



## 13.3 数据库存储

- 13.3.1 SQLite数据库
  - SQLite是一个开源的嵌入式关系数据库
  - 在2000年由D. Richard Hipp发布





## 13.3 数据库存储

- 13.3.1 SQLite数据库
  - SQLite数据库特点
    - 更加适用于嵌入式系统，嵌入到使用它的应用程序中
    - 占用非常少，运行高效可靠，可移植性好
    - 提供了零配置（zero-configuration）运行模式
  - SQLite数据库不仅提高了运行效率，而且屏蔽了数据库使用和管理复杂性，程序仅需要进行最基本的数据操作，其他操作可以交给进程内部的数据库引擎完成





## 13.3 数据库存储

- 13.3.1 SQLite数据库
- SQLite 是一个轻量级的软件库
  - 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：不支持ALTER TABLE）





## 13.3 数据库存储

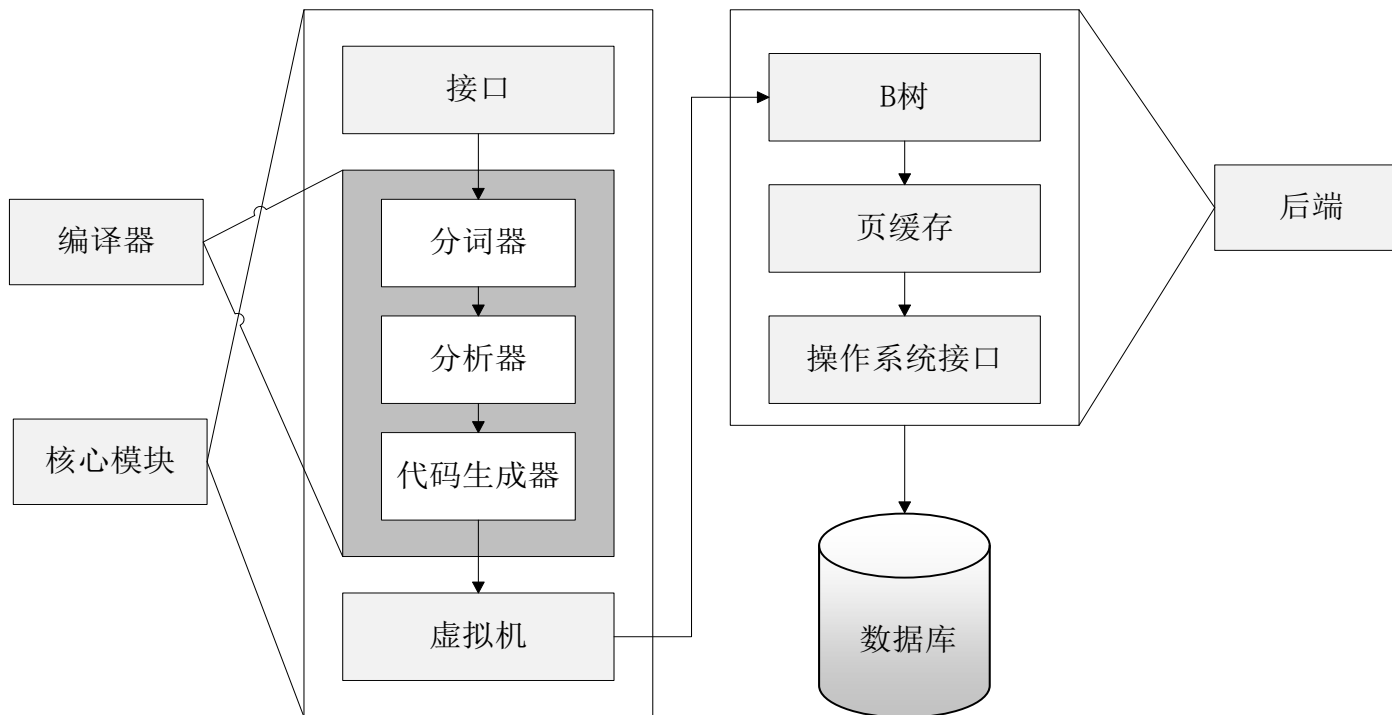
- 13.3.1 SQLite数据库
  - 这些数据库和其中的数据是应用程序**所私有的**
  - 不应该用其来存储文件
  - 如果要将其同其他应用程序共享，则必须把应用程序变为一个ContentProvider（后面将讲到）





## 13.3 数据库存储

- 13.3.1 SQLite数据库
  - SQLite数据库采用了模块化设计，由8个独立的模块构成，这些独立模块又构成了三个主要的子系统，模块将复杂的查询过程分解为细小的工作进行处理





## 13.3 数据库存储

- 13.3.1 SQLite数据库
  - SQLite数据库具有很强的移植性，可以运行在Windows, Linux, BSD, Mac OS X和一些商用Unix系统，比如Sun的Solaris, IBM的AIX
  - SQLite数据库也可以工作在许多嵌入式操作系统下，例如QNX, VxWorks, Palm OS, Symbian和Windows CE
  - SQLite的核心大约有3万行标准C代码，模块化的设计使这些代码更加易于理解





## 13.3 数据库存储

- 13.3.1 SQLite数据库
  - Android SDK包含了若干有用的SQLite数据库管理类
  - 大多都存在于android.database.sqlite包中
  - 包中含有许多功能包类：管理数据库的创建和版本信息、数据库管理以及查询生成类等
  - 利用这些包能帮助你生成正确的SQL表达式和查询

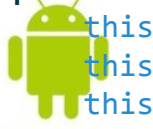




## 13.3 数据库存储

- 13.3.2 使用SQLite数据库
  - **第一步：创建实体类**
  - 数据库管理系统中的各种用于数据管理方便而设定的各种数据管理对象，如：数据库表、视图、存储过程等都是数据库实体。广义上讲，这些对象中所存储的数据也是数据库实体。因为它们也是确切存在着的实体。
  - 实现一个实体类需要有如下元素：
    - 属性、构造函数、属性Get函数、属性Set函数

```
public class Member {  
    private Integer id;  
    private String name;  
    private String info;  
  
    public Member(){  
  
    }  
  
    public Member(String name, String info) {  
        this.name = name;  
        this.info = info;  
    }  
  
    public Member(Integer id, String name, String info) {  
        this.id = id;  
        this.name = name;  
        this.info = info;  
    }  
}
```

  
ANDROID

```
    public Integer getId() {  
        return id;  
    }  
    public void setId(Integer id) {  
        this.id = id;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public String getInfo() {  
        return info;  
    }  
    public void setInfo(String info) {  
        this.info = info;  
    }  
}
```





## 13.3 数据库存储

- 13.3.2 使用SQLite数据库
  - 第二步：使用应用程序上下文创建SQLite数据库
    - 首先创建的类需要继承SQLiteOpenHelper类

```
public class MemberDAO extends SQLiteOpenHelper
```

- 然后定义数据库的名字、版本以及里面的TABLE的名字，定义SQL的创建表格命令：

```
private static final String DB_NAME = "member.db";
private static final int DB_VERSION = 1;
private static final String TABLE_NAME = "member";
private static final String SQL_CREATE_TABLE = "create table " + TABLE_NAME
    + " (_id integer primary key autoincrement,"
    + " name text not null, info text);";
```

- 在onCreate函数中调用execSQL函数执行创建表格指令即可成功创建表格

```
/**
 * 第一次调用 getWritableDatabase() 或 getReadableDatabase() 时调用
 */
@Override
public void onCreate(SQLiteDatabase db) {
    // 创建数据库表
    db.execSQL(SQL_CREATE_TABLE);
}
```





## 13.3 数据库存储

- 13.3.2 使用SQLite数据库
- 完整代码示例如下：

```
public class MemberDAO extends SQLiteOpenHelper {
    private static final String DB_NAME = "member.db";
    private static final int DB_VRESION = 1;
    private static final String TABLE_NAME = "member";
    private static final String SQL_CREATE_TABLE = "create table " + TABLE_NAME
        + " (_id integer primary key autoincrement,"
        + " name text not null, info text);";

    public MemberDAO(Context c) {
        /*
         * 创建数据库访问对象 它实际上没有创建数据库，马上返回。 只有调用 getWritableDatabase() 或
         * getReadableDatabase() 时才会创建数据库 数据库文件位于 /data/data/<包名>/databases
         */
        super(c, DB_NAME, null, DB_VRESION);
    }

    /* 第一次调用 getWritableDatabase() 或 getReadableDatabase() 时调用 */
    @Override
    public void onCreate(SQLiteDatabase db) {
        // 创建数据库表
        db.execSQL(SQL_CREATE_TABLE);
    }

    /* DB_VRESION 变化时调用此函数 */
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // 更新数据库版本（这里不使用）
        // db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        // onCreate(db);
    }
}
```





## 13.3 数据库存储

- 13.3.2 使用SQLite数据库
  - **第三步：实现数据操作**
    - 数据操作是指对数据的添加、删除、查找和更新操作；
    - 可以通过execsql函数直接执行SQL命令来完成数据操作，也可以通过调用SQLiteDatabase类的公共函数insert()、delete()、update()和query()这四个函数，封装了执行的添加、删除、更新和查询功能的SQL命令；
    - 下面通过前面的Member实例，分别介绍如何使用SQLiteDatabase类的公共函数，完成数据的添加、删除、更新和查询等操作





## 13.3 数据库存储

- 添加功能

- 首先构造一个ContentValues对象，然后调用ContentValues对象的put()方法，将每个属性的值写入到ContentValues对象中，最后使用 SQLiteDatabase对象的insert()函数，将ContentValues对象中的数据写入指定的数据库表中
- insert()函数的返回值是新数据插入的位置，即ID值。ContentValues类是一个数据承载容器，主要用来向数据库表中添加一条数据

```
/* 插入操作 */
public long insert(Member entity) {
    SQLiteDatabase db = getWritableDatabase();
    ContentValues values = new ContentValues();

    // 名称/值对，第一个参数是名称，第二个参数是值；
    values.put("name", entity.getName());
    values.put("info", entity.getInfo());

    // 必须保证 values 至少一个字段不为null，否则出错
    long rid = db.insert(TABLE_NAME, null, values);
    db.close();
    return rid;
}
```





## 13.3 数据库存储

- 删除功能

- 删除数据比较简单，只需要调用当前数据库对象的delete()函数，并指明表名称和删除条件即可

```
/* 删除操作 */  
public int deleteById(Integer id) {  
    SQLiteDatabase db = getWritableDatabase();  
    String whereClause = "_id = ?";  
    String[] whereArgs = { id.toString() };  
    int row = db.delete(TABLE_NAME, whereClause, whereArgs);  
    db.close();  
    return row;  
}
```

- delete()函数第1个参数是数据库表名，后面的参数是删除条件
- 参数id指明了需要删除数据的id值，因此deleteById()函数仅删除一条数据，此时delete()函数的返回值表示被删除的数据的数量；
- 如果后面两个参数均为null，那么表示删除数据库中的全部数据。





## 13.3 数据库存储

- 更新功能
  - 更新数据同样要使用ContentValues对象
  - 首先构造ContentValues对象
  - 然后调用put()函数将属性的值写入到ContentValues对象
  - 最后使用SQLiteDatabase对象的update()函数，并指定数据的更新条件，函数返回的值是更新的数据数量

```
/* 更新操作 */  
public int update(Member entity) {  
    SQLiteDatabase db = getWritableDatabase();  
    String whereClause = "_id = ?";  
    String[] whereArgs = { entity.getId().toString() };  
    ContentValues values = new ContentValues();  
    values.put("name", entity.getName());  
    values.put("info", entity.getInfo());  
    int rows = db.update(TABLE_NAME, values, whereClause, whereArgs);  
    db.close();  
    return rows;  
}
```





## 13.3 数据库存储

- 查询功能

- 首先介绍Cursor类。在Android系统中，数据库查询结果的返回值并不是数据集合的完整拷贝，而是返回数据集的指针，这个指针就是Cursor类；
- Cursor类支持在查询的数据集合中多种方式移动，并能够获取数据集合的属性名称和序号，Cursor类方法说明如下：

函数	说明
moveToFirst	将指针移动到第一条数据上
moveToNext	将指针移动到下一条数据上
moveToPrevious	将指针移动到上一条数据上
getCount	获取集合的数据数量
getColumnIndexOrThrow	返回指定属性名称的序号，如果属性不存在则产生异常
getColumnName	返回指定序号的属性名称
getColumnNames	返回属性名称的字符串数组
getColumnIndex	根据属性名称返回序号
moveToPosition	将指针移动到指定的数据上
getPosition	返回当前指针的位置





## 13.3 数据库存储

- 查询操作：
  - SQLiteDatabase类的query()函数参数说明

```
Cursor android.database.sqlite.SQLiteDatabase.query(String table,  
String[] columns, String selection, String[] selectionArgs,  
String groupBy, String having, String orderBy)
```

位置	类型+名称	说明
1	String table	表名称
2	String[] columns	返回的属性列名称
3	String selection	查询条件
4	String[] selectionArgs	如果在查询条件中使用的问号，则需要定义替换符的具体内容
5	String groupBy	分组方式
6	String having	定义组的过滤器
7	String orderBy	排序方式







## 13.3 数据库存储

- 查询操作：
  - 示例代码如下：

```
/* 查询操作 */
public Member getById(Integer id){
    Member m = null;
    SQLiteDatabase db = getReadableDatabase();
    String selection = "_id = ?";
    String[] selectionArgs = { id.toString() };
    Cursor c = db.query(TABLE_NAME, null, selection, selectionArgs, null,null, null);
    if (c.moveToNext()){
        m = new Member(c.getInt(0),c.getString(1),c.getString(2));
    }
    c.close();
    db.close();
    return m;
}
```

- 在从Cursor中提取数据之前，推荐先测试一下Cursor中的数据数量，避免在数据获取的过程中产生异常情况。



```
if (resultCounts == 0 || !cursor.moveToFirst()){
    return null;
}
```