Android中sqlite数据库

在Android中产生数据库文件的路径：/date/date/应用包名/databases/数据库文件

SQLite数据库,一个单独文件的数据库系统.以文件的形式存在硬盘，文件中包含数据库，数据库中是我们的数据表 轻型关系数据库管理系统，Android将其

作为内嵌数据库系统

通过JAVA中的类实现SQLite数据库的数据操作：

1、第一需要助手

自定义一个类extends SQLiteOpenHelper类

必须提供一个参多个参数的构造方法,在构造中super的参数必须手动指定

参数说明：

Context 上下文环境，不用指定参数 为默认值

String 数据库文件的文件名 要给出文件名，这个参数也可以在获取助手时给出

CursorFactory 获取查询所用游标的工厂 没有给null 默认也是null

int 数据库版本号 版本号 从1的数值，1、2、3...默认为1

重写onCreate(SQLiteDatabase db)方法 通过execSQL（）方法在该方法中向数据库中创建表

格式： db.execSQL("create table user(_uid integer primary key autoincrement,name varchar(8) not null,sex varchar(3),pwd int,age int)");

重写onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)方法

该方法用于更新数据库版本号，默认不做任何处理

2、在MainActivity类中获取助手

将助手的对象设置为成员变量 db = new Mydph(this);

3、写增加，删除，修改，查询的方法

增加：

```
//1、获得助手
SQLiteDatabase database = dbh.getWritableDatabase();
//2、设置SQL语句
String str ="insert into user(name,sex,pwd,age) values('张三','男',1238252,29)";
//3、向表里添加数据
database.execSQL(str);
//4、关闭助手
database.close();
```

修改，删除代码及步骤相同

```
//1、获得助手 注意是通过getReadableDatabase()方法获取助手的。
SQLiteDatabase database = dbh.getReadableDatabase();
//2、设置SQL语句 把姓名叫陈小强改成张玉玉
String str = "update user set name ='张玉玉' where name='陈小强'";
//3、向表里添加数据
database.execSQL(str);
//4、关闭助手
database.close();
```

查询：

```
//1、获得助手        注意是通过getReadableDatabase()
SQLiteDatabase  database  =  dbh.getReadableDatabase();
//2、查寻语句
String  str  =  "select  *  from  user";
//3、开始查询
Cursor  query  =  database.rawQuery(str,  null);
String  string  ="";
//4、
while(query.moveToNext()){
string  +=  query.getInt(0)+"    "+query.getString(1)+"    "+query.getString(2)+"    "+query.getInt
}
//5、
database.close();
```

```
    public DBHelper(Context context, String name) {
        /**
         * context :上下文
         * name: 数据库文件的文件名
         * factory : 游标工厂,
         * version 数据库版本号
         */
        super(context, name, null, 1);
    }

    @Override
    /**
     * 创建数据库，我们将建表代码写在此处
     */
    public void onCreate(SQLiteDatabase db) {
        // 通过SQL代码去建表
        db.execSQL("create table user(uid integer primary key autoincrement, " +
                "name varchar(18) not null,sex varchar(5),age int)");
    }
```

```
    // 获取数据库助手对象
    dbHelper = new DBHelper(this, "userinfo");
}
// 创建数据库
public void create(View v) {
    SQLiteDatabase db = dbHelper.getReadableDatabase();
}

public void insert(View v) {
    SQLiteDatabase db = dbHelper.getReadableDatabase();
    String sql = "insert into user(name,sex,age) values('张三','男',19)";
    db.execSQL(sql);
    db.close();
    Toast.makeText(this, "添加成功", 0).show();
}
public void select(View v) {
    SQLiteDatabase db = dbHelper.getReadableDatabase();
    String sql = "select * from user";
    // 通过游标进行数据读取
    Cursor cursor = db.rawQuery(sql, null);
    String content = "";
    while (cursor.moveToNext()) { // 如果能移动，就继续
        content += cursor.getInt(0);
        content += cursor.getString(1);
        content += cursor.getString(2);
        content += cursor.getInt(3);
        content += "\n";
```

```
    }
    db.close();
    txt.setText(content);
}
public void update(View v){
    SQLiteDatabase db = dbHelper.getReadableDatabase();
    String sql = "update user set age=22 where uid=1 ";
    db.execSQL(sql);
    db.close();
    Toast.makeText(this, "修改成功", 0);
}
public void delete(View v){
    SQLiteDatabase db = dbHelper.getReadableDatabase();
    String sql = "delete from user where uid=1";
    db.execSQL(sql);
    db.close();
    Toast.makeText(this, "删除成功", 0);
}
```

通过谷歌封装好的API操作SQLite数据库的数据

自定义一个类extends SQLiteOpenHelper类

必须提供一个参多个参数的构造方法,在构造中super的参数必须手动指定

参数说明:

Context 上下文环境，不用指定参数 为默认值

name 数据库文件的文件名 要给出文件名，这个参数也可以在获取助手时给出

CursorFactory 获取查询所用游标的工厂 没有给null 默认也是null

int 数据库版本号 版本号 从1的数值，1、2、3...默认为1

重写onCreate(SQLiteDatabase db)方法 通过execSQL（）方法在该方法中向数据库中创建表

格式： db.execSQL("create table user(_uid integer primary key autoincrement,name varchar(8) not null,sex varchar(3),pwd int,age int)");

重写onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)方法

该方法用于更新数据库版本号，默认不做任何处理

2、在MainActivity类中获取助手

将助手的对象设置为成员变量 db = new Mydph(this);

1、增加

```
        //1、获得助手
SQLiteDatabase ddb = db.getReadableDatabase();
         //2、设置要向数据库 表中要添加的列及值对象
  ContentValues values = new ContentValues();
        //3、通过put()方法以键值对的形式添加设置数据
             values.put("name", "张三");
             values.put("sex","男");
             values.put("pwd", 123852);
             values.put("age", 22);
       //4、通过谷歌内置的方法向数据库添加
```

```
            ddb.insert("user",  null,  values);
        //5、关闭助手
            ddb.close();
```

修改：

```
      //修改
    SQLiteDatabase  database  =  db.getReadableDatabase();
    ContentValues  values  =  new  ContentValues();
          values.put("name",  "王小Y");
   //通过助手向把指定表中age为22的    name改为  王小明
    database.update("user",  values,  "age=?",  new  String[]{"22"});
      //关闭助手
    database.close();
```

删除：

```
    SQLiteDatabase  database  =  db.getWritableDatabase();
    database.delete("user",  "_uid=?",  new  String[]{"2"});
          database.close();
```

查询：

```
SQLiteDatabase  database  =  db.getReadableDatabase();
```

参一：String table 要查询的那个表 " 表名 "
参二：String[] columns查那一列，如果参数是null,则返回所有列。 `new String[] { "CustomerName" , "SUM(OrderPrice)" };`
参三：String selection 查寻条件 `"列名=?" ;`
参四：String[] selectionArgs 查询条件值 `new String[]{ "列的值" };`
参五：String groupBy 一个过滤器,如何来分组。
参六：String having 分组后聚合的过滤条件。
参七：String orderBy排序，格式是SQL的ORDER一样。
参八：String limit返回的行数，设置为null表示没有限制条款。

```
Cursor  qu  =  database.query("user",  null,  null,  null,  null,  null,  null,null);
String  str  =  "";    用于接收查到的数据
if(qu.getCount()>0){   获取查寻数据有无记录，如果有记录有循环获取数据
while(qu.moveToNext()){   从0开始对应相应的数据类型
str  +=qu.getInt(0)+"    "+  qu.getString(1)  +"    "+qu.getString(2)  +"    "+qu.getInt(3)  +"    '
database.close();
```

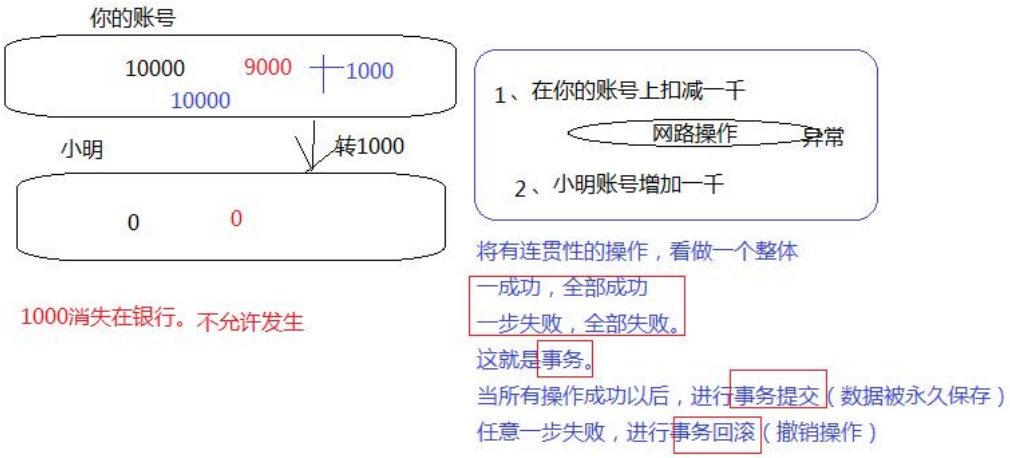
```
public class DBHelper extends SQLiteOpenHelper {

    public DBHelper(Context context, String name) {
        super(context, name, null, 3);
    }

    @Override
    //第一次创建数据库时执行
    public void onCreate(SQLiteDatabase db) {
        Log.w("mylog", "数据库创建");
        String sql = "create table user(_uid integer primary key autoincrement, name varchar(18),pwd varchar(18))";
        db.execSQL(sql);
    }

    @Override
    //版本升级时执行
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w("mylog", "数据库升级");
    }
```

```
    // 增
    public void insert(View v) {
        SQLiteDatabase db = dbHelper.getReadableDatabase();// 要用时，再去拿数据库，不要定义成全局变量
        /*
         * table:表名 nullColumnHack :给null ,用于解决 values没有值时的sql语法问题。 values :
         * 一行数据，就一个values
         */
        ContentValues values = new ContentValues();
        values.put("name", "张三");
        values.put("pwd", "123");
        long uid = db.insert("user", null, values); // 这就是GoogleAPI,需要一个值对象对每个列的值进行封装
        db.close();
        Toast.makeText(this, "增加成功,新增行的_uid是: " + uid, 0).show();
    }
    //修改
    public void update(View v) {
        SQLiteDatabase db = dbHelper.getReadableDatabase();

        ContentValues values = new ContentValues();
        values.put("name", "李四");
        values.put("pwd", "1234");
        int result = db.update("user", values, " _uid=?", new String[]{"1"});//修改uid为1的用户信息
        db.close();
        txt.setText("修改成功，修改了" + result +"行");
    }

    // 查
    public void select(View v) {
        SQLiteDatabase db = dbHelper.getReadableDatabase();
        /**
         * table:表名
         * columns: 列名的字符串数组，如果查询所有，可以给null
         * selection:查询条件
         * selectionArgs:查询条件的参数的字符串数组方式的表示，
         * groupBy: 指定分组方式（列名）
         * having: 分组操作的条件
         *  orderBy : 排序
         */
        Cursor query  = db.query("user", new String[]{"_uid","name","pwd"},null, null, null, null,null);
        //Cursor query = db.query("user", new String[]{"_uid","name","pwd"}," _uid>? and _uid<?", new String[]{"1","3"}, null,
        null," _uid desc");
        String conten = "";
        while(query.moveToNext()){
            conten += "_uid: " + query.getInt(0);
            conten += "name: " + query.getString(1);
            conten += "pwd: "+query.getString(2);
            conten+="\n";
        }
        db.close();
        txt.setText(conten);
    }

    public void delete(View v) {
        SQLiteDatabase db = dbHelper.getReadableDatabase();
        int result = db.delete("user", " _uid=?", new String[]{"2"});
        db.close();
        txt.setText("删除成功，删除了 " + result +" 行");
    }
}
```

Android中数据库的事务

将有连贯性的操作，看做一个整体，一操作成功，全成功，一步操作失败，全失败。

所有操作完成以后进行事务的提交，任意一步操作失败都将进行事务回滚（操作自动撤销）。



代码体现：

通过beginTransaction()开启事务 ------> setTransactionSuccessful()提交事务 ------> endTransaction()关闭事务

（案例意图：张三的钱转给王五，在转帐的过程中，如果出现意外那么转出就会失败，已经操作成功的将会撤销回去，避免出现张三已转出而王五没有收到的情况）

1、获得助手对象

```
SQLiteDatabase db = dbHelper.getReadableDatabase();
2、开启事务
db.beginTransaction();// 开启事务
3、获得修改数据库中，操作数据的对象
ContentValues values = new ContentValues();
4、设置要改的数据
values.put("umoney", 18000 - 1000);
5、修改更新数据
db.update("user", values, "uname=?", new String[] { "张三" });
                    // 出现意外
                    //int i = 10 / 0;

6、设置数据库中另一条数据
db.execSQL("update user set umoney=1000 where uname='王五'");
7、提交事务
db.setTransactionSuccessful();
8、关闭事务
db.endTransaction();
9、关闭助手对象
db.close();
```

```java
// 模拟转账
public void account(View v) {
    SQLiteDatabase db = dbHelper.getReadableDatabase();
    db.beginTransaction();
    try {
        // 张三转成1000 : 减1000 ,先查出张三的余额
        ContentValues values = new ContentValues();
        values.put("money", 8000);
        db.update("user", values, " name=?", new String[] { "张三" });
        Log.w("mylog", "张三转出");

        int i = 1 / 0; // 模拟网络错误

        // 李四转入1000 加1000，先查出李四的余额
        ContentValues values1 = new ContentValues();
        values1.put("money", 2000);
        db.update("user", values1, " name=?", new String[] { "李四" });
        // 提交事务
        db.setTransactionSuccessful();// 设置事务成功标志，就会提交事务 ,不设置就不会提交
    } catch (Exception e) {

    }
    // 结束事务 //如果不结束事务，会锁定数据库
    db.endTransaction();
    db.close();
}
```

ListView组件

它以列表的形式展示具体内容，并且能够根据数据的长度自适应显示。

显示需要三个元素：　通过控制器的管理控制将数据分配给ListVeiw组件然后显示到屏幕上

1. ListVeiw 用来展示列表的View。

2. （视图控制器）适配器 用来把数据映射到ListView上的中介。

3. 数据（Mode）具体的将被映射的字符串，图片，或者基本组件。

根据列表的适配器类型，列表分为三种

ArrayAdapter

```
ArrayAdapter<String>      arrayAdapter;//数组适配器
```

将数据放入布局中对应的ListView组件中

```
arrayAdapter = new ArrayAdapter<String>
(this,R.layout.activity_main_item,R.id.tv_message,messages);
```

SimpleAdapter

SimpleCursorAdapter

为ListView控件添加单击条目事件

ListView.setOnItemClickListener(new OnItemClickListener()

重写onItemClick(AdapterView<?> parent, View view,int position, long id)方法

position参数指代数据或集合中的每一个元素（对应的索引）

String[] messages=new String[]{" ", " ", " "};

messages = messages[position];

其中以ArrayAdapter最为简单，只能展示一行字。SimpleAdapter有最好的扩充性，可以自定义出各种效果。SimpleCursorAdapter可以认为是SimpleAdapter对数据库的简单结合，可以方面的把数据库的内容以列表的形式展示出来。

1、activity_main.xml布局中定义一个ListView组件并设置ID

2、新建item.xml布局，布局中定义定义其它的显示组件（图片、文字）

3、通过ID获取ListView组件

4、ListView lv_contact一定要.setAdapter（new 内部类()）继承BaseAdapter类的适配器类

5、自定义内部类适配器内部类一定要继承BaseAdapter重写方法，适配器主要是用来向ListView添加数据。

6、设置item的数量　　重写getCount()方法

7、ListView会调用该方法，重写getView(int position, View convertView, ViewGroup parent)方法。在没有设置item的时候可以将数据给TextView来显示。

position: 屏幕显示的item从0开始，给item位置编号，最大值为 getCount()-1

convertView : 随着用户滑动手机界面，而被隐藏的 Item, 适配器传回给你，方便重复使用

parent : 就是前面定义的 ListView (lv)


```
@Override
// 调用这个方法的是 listView ,
// 程序员可以在此创建 Item中的具体UI组件
public View getView(int position, View convertView, ViewGroup parent) {
    //适配器优化：重复使用UI组件，从而降低内存消耗
    TextView txt = null;
    if (convertView == null) {
        txt = new TextView(getApplicationContext());
        i++;
    }else{
        txt = (TextView)convertView;
    }
    txt.setText(" item"+i+": " + " position " +position );

    return txt;
}
}
}
```

打气筒inflate（）方法的获取方式，这个方法是View类的方法：

inflate()方法的三个参数：

Context context:上下文 返回一个int值。

resource ：通过R文件获取的一个布局，我们的例子中 itme

ViewGroup root ：　是指定打气筒创建的View的父UI组件。

为什么这个参数是null，而不是ListView？因为调用geiView（）的最终就是ListView通过适配器来完成的，而在适配器中，不用再指定因为适配器本身就会被set到一ListView，就有一个根了。

1、View自己获取打气筒，将布局xml ，转换成 View

v = View.inflate(getApplicationContext(),R.layout.item,null);

参数的意思：通过应用程序上下文对象，用打气筒给资源打气，然后给一个根视图（至于这个根是谁由控制器来分配处理所以给null）

```
// 自定义适配器类
public class MyAdapter extends BaseAdapter {
    int i = 0;
    @Override
    // 告诉LV , Item的总数
    public int getCount() {
        return 100;
    }
    @Override
    // 获取指定位置的item
    public Object getItem(int position) {
        return null;
    }
    @Override
    // item的ID
    public long getItemId(int position) {
        return 0;
    }
    @Override
    // 调用这个方法的是 listView ,
    // 程序员可以在此创建 Item中的具体UI组件
    public View getView(int position, View convertView, ViewGroup parent) {
        //适配器优化 : 重复使用 UI组件, 从而降低内存消耗
        TextView txt = null;
        if (convertView == null) {
            txt = new TextView(getApplicationContext());
            i++;
        }else{
            txt = (TextView)convertView;
        }
        txt.setText(" item"+i+": " + " position " +position );
        return txt;
    }
```

2、通过布局填充器获取打气筒

```
LayoutInflater layout =  LayoutInflater.from(getApplicationContext());
v =layout.inflate(R.layout.item, null);
```

3、通过系统服务获取打气筒

```
LayoutInflater layout =  (LayoutInflater) getSystemService(LAYOUT_INFLATER_SERVICE);

v = layout.inflate(R.layout.item, null);
```

布局填充器

LayoutInflater类在应用程序中比较实用，可以叫布局填充器，也可以成为打气筒，意思就是将布局文件填充到自己想要的位置，LayoutInflater是用来找res/layout/下的xml布局文件，并且实例化这个XML文件成为一个View，有点类似于类似于findViewById()，但是findViewById()是找xml布局文件下的具体widget控件(Button、TextView)。对于一个没有被载入或者想要动态载入的界面，都需要使用LayoutInflater.inflate()来载入；对于一个已经载入的界面，使用Activiyt.findViewById()方法来获得其中的界面元素。

第一种getLayoutInflater方式:

```
return getWindow().getLayoutInflater();
```

第二种实现方式:

```
public static LayoutInflater from(Context context) {
    LayoutInflater LayoutInflater =(LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
```

数组适配器 针对于ListView的应用

1、ArrayAdapter

需要定义一个数组，布局文件item及TextView组件

```
String[] array = new String[]{"张三","李四","王麦子","test"};
```

将数组中的数据在TextView组件中显示，TextView组件填充到item布局中，将布局显示到屏幕上。

ArrayAdapter<String> adapter = new ArrayAdapter<String(this,R.layout.item1,R.id.tx1,array);

2、SimpleAdapter

List<Map<String, String>> list = new ArrayList<Map<String,String>>();

Map<String, String> m3 = new HashMap<String, String>();

m3.put("name", "王五");

m3.put("pwd", "12345");

list.add(m3);

SimpleAdapter simpleAdapter = new SimpleAdapter(this, list, R.layout.item2, new String[]{"name","pwd"}, new int[]{R.id.user , R.id.pwd});

ListView lv = (ListView) findViewById(R.id.lv);

lv.setAdapter(simpleAdapter);

ListView显示数据库数据（将数据库中的数据显示在ListView中）

```
22      protected void onCreate(Bundle savedInstanceState) {
23          super.onCreate(savedInstanceState);
24          setContentView(R.layout.activity_main);
25          dbHelper = new DBHelpler(this, "userinfo");
26          //init();//数据初始化
27          ListView lv =(ListView) findViewById(R.id.lv);
28          lv.setAdapter(new MyAdapter());
29      }
30      public class MyAdapter extends BaseAdapter {
31          @Override
32          // count?
33          public int getCount() {
34              SQLiteDatabase db = dbHelper.getReadableDatabase();
35              Cursor cursor = db.rawQuery("select count(*) from user", null);
36              if (cursor.moveToNext()) {
37                  int count = cursor.getInt(0);
38                  db.close();
39                  return count;
40              } else {
41                  return 0;
42              }
43          }
44          @Override
45          public Object getItem(int position) {
46              return null;
47          }
48          @Override
49          public long getItemId(int position) {
50              return 0;
```

获取数据库中有多少条数据

```
    item布局文件中有两个TextView  一个用来保存数据库中的姓名一个用来保存数据库中的ID
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View v = null ;
        if(convertView == null){
            v = View.inflate(getApplicationContext(), R.layout.item,null);
        }else{
            v= convertView;
        }

        TextView name_txt = (TextView)v.findViewById(R.id.name);
        TextView money_txt = (TextView)v.findViewById(R.id.money);
        //填充数据开始
        SQLiteDatabase db = dbHelper.getReadableDatabase();
        Cursor query = db.query("user", null, null, null, null, null, null);
        //移动到指定位置
        if (query.moveToPosition(position)) {
            name_txt.setText("name: " + query.getString(0));
            money_txt.setText("money: " + query.getInt(1));
        }
        db.close();
        //数据填充结束
        return v;
    }
}
private void init() {
    SQLiteDatabase db = dbHelper.getReadableDatabase();
    db.execSQL("insert into user values('张三',10000)");
    db.execSQL("insert into user values('李四',0)");
    db.close();
}
```