

ANDROID



Data Storage in Android

Dimitar G. Dimitrov



What options do we have?

- [SharedPreferences](#)
- Internal storage (flash memory)
- External storage
- SQLite relational DB
- [ContentProvider](#)
- Network





SharedPreferences

- Persistent way to store key/value pairs
 - Primitive types and Strings
- Saved as XML in your application's folder in /data/data
- Removed when the app is uninstalled
- Can be used for general settings of the application
 - See [PreferenceActivity](#)



SharedPreferences Privacy

- Can be obtained with different modes
 - MODE_PRIVATE
 - MODE_WORLD_READABLE
 - MODE_WORLD_WRITABLE
- [android:sharedUserId](#) + MODE_PRIVATE
- How safe is this?





Working with SharedPreferences

- `getSharedPreferences(String, int)` or `getPreferences(int)`
- To write values:
 - obtain `SharedPreferences.Editor` by calling `edit()`
 - write stuff with the editor using methods such as `putBoolean()` and `putString()`
 - apply the changes by calling `commit()` to your editor
- To read values:
 - `SharedPreferences.getBoolean()`, `getString()`, etc.



אנדרואיד



Demo





Internal Storage

- Data saved on the internal storage of the device is located in your application's folder in /data/data
- Like SharedPreferences, these files are removed, when the app is uninstalled
- YAFFS (Yet Another Flash File System)
 - read (very fast)
 - write (not very fast)
 - erase (very slow)





Internal Storage - 2

Some methods in class Context:

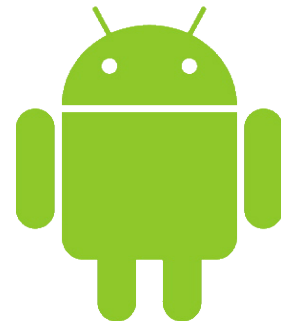
- [String\[\] fileList\(\)](#)
- [FileOutputStream openFileOutput\(String, int\)](#)
 - MODE_PRIVATE, ..., MODE_APPEND
- [FileInputStream openFileInput\(String\)](#)
- [boolean deleteFile\(String\)](#)
- [File getDir\(String, int\)](#)





Internal Storage - 3

- [File getCacheDir\(\)](#)
 - Application specific cache directory
(/data/data/<package_name>/cache/)
 - These files will be ones that get deleted first when the device runs low on storage





External Storage

- First, external storage is not always SD Card
 - Samsung Galaxy Tab has both `internal_sd` and `external_sd`
- If you rely on external storage:
 - always start with a check to [`getExternalStorageState\(\)`](#)
 - listen for broadcasts, regarding the state of the external storage (`ACTION_MEDIA_EJECT`, `ACTION_MEDIA_REMOVED`, `ACTION_MEDIA_UNMOUNTED`, `ACTION_MEDIA_BAD_REMOVAL`, etc.)
- Who can access the files on the external storage?





External Storage - 2

- With Android 2.2 (API Level 8), the ability to install applications on the external storage have been introduced
 - `getExternalFilesDir(String)` opens your application's folder there
 - `getExternalCacheDir()` works similar to `getCacheDir()`, but the system doesn't monitor it as much
 - available space isn't checked
 - there isn't application sandbox security
- Media scanner and pre-defined folders (API Level 8)





External Storage - 3

- For Android 2.1-update1 (API Level 7) or below, use [getExternalStorageDirectory\(\)](#) and the standard Java approach for creating and managing files
- Media scanner still recognizes specific folder names, but you have to create them manually
- “.nomedia” empty file - include in your folder, if you want the scanner to skip it
 - if you have the Android source, look at `/external/opencore/mediascanner.cpp`





Demo

- Manage files on a device/emulator using
 - DDMS
 - adb push/pull
 - mount





SQLite

- What is SQLite?
 - Embedded RDBMS in 275 KB
 - public domain (whether this classifies as open-source is still an open debate)
- Why use RDB?
- Why SQLite?
 - Android has full support for SQLite databases
 - Lightweight, no separate process
 - Very popular (iPhone, Skype, etc.)





SQLite - 2

- To create and use SQLite database, use [SQLiteOpenHelper](#)
 - _ use `getReadableDatabase()` or `getWritableDatabase()`
 - _ `onCreate()` of the helper is called (provide SQL CREATE statement here)
 - _ use some of the `query()` methods of `SQLiteDatabase`
 - _ [Cursor](#) is returned as a result of the query
- Databases are saved in your application's folder in `/data/data`
- The [sqlite3](#) tool is available for examining the contents of a table (`.dump`), the initial SQL CREATE statement (`.schema`) or executing queries dynamically (directly)





SQLite - 3

- Cursor can hold only about 1 MB, after which it has to use windowing (very slow). Be careful!
- For complex queries, use SQLiteQueryBuilder
- Using only SQLite can range from being very easy to being pretty hard
 - If you get to the pretty hard point, it's good to use ContentProvider, even if you don't have to share data





SQLite - Good Practices

- Consider creating a database adapter, which adds an abstraction layer that encapsulates database interactions.
- Files are not usually stored within database tables
- Auto-increment primary key is recommended



אנדרואיד

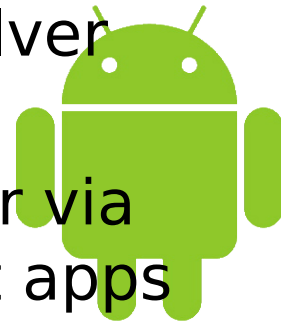


Demo



ContentProvider

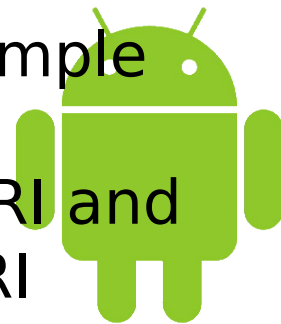
- One of the fundamental components of Android applications
- Encapsulates data and provides common interface for it, independent from the implementation details
- The primary use of most ContentProviders is sharing data between multiple applications
- Generally, its interface is used via ContentResolver objects
 - Usually you access the same ContentProvider via different ContentResolvers from the different apps





ContentProvider - 2

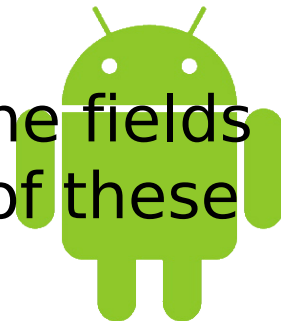
- The data model is similar to the RDB model
 - you can think of each record as represented by a row in a table with columns for each type of data
- One provider can contain multiple data sets (tables)
- Every data set in the provider has a unique URI
 - All URIs for providers begin with "content://"
 - Different data sets have different URIs, for example the built-in Contacts provider has both `android.provider.Contacts.Phones.CONTENT_URI` and `android.provider.Contacts.Photos.CONTENT_URI`





ContentProvider - 3

- Because of the similarities between ContentProviders and RDB models, very often SQLite databases are used together with ContentProvider
- Querying ContentProvider data can be done via either `ContentResolver.query()` or `Activity.managedQuery()`.
 - The difference is that the latter manages the lifecycle of the result Cursor.
 - Querying requires the URI of the provider, the fields that you want returned and the data types of these fields





ContentProvider - 4

- Cursors can be used only for reading data
- Adding, modifying or deleting data is done via ContentResolver objects
 - adding and modifying are similar; use [insert\(Uri, ContentValues\)](#)
 - for deleting use [delete\(Uri, String, String\[\]\)](#)





ContentProvider and REST

- Relation between REST HTTP methods and methods for using ContentProvider data
 - query() == GET
 - insert() == POST
 - update() == PUT
 - delete() == DELETE



Resources and Links

- <http://developer.android.com/guide/topics/data/data-storage.html>
- <http://www.youtube.com/watch?v=c4zvnD-7VDA#t=4m5s>
- <http://www.youtube.com/watch?v=xHXn3Kg2IQE>



Q&A + Feedback

- Questions?
- Feedback section:
 - Did you hear well?
 - Was there anything you didn't understand?
 - What would you like changed in our next lecture?

