Android数据库高手秘籍(6)：LitePal的修改和删除操作

2015-09-10 安卓应用频道

(点击上方蓝字，可快速关注我们)

> 来源：郭霖
> 链接：http://blog.csdn.net/guolin_blog/article/details/40083685

在上一篇文章中，我们学会了使用LitePal进行存储数据的功能。确实，比起直接使用Android原生的API，LitePal明显简单方便了太多。那么，在增删改查四种操作中，我们已经把“增”学完了，今天就让我们继续趁热打铁，学习一下如何使用LitePal进行修改和删除操作。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(5)：LitePal的存储操作 。

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

传统的修改和删除数据方式

上篇文章中我们已经得知，SQLiteDatabase类中提供了一个insert()方法用于插入数据，那么类似地，它还提供了update()和delete()这两个方法，分别用于修改和删除数据。先来看一下update()方法的方法定义：

```
public int update(String table, ContentValues values, String whereClause, String[] whereArgs)
```

update()方法接收四个参数，第一个参数是表名，第二个参数是一个封装了待修改数据的ContentValues对象，第三和第四个参数用于指定修改哪些行，对应了SQL语句中的where部分。

那么比如说我们想把news表中id为2的记录的标题改成“今日iPhone6发布”，就可以这样写：

```
SQLiteDatabase db = dbHelper.getWritableDatabase();
ContentValues values = new ContentValues();
values.put("title", "今日iPhone6发布");
db.update("news", values, "id = ?", new String[] {"2"});
```

其作用相当于如下SQL语句：

```
update news set title='今日iPhone6发布' where id=2;
```

可以看出，比起直接使用SQL语句，update()方法的语义性明显更强，也更容易让人理解。

接下来再看一下delete()方法的方法定义：

```
public int delete(String table, String whereClause, String[] whereArgs)
```

delete()方法接收三个参数，第一个参数同样是表名，第二和第三个参数用于指定删除哪些行，对应了SQL语句中的where部分。

那么比如说我们想把news表中所有没有评论的新闻都删除掉，就可以这样写：

```
SQLiteDatabase db = dbHelper.getWritableDatabase();
db.delete("news", "commentcount = ?", new String[] {"0"});
```

其作用相当于如下SQL语句：

```
delete from news where commentcount=0;
```

由此可见，Android给我们提供的这些帮助方法，在很大程度上确实简化了不少数据库操作的复杂度。不过LitePal显然做到了更好，下面就让我们学习一下如何使用LitePal来进行修改和删除操作。

使用LitePal修改数据

LitePal修改数据的API比较简单，并没有什么太多的用法，也比较好理解，方法都是定义在DataSupport类中的，我们先来看一下方法定义：

```
public static int update(Class<?> modelClass, ContentValues values, long id)
```

这个静态的update()方法接收三个参数，第一个参数是Class，传入我们要修改的那个类的Class就好，第二个参数是ContentValues对象，这三个参数是一个指定的id，表示我们要修改哪一行数据。

那么比如说我们想把news表中id为2的记录的标题改成“今日iPhone6发布”，就可以这样写：

```
ContentValues values = new ContentValues();
values.put("title", "今日iPhone6发布");
```

```
DataSupport.update(News.class, values, 2);
```

可以看出，总体来讲还是比原生的用法要简单一些的，首先我们避免掉了要去获取SQLiteDatabase对象的步骤，其次在指定修改某一条id记录的时候只需要传入这个id即可，语法更简练。

那么有的朋友可能会问了，也许我想修改的是某一个条件下的所有数据，而不是仅仅修改某个id的数据，那该怎么办呢？别担心，LitePal还提供了另外一个简便的方法，方法定义如下：

```
public static int updateAll(Class<?> modelClass, ContentValues values, String... conditions)
```

updateAll()方法表示修改多行记录，其中第一个参数仍然是Class，第二个参数还是ContentValues对象，第三个参数是一个conditions数组，用于指定修改哪些行的约束条件，返回值表示此次修改影响了多少行数据。

那么比如说我们想把news表中标题为“今日iPhone6发布”的所有新闻的标题改成“今日iPhone6 Plus发布”，就可以这样写：

```
ContentValues values = new ContentValues();
values.put("title", "今日iPhone6 Plus发布");
DataSupport.updateAll(News.class, values, "title = ?", "今日iPhone6发布");
```

前面都没什么好说的，重点我们看一下最后的这个conditions数组，由于它的类型是一个String数组，我们可以在这里填入任意多个String参数，其中最前面一个String参数用于指定约束条件，后面所有的String参数用于填充约束条件中的占位符(即?号)，比如约束条件中有一个占位符，那么后面就应该填写一个参数，如果有两个占位符，后面就应该填写两个参数，以此类推。

比如说我们想把news表中标题为“今日iPhone6发布”且评论数量大于0的所有新闻的标题改成“今日iPhone6 Plus发布”，就可以这样写：

```
ContentValues values = new ContentValues();
values.put("title", "今日iPhone6 Plus发布");
DataSupport.updateAll(News.class, values, "title = ? and commentcount > ?", "今日iPhone6发布", "0");
```

可以看出，通过占位符的方式来实现条件约束明显要比原生的API更加简单易用。

那么如果我们想把news表中所有新闻的标题都改成“今日iPhone6发布”，该怎么写呢？其实

这就更简单了，只需要把最后的约束条件去掉就行了，如下所示：

```
ContentValues values = new ContentValues();
values.put("title", "今日iPhone6 Plus发布");
DataSupport.updateAll(News.class, values);
```

怎么样，这种写法是不是感觉语义性非常强？updateAll()方法在不指定约束条件的情况下就是修改所有行的数据，的的确确是update all了。

当然有些朋友可能会觉得这样用起来还是有点复杂，因为这个ContentValues对象很烦人，每次创建它的时候都要写很多繁琐的代码。没关系，LitePal也充分考虑了这种情况，提供了一种不需要ContentValues就能修改数据的方法，下面我们尝试使用这种新方法来完成上述同样的功能。

比如把news表中id为2的记录的标题改成“今日iPhone6发布”，就可以这样写：

```
News updateNews = new News();
updateNews.setTitle("今日iPhone6发布");
updateNews.update(2);
```

这次我们并没有用ContentValues，而是new出了一个News对象，把要修改的数据直接set进去，最后调用一下update()方法并传入id就可以了。不仅不用创建ContentValues对象，连表名都不用指定了，因为News对象默认就是修改的news表。

这是其中一种用法，那么如果我们想把news表中标题为“今日iPhone6发布”且评论数量大于0的所有新闻的标题改成“今日iPhone6 Plus发布”，就可以这样写：

```
News updateNews = new News();
updateNews.setTitle("今日iPhone6发布");
updateNews.updateAll("title = ? and commentcount > ?", "今日iPhone6发布", "0");
```

还是非常好理解的，这里我就不再详细解释了。

但是这种用法有一点需要注意，就是如果我们想把某一条数据修改成默认值，比如说将评论数修改成0，只是调用updateNews.setCommentCount(0)这样是不能修改成功的，因为即使不调用这行代码，commentCount的值也默认是0。所以如果想要将某一列的数据修改成默认值的话，还需要借助setToDefault()方法。用法也很简单，在setToDefault()方法中传入要修改的字段名就可以了(类中的字段名)，比如说我们想要把news表中所有新闻的评论数清零，就可以这样写：

```
News updateNews = new News();
updateNews.setToDefault("commentCount");
updateNews.updateAll();
```

使用LitePal删除数据

LitePal删除数据的API和修改数据是比较类似的，但是更加的简单一些，我们先来看一下DataSupport类中的方法定义，如下所示：

```
public static int delete(Class<?> modelClass, long id)
```

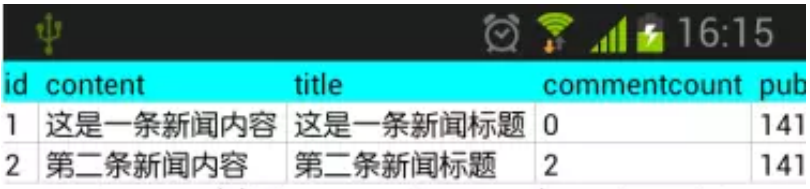
delete()方法接收两个参数，第一个参数是Class，传入我们要删除的那个类的Class就好，第二个参数是一个指定的id，表示我们要删除哪一行数据。

那么比如说我们想删除news表中id为2的记录，就可以这样写：

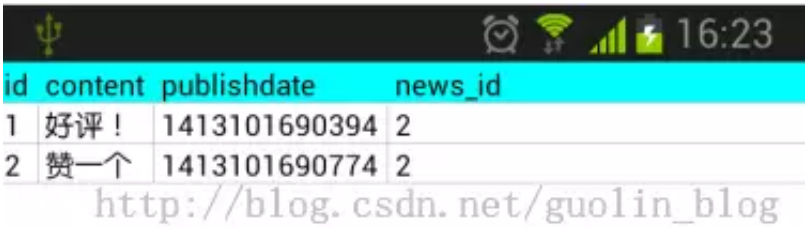
```
DataSupport.delete(News.class, 2);
```

需要注意的是，这不仅仅会将news表中id为2的记录删除，同时还会将其它表中以news id为2的这条记录作为外键的数据一起删除掉，因为外键既然不存在了，那么这么数据也就没有保留的意义了。

说起来可能有点拗口，我们还是举例看一下。比如news表中目前有两条数据，如下图所示：



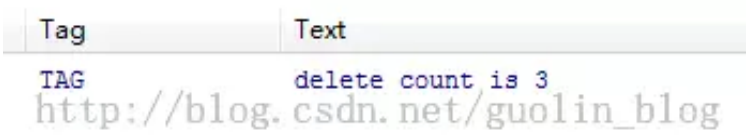
然后comment表中也有两条数据，如下图所示：



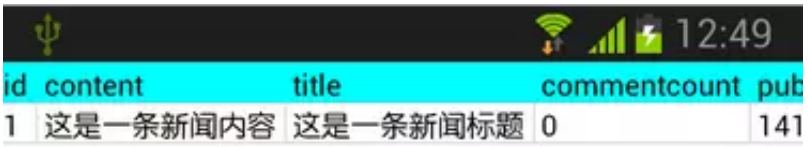
其中comment表中两条数据的外键都是2，指向的news表中id为2的这条记录。那么下面我们执行如下删除语句：

```
int deleteCount = DataSupport.delete(News.class, 2);
Log.d("TAG", "delete count is " + deleteCount);
```

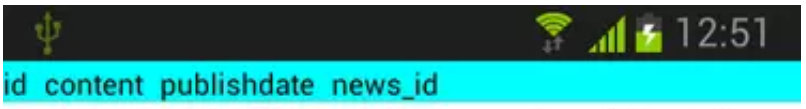
其中delete()方法的返回值表示被删除的记录数，打印结果如下所示：



可以看到，有三条记录被删除了，那我们再到news表中查询一下：



OK，只剩下一条记录了，id为2的那条记录确实被删除了。那么再到comment表中看一下呢，如下图所示：



数据全没了！为什么呢？因为comment表中的两条数据都是以news表中id为2的数据作为外键的，现在外键不存在了，那么这两条数据自然也没有存在的意义了，因此被删除的记录数一共是3条。这样是不是就好理解了很多呢？

除了删除指定id的数据之外，DataSupport中也提供了一个通过where语句来批量删除数据的方法，先看一下方法定义：

```
public static int deleteAll(Class<?> modelClass, String... conditions)
```

看起来很眼熟吧？非常简单，deleteAll()方法接收两个参数，第一个参数是Class，传入我们要删除的那个类的Class就好，第二个参数是一个conditions数组，用于指定删除哪些行的约

束条件，返回值表示此次删除了多少行数据，用法和updateAll()方法是基本相同的。

那么比如说我们想把news表中标题为“今日iPhone6发布”且评论数等于0的所有新闻都删除掉，就可以这样写：

```
DataSupport.deleteAll(News.class, "title = ? and commentcount = ?", "今日iPhone6发布", "0");
```

而如果我们想把news表中所有的数据全部删除掉，就可以这样写：

```
DataSupport.deleteAll(News.class);
```

在不指定约束条件的情况下，deleteAll()方法就会删除表中所有的数据了。

除了DataSupport类中提供的静态删除方法之外，还有一个删除方法是作用于对象上的，即任何一个继承自DataSupport类的实例都可以通过调用delete()这个实例方法来删除数据。但前提是这个对象一定是要持久化之后的，一个非持久化的对象如果调用了delete()方法则不会产生任何效果。

比如说下面这种写法：

```
News news = new News();
news.delete();
```

这里new出了一个News对象，这个对象明显是没有持久化的，那么此时调用delete()方法则不会删除任何数据。

但如果我们之前将这个对象持久化过了，那么再调用delete()方法就会把这个对象对应的数据删除掉了，比如：

```
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.save();
...
news.delete();
```

一个对象如果save过了之后，那就是持久化的了。除了调用save()方法之外，通过DataSupport中提供的查询方法从数据库中查出来的对象也是经过持久化的，查询的功能我们会在下篇博客中讲解。

另外还有一个简单的办法可以帮助我们判断一个对象是否是持久化之后的，DataSupport类中提供了一个isSaved()方法，这个方法返回true就表示该对象是经过持久化的，返回false则表示该对象未经过持久化。那么删除一个对象对应的数据也就可以这样写了：

```
News news;
...
if (news.isSaved()) {
    news.delete();
}
```

好了，这样我们就把LitePal中提供的修改和删除数据操作的用法基本都学习完了，那么今天的文章就到这里，下一篇文章中会开始讲解查询数据的用法，感兴趣的朋友请继续阅读Android数据库高手秘籍(7)：体验LitePal的查询艺术 。

阅读原文

微信扫一扫
关注该公众号