在某些情况下，文件不是有效的
　　如果多线程数据访问是相关的
　　如果应用程序处理可能变化的复杂数据结构
　　等等

因此，Android 带来了内置 SQLite 数据库支持

数据库对于创建它们的包套件是私有的

数据库不应该用来存贮文件

SQLite 是一个轻量级的软件库

实现了一个完全适应严峻环境的数据库
　　原子量性
　　坚固性
　　独立性
　　耐久性

体积大小只用几千字节

一些 SQL 的指令只是部分支持，例如：ALTER、TABLE

下面来看下怎样使用 SQLiteDatabase.

第一种方案：

1.首先要创建一个类，该类继承自 android.database.sqlite.SQLiteOpenHelper，由于这个是 abstract class，因此你需要实现该类的两个方法，一个是 onCreate(),一个是 onUpgrade().注：两个方法的参数都省略了。

示例如下：

```
public class MySQLhelper extends SQLiteOpenHelper {
        public static final String tableName = "localContact";
        public static final String ID = "_id";
        public static final String nameRow = "name";
        public static final String numRow = "mobileNum";
        /*
        * version 很重要，由它确定是否需要升级数据库，当 version 大于数据库自己保存的 version 事，就会调用
        *onUpgrade 方法
        */
        public MySQLhelper(Context context, String name, CursorFactory factory, int version) {
                super(context, name, factory, version);
        }
        @Override
        public void onCreate(SQLiteDatabase db) {
        //只有在数据库被创建时才调用
             this.db = db;
            //使用_writeLock锁住数据库的修改，以保证线程安全
            synchronized (_writeLock) {
              db.beginTransaction();
              try{
```

```
                    db.execSQL("CREATE TABLE IF NOT EXISTS "+tableName
                            +" ("/*+ID+" INTEGER PRIMARY KEY,"*/+nameRow+" VARCHAR,"
                            +numRow+" VARCHAR)");
                    db.setTransactionSuccessful();
                }
                finally {
                    db.endTransaction();//此时不能关闭数据库，不然创建数据库完成后，数据库不能使用
                }
            }

        }
        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        //该方法只有在数据库版本号 version 修改时才会被调用
            synchronized (_writeLock) {
                db.beginTransaction();
                try{
                    // 删除以前的旧表，创建一张新的空表
                    db.execSQL("DROP TABLE IF EXISTS " + tableName);
                    db.setTransactionSuccessful();
                }
                finally {
                    db.endTransaction();
                }
            }
            onCreate(db);
        }
    }
```

2.在 activity 中，创建一个 MySQLhelper 的实例

```
MySQLhelper sqlHelper = new MySQLhelper(this,DBname,null,1);
SQLiteDatabase sql = sqlHelper.getWritableDatabase();
```

此时便获得了刚刚创建的数据库实例的写入权限。

3.通过调用相应的方法如 Insert(),delete()等，对数据库进行插入和删除的操作

4.示例：例如进行插入操作时，可以直接调用 Insert()方法，代码如下：

```
ContentValues tcv = new ContentValues();
tcv.put(MySQLhelper.nameRow, name);
tcv.put(MySQLhelper.numRow, mobilePhone);
sql.insert(MySQLhelper.tableName, null, tcv);
```

一个整体代码的实例：

```
package com.openwudi.service;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
```

```
import android.database.sqlite.SQLiteOpenHelper;

public class DBOpenHelper extends SQLiteOpenHelper {
    private static String NAME = "person.db";
    private static int VERSION = 1;

    public DBOpenHelper(Context context) {
       super(context, NAME, null, VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
       db.execSQL("CREATE TABLE IF NOT EXISTS person (personid integer primary key autoincrement, name
varchar(20), age INTEGER)");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
       db.execSQL("DROP TABLE IF EXISTS person");
      onCreate(db);
    }

}
```

由于考虑到移动设备资源的消耗问题，在 Android 应用中将不创建 DAO 接口，为了节省开销。

并且暂时 Android 应用不需要做解耦操作，DAO 接口此时显得有些多余。

对数据库的操作将创建一个类，来直接实现。下面此类实现了增，删，改，查，分页，取得数据总数的操作。

```
package  com.openwudi.service;

import  java.util.ArrayList;
import  java.util.List;

import  android.content.Context;
import  android.database.Cursor;
import  android.database.sqlite.SQLiteDatabase;

import  com.openwudi.domain.Person;

public  class  PersonService  {
    private  DBOpenHelper  dbOpenHelper;
     private  final  static  byte[]  _writeLock  =  new  byte[0];  //定义一个 Byte 作为写锁,解决多线程同时操作数据库问题
    public  PersonService(Context  context)  {
       this.dbOpenHelper  =  new  DBOpenHelper(context);
    }

    public  void  save(Person  person)  {
       //使用_writeLock 锁住数据库的修改，以保证线程安全
```

```
  synchronized (_writeLock) {
  // 要对数据进行更改，调用此方法，该方法以读和写的方式打开数据库
            SQLiteDatabase db = dbOpenHelper.getWritableDatabase();
              //使用事物处理保证事物安全
             db.beginTransaction();
             try{
                  db.execSQL("insert into person(name,age) values (?,?)", new Object[] {
                   person.getName(), person.getAge() });
                 db.setTransactionSuccessful();
             }finally{
                  db.endTransaction();
                 db.close();
             }
     }
}

public void update(Person person) {
    //使用_writeLock 锁住数据库的修改，以保证线程安全
  synchronized (_writeLock) {
           // 要对数据进行更改，调用此方法，该方法以读和写的方式打开数据库
             SQLiteDatabase db = dbOpenHelper.getWritableDatabase();
              //使用事物处理保证事物安全
             db.beginTransaction();
             try{
                       db.execSQL("update person set name=?,age=? where personid=?",
                                 new Object[] { person.getName(), person.getAge(),
                                 person.getId() });
                      db.setTransactionSuccessful();
             }finally{
                  db.endTransaction();
                 db.close();
             }
     }
 }

public void delete(Integer id) {
    //使用_writeLock 锁住数据库的修改，以保证线程安全
  synchronized (_writeLock) {
             // 要对数据进行更改，调用此方法，该方法以读和写的方式打开数据库
             SQLiteDatabase db = dbOpenHelper.getWritableDatabase();
              //使用事物处理保证事物安全
             db.beginTransaction();
             try{
                       db.execSQL("delete from person where personid=?", new Object[] { id.toString() });
                        db.setTransactionSuccessful();
             }finally{
                  db.endTransaction();
                  db.close();
```

```
            }
        }

    }

    public Person find(Integer id) {
        Person person = null;
        synchronized (DBHelper._writeLock) {
            // 要对数据进行读取，建议调用此方法
            SQLiteDatabase db = helper.getReadableDatabase();
            Cursor c = null;
            try{ //使用 try/finally 方式保证数据库在每次使用完毕后能被关闭
                cursor = db.rawQuery("select * from person where personid=?",
                    new String[] { id.toString() });
                if (cursor.moveToFirst()) {
                    Integer personid = cursor.getInt(cursor.getColumnIndex("personid"));
                    String name = cursor.getString(cursor.getColumnIndex("name"));
                    Integer age = cursor.getInt(cursor.getColumnIndex("age"));
                    person  = Person(personid, name, age);
                }
            }finally{
                if(c!=null)c.close();
                db.close();
            }
            return    person ;
        }

    public List<Person> getScrollData(Integer offset, Integer maxResult) {
         List<Person> persons = new ArrayList<Person>();
        synchronized (DBHelper._writeLock) {
                // 要对数据进行读取，建议调用此方法
                SQLiteDatabase db = helper.getReadableDatabase();
                Cursor c = null;
                try{ //使用 try/finally 方式保证数据库在每次使用完毕后能被关闭
                    cursor = db.rawQuery("select * from person limit ?,?",
                        new String[]{offset.toString(),maxResult.toString()});
                     while (cursor.moveToNext()) {
                        Integer personid = cursor.getInt(cursor.getColumnIndex("personid"));
                        String name = cursor.getString(cursor.getColumnIndex("name"));
                        Integer age = cursor.getInt(cursor.getColumnIndex("age"));
                        persons.add(new Person(personid, name, age));
                    }

                }finally{
                     if(c!=null)c.close();
                     db.close();
                }
          }
```

```
            return persons;
    }

    public long getCount() {
            long count = 0l;
            synchronized (DBHelper._writeLock) {
                    // 要对数据进行读取，建议调用此方法
                    SQLiteDatabase db = helper.getReadableDatabase();
                    Cursor c = null;
                    try{ //使用 try/finally 方式保证数据库在每次使用完毕后能被关闭
                            cursor = db.rawQuery("select count(*) from person",null);
                        if (cursor.moveToFirst()) {
                                count = cursor.getLong(0);
                            }
                    }finally{
                            if(c!=null)c.close();
                            db.close();
                    }
              }
              return count;
    }
}
```

解释一下 getReadableDatabase () 和 getWritableDatabase () 的区别。

这两个方法都可以获得 SQLiteDatabase 对象。

getWritableDatabase () 获得一个可进行操作的数据库类。并且查看源代码，了解到获得的是单实例的 SQLiteDatabase。因为不论是调用几次 getWritableDatabase () 方法返回的都是同一个连接。

getReadableDatabase () 同样是获得 SQLiteDatabase 对象。一样查看源代码，了解到 getReadableDatabase () 内部同样是调用 getWritableDatabase ()来实现获得 SQLiteDatabase 对象的。但是，getReadableDatabase () 判断一旦 getWritableDatabase () 抛出异常（比如存储空间满了），将以只读的方式读取数据库，此时获得的 SQLiteDatabase 对象将和 getWritableDatabase () 获得的不一样。

第二种方案：

Context.createDatabase(String name,int version ,int mode,CursorFactory factory)创建一个新的数据库并返回一个 SQLiteDatabase 对象

假如数据库不能被创建，则抛出 FileNotFoundException 异常

新创建 SQLite 数据库方法

```
SQLiteDatabase myDataBase=this.openOrCreateDatabase("myDataBase.db",
                        MODE_PRIVATE, new CursorFactory(){
//创建新的数据库，名称 myDatabase，模式 MODE_PRIVATE，鼠标工厂
//工厂类，一个可选工厂类，当查询时调用来实例化一个光标
    @Override
    public Cursor newCursor(SQLiteDatabase db,
            SQLiteCursorDriver masterQuery, String editTable,
```

```
            SQLiteQuery query) {
        // TODO Auto-generated method stub
            return null;
            }
});
```

删除数据库

Context.deleteDatabase(String name)删除指定名称的数据库

假如数据库成功删除则返回 true，失败则为 false(例如数据库不存在)

```
        //删除指定名称的数据库
        this.deleteDatabase("myDatabase.db");
```

打开数据库

Context.openDatabase(String file,CursorFactory factory)打开一个存在的数据库并返回一个 SQLiteDatabase 对象

如果数据库不存在则抛出 FileNotFoundException 异常

```
//创建一个名为：myDataBase 的数据库，后缀为.db
SQLiteDatabase my_DataBase=this.openOrCreateDatabase("myDateBase.db",MODE_PRIVATE, null);
my_DataBase.close();//不要忘记关闭数据库
```

非查询 SQL 指令

SQLiteDatabase.execSQL(String sql)可以用来执行非查询 SQL 指令，这些指令没有结果

包括：CREATE TABLE / DROP TABLE / INSERT 等等

例如：

创建一个名为"test"并带两个参数的表

```
//创建一个名为"test"并带两个参数的表
my_DataBase.execSQL("CREATE TABLE test (_id INTEGER PRIMARY KEY,
                                        someNumber INTERGER);");
```

在数据库中插入一个元组

```
//在数据库中插入一个元组
my_DataBase.execSQL("INSERT INTO test (_id,someNumber) values(1,8);");
```

删除表

```
//删除表
my_DataBase.execSQL("DROP TABLE test");
```

网上强烈建议使用第一种，原因在于使用数据库时读写分开，利于控制