Android 软件开发

数据存储

秦兴国

xgqin@guet.edu.cn

计算机与信息安全学院
桂林电子科技大学

2019 年 5 月 6 日

Outline

数据存储简介

Android 系统提供了多种方式进行 App 数据存储：

- 内部文件存储 (Internal file storage)，用于存储 App 私有数据文件；
- 外部文件存储 (External file storage)，将文件存储于外部共享文件系统中[1]；
- Shared preferences，以键值对形式 (key-value pairs) 存储 App 私有的原子数据；
- 数据库 (Databases)，将结构化的数据存储于 App 私有的数据库；

[1]通常用于 App 间共享用户数据，例如图片等

数据存储简介

Android 系统提供了多种方式进行 App 数据存储：

- 内部文件存储 (Internal file storage)，用于存储 App 私有数据文件；
- 外部文件存储 (External file storage)，将文件存储于外部共享文件系统中[1]；
- Shared preferences，以键值对形式 (key-value pairs) 存储 App 私有的原子数据；
- 数据库 (Databases)，将结构化的数据存储于 App 私有的数据库；

App 可以根据自身的需要选择合适的方式，例如：数据存储所需的空间大小，数据存储的类型，数据是否仅 App 内可用，是否可供其他 App 访问。

[1]通常用于 App 间共享用户数据，例如图片等

Shared preferences 简介

Shared preferences 以键值对形式存储非结构化的原子数据类型：boolean, float, int, long 以及 string 等。

Shared preferences 具备以下特点：

- 以 XML 文件形式存储键值对；
- 存储的数据跨用户会话，当 App 被重新启动后仍可访问；
- 可指定 Shared preferences 存储文件名，可指定多个文件名；

SharedPreferences 对象

SharedPreferences 类提供了读取键值对所需的方法，可以通过如下两种形式获得该对象：

1. **getSharedPreferences()**，如果需要多个不同的 shared preferences 文件，则使用该方法，并在该方法的第一个参数中指明文件名，使用 **Context** 上下文对象可以调用该方法。
2. **getPreferences()**，该方法得到的 shared preferences 是 Activity 相关的，每个 Activity 中的 shared preferences 对象均不一样，使用该方法时不需要指定其文件名。

获得 SharedPreferences 对象

以下代码使用 **getSharedPreferences()** 方法获得 SharedPreferences 对象，其中：

- **R.string.preference_file_key** 指明 Shared preferences 文件名[2]；
- **Context.MODE_PRIVATE**[3]指明该文件为 App 私有文件，仅 App 自身可访问；

```
1   Context context = getActivity();
2   SharedPreferences sharedPref = context.getSharedPreferences(
3           getString(R.string.preference_file_key), Context.MODE_PRIVATE);
```

[2]确保指定的文件名唯一，通常使用 App 的 package name 作为文件前缀，例如:com.example.myapp.PREFERENCE_FILE_KEY

[3]从 API level 17 开始，MODE_WORLD_READABLE 和 MODE_WORLD_WRITEABLE 模式已不推荐使用(deprecated)

向 Shared preferences 文件写入数据

向 Shared preferences 文件写入数据，分为三个步骤：

1. 1. 首先必须获得 **SharedPreferences.Editor** 对象；
2. 2. 通过 **putInt()**、**putString()** 等方式写入键值对数据；
3. 3. 最后通过调用 **apply()** 或 **commit()** 方法保存数据[4]。

[4] **apply()** 使用异步的方式将内存中的 SharedPreferences 对象写回磁盘的文件中，**commit()** 则使用同步的方式完成该操作

向 Shared preferences 文件写入数据

向 Shared preferences 文件写入数据，分为三个步骤：

1. 1. 首先必须获得 **SharedPreferences.Editor** 对象；
2. 2. 通过 **putInt()**、**putString()** 等方式写入键值对数据；
3. 3. 最后通过调用 **apply()** 或 **commit()** 方法保存数据[4]。

```
1    SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
2    SharedPreferences.Editor editor = sharedPref.edit();
3    editor.putInt(getString(R.string.saved_high_score_key), newHighScore);
4    editor.commit();
```

[4] **apply()** 使用异步的方式将内存中的 SharedPreferences 对象写回磁盘的文件中，**commit()** 则使用同步的方式完成该操作

从 Shared preferences 文件读取数据

从 Shared preferences 文件中读取数据：

1. 首先获得文件对应的 SharedPreferences 对象；
2. 通过 **getInt()**、**getString()** 等方法指定 key 获取数据；

从 Shared preferences 文件读取数据

从 Shared preferences 文件中读取数据：

1. 首先获得文件对应的 SharedPreferences 对象；
2. 通过 **getInt()**、**getString()** 等方法指定 key 获取数据；

```
1    SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
2    int defaultValue = getResources().getInteger(R.integer.saved_high_score_default_key);
3    int highScore = sharedPref.getInt(
4                getString(R.string.saved_high_score_key),
5                defaultValue
6                );
```

从 Shared preferences 文件读取数据

从 Shared preferences 文件中读取数据：

1. 首先获得文件对应的 SharedPreferences 对象；
2. 通过 **getInt()**、**getString()** 等方法指定 key 获取数据；

```
1     SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
2     int defaultValue = getResources().getInteger(R.integer.saved_high_score_default_key);
3     int highScore = sharedPref.getInt(
4                 getString(R.string.saved_high_score_key),
5                 defaultValue
6                 );
```

getInt() 等方法的第二个参数可指定默认值，即不存在指定的键值对时，方法返回的默认值。

Android 系统中关系型数据库简介

Android 系统及第三方库为存储结构型、关系型数据提供了相应的接口包。主要包括：

- **android.database.sqlite 与 android.database 包**，提供底层 API 对 sqlite 数据库进行操作；
- **Room Persistence Library**，Android 官方数据库支持库；
- **LitePal、GreenDAO 等**，第三方数据库支持库；

底层数据库 API 操作包

Android 操作系统提供了两个包用于对 sqlite 数据库的操作，其中：

- **android.database.sqlite**，是与 SQLite 数据库操作相关的包；
- **android.datase**，则是用于操作数据库及 Content Provider 返回数据相关的包；

底层数据库 API 操作包

Android 操作系统提供了两个包用于对 sqlite 数据库的操作，其中：

- **android.database.sqlite**，是与 SQLite 数据库操作相关的包；
- **android.datase**，则是用于操作数据库及 Content Provider 返回数据相关的包；

App 创建的数据库被存储于设备的**/data/data/<package_name>/databases** 目录中，其中 *<package_name>* 是 App 的包名。

底层数据库操作常用类及接口

使用 android.database.sqlite，android.database 包进行数据库操作主要涉及的类：

- **SQLiteOpenHelper**，创建数据库及进行数据库版本管理的帮助助手类 (helper class)；
- **SQLiteDatabase**，对数据库进行操作的类；
- **Cursor 接口**，主要提供对数据库返回数据进行操作的方法；

定义数据库表 schema 及对应的合约类

schema(模式) 通常指数据库的组织和结构，可以是数据库中的表、列、视图等。

在使用 android.database.sqlite 等包操作数据库时，通常为每张表创建一个伙伴类 (companion class) 或者称为合约类 (contract class)。合约类的作用：

- 反映数据库 schema 结构；
- 提供与该类对应表信息的常量，例如表名、列名、表及数据 URI 等；

合约类举例

在合约类中，将与数据库相关的定义作为类的顶层元素 (成员变量)，并在该类中定义表示某一个数据库表相关的内部类 (inner class)，内部类中包含该表的所有列名等信息[5]。

[5] 内部类通常实现一个名为 **BaseColumns** 接口，从而使得内部类继承名为 **_ID** 的列用于表示数据表的主键。该 **_ID** 列在 Android 系统中的其他类 (例如：CursorAdapter) 中将被使用。

合约类举例

在合约类中，将与数据库相关的定义作为类的顶层元素 (成员变量)，并在该类中定义表示某一个数据库表相关的内部类 (inner class)，内部类中包含该表的所有列名等信息[5]。

```
1  public final class FeedReaderContract {
2      // To prevent someone from accidentally instantiating the contract class,
3      // make the constructor private.
4      private FeedReaderContract() {}
5
6      /* Inner class that defines the table contents */
7      public static class FeedEntry implements BaseColumns {
8          public static final String TABLE_NAME = "entry";
9          public static final String COLUMN_NAME_TITLE = "title";
10         public static final String COLUMN_NAME_SUBTITLE = "subtitle";
11     }
12 }
```

[5] 内部类通常实现一个名为 **BaseColumns** 接口，从而使得内部类继承名为 **_ID** 的列用于表示数据表的主键。该 **_ID** 列在 Android 系统中的其他类 (例如：CursorAdapter) 中将被使用。

创建 SQLiteOpenHelper 子类

应用需要继承 SQLiteOpenHelper 类，并在子类中重载创建和管理数据库及表格的方法。

创建 SQLiteOpenHelper 子类

应用需要继承 SQLiteOpenHelper 类，并在子类中重载创建和管理数据库及表格的方法。

1. 创建数据库及数据表

通常在 SQLiteOpenHelper 子类中定义创建及删除数据表的静态字符串常量。

创建 SQLiteOpenHelper 子类

应用需要继承 SQLiteOpenHelper 类，并在子类中重载创建和管理数据库及表格的方法。

1. 创建数据库及数据表

通常在 SQLiteOpenHelper 子类中定义创建及删除数据表的静态字符串常量。

```
1  public class FeedReaderDbHelper extends SQLiteOpenHelper {
2      private static final String SQL_CREATE_ENTRIES =
3          "CREATE␣TABLE␣" + FeedEntry.TABLE_NAME + "␣(" +
4          FeedEntry._ID + "␣INTEGER␣PRIMARY␣KEY," +
5          FeedEntry.COLUMN_NAME_TITLE + "␣TEXT," +
6          FeedEntry.COLUMN_NAME_SUBTITLE + "␣TEXT)";
7
8      private static final String SQL_DELETE_ENTRIES =
9          "DROP␣TABLE␣IF␣EXISTS␣" + FeedEntry.TABLE_NAME;
10     ...
11
12 }
```

创建 SQLiteOpenHelper 子类

应用需要继承 SQLiteOpenHelper 类，并在子类中重载创建和管理数据库及表格的方法。

2. 重载 **onCreate()** 方法[6]

[6] onCreate 方法在数据库不存在时被执行

创建 SQLiteOpenHelper 子类

应用需要继承 SQLiteOpenHelper 类，并在子类中重载创建和管理数据库及表格的方法。

2. 重载 **onCreate()** 方法[6]

```
1  public class FeedReaderDbHelper extends SQLiteOpenHelper {
2      ...
3      // If you change the database schema, you must increment the database version.
4      public static final int DATABASE_VERSION = 1;
5      public static final String DATABASE_NAME = "FeedReader.db";
6
7      public FeedReaderDbHelper(Context context) {
8          super(context, DATABASE_NAME, null, DATABASE_VERSION);
9      }
10     public void onCreate(SQLiteDatabase db) {
11         db.execSQL(SQL_CREATE_ENTRIES);
12     }
13     ...
14 }
```

[6] onCreate 方法在数据库不存在时被执行

创建 SQLiteOpenHelper 子类

应用需要继承 SQLiteOpenHelper 类，并在子类中重载创建和管理数据库及表格的方法。

3. 重载 **onUpgrade()**、**onDowngrade()** 方法[7]

[7] 当 App 进行版本更新时，对数据库结构进行了变更，则需要设置数据库版本 (**DATABASE_NAME**) 为一个新的值。

创建 SQLiteOpenHelper 子类

应用需要继承 SQLiteOpenHelper 类，并在子类中重载创建和管理数据库及表格的方法。

3. 重载 **onUpgrade()**、**onDowngrade()** 方法[7]

```
1  public class FeedReaderDbHelper extends SQLiteOpenHelper {
2      ...
3
4      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
5          // This database is only a cache for online data, so its upgrade policy is
6          // to simply to discard the data and start over
7          db.execSQL(SQL_DELETE_ENTRIES);
8          onCreate(db);
9      }
10     public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
11         onUpgrade(db, oldVersion, newVersion);
12     }
13 }
```

[7] 当 App 进行版本更新时，对数据库结构进行了变更，则需要设置数据库版本 (**DATABASE_NAME**) 为一个新的值。

获取 SQLiteDatabase 对象

SQLiteDatabase 类提供了对数据库进行操作的所有方法，可通过 App 继承 SQLiteOpenHelper 的子类提供的方法获取。

- **getWritableDatabase()**，获取一个可写数据库对象；
- **getReadableDatabase()**，获取一个可读数据库对象；

获取 SQLiteDatabase 对象

SQLiteDatabase 类提供了对数据库进行操作的所有方法，可通过 App 继承 SQLiteOpenHelper 的子类提供的方法获取。

- **getWritableDatabase()**，获取一个可写数据库对象；
- **getReadableDatabase()**，获取一个可读数据库对象；

```
1    FeedReaderDbHelper dbHelper = new FeedReaderDbHelper(getContext());
2    // Gets the data repository in write mode
3    SQLiteDatabase db = dbHelper.getWritableDatabase();
```

获取 SQLiteDatabase 对象

getWriteDatabase() 与 getReadDatabase() 方法调用开销很大，尽可能在 Activity 的生命周期内保持一个 SQLiteDatabase 对象。

```
1   SQLiteDatabase db = null;
2   ...
3   @Override
4   protected void onCreate() {
5   ...
6    FeedReaderDbHelper dbHelper = new FeedReaderDbHelper(getContext());
7    db = dbHelper.getWritableDatabase();
8   ...
9   }
10
11  @Override
12  protected void onDestroy() {
13      dbHelper.close();
14      super.onDestroy();
15  }
```

使用 insert() 方法插入数据

SQLiteDatabase 类提供了 insert() 方法用于插入数据。insert() 方法原型如下：

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

[8] SQLite 不允许插入一条空的数据记录，当 values 为空时，nullColumnHack 指定的列的值被设为 *NULL*。

使用 insert() 方法插入数据

SQLiteDatabase 类提供了 insert() 方法用于插入数据。insert() 方法原型如下：

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

参数及返回值含义：

- **table**，插入数据的表名。
- **nullColumnHack**，可为 *NULL* 的列名。[8]
- **values**，键值对数据，key 为列名，value 为列的值。
- **返回值**，返回新插入的数据记录 ID 或 -1。

[8] SQLite 不允许插入一条空的数据记录，当 values 为空时，nullColumnHack 指定的列的值被设为 *NULL*。

使用 insert() 方法插入数据

SQLiteDatabase 类提供了 insert() 方法用于插入数据。

```
1  // Gets the data repository in write mode
2  SQLiteDatabase db = dbHelper.getWritableDatabase();
3
4  // Create a new map of values, where column names are the keys
5  ContentValues values = new ContentValues();
6  values.put(FeedEntry.COLUMN_NAME_TITLE, title);
7  values.put(FeedEntry.COLUMN_NAME_SUBTITLE, subtitle);
8
9  // Insert the new row, returning the primary key value of the new row
10 long newRowId = db.insert(FeedEntry.TABLE_NAME, null, values);
```

使用 query() 方法查询数据

SQLiteDatabase 类提供了 query() 方法用于查询数据。query() 方法原型如下：

```
1  public Cursor query (String table, String[] columns,
2                String selection, String[] selectionArgs,
3                String groupBy, String having, String orderBy);
4
5  public Cursor query (boolean distinct,  String table, String[] columns,
6                String selection, String[] selectionArgs, String groupBy,
7                String having, String orderBy, String limit);
8
9  public Cursor query (String table, String[] columns, String selection,
10               String[] selectionArgs, String groupBy, String having,
11               String orderBy, String limit);
12
13 public Cursor query (boolean distinct, String table, String[] columns,
14               String selection, String[] selectionArgs, String groupBy,
15               String having, String orderBy, String limit,
16               CancellationSignal cancellationSignal);
```

使用 query() 方法查询数据

SQLiteDatabase 类提供了 query() 方法用于查询数据。query() 方法原型如下：

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

使用 query() 方法查询数据

SQLiteDatabase 类提供了 query() 方法用于查询数据。query() 方法原型如下：

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

参数及返回值含义：

- **table**，查询操作的表名；
- **columns**，查询的列名；
- **selection**，selection 子句[9]，与 selctionArgs 参数结合使用；
- **selectionArgs**，selection 子句中? 占位符对应的参数[10]；

[9] selection 中不包含 WHERE 关键字

[10] selection 子句与 selectionArgs 参数分离主要为防止 SQL 注入

使用 query() 方法查询数据

SQLiteDatabase 类提供了 query() 方法用于查询数据。query() 方法原型如下：

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

参数及返回值含义：

- **groupBy**，group by 子句[9]；
- **having**，having 子句[10]；
- **orderBy**，order by 子句[11]；
- **返回值**，返回 Cursor 对象，指向查询结果集。

[9] groupBy 中不包含 GROUP BY 关键字

[10] having 中不包含 HAVING 关键字

[11] orderBy 中不包含 ORDER BY 关键字

使用 query() 方法查询数据

```
1   SQLiteDatabase db = dbHelper.getReadableDatabase();
2
3  String[] projection = {
4      BaseColumns._ID,
5      FeedEntry.COLUMN_NAME_TITLE,
6      FeedEntry.COLUMN_NAME_SUBTITLE
7      };
8
9  String selection = FeedEntry.COLUMN_NAME_TITLE + " = ?";
10 String[] selectionArgs = { "My Title" };
11 String sortOrder = FeedEntry.COLUMN_NAME_SUBTITLE + " DESC";
12
13 Cursor cursor = db.query(
14     FeedEntry.TABLE_NAME,    // The table to query
15     projection,              // The array of columns to return (pass null to get all)
16     selection,               // The columns for the WHERE clause
17     selectionArgs,           // The values for the WHERE clause
18     null,                    // don't group the rows
19     null,                    // don't filter by row groups
20     sortOrder                // The sort order
21     );
```

使用 query() 方法查询数据

Cursor 类提供了多个方法对查询的结果集进行操作：

- int getCount()
- boolean moveToNext()
- boolean moveToPrevious()
- boolean movetoFirst()
- boolean moveToLast()
- boolean move(int offset)
- boolean moveToPosition(int position)

使用 query() 方法查询数据

Cursor 类提供了多个方法对查询的结果集进行操作：

- int getColumnIndex(String columnName)
- double getDouble(int columnIndex)
- float getFloat(int columnIndex)
- int getInt(int columnIndex)
- long getLong(int columnIndex)
- String getString(int columnIndex)
- byte[] getBlob(int columnIndex)

Cursor 类的使用方式

1. 可以通过循环遍历 Cursor 对象指向的数据集；

```
1       ...
2       List itemIds = new ArrayList<>();
3       while(cursor.moveToNext()) {
4               long itemId = cursor.getLong(
5               cursor.getColumnIndexOrThrow(FeedEntry._ID));
6               itemIds.add(itemId);
7       }
8       cursor.close();
9       ...
```

Cursor 类的使用方式

2. 可以将 Cursor 对象传递给 CursorAdapter；

```
1    ...
2    CursorAdapter cursorAdapter = new FeeEntryAdapter(context,
3            R.layout.list_litem, cursor);
4    feedEntryListView.setAdapter(cursorAdapter);
5    ...
```

使用 delete() 方法删除数据

SQLiteDatabase 类提供了 delete() 方法用于删除数据。delete() 方法原型如下：

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

使用 delete() 方法删除数据

SQLiteDatabase 类提供了 delete() 方法用于删除数据。delete() 方法原型如下：

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

参数及返回值含义：

- **table**，删除操作的表名；
- **whereClause**，where 子句；
- **whereArgs**，where 子句的? 占位符对应的参数；
- **返回值**，返回删除的行数。

使用 delete() 方法删除数据

SQLiteDatabase 类提供了 delete() 方法用于删除数据。

```
1  // Define 'where' part of query.
2  String selection = FeedEntry.COLUMN_NAME_TITLE + " LIKE ?";
3
4  // Specify arguments in placeholder order.
5  String[] selectionArgs = { "MyTitle" };
6
7  // Issue SQL statement.
8  int deletedRows = db.delete(FeedEntry.TABLE_NAME, selection, selectionArgs);
```

使用 update() 方法更新数据

SQLiteDatabase 类提供了 update() 方法用于更新数据。update() 方法原型如下：

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

使用 update() 方法更新数据

SQLiteDatabase 类提供了 update() 方法用于更新数据。update() 方法原型如下：

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

参数及返回值含义：

- **table**，更新操作的表名；
- **values**，键值对数据，key 为列名，value 为值；
- **whereClause**，where 子句；
- **whereArgs**，where 子句的? 占位符对应的参数；
- **返回值**，返回更新的行数。

使用 update() 方法更新数据

SQLiteDatabase 类提供了 update() 方法用于更新数据。

```
1  SQLiteDatabase db = dbHelper.getWritableDatabase();
2
3  // New value for one column
4  String title = "MyNewTitle";
5  ContentValues values = new ContentValues();
6  values.put(FeedEntry.COLUMN_NAME_TITLE, title);
7
8  // Which row to update, based on the title
9  String selection = FeedEntry.COLUMN_NAME_TITLE + " LIKE ?";
10 String[] selectionArgs = { "MyOldTitle" };
11
12 int count = db.update( FeedReaderDbHelper.FeedEntry.TABLE_NAME,
13                 values,
14                 selection,
15                 selectionArgs);
```