Android数据库高手秘籍(5)：LitePal的存储操作

2015-09-09 安卓应用频道

(点击上方蓝字，可快速关注我们)

> 来源：郭霖
> 网址：http://blog.csdn.net/guolin_blog/article/details/39345833

经过前面几篇文章的学习，我们已经把LitePal的表管理模块的功能都很好地掌握了，相信大家都已经体会到了使用LitePal来创建表、升级表、以及建立表关联所带来的便利。那么从本篇文章开始，我们将进入到一个新模块的学习旅程当中，使用LitePal来进行表的CRUD操作。还没有看过前一篇文章的朋友建议先去参考 Android数据库高手秘籍(4)：使用LitePal建立表关联 。

LitePal提供的CRUD操作的API还是颇为丰富的，一篇文章肯定是介绍不全的，因此这里我们仍然是分几篇文章进行讲解，本篇主要是介绍存储方面的API。

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

传统的存储数据方式

其实最传统的存储数据方式肯定是通过SQL语句拼接字符串来进行存储的，不过这种方式有点过于“传统”了，今天我们在这里就不讨论这种情况。实际上，Android专门提供了一种用于存储数据的简便方法，使得我们不用编写SQL语句就可以执行存储操作。下面来看一下SQLiteDatabase中的insert()方法：

```
public long insert(String table, String nullColumnHack, ContentValues values)
```

可以看到，insert方法接收三个参数，第一个参数是表名，第二个参数通常都用不到，直接传null，第三个参数则是一个封装了待存储数据的ContentValues对象。因此，比如说我们想往news表中插入一条新闻，就可以这样写：

```
SQLiteDatabase db = dbHelper.getWritableDatabase();
ContentValues values = new ContentValues();
values.put("title", "这是一条新闻标题");
values.put("content", "这是一条新闻内容");
values.put("publishdate", System.currentTimeMillis());
```

```
long id = db.insert("news", null, values);
```

其中，调用ContentValues的put()方法来添加待存储数据，put()方法接收两个参数，第一个参数是数据库表中对应的列名，第二个参数就是要存储的值，最后调用一下insert()方法，这条新闻就会插入到news表当中了，并且该数据行对应的id会作为返回值进行返回。

用法很简单是吗？确实，比起直接使用SQL语句，SQLiteDatabase中提供的insert()方法的确简单了很多。但insert()方法也并非是那么的完美，它还是有很多不方便的地方的，比如说没有考虑表关联的情况，我们需要手动对关联表的外键进行存储。再比如说，没有提供批量存储的功能，当我们有一个集合的数据需要存储时，需要通过循环来遍历这个集合，然后一次次地调用insert()方法来插入数据。

好了，那么关于传统存储数据的用法就简单介绍到这里，因为确实没什么的更多的用法了，并且它也不是我们今天的主角。接下来，就让我们看一看今天的惊喜，学习如何使用LitePal来进行数据库存储的操作。

使用LitePal存储数据

LitePal中与存储相关的API其实并不多，但用法还是颇为丰富的，而且比起传统的insert()方法，使用LitePal来存储数据可以简单到让你惊叹的地步，那么今天我们就来完整地学习一下LitePal存储数据的所有用法。

在前面几篇文章当中，我们在项目里已经建好了News、Comment、Introduction、Category这几个实体类，通过这些实体类，LitePal就可以把相应的表自动创建出来。现在来观察这几个实体类，我们发现这几个类都是没有继承结构的。没错，因为LitePal进行表管理操作时不需要这些实体类有任何的继承结构，当时为了简单起见就没有写。但是进行CRUD操作时就不行了，LitePal要求所有的实体类都要继承自DataSupport这个类，因此这里我们就要把继承结构给加上才行。修改News类的代码，如下所示：

```
public class News extends DataSupport{

    ......

    // 自动生成get、set方法
}
```

可以看到，这里只是让News类继承自了DataSupport，其它什么都没有改变。另外几个Comment、Introduction、Category类也使用同样的改法，这里就不一一演示了。

继承了DataSupport类之后，这些实体类就拥有了进行CRUD操作的能力，那么比如想要存储一条数据到news表当中，就可以这样写：

```
News news = new News();
news.setTitle("这是一条新闻标题");
news.setContent("这是一条新闻内容");
news.setPublishDate(new Date());
news.save();
```

怎么样？是不是非常简单，不需要SQLiteDatabase，不需要ContentValues，不需要通过列名组装数据，甚至不需要指定表名，只需要new出一个News对象，然后把要存储的数据通过setter方法传入，最后调用一下save()方法就好了，而这个save()方法自然就是从DataSupport类中继承而来的了。

除此之外，save()方法还是有返回值的，我们可以根据返回值来判断存储是否成功，比如说这样写：

```
if (news.save()) {
    Toast.makeText(context, "存储成功", Toast.LENGTH_SHORT).show();
} else {
    Toast.makeText(context, "存储失败", Toast.LENGTH_SHORT).show();
}
```

可以看出，save()方法返回的是一个布尔值，用于表示存储成功还是失败，但同时也说明这个方法是不会抛出异常的。有些朋友希望如果存储失败的话就抛出异常，而不是返回一个false，那就可以使用saveThrows()方法来代替，如下所示：

```
News news = new News();
news.setTitle("这是一条新闻标题");
news.setContent("这是一条新闻内容");
news.setPublishDate(new Date());
news.saveThrows();
```

使用saveThrows()方法来存储数据，一旦存储失败就会抛出一个DataSupportException异常，我们可以通过对这个异常进行捕获来处理存储失败的情况。

那有些细心的朋友可能已经注意到，使用的insert()方法来存储数据时是有返回值的，返回的是插入行对应的id。但LitePal中的save()方法返回的是布尔值，那么我们怎样才能拿到存储成功之后这条数据对应的id呢？对此，LitePal使用了一种非常巧妙的做法，还记得我们在每

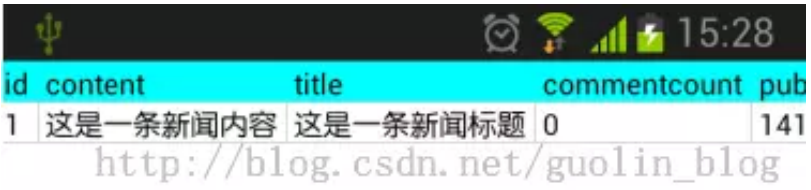
个实体类中都定义了一个id字段吗？当调用save()方法或saveThrows()方法存储成功之后，LitePal会自动将该条数据对应的id赋值到实体类的id字段上。让我们来做个试验吧，代码如下所示：

```
News news = new News();
news.setTitle("这是一条新闻标题");
news.setContent("这是一条新闻内容");
news.setPublishDate(new Date());
Log.d("TAG", "news id is " + news.getId());
news.save();
Log.d("TAG", "news id is " + news.getId());
```

在save之前打印一下news的id，在save之后再打印一次，现在运行一下，打印结果如下所示：

Application	Tag	Text
com.example.database...	TAG	news id is 0
com.example.database...	TAG	news id is 1

OK，在save之前打印的id是0，说明此时id这个字段还没有被赋值，在save之后打印的id是1，说明此时id已经被赋值了。那么我们再到数据库表中再查看一下这条记录到底有没有存储成功吧，如下图所示：



15:28

id	content	title	commentcount	pub
1	这是一条新闻内容	这是一条新闻标题	0	141

可以看到，这条新闻确实已经存储成功了，并且对应的id正是1，和我们前面打印的结果是一致的。

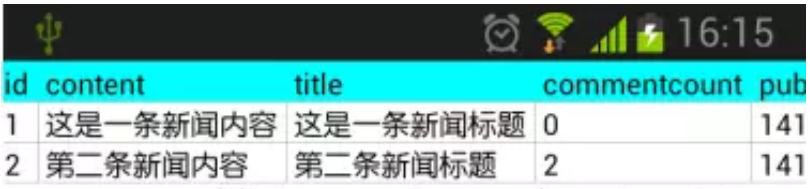
不过LitePal的存储功能显示不仅仅只有这些用法，事实上，LitePal在存储数据的时候默默帮我们做了很多的事情，比如多个实体类之间有关联关系的话，我们不需要考虑在存储数据的时候怎么去建立数据与数据之间的关联，因为LitePal一切都帮我们做好了。

还是通过一个例子来看一下吧，Comment和News之间是多对一的关系，一条News中是可以包含多条评论的，因此我们就可以这样写：

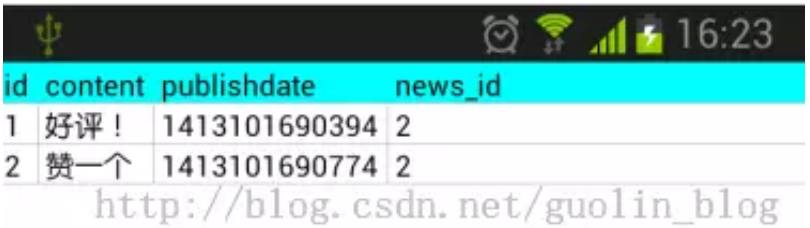
```
Comment comment1 = new Comment();
comment1.setContent("好评！");
```

```
comment1.setPublishDate(new Date());
comment1.save();
Comment comment2 = new Comment();
comment2.setContent("赞一个");
comment2.setPublishDate(new Date());
comment2.save();
News news = new News();
news.getCommentList().add(comment1);
news.getCommentList().add(comment2);
news.setTitle("第二条新闻标题");
news.setContent("第二条新闻内容");
news.setPublishDate(new Date());
news.setCommentCount(news.getCommentList().size());
news.save();
```

可以看到，这里先是存储了一条comment1数据，然后存储一条comment2数据，接着在News之前先把刚才的两个Comment对象添加到了News的commentList列表当中，这样就表示这两条Comment是属于这个News对象的，最后再把News存储到数据库中，这样它们之间的关联关系就会自动建立了。让我们查看数据库表检查一下吧，首先看一下news表，如下所示：



OK，第二条新闻已经成功存储到news表中了，这条新闻的id是2。那么从哪里可以看出来关联关系呢？我们在上一篇文章中学过，多对一关联的时候，外键是存放在多方的，因此关联关系我们要到comment表中去查看，如下所示：



可以看到，两条评论都已经成功存储到comment表中了，并且这两条评论的news_id都是2，说明它们是属于第二条新闻的。怎么样，仅仅是在存储数据之前建立好实体类之间的关系，再调用一下save()方法，那么数据之间的关联关系就会自动建立了，是不是非常简单？上面的代码只是多对一情况的一种用法，还有一对一和多对多的情况，其实用法都是差不多

的，相信你已经能举一反三了。

另外，LitePal对集合数据的存储还专门提供了一个方法，比如说我们有一个News集合，那么应该怎样去存储这个集合中的每条News呢？传统情况下可以这样写：

```
List<News> newsList;
...
for (News news : newsList) {
    news.save();
}
```

通过一个循环来遍历出这个集合中的每一个News对象，然后逐个调用save()方法。这样的写法当然是可以的，但是效率会比较低，因为调用save()方法的时候除了会执行存储操作之外，还会去分析News类的关联关系，那么每次循环都去重新分析一遍关联关系显然是比较耗时的。因此，LitePal提供了一个saveAll()方法，专门用于存储集合数据的，用法如下所示：

```
List<News> newsList;
...
DataSupport.saveAll(newsList);
```

saveAll()方法接收一个Collection集合参数，只要把待存储的集合数据传入即可。这个方法可以完成和上面一段代码完全一样的功能，但效率却会高得多，而且写法也更加简单。

好了，这样我们就把LitePal中提供的存储操作的用法全部都学完了，那么今天的文章就到这里，下一篇文章当中会开始讲解更新和删除操作的用法。感兴趣的朋友请继续阅读 Android数据库高手秘籍(6)：LitePal的修改和删除操作 。

阅读原文

微信扫一扫
关注该公众号