



北京化工大学

BEIJING UNIVERSITY OF CHEMICAL TECHNOLOGY

Android Development Practice Course

---

## LitePal 数据库

---

计科 1701-2017040271-赵振山

### 目录

第 1 章 LitePal 数据库	1
1.1 简介	1
1.2 实现过程	1
1.2.1 配置 LitePal	1
1.2.2 创建和升级数据库	2
1.2.3 添加数据	4
1.2.4 查询数据	6
1.2.5 删除数据	7
1.2.6 更新数据	8
1.3 实例总结	9
1.4 总结	14

创建日期：2020 年 7 月 10 日

更新日期：2020 年 7 月 14 日

# 第 1 章 LitePal 数据库

## 1.1 简介

LitePal 是一款开源的 Android 数据库框架，采用对象关系映射（ORM）模式，将常用的数据库功能进行封装，可以不用写一行 SQL 语句就可以完成创建表、增删改查的操作。

## 1.2 实现过程

### 1.2.1 配置 LitePal

第一步: 编辑 app/build.gradle(注意这里是 app 目录下的 build.gradle 文件) 添加代码如下:

```
dependencies {  
    implementation 'org.litepal.guolindev:core:3.1.1'  
}
```

该步骤完成后在 Android Studio 右上方有一个 Syn, 点击”Syn” 即可安装 jar 文件

第二步: 配置 litepal.xml 文件。右击 app/src/main 目录->New->Directory, 文件夹名字为 assets, 在 assets 目录下创建 litepal.xml 文件, 接着编辑该文件内容如下所示:

```
<?xml version="1.0" encoding="utf-8"?>  
<litepal>  
    <dbname value="demo" />  
    <version value="1" />  
    <list>  
    </list>  
</litepal>
```

其中 <dbname> 标签用于指定数据库名字, <version> 标签用于指定数据库的版本号,<list> 标签用于指定所有的映射模型

第三步: 配置 AndroidManifest.xml 清单文件. 在该清单文件下加入如下内容:

```
<manifest>  
    <application  
        android:name="org.litepal.LitePalApplication"  
        ...  
    >  
        ...  
    </application>  
</manifest>
```

### 1.2.2 创建和升级数据库

第一步：创建 Song 类。为测试创建数据库成功，在布局文件 activity-main.xml 文件中添加一个按钮。因为 LitePal 是使用面向对象的思维来使用数据库，下面我们在与 MainActivity 并行的位置定义一个 Song 类并且继承 LitePalSupport 类，代码如下：

```
package com.example.litepal;

import org.litepal.crud.LitePalSupport;

public class Song extends LitePalSupport {

    private String name;

    private int duration;

    public void setName(String s)
    {
        this.name=s;
    }

    public String getName()
    {
        return name;
    }

    public void setDuration(int s)
    {
        this.duration=s;
    }

    public int getDuration()
    {
        return duration;
    }

}
```

这是我们定义的一个 Song 类，里面有两个参数分别是 String 类型的 name 和 int 类型的 duration。当我们创建完成数据库的时候，数据库会自动的生成一个 Song 表，表中有两列，分别是 name 和 duration，而对应的 getName 和 setName 是获取 name 和设置 name 的方法。当你定义了 getName 和 setName 方法，系统就会自动生成 getter 和 setter 方法，如果不懂也没有关系，你只需要知道 Song 类会对应数据库中的一个 Song 表，Song 表有 name 和 duration 两项，我们还必须在 Song 类中实现对应的 getName 和 setName 方法，这样程序就不会出错啦。

第二步: 修改 litepal.xml 文件. 我们已经创建了 Song 类, 并且我们知道 Song 类会对应数据库中的一个表, 那么我们如何将 Song 类和数据库中的表建立起一个对应关系呢? 来看下面的代码:

```
<?xml version="1.0" encoding="utf-8"?>
<litepal>
    <dbname value="demo" />
    <version value="1" />
    <list>
        <mapping class="com.example.litepal.Song" />
    </list>
</litepal>
```

这里一定注意上面新增的代码一定要与 Song 类所在的位置对应好

第三步: 创建数据库. 修改 MainActivity 中的代码如下:

```
setContentView(R.layout.activity_main);
Button createDatabase = findViewById(R.id.create_database);
createDatabase.setOnClickListener(new View.OnClickListener(){
    public void onClick(View v)
        LitePal.getDatabase();
});
```

当你执行程序并且点击按钮时, 数据库就创建好了

第四步: 升级数据库. 如果你想在 Song 表中再加一个字段 price 怎么办呢? 很简单, 只需要在 Song 类中添加如下代码 (一共添加一个数据元素以及两个方法):

```
package com.example.litepal;

import org.litepal.crud.LitePalSupport;

public class Song extends LitePalSupport {

    private double price;

    public void setPrice(double s)
    {
        this.price=s;
    }

    public double getPrice()
    {
        return price;
    }
}
```

```
}
```

如果你想在数据库中再添加一个表该怎么办呢？同理，去创建一个新的类，然后这个类同上面的 Song 类创建的方法类似，不要忘记在 litepal.xml 中把你新创建的类进行和数据库的一一对应就可以啦。当一切想要改的数据改好之后，我们把 litepal.xml 中的版本号 version 自增 1 就可以，比如上面我们的 version value=1，我们改成 version value=2 再运行即可

我们的数据库已经创建好了，实际效果图如下 (demo.db)：

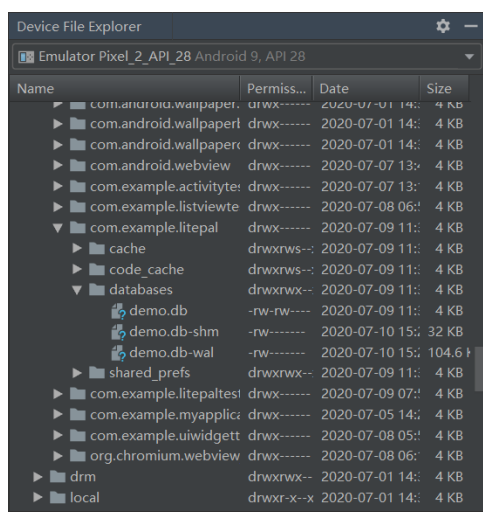


图 1.1: database

我们可以去打开这个数据库，如图所示，Song 表有两个表项，分别是我们在上面在 Song 类中定义的名字和 duration：

对象	album @main (litepal) - 表
	id duration name
	1 320 song1
	2 356 song2
	3 320 song1
	4 356 song2
	5 320 song1
	6 356 song2
	7 320 song1
	8 356 song2

图 1.2: table

### 1.2.3 添加数据

使用 Litepal 添加数据十分的简单，给人一种舒适的感觉。我们已经知道数据库中的 Song 表和我们前面定义的 Song 类是一一对应的，所以我们只需要在我们的 MainActivity 中新建一个 Song 对象，然后对该对象进行初始化，最后使用 save() 函数即可实现在数据库表中添加数据。MainActivity 中的代码如下图所示：

```
package com.example.litepal;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;

import org.litepal.LitePal;

import java.util.List;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button adddata = findViewById(R.id.add_data);
        adddata.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                Song song1 = new Song();
                song1.setName("song1");
                song1.setDuration(320);
                song1.save();

                Song song2 = new Song();
                song2.setName("song2");
                song2.setDuration(356);
                song2.save();
            }
        });
    }
}
```

这个 `save()` 方法是从哪里来的呢？还记得上面定义 `Song` 类继承了 `LitePalSupport` 类，那么这个 `save()` 方法就是该类中的方法，当你运行程序并且点击添加按钮时，`song1` 和 `song2` 就会被添加到数据库中了，我们来看一下实际效果。



图 1.3: 运行截图

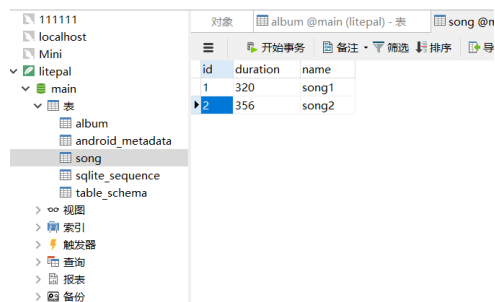


图 1.4: 效果截图

## 1.2.4 查询数据

使用 litepal 查询数据一共有三种方式。

第一种方式: 查询表中所有信息

```
List<Song> allSongs = LitePal.findAll(Song.class);
```

第二种方式: 根据 id 查询表中某一行

```
Song song = LitePal.find(Song.class, id);
```

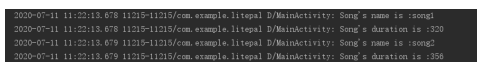
第三种方式: 条件查询

```
List<Song> songs = LitePal.where("name like ? and duration < ?", "song%", "200").order("duration").find(Song.class);
```

为查看运行效果，我在 MainActivity 设置了一个按钮，用来查询 Song 表中的所有信息代码如下：

```
Button lookdata = findViewById(R.id.query_data);
lookdata.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        List<Song> allSongs = LitePal.findAll(Song.class);
        for(Song song:allSongs){
            Log.d("MainActivity", "Song's name is :"+song.getName());
            Log.d("MainActivity", "Song's duration is :"+song.getDuration());
        }
    }
});
```

运行效果我是利用了 Log 函数，用来查看运行结果，下面是结果截图：



```
2020-07-11 11:22:13.678 11215-11215/com.example.litepal D/MainActivity: Song's name is :song1
2020-07-11 11:22:13.678 11215-11215/com.example.litepal D/MainActivity: Song's duration is :320
2020-07-11 11:22:13.679 11215-11215/com.example.litepal D/MainActivity: Song's name is :song2
2020-07-11 11:22:13.679 11215-11215/com.example.litepal D/MainActivity: Song's duration is :356
```

图 1.5: 查询结果

### 1.2.5 删除数据

使用 LitePal 删除数据有两种方法。

第一种方法: 删除表中的所有数据

```
LitePal.deleteAll(Song.class);
```

第二种方法: 根据条件删除表格中的数据

```
LitePal.deleteAll(Song.class, "duration > ?", "320");
//这一行也很简单，这行代码的意思是删除表格中duration>320的数据。
```

测试代码如下：

```
Button deletedata = findViewById(R.id.delete_data);
deletedata.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // LitePal.deleteAll(Song.class);
        LitePal.deleteAll(Song.class, "duration > ?", "320");
    }
});
```

运行结果截图：



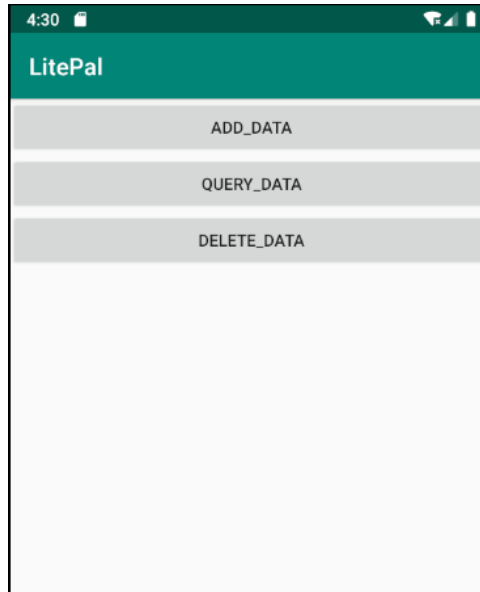


图 1.6: 运行界面

```
I/System.out: name:First duration:320  
name:Second duration:356
```

图 1.7: 原始表

```
I/System.out: name:First duration:320
```

图 1.8: 删除后的表

### 1.2.6 更新数据

LitePal 数据库更新数据一共有以下几种方法，下面进行依次介绍。

第一种方法：这个是最简单的一种方法，首先通过 `find()` 函数来获取数据库表中的一行数据，也就是得到一个对象，然后对该对象进行数据变更操作，最后再使用 `save()` 函数保存即可。

```
Album albumToUpdate = LitePal.find(Album.class, 1);  
albumToUpdate.setPrice(20.99f); // raise the price  
albumToUpdate.save();
```

第二种方法: 通过 id 更新数据库中的数据

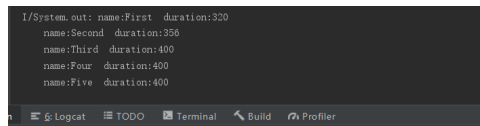
```
Album albumToUpdate = new Album();  
albumToUpdate.setPrice(20.99f); // raise the price  
albumToUpdate.update(id);
```

第三种方法: 这种方法是最常用的方法，即根据条件变更数据

```
Album albumToUpdate = new Album();
albumToUpdate.setPrice(20.99f); // raise the price
albumToUpdate.updateAll("name = ?", "album");
```

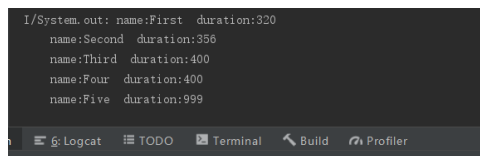
下面给出我的测试代码:

```
Button update = findViewById(R.id.update_data);
update.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Song newsong = new Song();
        newsong.setDuration(999);
        newsong.updateAll("name = ?", "Five");
    }
});
```



```
I/System.out: name:First duration:320
name:Second duration:356
name:Third duration:400
name:Four duration:400
name:Five duration:400
```

图 1.9: 变更前的表



```
I/System.out: name:First duration:320
name:Second duration:356
name:Third duration:400
name:Four duration:400
name:Five duration:999
```

图 1.10: 变更后的表

## 1.3 实例总结

在这里我把我实验使用的代码展示出来, 并且通过 ListView 来展示数据库中的内容

第一个文件: 类 Song

```
package com.example.litepal;

import org.litepal.crud.LitePalSupport;

public class Song extends LitePalSupport {

    private String name;

    private int duration;

    public void setName(String s)
```

```
{
    this.name=s;
}

public String getName()
{
    return name;
}

public void setDuration(int s)
{
    this.duration=s;
}

public int getDuration()
{
    return duration;
}
}
```

第二个文件:ListView 针对 Song 类定义的适配器

```
package com.example.litepal;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.TextView;

import java.util.List;

public class SongAdapter extends ArrayAdapter<Song> {
    private int resourceId;

    public SongAdapter(Context context, int textViewResourceId, List<Song> Objects) {
        super(context, textViewResourceId, Objects);
        resourceId = textViewResourceId;
    }

    public View getView(int position, View convertView, ViewGroup parent)
    {
        Song song = getItem(position);
        View view = LayoutInflater.from(getContext()).inflate(resourceId,parent,false);
        TextView songname=view.findViewById(R.id.song_name);
        songname.setText(song.getName());
        return view;
    }
}
```

第三个文件: 数据库需要使用的 litepal.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<litepal>
    <dbname value="demo" />

    <version value="1" />

    <list>
        <mapping class="com.example.litepal.Album" />
        <mapping class="com.example.litepal.Song" />
    </list>

    -->
</litepal>
```

#### 第四个文件: 类 MainActivity

```
package com.example.litepal;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ListView;

import org.litepal.LitePal;

import java.util.List;

public class MainActivity extends AppCompatActivity {
    private List<Song> SongList= LitePal.findAll(Song.class);

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button adddata = findViewById(R.id.add_data);
        adddata.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                Song song1 = new Song();
                song1.setName("First");
                song1.setDuration(320);
                song1.save();
            }
        });
    }
}
```

```
        Song song2 = new Song();
        song2.setName("Second");
        song2.setDuration(356);
        song2.save();

        Song song3 = new Song();
        song3.setName("Third");
        song3.setDuration(400);
        song3.save();

        Song song4 = new Song();
        song4.setName("Four");
        song4.setDuration(400);
        song4.save();

        Song song5 = new Song();
        song5.setName("Five");
        song5.setDuration(400);
        song5.save();
    }
});
Button lookdata = findViewById(R.id.query_data);
lookdata.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        List<Song> allSongs = LitePal.findAll(Song.class);
        for(Song song:allSongs){
            System.out.println("name:"+song.getName()+" duration:"+song.getDuration());
        }
    }
});
Button deletedata = findViewById(R.id.delete_data);
deletedata.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        LitePal.deleteAll(Song.class);
    }
});
Button update = findViewById(R.id.update_data);
update.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Song newsong = new Song();
        newsong.setDuration(999);
        newsong.updateAll("name = ?", "Five");
    }
});
```

```

        SongAdapter adapter = new SongAdapter(MainActivity.this,R.layout.song_item,SongList)
        ;
        ListView listview = findViewById(R.id.list_view);
        listview.setAdapter(adapter);

    }
}

```

第五个文件:activitymain.xml 布局文件

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/add_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="add_data"
    />
    <Button
        android:id="@+id/query_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="query_data"
    />
    <Button
        android:id="@+id/delete_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="delete_data"
    />
    <Button
        android:id="@+id/update_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="update_data"
    />
    <ListView
        android:id="@+id/list_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>

```

第六个文件: songitem.xml 文件

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

```

```
android:layout_height="match_parent"
android:layout_width="wrap_content">

<TextView
    android:id="@+id/song_name"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:layout_marginLeft="10dp" />
</LinearLayout>
```

下面是运行效果截图: 分别是四个按钮, 以及一个展示数据库 Song 表的 ListView

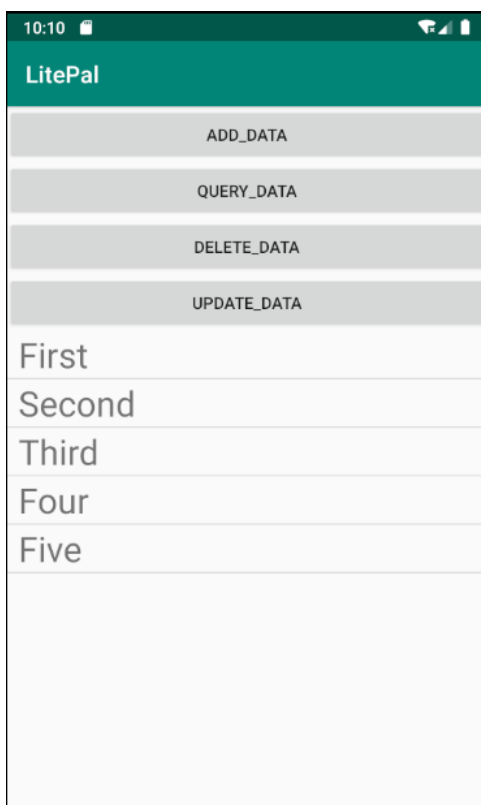


图 1.11: 效果实验效果

## 1.4 总结

LitePal 是一款开源的 Android 数据库框架, 采用对象关系映射 (ORM) 模式, 将常用的数据库功能进行封装, 可以不用写一行 SQL 语句就可以完成创建表、增删改查的操作。总的来说 LitePal 框架功能就是将自定义的类自动转换成内置数据库中的表, 除此之外, 它还把对数据库的原生语言操作变为了更简单的增删改查操作, LitePal 是一个容易掌握, 简洁轻便的 Android 内置数据库框架, 推荐使用

## 参考文献

- [1] 官方文档. <https://github.com/guolindev/LitePal>, 2020.
- [2] 《第一行代码》< 郭霖 >.