Mobile App Development

Lec 6: Saving data (File)

Ekarat Rattagan, PhD

Outline

- Saving Key-Value Sets
- Saving Files
- Saving Data in SQL Databases

Saving Files

- Android uses disk-based file systems.
  - https://docs.oracle.com/cd/E19683-01/817-3814/fsoverview-1/index.html

- A File object is suited to read or write large amounts of data, e.g., image files or anything ex changed over a network.

- API: Java.io

Internal or External Storage

Two file storage areas

- Internal storage: built-in non-volatile memory
- External storage: a removable storage medium (micro SD card)

Some devices divide the permanent storage space into two storage spaces (even without a removable storage medium)

Internal or External Storage

Internal storage	External storage
Always available	not always available
Files saved here are accessible by only your app	files saved here may be read outside of your control
When the user uninstalls his app, the system removes all his app's files from internal storage	When the user uninstalls his app, the system removes his app's files from here only if user save them in the directory from getExternalFilesDir()
best when you want to be sure that neither the user nor other apps can access your files	best place for files that don't require access restrictions and for files that you want to share with other apps or allow the user to access with a computer

Permissions

- Permissions for External storage

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

Don’t need any permissions to save files on Internal storage. Your application always has permission to read and write files in its internal storage directory.

Save a File on Internal Storage

```
String filename  = "myfile";
String string    = "Hello world!";
File file        = new File(this.getFilesDir(), filename);

try
{
  FileOutputStream fos = openFileOutput(filename, Context.MODE_PRIVATE);
  fos.write(string.getBytes());
  fos.close();
}
catch (Exception e)
{
  e.printStackTrace();
}
```

Read a File on Internal Storage

```
try {

    InputStream is = this.openFileInput(fileName);

    if ( is != null )
    {
        InputStreamReader isr = new InputStreamReader(is);
        BufferedReader br = new BufferedReader(isr);
        String rs = "";
        StringBuilder sb = new StringBuilder();

        while ( (rs = br.readLine()) != null )
        {
            sb.append(rs);
        }

        is.close();
        String ret = sb.toString();
    }

} catch (IOException e) {

}
```

Example 1

- Implement Ex.1 (Please see the attached file, Ex1.pdf)
- In Ex1.pdf, you will see the new View (SnackBar)

- **To use SnackBar**

  Insert `compile 'com.android.support:design:25.1.1'`
  into build.gradle > dependencies and sync now

- **More about Snackbar**
  - http://www.akexorcist.com/2015/07/android-design-support-library-snackbar.html

Write & Read a File on External Storage

```
File file = new File(this.getExternalFilesDir(null), fileName);
```

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}
```

Because the external storage may be unavailable—such as when the user has mounted the storage to a PC or has removed the SD card that provides the external storage—you should always verify that the volume is available before accessing it. You can query the external storage state by calling `getExternalStorageState()`. If the returned state is equal to `MEDIA_MOUNTED`, then you can read and write your files.

More about External Storage

- To check the size of external storage

```
long freeSize = this.getExternalFilesDir(null).getFreeSpace();
long totalSize = this.getExternalFilesDir(null).getTotalSpace();
```

However, the system does not guarantee that you can write as many bytes as are indicated by `getFreeSpace()`.

- The storage can be written if the number returned is a few MB more than the size of the data you want to save, or if the file system is less than 90% full. Otherwise, you probably shouldn't write to storage.

Exercise 1

1.1 Modify the writeFile method,

- to get the percentage of free space size?
- to allow to save file, if after saving, the percentage of free space size is more than 10%.

Delete a File

You should always delete files that you no longer need. The most straightforward way to delete a file is to have the opened file reference call `delete()` on itself.

```
myFile.delete();
```

If the file is saved on internal storage, you can also ask the `Context` to locate and delete a file by calling `deleteFile()`:

```
myContext.deleteFile(fileName);
```

More file types

- Load an image from SD Card

```
ImageView myImageView = (ImageView)findViewById(R.id.imageview);

File    sdCard = Environment.getExternalStorageDirectory();
File    directory = new File (sdCard.getAbsolutePath() + “/Your_Path”);
File    file = new File(directory, "image_name.jpg“);

Bitmap  bitmap = BitmapFactory.decodeFile(file);

myImageView.setImageBitmap(bitmap);
```

Interesting External Library for files

- Encrypt/Decrypt library
  - https://github.com/tozny/java-aes-crypto

Android External Library

- https://github.com/eluleci/FlatUI

https://codecanyon.net/item/android-clean-flat-ui-components/11316073

Constraint Layout tutorial

https://codelabs.developers.google.com/codelabs/constraint-layout/index.html?index=..%2F..%2Findex#0

Keyword to search: codelabs constraint layout