Android: Saving Data

Outline

- Saving data
  - Files
  - SQLite Database
  - Shared Preferences

Outline

- Saving data
  - Files
    - 140.114.79.79:/dropbox/SaveFile.rar
  - SQLite Database
  - Shared Preferences

Java Saving Data in Files

- Recall what did in Java SE

```
File file = new File("./Taiwan.txt");
FileWriter fw = new FileWriter(file.getAbsoluteFile());
BufferedWriter bw = new BufferedWriter(fw);
bw.write(content);
bw.close();
```

```
package org.nmsl;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
public class Main3 {
    public static void main(String[] args) {
        try {
            String content = "I love Taiwan";
            //Windows read file
            //File file = new File(".\\Taiwan.txt");
            //Linux read file
            File file = new File("./Taiwan.txt");
```

```
            if (!file.exists()) {
                file.createNewFile();
            }
            FileWriter fw = new FileWriter(file.
        getAbsoluteFile());
            BufferedWriter bw = new BufferedWriter(fw);
            bw.write(content);
            bw.close();
            System.out.println("Done");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Features of internal and external storage

Internal storage (built-in non-volatile memory):

- it's always available
- files can only be accessed by your app
- while uninstalling the app, all files are removed

External storage (SD card):

- not always available
- files are world-readable
- while uninstalling the app, files saved in getExternalFilesDir() are removed

Install your app in external storage

- apps are installed onto the internal storage by default
- if the internal storage is not enough…
  - install your apps in external storage

```
<manifest xmlns:android="http://schemas.android.
com/apk/res/android"
    android:installLocation="preferExternal"
    ... >
```

Save Data in External Storage

- Always check the permission first

```
<manifest ...>
    <uses-permission android:name="android.permission.
    WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.
    RAED_EXTERNAL_STORAGE" />
    ...
</manifest>
```

- Your application always has permission to access the internal storage

Two Categories of External Storage

Files are technically accessible by the user and other apps

- Public files
  - When the user uninstalls your app, these files should remain available to the user
- Private files
  - When the user uninstalls your app, the system deletes all files in your app's external private directory

Get Path to Save Data

Internal Storage:

- context.getFilesDir()

External Storage:

- Environment.
  getExternalStoragePublicDirectory(
  Environment.DIRECTORY_PICTURES)
- getApplicationContext().getExternalFilesDir(
  Environment.DIRECTORY_PICTURES)

Can We Specify the Path by Ourselves?

- Yes, but you cannot make sure that the specified path is exist.
  - The structure of the folders will be different in different devices
- Also, the file may not be treated properly by your system.
  - for example, you save a ringtone in your specified path, your media scanner may not be able to classify it as a ringtone

Important Concept of UI Thread

- To avoid overloading UI thread, we cannot execute tasks which takes time to finish
  - Use worker thread to the tasks
    - Handler
    - AsyncTask
    - …
- You can only update UI in your UI thread

How to Update UI?

- In our sample code, we need to use a thread to download an image file. How can we update the image view after we download it?
  - implement a callback function
    - define your interface in thread class
    - implement it in your main thread (UI thread)

Steps of AsyncTask

- onPreExecute():
  - used to set up the task
- doInBackground(Params...):
  - perform background computation that can take a long time
- onProgressUpdate(Progress...):
  - This method is used to display progress
- onPostExecute(Result):
  - result of the background computation is passed to this step

Rules of AsyncTask

- The AsyncTask class must be loaded on the UI thread
- execute(Params...) must be invoked on the UI thread
- Do not call the functions of 4 steps manually
- The task can be executed only once

Save Image to File

- Use `FileOutputStream(filename)`

```
Bitmap savePic = ....
FileOutputStream fos = new FileOutputStream(
SDStorage+filename+".png");
savePic.compress(Bitmap.CompressFormat.PNG, 90, fos);
fos.close();
```

Practice

- Download the sample code and try to show the downloaded image in your image View

Outline

- Saving data
  - Files
  - SQLite Database
    - 140.114.79.79:/dropbox/SQLite.zip
  - Shared Preferences

SQLite DB

- SQLite DB is the default DB of Android
- It has several features
  - Serverless
  - Single Database File: An SQLite database is a single ordinary disk file
  - Compact: The whole SQLite library with everything enabled can be less than 400KB in size

Define a Schema and Create DB

```
public static final String TABLE_COMMENTS = "comments";
public static final String COLUMN_ID = "_id";
public static final String COLUMN_COMMENT = "comment";

private static final String DATABASE_NAME = "commments.db";
private static final int DATABASE_VERSION = 1;

// Database creation sql statement
private static final String DATABASE_CREATE = "create table "
                + TABLE_COMMENTS + "(" + COLUMN_ID
                + " integer primary key autoincrement, " + COLUMN_COMMENT
                + " text not null);";
```

Define a SQL Helper which extends SQLiteOpenHelper

- A useful set of APIs provided by SQLiteOpenHelper helps us access the database
- getWritableDatabase()
- getReadableDatabase()
- onCreate(SQLiteDatabase database)
  - Create DB
- onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)

Open your database and insert/update/delete entry

```
private SQLiteDatabase database=new MySQLiteHelper(context).
getWritableDatabase();
ContentValues values = new ContentValues();
values.put(MySQLiteHelper.COLUMN_COMMENT, comment);
```

- insert

```
database.insert(MySQLiteHelper.TABLE_COMMENTS, null,values);
```

- update

```
database.update(MySQLiteHelper.TABLE_COMMENTS , values, MySQLiteHelper.
COLUMN_ID + " = " + id, null);
```

- delete

```
database.delete(MySQLiteHelper.TABLE_COMMENTS,
MySQLiteHelper.COLUMN_ID+ " = " + id, null);
```

Practice

- Download the sample code and try to implement update function to update the content of existing entry

Outline

- Saving data
  - Files
  - SQLite Database
  - Shared Preferences
    - 140.114.79.79:/dropbox/SharedPreferences.zip

Shared Preferences

- Shared preferences are used for a small key-value set
- Get sharedpreferences

```
Context context = getActivity();

SharedPreferences sharedPref = context.getSharedPreferences(

        getString(R.string.preference_file_key), Context.

MODE_PRIVATE);
```

Write SharedPreferences

- To write to a shared preferences file, create a SharedPreferences. Editor by calling edit() on your SharedPreferences.

```
SharedPreferences sp = this.getPreferences(Context.MODE_PRIVATE);

    sp.edit()
            .putString(nameField, name.getText().toString())
            .putString(phoneField, phone.getText().toString())
            .putString(sexField, sex.getText().toString())
            .commit();
```

Read SharedPreferences

- To retrieve values from a shared preferences, call methods such as getInt() and getString(), providing the key for the value you want

```
SharedPreferences sp = this.getPreferences(Context.MODE_PRIVATE);

    name.setText(sp.getString(nameField, ""));
    phone.setText(sp.getString(phoneField, ""));
    sex.setText(sp.getString(sexField, ""));
```