FPT Software

ANDROID TRAINING
LESSON 8

Version 0.1

Agenda

- **Persistent Storage**
- **Shared Preferences**
- **Android File API**
- **SQLite**

Introduction to Android Persistent Storage

- As mentioned in the Android SDK, there are five different methods:
  - Shared Preferences
  - Internal Storage
  - External Storage
  - SQLite Database
  - Network

Introduction to Android Persistent Storage

- **Shared Preferences**
  - General framework in Android that will allow you to save/retrieve key value pairs for primitive objects such as boolean, float, int, long and string.
- **Pros**
  - Really built-in to the existing Android framework, so easy to use, and easy to create a preference activity to view and edit the preferences
  - Quick
  - Can really be used to store any data with manipulation with the string type.
- **Cons**
  - Not suitable for storing very large amounts of data. As it will be slow in retrieving and sorting through the data.
  - Use internal device memory (more limited than external)
  - Requires customization for any non-primitive data types.

Introduction to Android Persistent Storage

- **Internal Storage**
  - Uses the devices internal memory to store data in files within the devices internal memory.
- **Pros**
  - Many modes of permissions makes this fairly versatile. You can have it usable only by your app, or shared to other applications as read-only or writable.
  - Uses the quicker internal device memory
  - Can use the cache folder to cache files (that will be removed by Android if it needs space)
  - Can be used to store binary files.
- **Cons**
  - Uses the space limited internal device memory
  - No real built-in framework, so lots of customization necessary.
  - Not suitable for retrieving/sorting through large amounts of data.

Introduction to Android Persistent Storage

- **External Storage**
  - Uses the devices external memory (i.e. SD Cards) to store data in files within the devices internal memory.
- **Pros**
  - Access to both public shared directories and application specific directory
  - Can store binary files
  - Space is not an issue with external storage.
  - Easy access by the user to backup data to the computer if necessary.
- **Cons**
  - Usually, external storage is slower than internal storage
  - No real built-in framework, so lots of customization necessary.
  - Not suitable for retrieving/sorting through large amounts of data.
  - Not reliable as user has access and can modify data/remove the external storage at any time. So also requires more error checking.

Introduction to Android Persistent Storage

- **SQLite Database**
  - Uses the devices internal memory to store data in a SQLite database file.
- **Pros**
  - Quick to sort and retrieve large data sets
  - Can easily be used to store customized data
  - Can enforce relationships and integrity between data
- **Cons**
  - Uses space limited internal device memory
  - More difficult to setup.
  - More difficult and error prone when doing structure changes.

Introduction to Android Persistent Storage

- **Network**
  - Stores the data via web-based services.
- **Pros**
  - Easy to share data between devices and computer
  - Easy backup of data
- **Cons**
  - Not reliable, can’t guarantee network connection
  - Most difficult to setup with service and device components
  - Uses up data bandwidth(which costs money for users)
  - Slowest method as ping rates are usually much much slower than local storage.

SharedPreferences

- **SharedPreferences** are a simple, lightweight key/value pair mechanism for saving primitive application data, most commonly a user's application preferences.

SharedPreferences

- How we get access to the preference?
  - **getPreference()** from within Activity:
    to access activity specific preference.
  - **getSharedPreferences()** from within Activity or other application Context:
    to access application-level preference.
  - **getDefaultSharedPreferences()** on **PreferencesManager**
    to get the shared preferences that work in concert with Android's overall preference framework.

SharedPreferences

- Given the appropriate **SharedPreferences** object:
  - **Edit():** This object has a set of setters that mirror the getters for the primitive types Boolean, string, float, long, and integer.
  - **remove()**: Deletes a single named preference.
  - **clear()**: Deletes all preferences.
  - **commit()**: Persists our changes made via the editor.

SharedPreferences

- We can describe our application's preferences in an XML file stored in project's **res/xml/** directory. Android, then, present a UI for manipulating those preferences, which are then stored in the **SharedPreferences** which we get back from **getDefaultSharedPreferences()**.

Android File API

- Access to the file system is performed via the standard java.io classes.
- Android provides also helper classes for creating and accessing new files and directories. For example the getDir(String, int) method would create or access a directory. The openFileInput(String s) method would open a file for input and openFileOutput(String s, int) would create a file.
- int specifies the permissions which are:
  - MODE_PRIVATE - No access for other applications
  - MODE_WORLD_READABLE - Read access for other applications
  - MODE_WORLD_WRITABLE - Write access for other applications
  - MODE_WORLD_READABLE | MODE_WORLD_WRITABLE - Read / Write access

Internal storage

```
private void writeFileToInternalStorage() {
        String eol = System.getProperty("line.separator");
        BufferedWriter writer = null;
         try {
                writer = new BufferedWriter(new OutputStreamWriter(openFileOutput( "myfile",
                    MODE_WORLD_WRITEABLE)));
                writer.write("This is a test1." + eol);
                writer.write("This is a test2." + eol);
        } catch (Exception e) {
              e.printStackTrace();
        }
         finally {
                if (writer != null) {
                         try { writer.close();
                         } catch (IOException e) {
                                e.printStackTrace();
                         }
                }
         }
}
```

Internal storage

```
private void readFileFromInternalStorage() {
String eol = System.getProperty("line.separator"); BufferedReader input = null;
try {
        input = new BufferedReader(new InputStreamReader( openFileInput("myfile")));
        String line;
        StringBuffer buffer = new StringBuffer();
        while ((line = input.readLine()) != null) {
                buffer.append(line + eol);
        }
} catch (Exception e) {
         e.printStackTrace();
} finally {
                if (input != null) {
                        try { input.close();
                        } catch (IOException e) {
                            e.printStackTrace();
                         }
                }
        }
}
```

External storage

- **External storage**
  - Android supports also access to an external storage system e.g. the SD card. All files and directories on the external storage system are readable for all applications.
  - To write to the external storage system your application needs the android.permission.WRITE_EXTERNAL_STORAGE permission. You get the path to the external storage system via the Environment.getExternalStorageDirectory() method.
  - Via the following method call you can check the state of the external storage system. If the Android device is connected via USB to a computer, a SD card which might be used for the external storage system is not available.

```
Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)
```

External storage

```
private void readFileFromSDCard() {
File directory = Environment.getExternalStorageDirectory();
// Assumes that a file article.rss is available on the SD card
File file = new File(directory + "/article.rss");
if (!file.exists()) {
throw new RuntimeException("File not found");
}
Log.e("Testing", "Starting to read");
BufferedReader reader = null;
try {
        reader = new BufferedReader(new FileReader(file));
        StringBuilder builder = new StringBuilder();
        String line;
        while ((line = reader.readLine()) != null) {
                builder.append(line);
         }
         } catch (Exception e) {
                 e.printStackTrace();
         }
         finally {
                 if (reader != null) {
```

SQLite

- Android default Database engine is Lite. SQLite is a lightweight transactional database engine that occupies a small amount of disk storage and memory.
- Things to consider when dealing with SQLite:
  - Data type integrity is not maintained in SQLite, you can put a value of a certain data type in a column of another datatype (put string in an integer and vice versa).
  - Referential integrity is not maintained in SQLite, there is no FOREIGN KEY constraints or JOIN statements.
  - SQLite Full Unicode support is optional and not installed by default.

SQLite

- **Creating SQLite Database**
  - The first step is to create a class that inherits from SQLiteOpenHelper class. This class provides two methods to override to deal with the database:
    - onCreate(SQLiteDatabase db): invoked when the database is created, this is where we can create tables and columns to them, create views or triggers.
    - onUpgrade(SQLiteDatabse db, int oldVersion, int newVersion): invoked when we make a modification to the database such as altering, dropping , creating new tables.

SQLite

- **Managing Foreign-Key Constraints**
  - SQLite 3 by default does not support foreign key constraint, however we can force such a constraint using **TRIGGERS**

SQLite

- **Executing SQL Statements**
  - execute any SQL statement : insert, delete, update or DDL using db.execSQL(String statement)

```
db.execSQL("CREATE TABLE "+deptTable+" ("+colDeptID+
  " INTEGER PRIMARY KEY , "+ colDeptName+ " TEXT)");
```

SQLite

- **Inserting Records**
  - call this.getWritableDatabase() to open the connection with the database for **reading/writing**.
  - The ContentValues.put has two parameters: Column Name and the value to be inserted.
  - close the database after executing statements.

SQLite

- **Updating Values**
  - To execute an update statement, we have two ways:
    - To execute db.execSQL
    - To execute db.update method, the update method has the following parameters:
      - String Table: The table to update a value in
      - ContentValues cv: The content values object that has the new values
      - String where clause: The WHERE clause to specify which record to update
      - String[] args: The arguments of the WHERE clause

SQLite

- **Deleting Rows**
  - As in update to execute a delete statement, we have two ways:
    - To execute db.execSQL
    - To execute db.delete method

SQLite

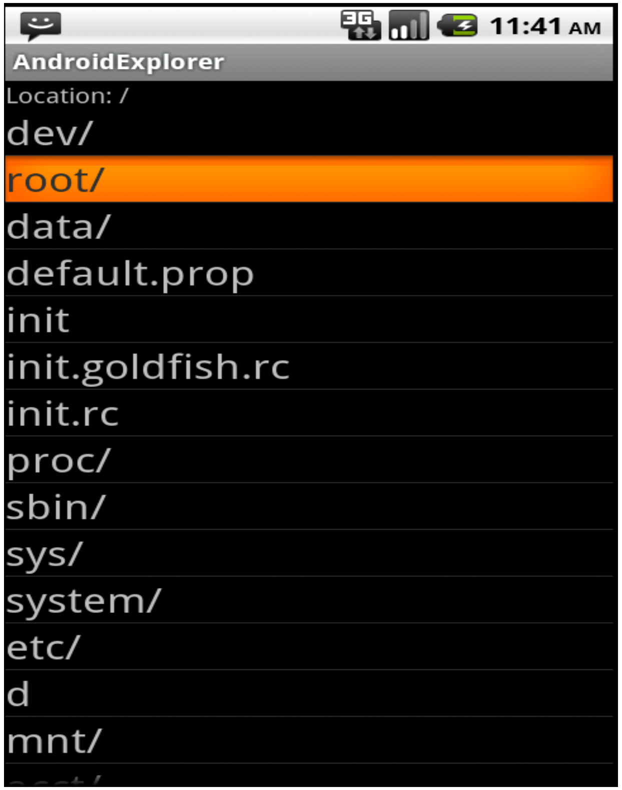
- **Executing Queries**
  - To execute queries, there are two methods:
    - Execute db.rawQuery method
    - Execute db.query method. The db.query has the following parameters:
      - String Table Name: The name of the table to run the query against
      - String [ ] columns: The projection of the query, i.e., the columns to retrieve
      - String WHERE clause: where clause, if none pass null
      - String [ ] selection args: The parameters of the WHERE clause
      - String Group by: A string specifying group by clause
      - String Having: A string specifying HAVING clause
      - String Order By by: A string Order By by clause

SQLite

- **Managing Cursors**
  - Result sets of queries are returned in Cursor objects. There are some common methods that you will use with cursors:
  - boolean moveToNext(): moves the cursor by one record in the result set, returns false if moved past the last row in the result set.
  - boolean moveToFirst(): moves the cursor to the first row in the result set, returns false if the result set is empty.
  - boolean moveToPosition(int position): moves the cursor to a certain row index within the boolean result set, returns false if the position is un-reachable
  - boolean moveToPrevious(): moves the cursor to the previous row in the result set, returns false if the cursor is past the first row.
  - boolean moveToLast(): moves the cursor to the lase row in the result set, returns false if the result set is empty.

Homework

- Create simple File browser using listFiles API



```
11:41 AM
AndroidExplorer
Location: /
dev/
root/
data/
default.prop
init
init.goldfish.rc
init.rc
proc/
sbin/
sys/
system/
etc/
d
mnt/
```

Thank you!