SQLite 详细信息请参考: http://www.yiibai.com/sqlite/

SQLite简单介绍

SQLite是目前最流行的开源嵌入式数据库，和很多其他嵌入式存储引擎相比(NoSQL)，如BerkeleyDB、MemBASE等，SQLite可以很好的支持关系型数据库所具备的一些基本特征，如标准SQL语法、事务、数据表和索引等。事实上，尽管SQLite拥有诸多关系型数据库的基本特征，然而由于应用场景的不同，它们之间并没有更多的可比性。下面我们将列举一下SQLite的主要特征：

- 管理简单，甚至可以认为无需管理。
- 操作方便，SQLite生成的数据库文件可以在各个平台无缝移植。
- 可以非常方便的以多种形式嵌入到其他应用程序中，如静态库、动态库等。
- 易于维护

综上所述，SQLite的主要优势在于灵巧、快速和可靠性高。SQLite的设计者们为了达到这一目标，在功能上作出了很多关键性的取舍，与此同时，也失去了一些对RDBMS关键性功能的支持，如高并发、细粒度访问控制(如行级锁)、丰富的内置函数、存储过程和复杂的SQL语句等。正是因为这些功能的牺牲才换来了简单，而简单又换来了高效性和高可靠性

SQLite的主要优点

一致性的文件格式：在SQLite的官方文档中是这样解释的，我们不要将SQLite与Oracle或PostgreSQL去比较，而是应该将它看做fopen和fwrite。与我们自定义格式的数据文件相比，SQLite不仅提供了很好的移植性，如大端小端、32/64位等平台相关问题，而且还提供了数据访问的高效性，如基于某些信息建立索引，从而提高访问或排序该类数据的性能，SQLite提供的事务功能，也是在操作普通文件时无法有效保证的

在嵌入式或移动设备上的应用：由于SQLite在运行时占用的资源较少，而且无需任何管理开销，因此对于PDA、智能手机等移动设备来说，SQLite的优势毋庸置疑

数据分析：可以充分利用SQLite提供SQL特征，完成简单的数据统计分析的功能。这一点是CSV文件无法比拟的。

产品Demo和测试：在需要给客户进行Demo时，可以使用SQLite作为我们的后台数据库，和其他关系型数据库相比，使用SQLite减少了大量的系统部署时间。对于产品的功能性测试而言，SQLite也可以起到相同的作用

和RDBMS相比SQLite的一些劣势

只适合C/S应用：如果你有多个客户端需要同时访问数据库中的数据，特别是他们之间的数据操作是需要通过网络传输来完成的。在这种情况下，不应该选择SQLite。由于SQLite的数据管理机制更多的依赖于OS的文件系统，因此在这种操作下其效率较低

只适合小规模数据：受限于操作系统的文件系统，在处理大数据量时，其效率较低。对于超大数据量的存储，甚至不能提供支持

不适合高并发数据：由于SQLite仅仅提供了粒度很粗的数据锁，如读写锁，因此在每次加锁操作中都会有大量的数据被锁住，即使仅有极小部分的数据会被访问。换句话说，我们可以认为SQLite只是提供了表级锁，没有提供行级锁。在这种同步机制下，并发性能很难高效

个性化特征：

没有独立的服务器：

和其他关系型数据库不同的是，SQLite没有单独的服务器进程，以供客户端程序访问并提供相关的服务。SQLite作为一种嵌入式数据库，其运行环境与主程序位于同一进程空间，因此它们之间的通信完全是进程内通信，而相比于进程间通信，其效率更高。然而需要特别指出的是，该种结构在实际运行时确实存在保护性较差的问题，比如此时，应用程序出现问题导致进程崩溃，由于SQLite与其所依赖的进程位于同一进程空间，那么此时SQLite也将随之退出。但是对于独立的服务器进程，则不会有此问题，它们将在密闭性更好的环境下完成它们的工作。

单一磁盘文件：

SQLite的数据库被存放在文件系统的单一磁盘文件内，只要有权限便可随意访问和拷贝，这样带来的主要好处是便于携带和共享。其他的数据库引擎，基本都会将数据库存放在一个磁盘目录下，然后由该目录下的一组文件构成该数据库的数据文件。尽管我们可以直接访问这些文件，但是我们的程序却无法操作它们，只有数据库实例进程才可以做到。这样的好处是带来了更高的安全性和更好的性能，但是也付出了安装和维护复杂的代价。

平台无关性：

这一点在前面已经解释过了。和SQLite相比，很多数据库引擎在备份数据时不能通过该方式直接备份，只能通过数据库系统提供的各种dump和restore工具，将数据库中的数据先导出到本地文件中，之后在load到目标数据库中。这种方式存在显而易见的效率问题，首先需要导出到另外一个文件，如果数据量较大，导出的过程将会比较耗时。然而这只是该操作的一小部分，因为数据导入往往需要更多的时间。数据在导入时需要很多的验证过程，在存储时，也并非简简单单的顺序存储，而是需要按照一定的数据结构、算法和策略存放在不同的文件位置。因此和直接拷贝数据库文件相比，其性能是非常拙劣的。

弱类型：

和大多数支持静态类型的数据库不同的是，SQLite中的数据类型被视为数值的一个属性。因此对于一个数据表列而言，即便在声明该表时给出了该列的类型，我们在插入数据时仍然可以插入任意类型，比如Integer的列被存入字符串'hello'。针对该特征唯一的例外是整型的主键列，对于此种情况，我们只能在该列中存储整型数据

每个存放在sqlite数据库中（或者由这个数据库引擎操作）的值都有下面中的一个存储类：

- NULL，值是NULL
- INTEGER，值是有符号整形，根据值的大小以1,2,3,4,6或8字节存放
- REAL，值是浮点型值，以8字节IEEE浮点数存放
- TEXT，值是文本字符串，使用数据库编码（UTF-8，UTF-16BE或者UTF-16LE）存放
- BLOB，只是一个数据块，完全按照输入存放（即没有准换）

SQLite常见的SQL操作命令:

```
/*  建表如果不存在则创建 */
create table if not exists t1 (id integer primary key autoincrement,name text,money real,pic BLOB);

/* 查看表结构,sqlite_master是用来存表、视图、索引的系统表 */
select * from sqlite_master where type="table"

/* 增加数据, pic 在app时在测试 */
insert into t1 (name,money) values ('小强', 8000.00);
insert into t1 (name,money) values ('旺财', 9000.00);
insert into t1 (name,money) values ('主管', 15000.00);
insert into t1 (name,money) values ('小王', 4000.00);
insert into t1 (name,money) values ('小李', 7000.00);

/* 更新相关数据,顺便测试SQLite的弱类型特性 */
update t1 set name = '小王', money = '12000' where id = 3;

/* 模糊查询 + 分页查询 */
select * from t1 where name like '%小%' and money > 6000 limit 0,2;

/* 删除数据 */
delete from t1 where money > 10000;

/* 按工资进行排序 */
select * from t1 order by money desc;

/* 常见的聚合函数使用 */
select min(money) '最低工资',max(money) '最高工资',avg(money) '平均工资',count(*) '总人数' from t1;

/* 查询出来工资相同的人信息*/
select * from t1 where money=(select money from t1 group by money having count(money)>1);

/* 创建视图：查询个人信息时屏蔽相应的工资 */
create view tv as select id,name,pic from t1;
select * from tv;

/*
   对常见的关键字建立一个索引,在后期查询的时候加快查询速度

   注意：索引有助于加快SELECT查询和WHERE子句，但它会减慢数据的输入，UPDATE和INSERT语句。索引可以创建或删除，但数据不会影响

   索引在什么时候避免使用:

      1: 索引不应该使用较小的表上

      2: 有频繁的，大批量的更新或插入操作的表

      3: 索引不应使用含有大量的NULL值的列
*/
CREATE INDEX ti ON t1 (name);
```

```
50.    DROP INDEX ti;
```

测试的相关类型介绍：

1. 测试功能来区分：
2. 黑盒测试：测试逻辑业务(功能OK就行,不管输入与输出)
3. 白盒测试：测试里面的代码(在功能OK的情况测试性能)
4. 测试按照粒度来区分：
5. 方法测试、单元测试、集成测试、系统测试
6. 其它测试：
7. 压力测试、模拟测试(模拟：HTTP Servlet 例如：HTTPClient)

AndroidTestCase 测试用例使用

```
1.    public class AndroidTestDemo extends AndroidTestCase {
2.
3.     @Override
4.     protected void setUp() throws Exception {
5.      super.setUp();
6.      Log.i("test","所有测试方法之前执行");
7.    }
8.
9.    @Override
10.   protected void tearDown() throws Exception {
11.    super.tearDown();
12.    Log.i("test","所有测试方法之后执行");
13.   }
14.
15.   public void test01(){
16.    Log.i("test","测试方法执行了!");
17.   }
```

运行完毕之后会出现异常,通过异常可以看出AndroidManifest.xml缺少相应的配置文件
启动测试的工具类:InstrumentationTestRunner 负责运行单元测试的targetPackage:要测试的目标包名(通过包名来确定应用程序的名称)

```
1.    <instrumentation android:name="android.test.InstrumentationTestRunner" android:targe
      tPackage="com.example.first" />
2.
```

导入测试时需要的包路径,此库是在模拟器的系统中,而不是在android.jar中

```
1.    <uses-library android:name="android.test.runner" />
```

创建一个工具类，此工具类用来实现与SQLite的库与表的创建.必须继承:SQLiteOpenHelper,但是并不负责数据的CRUD操作

```
1.    public class SQLiteHelp extends SQLiteOpenHelper {
```

```
    /*
     * Context: 上下文 , name: 数据库名称   cursorFactory: 用来生成游标的工厂, version: 数据库的版本号
     * */
    public SQLiteHelp(Context context, String name, CursorFactory factory,
      int version) {
     super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
     Log.i("demo","数据库创建调用此方法");
     db.execSQL("create table t1 (id integer primary key autoincrement,name text,money real)");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
     Log.i("demo","数据库的更新调用此方法");
           db.execSQL("drop table if exists t1");
     db.execSQL("create table t1 (id integer primary key autoincrement,name text,money real,pic BLOB)");
    }
   }
```

基本的增、删、查、改操作实现

```
   public void insert() {
     db.execSQL("insert into t1 (name,money) values (?,?) ", new Object[] {"admin", 1800.00 });
   }

   public void query() {
    Cursor cursor = db.rawQuery("select * from t1 limit ?,?", new String[] { "0", "2" });
    while (cursor.moveToNext()) {
      int id = cursor.getInt(0);
      String name = cursor.getString(cursor.getColumnIndex("name"));
      Log.i("demo","id:" + id + "\t name:" + name);
    }
   }

   public void update() {
    db.execSQL("update t1 set name=?,money=? where id=?", new Object[] {"admin2", 1900.00, 2 });
   }

   public void delete(){
    db.execSQL("delete from t1 where id = ?",new Object[]{1});
   }
```

事务的实现操作：

```
   public void trans() {
```

```
    try{
    // 手动开启事务,默认情况下每条SQL语句都是一个事务
    db.beginTransaction();
    db.execSQL("update t1 set name=? where id=?", new Object[] {"admin2",4});
    int num = 10 / 0;
    db.setTransactionSuccessful();
    // 结束事务,如果没有执行 setTransactionSuccessful 则默认回滚
    }finally{
     db.endTransaction();
    }
   }
```