浙江大学
ZHEJIANG UNIVERSITY

嵌入式软件开发

第13课 嵌入式数据库设计

王总辉
zhwang@zju.edu.cn

http://course.zju.edu.cn

提纲

- 嵌入式数据库概述
- 嵌入式数据库体系结构
- 嵌入式数据库基本设计
- 嵌入式数据库应用设计

嵌入式数据库概述

- 嵌入式数据库简介
- 嵌入式数据库特点
- 嵌入式数据库分类
- 典型嵌入式数据库系

嵌入式数据库简介

- 嵌入式数据库系统是指支持移动计算或某种特定计算模式的数据库管理系统；通常与操作系统和具体应用集成在一起，运行在智能型嵌入式设备或移动设备上
- 嵌入式数据库系统是一种安装在嵌入式设备中，可单独运行的微型数据库系统；针对嵌入式应用的具体需求，利用成熟的数据库技术，实现对嵌入式设备上数据的组织、存取和管理

嵌入式数据库特点

- 占用存储空间小
- 可靠性、可管理性和安全性
- 互操作性和可移植性
- 可剪裁性

嵌入式数据库分类

- 基于内存方式

  考虑内存直接快速存取的特点，以CPU和内存空间的高效利用为目标

- 基于文件方式

  数据按照一定格式储存在磁盘里，由应用程序通过相应的驱动程序甚至直接对数据文件进行读取

- 基于网络的嵌入式数据库
  - 客户端为嵌入式设备，通过网络协议，可以使用SQL接口或者其他接口访问远程数据信息

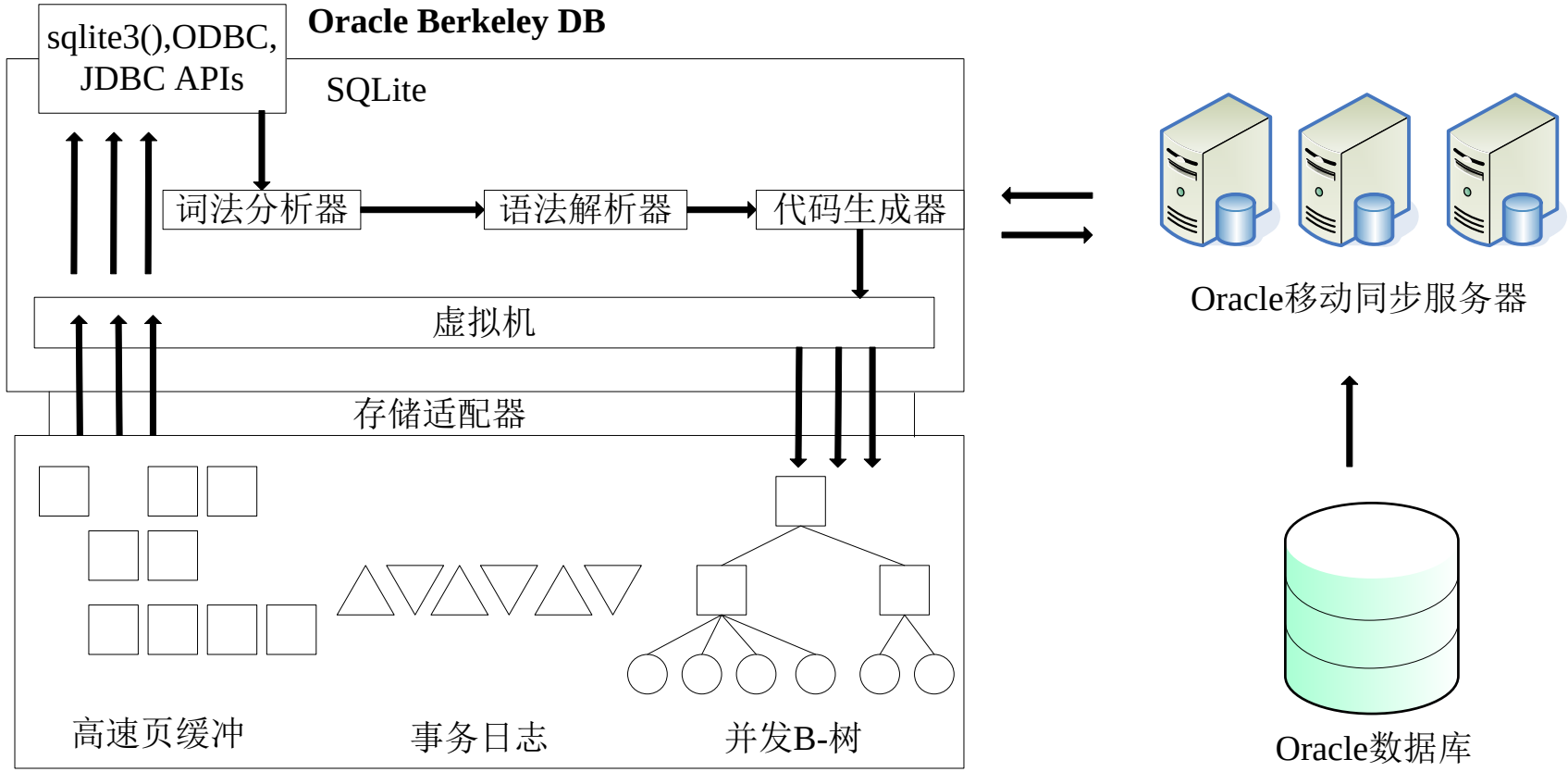
典型嵌入式数据库系

- 国外嵌入式数据库
  - Sybase iAnywhere
  - Oracle Lite Edition
  - DB2 Everyhere
  - MS SQL Server for Windows CE
  - Berkeley DB
  - SQLite
- 国内嵌入式数据库
  - OpenBASE Mini
  - Kingbase Lite

嵌入式数据库体系结构

- 嵌入式数据库体系结构
- 嵌入式数据库基本组件
- 嵌入式数据库关键技术

嵌入式数据库体系结构


sqlite3(),ODBC,
JDBC APIs
Oracle Berkeley DB
SQLite
词法分析器
语法解析器
代码生成器
虚拟机
存储适配器
高速页缓冲
事务日志
并发B-树
Oracle移动同步服务器
Oracle数据库

嵌入式数据库基本组件

- 存储管理模块
- 核心模块
- 词法分析模块
- 语法分析模块
- 预处理模块
- 查询执行模块
- 索引系统模块
- 数据同步模块

嵌入式数据库关键技术

- 系统微型化
- 系统可移植性与多平台支持
- 数据同步、复制技术
- 系统定制能力
- 系统实时处理能力
- 网络支持与数据安全

系统微型化

- 数据库微型化
  - 提高数据存储空间的利用率，增加嵌入式设备的数据存储能力
  - 对关系模式的优化和数据的压缩存储
- 数据库管理系统微型化
  - 根据应用需要，选择系统必须的功能
  - 微型化是以放弃系统功能的完备性为代价的

系统可移植性与多平台支持

- 嵌入式操作系统种类很多、特性不一、更新速度快
- 嵌入式硬件发展速度快，设备更新迅速

数据同步技术

- 嵌入式数据库一般采用某种数据复制模式（上载、下载或混合模式）与服务器数据库进行映射，满足人们在任意地点、任意时刻访问任意数据的需求
- 数据同步作为嵌入式数据库系统最重要的一个功能特点，包括数据传输、同步冲突检测和解决、同步过程中事务完整性保持、主动同步、异构数据源同步以及异构数据源之间数据类型变化等内容

系统定制能力

- 根据实际应用需求定制数据库系统的功能，真正做到量体裁衣
- 将数据库系统和应用程序结合在一起的方法就是数据库系统的定制技术

系统实时处理能力

- 如果系统所嵌入的某种移动设备支持实时应用，则嵌入式数据库系统还要考虑实时处理的要求

- 设备的移动性，如果应用请求的处理时间过长，任务就可能在执行完成后得到无效的逻辑结果，或有效性大大降低

网络支持与数据安全

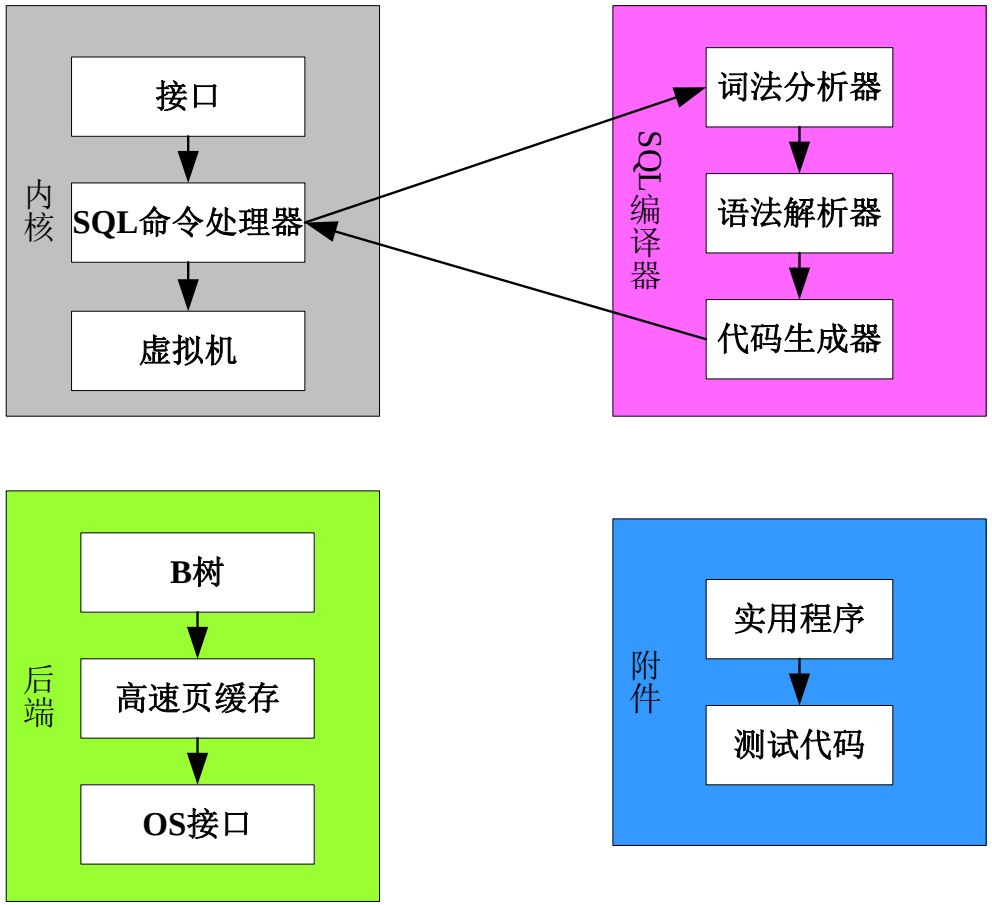
- 网络支持使数据库系统可通过网络与其他终端，或与后台主数据源进行交互

- 随着网络的到来，嵌入式数据库的数据存储安全和访问安全也成为一个重要问题

嵌入式数据库基本设计

- SQLite体系结构
- SQLite基本设计
- SQLite API接口

SQLite体系结构


内核
接口
SQL命令处理器
虚拟机
SQL编译器
词法分析器
语法解析器
代码生成器
后端
B树
高速页缓存
OS接口
附件
实用程序
测试代码

SQLite基本设计（1）

- 数据类型
  - 空、整型、实型、文本型和大数据（BLOB）类型
- 数据库大小
  - 支持高达2T的数据库
- 安全机制
  - 没有用户管理、访问控制和授权机制；
  - 利用操作系统对文件的访问控制能力实施文件级别的访问控制

SQLite基本设计（2）

- SQL标准支持

  实现了大部分SQL92标准，包括索引、限制、触发器和视图等，未实现外部关键字和行限制，但支持ACID（原子性、一致性、隔离性、持久性）

- 备份支持

  SQLite提供了两种方式备份数据库

- 应用场景

  有中小规模流量的网站、嵌入式设备和应用软件、应用程序专有文件、内部或临时数据库等

SQLite API接口（1）

- SQLite共支持83个API函数
- 最简单的程序使用三个函数就可以完成：sqlite3_open()，sqlite3_exec()，和sqlite3_close()
- 许多接口函数都是成对出现的，同时有UTF-8和UTF-16两个版本
- 提供了一组函数用来执行用户自定义的SQL函数和文本排序函数

SQLite API接口（2）

- 打开/关闭数据库

```
int sqlite3_open(const char*, sqlite3**);
int sqlite3_open16(const void*, sqlite3**);
int sqlite3_close(sqlite3*);
const char *sqlite3_errmsg(sqlite3*);
const void *sqlite3_errmsg16(sqlite3*);
int sqlite3_errcode(sqlite3*);
```

SQLite API接口（3）

- 执行 SQL 语句
  - int sqlite3_exec(sqlite3*, const char *sql, sqlite_callback, void*, char**);
  - int sqlite3_prepare(sqlite3*, const char*, int, sqlite3_stmt**, const char**);
  - int sqlite3_prepare16(sqlite3*, const void*, int, sqlite3_stmt**, const void**);
  - int sqlite3_finalize(sqlite3_stmt*);
  - int sqlite3_reset(sqlite3_stmt*);

SQLite API接口（4）

- 用户自定义函数
  - int sqlite3_create_function(sqlite3 *, const char *zFunctionName, int nArg,int eTextRep, void*,void (*xFunc)(sqlite3_context*,int,sqlite3_value**),void (*xStep)(sqlite3_context*,int,sqlite3_value**),void (*xFinal)(sqlite3_context*));
  - int sqlite3_create_function16(sqlite3*, const void *zFunctionName, int nArg,int eTextRep, void*, void (*xFunc)(sqlite3_context*, int,sqlite3_value**),void (*xStep)(sqlite3_context*,int,sqlite3_value**),void (*xFinal)(sqlite3_context*));

SQLite API接口（5）

- 用户自定义排序规则
  - sqlite3_create_collation(sqlite3*, const char *zName, int eTextRep, void*,int(*xCompare)(void*, int,const void*, int, const void*));
  - sqlite3_create_collation16(sqlite3*, const void *zName, int eTextRep, void*,int(*xCompare)(void*, int, const void*, int, const void*));
  - sqlite3_collation_needed(sqlite3*, void*,void(*)(void*, sqlite3*, int eTextRep, const char*));
  - sqlite3_collation_needed16(sqlite3*, void*,void(*)(void*,sqlite3*, int eTextRep, const void*));

嵌入式数据库应用设计

- SQLite安装与使用
- SQLite Linux应用

SQLite安装与使用（1）

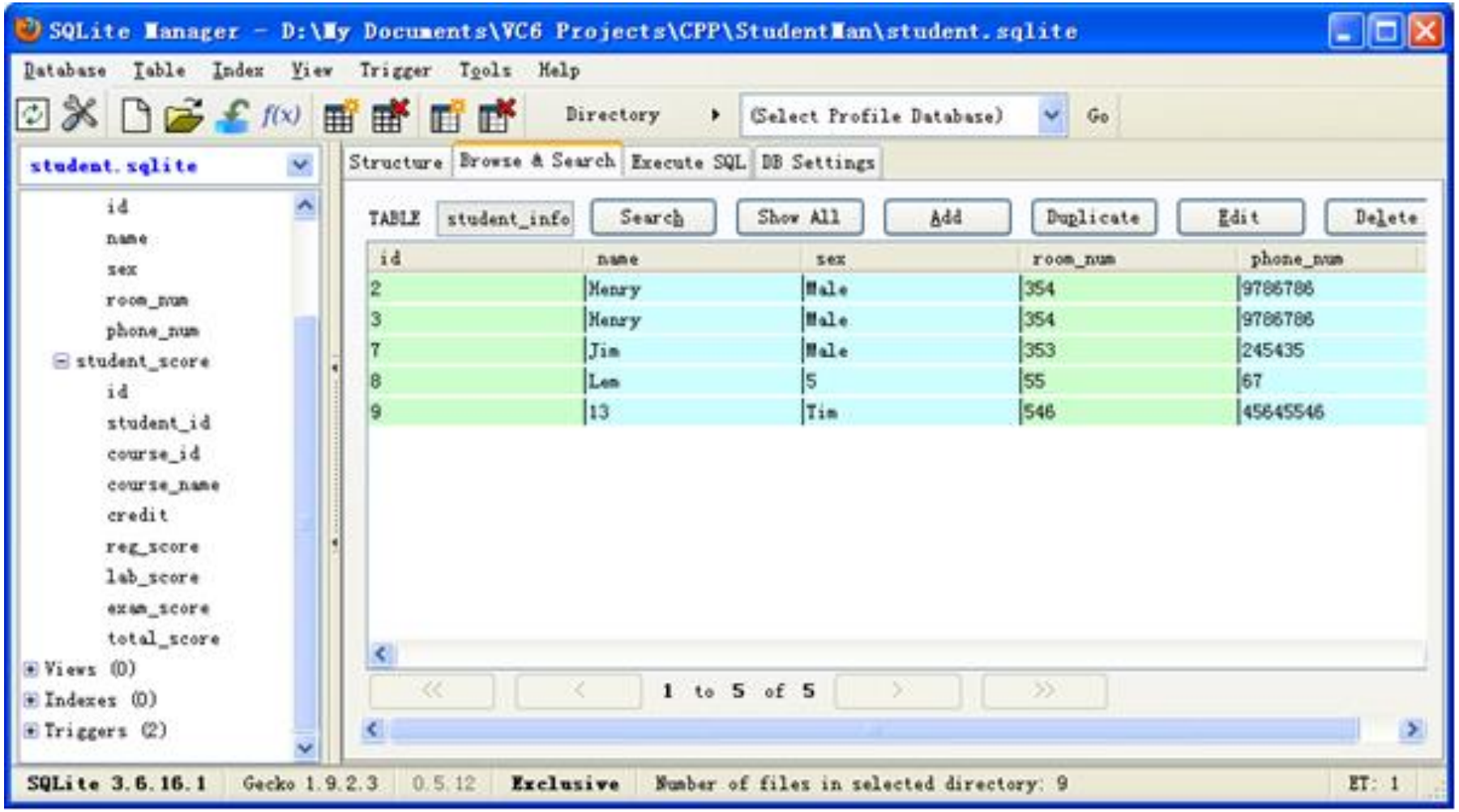
- SQLite是一个完全免费的开源数据库，主要应用的版本是SQLite3，源代码无任何版权限制，http://www.sqlite.org

- sqlite3ext.h, sqlite3.h, sqlite3.c三个文件复制到系统lib库或用户库目录中，在源代码中包含所需头文件即可

SQLite安装与使用（2）

- SQLite Manager是开放源代码的SQLite管理工具，用来管理SQLite数据库，可以独立运行，也可以作为Firefox、Thunderbird等软件的插件运行
- SQLite Manager以直观的树形结构显示数据库中的对象，可以用来管理表、索引、视图和触发器，包括增加、删除、更改、查询表中的数据，编辑表及其他对象，以及执行SQL语句等

SQLite安装与使用（3）



浙江大学 ZheJiang University

SQLite Linux应用（1）

```
#include <sqlite3.h>
#include <string>
#include <stdio.h>
int main()
{
    sqlite3* conn = NULL;
    //1. 打开数据库
    int result = sqlite3_open("mytest.db",&conn);
    if (result != SQLITE_OK) {
        sqlite3_close(conn);
        return;
    }

    const char* createTableSQL =
    "CREATE TABLE TESTTABLE (int_col INT, float_col REAL, string_col TEXT)";
    sqlite3_stmt* stmt = NULL;
    int len = strlen(createTableSQL);
```

SQLite Linux应用（2）

```
    //2. 准备创建数据表，如果创建失败，需要用sqlite3_finalize释放
sqlite3_stmt对象，以防止内存泄露。
    if (sqlite3_prepare_v2(conn,createTableSQL,len,&stmt,NULL) != SQLITE_OK) {
        if (stmt)
        sqlite3_finalize(stmt);
        sqlite3_close(conn);
        return;
    }

    //3. 通过sqlite3_step命令执行创建表的语句。对于DDL和DML语句而言，
sqlite3_step执行正确的返回值只有SQLITE_DONE，对于SELECT查询而言，如
果有数据返回SQLITE_ROW，当到达结果集末尾时则返回SQLITE_DONE。
    if (sqlite3_step(stmt) != SQLITE_DONE) {
        sqlite3_finalize(stmt);
        sqlite3_close(conn);
        return;
}
```

SQLite Linux应用（3）

```
    //4. 释放创建表语句对象的资源。
    sqlite3_finalize(stmt);
    printf("Succeed to create test table now.\n");
    //5. 显式的开启一个事物。
    sqlite3_stmt* stmt2 = NULL;
    const char* beginSQL = "BEGIN TRANSACTION";
    if (sqlite3_prepare_v2(conn,beginSQL,strlen(beginSQL),&stmt2,NULL) !=
SQLITE_OK) {
        if (stmt2)
            sqlite3_finalize(stmt2);
        sqlite3_close(conn);
        return;
    }
    if (sqlite3_step(stmt2) != SQLITE_DONE) {
        sqlite3_finalize(stmt2);
        sqlite3_close(conn);
        return;
```

SQLite Linux应用（4）

```
    }
    sqlite3_finalize(stmt2);

    //6. 构建基于绑定变量的插入数据。
    const char* insertSQL = "INSERT INTO TESTTABLE VALUES(?,?,?)";
    sqlite3_stmt* stmt3 = NULL;
    if (sqlite3_prepare_v2(conn,insertSQL,strlen(insertSQL),&stmt3,NULL) !=
SQLITE_OK) {
        if (stmt3)
            sqlite3_finalize(stmt3);
        sqlite3_close(conn);
        return;
    }
    int insertCount = 10;
    const char* strData = "This is a test.";
```

SQLite Linux应用（5）

```
//7. 基于已有的SQL语句，迭代的绑定不同的变量数据
for (int i = 0; i < insertCount; ++i) {
//在绑定时，最左面的变量索引值是1。
   sqlite3_bind_int(stmt3,1,i);
   sqlite3_bind_double(stmt3,2,i * 1.0);
   sqlite3_bind_text(stmt3,3,strData,strlen(strData),SQLITE_TRANSIENT);
   if (sqlite3_step(stmt3) != SQLITE_DONE) {
      sqlite3_finalize(stmt3);
      sqlite3_close(conn);
      return;
   }
   //重新初始化该sqlite3_stmt对象绑定的变量。
   sqlite3_reset(stmt3);
   printf("Insert Succeed.\n");
}
sqlite3_finalize(stmt3);
```

SQLite Linux应用（6）

```
    //8. 提交之前的事物。
    const char* commitSQL = "COMMIT";
    sqlite3_stmt* stmt4 = NULL;
    if (sqlite3_prepare_v2(conn,commitSQL,strlen(commitSQL),&stmt4,NULL) !=
SQLITE_OK) {
        if (stmt4)
            sqlite3_finalize(stmt4);
        sqlite3_close(conn);
        return;
    }
    if (sqlite3_step(stmt4) != SQLITE_DONE) {
        sqlite3_finalize(stmt4);
        sqlite3_close(conn);
        return;
    }
    sqlite3_finalize(stmt4);
```

SQLite Linux应用（7）

```
    //9. 为了方便下一次测试运行，我们这里需要删除该函数创建的数据表，
否则在下次运行时将无法创建该表，因为它已经存在。
    const char* dropSQL = "DROP TABLE TESTTABLE";
    sqlite3_stmt* stmt5 = NULL;
    if (sqlite3_prepare_v2(conn,dropSQL,strlen(dropSQL),&stmt5,NULL) !=
SQLITE_OK) {
        if (stmt5)
            sqlite3_finalize(stmt5);
        sqlite3_close(conn);
        return;
    }
    if (sqlite3_step(stmt5) == SQLITE_DONE) {
        printf("The test table has been dropped.\n");
    }
    sqlite3_finalize(stmt5);
    sqlite3_close(conn);
}
```

□谢　谢！