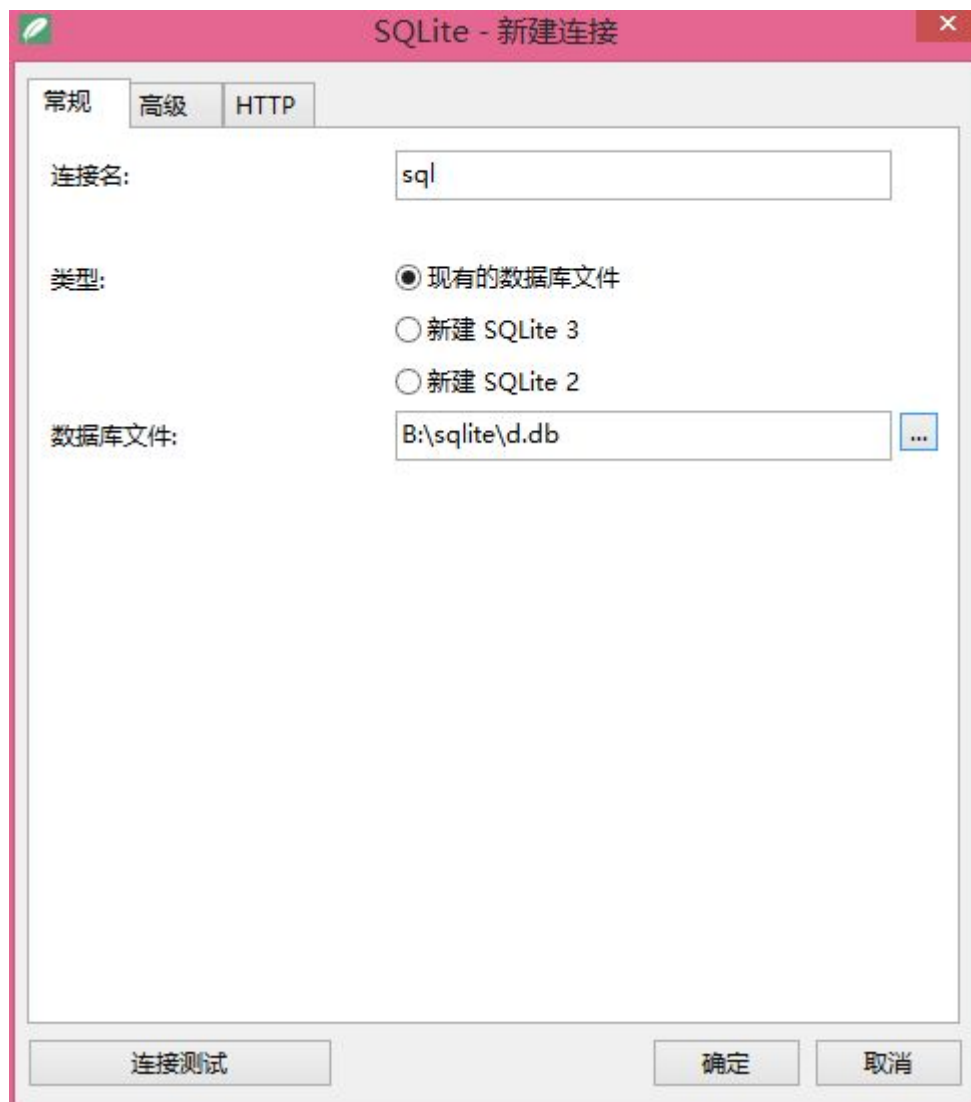


```
"b:\sqlite\Debug\sqlite.exe"
OnExecuteSQL sql1[drop table users] 成功
OnExecuteSQL sql1[CREATE TABLE users (id int NOT NULL,username varchar(30) NOT
NULL,pwd varchar(30) NOT NULL,regtime datetime,PRIMARY KEY ('id'))];1 成功
OnExecuteSQL sql1 INSERT INTO users VALUES ( 1, 'zhangsan', '123456', date
time('now'))>;1 成功
Press any key to continue_
```

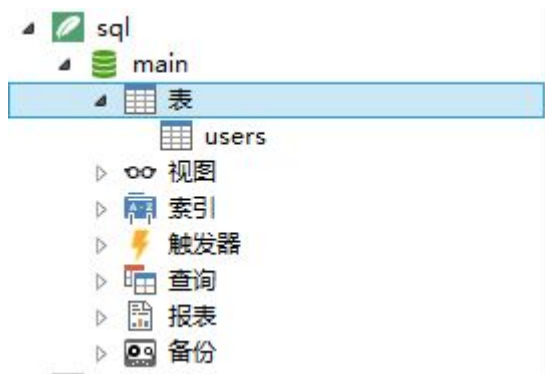
3.6 SQL 数据库管理

SQL 数据库通常都有提供管理器方便管理，mssql 有提供 sql 查询分析器，mysql 有很多非官方提供的查询工具，比较著名的有 navicat，navicat 不仅支持 mysql，还支持 sqlite，我们开发中可以使用它来帮助自己。

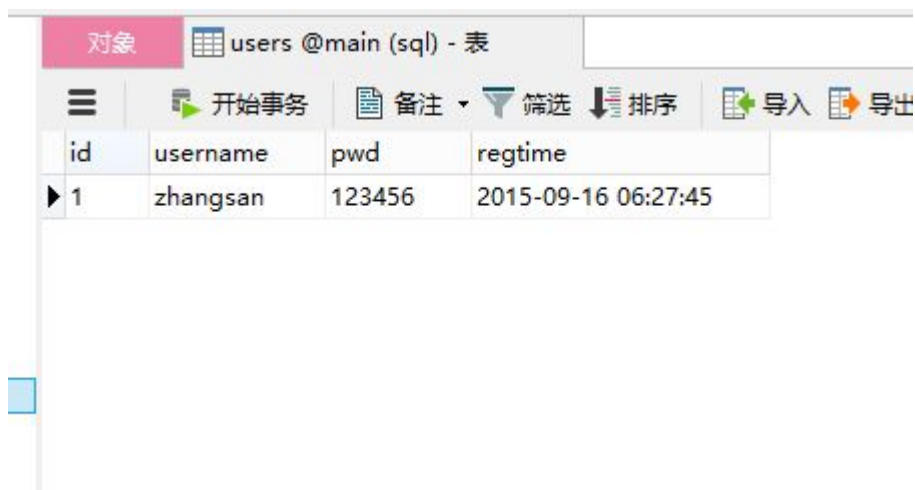
假设读者已经安装了 navicat。打开它，点击 文件—新建连接—sqlite 确定。在弹出的窗口中，连接名 填入 sql，选 ‘现有的数据库文件’，下面选择好 d.db 数据库文件，点确定，若没问题，便添加好了一个数据库连接。



双击左边的 sql 数据库项目，双击 main，双击 表，如图所示



d.db 数据库有一个名叫 users 的表。双击 users 表，如下图



有我们上面插入的一条记录，这个界面可以直接编辑数据，

Navicat 可以对数据库进行完全的操作。增删改查都可以。就不一一细说了。

3.7 SQL 查询

现在开始学习查询，SQL 查询用到的关键字是 select，

SELECT 语句用于从表中选取数据。

结果被存储在一个结果表中（称为结果集）。

SELECT 列名称 FROM 表名称

以及：

SELECT * FROM 表名称

注释：SQL 语句对大小写不敏感。SELECT 等效于 select。

如需获取名为 "username" 和 "pwd" 的列的内容（从名为 "users" 的数据库表），请使用类似这样的 SELECT 语句：

```
SELECT username, pwd FROM users
```

现在我们希望从 "users" 表中选取所有的列。

使用符号 * 取代列的名称，就像这样：

```
SELECT * FROM users
```

星号（*）是选取所有列的快捷方式。

下面通过代码方式查询，

```
std::string QuerySQL( const char* sql )
{
    std::string ret;

    try
    {

        //查询
        CppSQLite3Query query = db.execQuery( sql );

        while(!query.eof())
        {

            //输出所有列
            for(int i=0;i<query.numFields();i++)
            {
                //query.fieldName(i), query.getStringField(i );
                printf("%s ",query.getStringField(i ));
            }
        }
    }
}
```

```

        printf("\n");
        query.nextRow();
    }
    query.finalize();

    printf("OnQuerySQL sql[%s] 成功 \n", sql );
}
catch (CppSQLite3Exception & e)
{
    printf("OnExecuteSQL sql[%s] 异常[%s] \n", sql, e.errorMessage() );

    ret=e.errorMessage();
}

return ret;
}

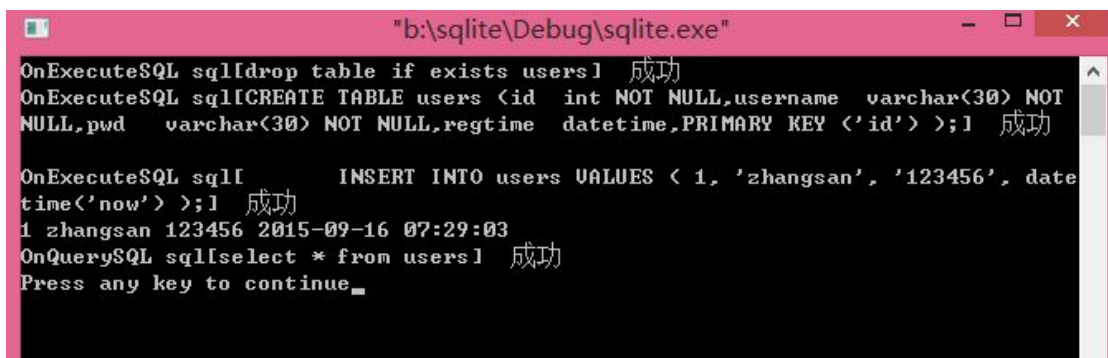
```

Main 函数添加

//查询

QuerySQL("select * from users");

运行结果如下



```

b:\sqlite\Debug\sqlite.exe
OnExecuteSQL sql[drop table if exists users] 成功
OnExecuteSQL sql[CREATE TABLE users (id int NOT NULL,username varchar(30) NOT
NULL,pwd varchar(30) NOT NULL,regtime datetime,PRIMARY KEY (id) );] 成功

OnExecuteSQL sql[INSERT INTO users VALUES ( 1, 'zhangsan', '123456', date
time('now') );] 成功
1 zhangsan 123456 2015-09-16 07:29:03
OnQuerySQL sql[select * from users] 成功
Press any key to continue_

```

可以看见打印了一行记录。查询成功。

3.8 SQL 删除

删除表我们已经会了，现在学习删除记录，有时我们添加的记录想删除，那么要用到 `delete` 关键字。

删除语法如下：

`DELETE FROM 表名称 WHERE 列名称 = 值`

例如：

要删除 id 为 1 的用户记录。`DELETE FROM users WHERE id = 1`

删除用户名为 `zhangsan` 的用户记录。`DELETE FROM users WHERE username = 'zhangsan'`

删除 `users` 表所有记录。`DELETE FROM users` 或 `DELETE * FROM users`

下面用代码运行演示

```
//删除zhangsan
```

```
ExecuteSQL(" DELETE FROM users WHERE username = 'zhangsan' ");
```

3.9 SQL 更新

有时我们需要更新某些记录，比如某行某列的值，那么需要用到更新操作，更新操作的关键字是 `update`

`Update` 语句用于修改表中的数据。

语法：

`UPDATE 表名称 SET 列名称 = 新值 WHERE 列名称 = 某值`

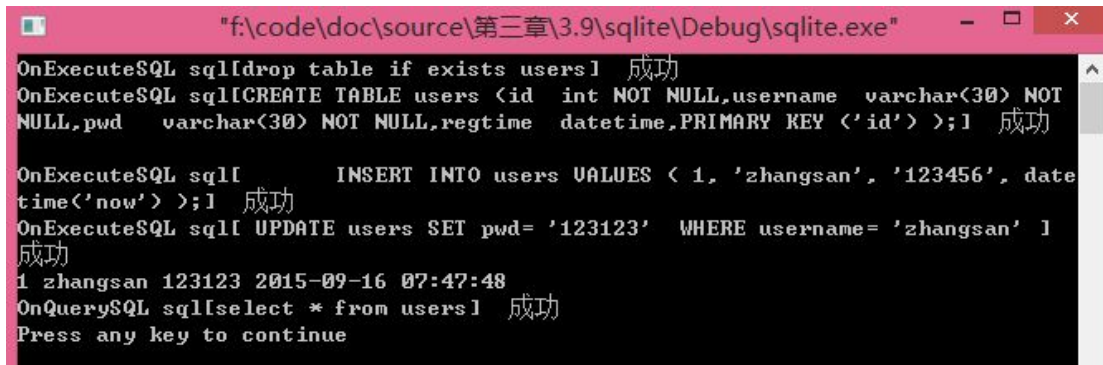
例如：我们要更新 `zhangsan` 的密码为 `123123`，那么语句为

`UPDATE users SET pwd= '123123' WHERE username= 'zhangsan'`

下面用代码运行演示

```
//更新
```

```
ExecuteSQL(" UPDATE users SET pwd= '123123' WHERE username= 'zhangsan' ");
```



```
"f:\code\doc\source\第三章\3.9\sqlite\Debug\sqlite.exe"
OnExecuteSQL sqlldrop table if exists users 成功
OnExecuteSQL sqlCREATE TABLE users (id int NOT NULL,username varchar(30) NOT
NULL,pwd varchar(30) NOT NULL,regtime datetime,PRIMARY KEY (<id> )) 成功
OnExecuteSQL sqlINSERT INTO users VALUES ( 1, 'zhangsan', '123456', date
time('now')) 成功
OnExecuteSQL sqlUPDATE users SET pwd= '123123' WHERE username= 'zhangsan' 1
成功
1 zhangsan 123123 2015-09-16 07:47:48
OnQuerySQL sqlselect * from users 成功
Press any key to continue
```

可以看见密码已经改为 123123 了

3.10 数据库模块

现在我们把数据库模块合并到服务端，改造服务端，实现客户端登陆查询数据库的账号密码。

将数据库相关代码复制到netserver服务端目录的db目录下，项目添加模块文件引用。Server 添加

```
#include "db/sqlite3.h"
#include "db/CppSQLite3.h"
```

Server 添加一个 db 成员 CppSQLite3DB db; //sqlite 数据库

Server 封装 BOOL ExecuteSQL(const char* sql); 执行 sql 的方法

Server 添加 BOOL Login(const char* username,const char* pwd);封装登陆验证的方法

Server::Init 添加 db.open("db/d.db");

PacketHandler::On_C2S_LOGIN 登陆相关代码改为

```
if (Server::GetInstance().Login(s.name,s.pwd) )
```

整合完成后。

当数据库没有 abc 账号或密码不是 123456 时，客户端登陆失败。Netserver 提示密码 Err

当数据库 abc 账号密码 123456 时，客户端登陆成功。Netserver 提示密码 OK

3.11 数据库模块 总结

这样，我们完成了数据库模块的封装整合，使服务端拥有了数据库的功能，实现了最基本的登陆账号数据库查询。但是数据库还有很多知识需要读者自行学习。

2016.1.3 作者：司马威
作者博客 <http://blog.csdn.net/smwhatjay>

编程交流 q 群: 316641007

