Android数据库高手秘籍(7)：体验LitePal的查询艺术

2015-09-24 安卓应用频道

(点击上方公众号，可快速关注)

> 来源：郭霖
> 链接：http://blog.csdn.net/guolin_blog/article/details/40153833

本系列：
Android数据库高手秘籍(1)：SQLite命令
Android数据库高手秘籍(2)：创建表和LitePal的基本用法
Android数据库高手秘籍(3)：使用LitePal升级表
Android数据库高手秘籍(4)：使用LitePal建立表关联
Android数据库高手秘籍(5)：LitePal的存储操作
Android数据库高手秘籍(6)：LitePal的修改和删除操作

经过了多篇文章的学习，我们已经把LitePal中的绝大部分内容都掌握了。现在回想起来了，增删改查四种操作中的前三种我们都已经学完了，不知道现在使用起数据库来，你有没有感觉到格外的轻松和简单。但是呢，我们都知道，在所有的数据库操作当中，查询操作肯定是最复杂的，用法也是最多的，因此LitePal在查询方面提供的API也是比较丰富，而且LitePal在查询方面的API设计也是颇为艺术的。那么今天我们就专门使用一篇博客来讲解一下查询操作的用法，体验一下LitePal查询的艺术。还没有看过前面一篇文章的朋友建议先去参考Android数据库高手秘籍(6)：LitePal的修改和删除操作 。

LitePal的项目地址是：https://github.com/LitePalFramework/LitePal

传统的查询数据方式

其实最传统的查询数据的方式当然是使用SQL语句了，Android当中也提供了直接使用原生SQL语句来查询数据库表的方法，即SQLiteDatabase中的rawQuery()方法，方法定义如下：

```
public Cursor rawQuery(String sql, String[] selectionArgs)
```

其中，rawQuery()方法接收两个参数，第一个参数接收的就是一个SQL字符串，第二个参数是用于替换SQL语句中占位符（？）的字符串数组。rawQuery()方法返回一个Cursor对象，所有查询到的数据都是封闭在这个对象当中的，我们只要一一取出就可以了。

当然这种用法其实并不是很常用，因为相信大多数人都还是不喜欢编写SQL语句的。所以，Android专门提供了一种封装好的API，使得我们不用编写SQL语句也能查询出数据，即SQLiteDatabase中的query()方法。query()提供了三个方法重载，其中参数最少的一个也有七个参数，我们来看下方法定义：

```
public Cursor query(String table, String[] columns, String selection,
        String[] selectionArgs, String groupBy, String having,
        String orderBy)
```

其中第一参数是表名，表示我们希望从哪张表中查询数据。第二个参数用于指定去查询哪几列，如果不指定则默认查询所有列。第三、第四个参数用于去约束查询某一行或某几行的数据，不指定则默认是查询所有行的数据。第五个参数用于指定需要去group by的列，不指定则表示不对查询结果进行group by操作。第六个参数用于对group by之后的数据进行进一步的过滤，不指定则表示不进行过滤。第七个参数用于指定查询结果的排序方式，不指定则表示使用默认的排序方式。

这个方法是query()方法最少的一个方法重载了，另外还有两个方法重载分别是八个和九个参数。虽说这个方法在Android数据库表查询的时候非常常用，但重多的参数让我们在理解这个方法的时候可能会很费力，另外使用起来的时候也会相当的不爽。比如说，我们想查询news表中的所有数据，就应该要这样写：

```
SQLiteDatabase db = dbHelper.getWritableDatabase();
Cursor cursor = db.query("news", null, null, null, null, null, null);
```

可以看到，将第一个表名参数指定成news，然后后面的六个参数我们都用不到，就全部指定成null。
那如果是我们想查询news表中所有评论数大于零的新闻该怎么写呢？代码如下所示：

```
SQLiteDatabase db = dbHelper.getWritableDatabase();
Cursor cursor = db.query("news", null, "commentcount>?", new String[]{"0"}, null, null, null);
```

由于第三和第四个参数是用于指定约束条件的，所以我们在第三个参数中指明了commentcount>?，然后在第四个参数中通过一个String数组来替换占位符，这样查到的结果就是news表中所有评论数大于零的新闻了。那么其它的几个参数呢？仍然用不到，所以还是只能传null。

然后我们可以看到，query()方法的返回值是一个Cursor对象，所有查询到的数据都是封装在这个对象中的，所以我们还需要将数据逐一从Cursor对象中取出，然后设置到News实体类当中，如下所示：

```
List<News> newsList = new ArrayList<News>();
if (cursor != null && cursor.moveToFirst()) {
    do {
        int id = cursor.getInt(cursor.getColumnIndex("id"));
        String title = cursor.getString(cursor.getColumnIndex("title"));
        String content = cursor.getString(cursor.getColumnIndex("content"));
        Date publishDate = new Date(cursor.getLong(cursor.getColumnIndex("publishdate")));
        int commentCount = cursor.getInt(cursor.getColumnIndex("commentcount"));
        News news = new News();
        news.setId(id);
        news.setTitle(title);
        news.setContent(content);
        news.setPublishDate(publishDate);
        news.setCommentCount(commentCount);
        newsList.add(news);
    } while (cursor.moveToNext());
}
```

这大概就是传统查询数据方式的用法了，总体来看，用法确实非常不友好，尤其是query()方法冗长的参数列表，即使我们用不到那些参数，也必须要传入许多个null。另外，查询到的数据还都只是封装到了一个Cursor对象中，我们还需要将数据一一取出然后再set到实体类对象当中。麻烦吗？可能你觉得不麻烦，因为你已经习惯了这种用法。但是习惯总是可以改变的，也许当你体验了LitePal中查询API给我们带来的便利之后，就会有了新的看法了，那么下面我们就一起来体验一下LitePal的查询艺术。

使用LitePal查询数据

LitePal在查询方面提供了非常丰富的API，功能多种多样，基本上已经能够满足我们平时所有的查询需求了。不仅如此，LitePal在查询API的设计方面也是非常用心，摒弃了原生query()方法中繁琐的参数列表，而是改用了一种更为灵巧的方式——连缀查询。除此之外，LitePal查询的结果也不再返回Cursor对象，然后再由开发者自己去逐个取出，而是直接返回封装好的对象。这些改变都使得查询数据变得更加简单，也更加合理，那么下面我们就来完整地学习一下LitePal中查询数据的所有用法。

简单查询

比如说现在我们想实现一个最简单的功能，查询news表中id为1的这条记录，使用LitePal就可以这样写：

```
News news = DataSupport.find(News.class, 1);
```

天呐！有没有觉得太轻松了？仅仅一行代码，就可以把news表中id为1的记录查出来了，而且结果还是自动封装到News对象里的，也不需要我们手动再从Cursor中去解析。如果是用原生的SQL语句，或者query()方法来写，至少要20行左右的代码才能完成同样的功能！

那我们先冷静一下，来分析分析这个find()方法。可以看到，它的参数列表也比较简单，只接收两个参数，第一个参数是一个泛型类，也就是说我们在这里指定什么类，返回的对象就是什么类，所以这里传入News.class，那么返回的对象也就是News了。第二个参数就更简单了，就是一个id值，如果想要查询id为1的记录就传1，想查id为2的记录就传2，以此类推。

本来一个还算颇为复杂的功能，通过LitePal之后就变得这么简单了！那么你可能已经迫不及待地想要学习更多LitePal中更多的查询用法了，别着急，我们一个个来看。

你也许遇到过以下场景，在某些情况下，你需要取出表中的第一条数据，那么传统的做法是怎么样的呢？在SQL语句中指定一个limit值，然后获取返回结果的第一条记录。但是在LitePal中不用这么麻烦，比如我们想要获取news表中的第一条数据，只需要这样写：

```
News firstNews = DataSupport.findFirst(News.class);
```

OK，语义性非常强吧，让人一眼就看懂是什么意思了，只需调用findFirst()方法，然后传入News类，得到的就是news表中的第一条数据了。

那我们举一翻三，如果是想要获取News表中的最后一条数据该怎么写呢？同样简单，如下所示：

```
News lastNews = DataSupport.findLast(News.class);
```

因为获取表中第一条或者是最后一条数据的场景比较常见，所以LitePal特意提供了这两个方法来方便我们的操作。

那么我们看到这里，目前都只是查询单条数据的功能，如果想要查询多条数据该怎么办呢？比如说，我们想把news表中id为1、3、5、7的数据都查出来，该怎么写呢？也许有的朋友会比较聪明，立马就想到可以一个个去查，就调用四次find()方法嘛，然后把1、3、5、7这四个id分别传进去不就可以了。没错，这样做完全是可以的，而且效率也并不低，但是LitePal给我们提供了一个更简便的方法——findAll()。这个方法的用法和find()方法是非常类似的，只不过它可以指定多个id，并且返回值也不再是一个泛型类对象，而是一个泛型类集

合，如下所示：

```
List<News> newsList = DataSupport.findAll(News.class, 1, 3, 5, 7);
```

可以看到，首先我们是调用的findAll()方法，然后这个方法的第一个参数仍然是指定的泛型类，但是后面的参数就很随意了，你可以传入任意个id进去，findAll()方法会把所有传入的id所对应的数据全部查出来，然后一起返回到List<News>这个泛型集合当中。

虽说这个语法设计算是相当人性化，但是在有些场景或许不太适用，因为可能要你要查询的多个id已经封装到一个数组里了。那么没关系，findAll()方法也是接收数组参数的，所以说同样的功能你也可以这样写：

```
long[] ids = new long[] { 1, 3, 5, 7 };
List<News> newsList = DataSupport.findAll(News.class, ids);
```

看到这里，那有的朋友可能会奇怪了，说findAll()方法不应该是查询所有数据的意思吗？怎么总是查询几个id所对应数据呢？哈！这个问题问得好，因为findAll()方法也是可以查询所有数据的，而且查询所有数据的写法更简单，只需要这样写：

```
List<News> allNews = DataSupport.findAll(News.class);
```

看到没有，我们只需要把后面的参数都去掉，在不指定具体id的情况下，findAll()方法查询出的就是news表中的所有数据了，是不是语义性非常强？

而且大家不要以为刚才这些都只是findAll()的几个方法重载而已，实际上刚才我们的这几种用法都是调用的同一个findAll()方法！一个方法却能够实现多种不同的查询效果，并且语义性也很强，让人一看就能理解，这就是LitePal的查询艺术！

连缀查询

当然了，LitePal给我们提供的查询功能还远远不只这些，好戏还在后头。相信大家现在也已经发现了，我们目前的查询功能都是基于id来进行查询的，并不能随意地指定查询条件。那么怎样才能指定查询条件呢？让我们回想一下传统情况应该怎么做，query()方法中接收七个参数，其中第三和第四个参数就是用于指定查询条件的，然后其它几个参数都填null就可以了。但是呢，前面我们已经痛批过了这种写法，因为冗长的参数列表太过繁琐，那么LitePal又是怎么解决这个问题的呢？我们现在就来学习一下。

为了避免冗长的参数列表，LitePal采用了一种非常巧妙的解决方案，叫作连缀查询，这种查询很灵活，可以根据我们实际的查询需求来动态配置查询参数。 那这里举个简单的例子，

比如我们想查询news表中所有评论数大于零的新闻，就可以这样写：

```
List<News> newsList = DataSupport.where("commentcount > ?", "0").find(News.class);
```

可以看到，首先是调用了DataSupport的where()方法，在这里指定了查询条件。where()方法接收任意个字符串参数，其中第一个参数用于进行条件约束，从第二个参数开始，都是用于替换第一个参数中的占位符的。那这个where()方法就对应了一条SQL语句中的where部分。

接着我们在where()方法之后直接连缀了一个find()方法，然后在这里指定一个泛型类，表示用于查询哪张表。那么上面的一段代码，查询出的结果和如下SQL语句是相同的：

```
select * from users where commentcount > 0;
```

但是这样会将news表中所有的列都查询出来，也许你并不需要那么多的数据，而是只要title和content这两列数据。那么也很简单，我们只要再增加一个连缀就行了，如下所示：

```
List<News> newsList = DataSupport.select("title", "content")
        .where("commentcount > ?", "0").find(News.class);
```

可以看到，这里我们新增了一个select()方法，这个方法接收任意个字符串参数，每个参数要求对应一个列名，这样就只会把相应列的数据查询出来了，因此select()方法对应了一条SQL语句中的select部分。

那么上面的一段代码，查询出的结果和如下SQL语句是相同的：

```
select title,content from users where commentcount > 0;
```

很好玩吧？不过这还不算完呢，我们还可以继续连缀更多的东西。比如说，我希望将查询出的新闻按照发布的时间倒序排列，即最新发布的新闻放在最前面，那就可以这样写：

```
List<News> newsList = DataSupport.select("title", "content")
        .where("commentcount > ?", "0")
        .order("publishdate desc").find(News.class);
```

order()方法中接收一个字符串参数，用于指定查询出的结果按照哪一列进行排序，asc表示正序排序，desc表示倒序排序，因此order()方法对应了一条SQL语句中的order by部分。

那么上面的一段代码，查询出的结果和如下SQL语句是相同的：

```
select title,content from users where commentcount > 0 order by publishdate desc;
```

然后呢，也许你并不希望将所有条件匹配的结果一次性全部查询出来，因为这样数据量可能会有点太大了，而是希望只查询出前10条数据，那么使用连缀同样可以轻松解决这个问题，代码如下所示：

```
List<News> newsList = DataSupport.select("title", "content")
        .where("commentcount > ?", "0")
        .order("publishdate desc").limit(10).find(News.class);
```

这里我们又连缀了一个limit()方法，这个方法接收一个整型参数，用于指定查询前几条数据，这里指定成10，意思就是查询所有匹配结果中的前10条数据。

那么上面的一段代码，查询出的结果和如下SQL语句是相同的：

```
select title,content from users where commentcount > 0 order by publishdate desc limit 10;
```

刚才我们查询到的是所有匹配条件的前10条新闻，那么现在我想对新闻进行分页展示，翻到第二页时，展示第11到第20条新闻，这又该怎么实现呢？没关系，在LitePal的帮助下，这些功能都是十分简单的，只需要再连缀一个偏移量就可以了，如下所示：

```
List<News> newsList = DataSupport.select("title", "content")
        .where("commentcount > ?", "0")
        .order("publishdate desc").limit(10).offset(10)
        .find(News.class);
```

可以看到，这里我们又添加了一个offset()方法，用于指定查询结果的偏移量，这里指定成10，就表示偏移十个位置，那么原来是查询前10条新闻的，偏移了十个位置之后，就变成了查询第11到第20条新闻了，如果偏移量是20，那就表示查询第21到第30条新闻，以此类推。因此，limit()方法和order()方法共同对应了一条SQL语句中的limit部分。

那么上面的一段代码，查询出的结果和如下SQL语句是相同的：

```
select title,content from users where commentcount > 0 order by publishdate desc limit 10,10;
```

这大概就是LitePal中连缀查询的所有用法了。看出区别了吧？这种查询的好处就在于，我们可以随意地组合各种查询参数，需要用到的时候就把它们连缀到一起，不需要用到的时候不用指定就可以了。对比一下query()方法中那冗长的参数列表，即使我们用不到那些参数，也

必须要传null，是不是明显感觉LitePal中的查询更加人性化？

激进查询

不过，上述我们的所有用法中，都只能是查询到指定表中的数据而已，关联表中数据是无法查到的，因为LitePal默认的模式就是懒查询，当然这也是推荐的查询方式。那么，如果你真的非常想要一次性将关联表中的数据也一起查询出来，当然也是可以的，LitePal中也支持激进查询的方式，下面我们就来一起看一下。

不知道你有没有发现，刚才我们所学的每一个类型的find()方法，都对应了一个带有isEager参数的方法重载，这个参数相信大家一看就明白是什么意思了，设置成true就表示激进查询，这样就会把关联表中的数据一起查询出来了。

比如说，我们想要查询news表中id为1的新闻，并且把这条新闻所对应的评论也一起查询出来，就可以这样写：

```
News news = DataSupport.find(News.class, 1, true);
List<Comment> commentList = news.getCommentList();
```

可以看到，这里并没有什么复杂的用法，也就是在find()方法的最后多加了一个true参数，就表示使用激进查询了。这会将和news表关联的所有表中的数据也一起查出来，那么comment表和news表是多对一的关联，所以使用激进查询一条新闻的时候，那么该新闻所对应的评论也就一起被查询出来了。

激进查询的用法非常简单，就只有这么多，其它find()方法也都是同样的用法，就不再重复介绍了。但是这种查询方式LitePal并不推荐，因为如果一旦关联表中的数据很多，查询速度可能就会非常慢。而且激进查询只能查询出指定表的关联表数据，但是没法继续迭代查询关联表的关联表数据。因此，这里我建议大家还是使用默认的懒加载更加合适，至于如何查询出关联表中的数据，其实只需要在模型类中做一点小修改就可以了。修改News类中的代码，如下所示：

```
public class News extends DataSupport{

    ...

    public List<Comment> getComments() {
        return DataSupport.where("news_id = ?", String.valueOf(id)).find(Comment.class);
    }
```

```
}
```

可以看到，我们在News类中添加了一个getComments()方法，而这个方法的内部就是使用了一句连缀查询，查出了当前这条新闻对应的所有评论。改成这种写法之后，我们就可以将关联表数据的查询延迟，当我们需要去获取新闻所对应的评论时，再去调用News的getComments()方法，这时才会去查询关联数据。这种写法会比激进查询更加高效也更加合理。

原生查询

相信你已经体会到，LitePal在查询方面提供的API已经相当丰富了。但是，也许你总会遇到一些千奇百怪的需求，可能使用LitePal提供的查询API无法完成这些需求。没有关系，因为即使使用了LitePal，你仍然可以使用原生的查询方式(SQL语句)来去查询数据。DataSuppport类中还提供了一个findBySQL()方法，使用这个方法就能通过原生的SQL语句方式来查询数据了，如下所示：

```
Cursor cursor = DataSupport.findBySQL("select * from news where commentcount>?", "0");
```

findBySQL()方法接收任意个字符串参数，其中第一个参数就是SQL语句，后面的参数都是用于替换SQL语句中的占位符的，用法非常简单。另外，findBySQL()方法返回的是一个Cursor对象，这和原生SQL语句的用法返回的结果也是相同的。

好了，这样我们就把LitePal中提供的查询数据的方法全部都学完了，那么今天的文章就到这里，下一篇文章当中会开始讲解聚合函数的用法，感兴趣的朋友请继续阅读 Android数据库高手秘籍(8)：使用LitePal的聚合函数 。

LitePal开源项目地址：https://github.com/LitePalFramework/LitePal