Contents

1 Logical Data Storage on Android

File-based abstractions

- Shared Preferences: Simple key value pairs
- File-based storage:
  - Internal Data Storage: Soldered RAM. Internal APK resources, temporary files
  - External Data Storage: SD Card. Large media files
- SQLite Database: Structured data, small binary files

Network sync adapters:

- Shared contact lists, backups
- Synchronising local and remote files

2 Internal File Storage

Internal Data storage is **private to the application**

- Other apps (and the user) cannot access it: Kernel enforced user permissions
- Removed on uninstall, so tbe sensitive data cannot be obtained
- Data is stored in files:
  - /data/data/com.example.martinstorage/files/
  - openRawResource: Can be used to read our own packaged resources

Android provides a standard place to store (small) cache files

- /data/data/com.example.martinstorage/cache
- getCacheDir() to get a File for the directory
- We should still manage the files ourselves
  - May be deleted when internal storage becomes full / contested
  - Will be deleted when the application is uninstalled
  - A well behaved application will delete them when no longer in use. Recommended to use less than 1MB

2.1 Shared Preferences

- In internal Storage , stored on a per-application basis
- I.e. all components in an application may access the same Shared Preferences
- But should not be used for data transfer (instead of Intents, Binder etc)
- Primitive data in key-value pairs. Primitives: strings, integers etc.
- Can have multiple preference files per application

3 External storage

Every Android device provides externally-accessible storage, e.g. SD card

- Even those phones without an SD card. They have a logical representation of external storage. Achieved by partinioning a single storage device into internal / external. This is because all Android devices must **conform to the Android API** in order to be Android devices
- Private application files. Internal Storage on the External partition (Else permissions cant be overriden with other device)
- Public general files
  - World readable
  - Other applications can read and modify these files
  - Each user has their own virtual SD card

Can be mounted externally (and/or disconnected). Before accessing files need to check the state of external storage. It may not be there, or mounted by something else

3.1 Using external storage

- Should check state with `Environment.getExternalStorageState()`. It is a separate file system, potentially removable. `Environment.MEDIA_MOUNTED`, `Environment.MEDIA_MOUNTED_READ_ONLY`.
- Use `getExternalStoragePublicDirectory(String type)` to obtain a File for the directory
  - Pass a type to obtain a sub-directory for that type
  - Used to enable the Media scanner to categorize material
  - Use File object returned to createNewFile()
  - Scoped Directory access
  - With each new release, developers have been provided with new and updated APIs to work with
- `getExternalFilesDir()` provides private external storage
- /sdcard/Android/data/com.example.pszmdf.fingerpainter/

4 Android Databases

- Often the data we are storing is logically structured
  - Need to query it based on that structure
  - Could store this in a file and write our own routines to access it
  - Obb virtual file system
  - StorageManager (wrapper for MountService system service)
  - Opaque Binary Blobs
  - Normally, we'd use a database to store it. E.g. An address book, music library
- Android provides local database support
  - Complete with the ability to run full SQL queries. Each apps databases are local to it
  - Database.db stored in Internal Storage (SQLite)

4.1 Android and SQLite

- Wrapped up in two main classes
  - Database represented by SQLiteDatabase
  - Lets us run SQL queries on the database
  - SQLiteOpenHelper: Supports the Application lifecycle. `onCreate()`. Create the database the first time the application is opened. `onUpgrade(int oldVersion, int newVersion`. Changing the version number affords drop and recreation of the database
- Create an instance of our SQLiteOpenHelper subclass
- Obtain reference to SQLiteDatabase using: `getReadableDatabase(), getWriteableDatabase()` Both return the same object, unless memory is low and can only open the DB readonly

4.2 Querying

- `execSQL` used for SQL queries that do not require a return value
- `rawQuery` used for SQL queries that will return a result
- `query` used for **building and executing** SQL queries for a result

4.3 Cursors

- Provides random access to results of a query
- Enable us to step over all the rows returned by a query `moveToFirst(), moveToNext()`
- `getString(columnIndex), getInt(columnIndex)`. Where column index is index of projection result
- Has a `close()` method to close the query when finished. Shouldnt wait for it to be garbage collected
- IPC implications. Can't be passed to another activity, as when querying the database context is passed.

4.4 Data Driven Views

Connect a cursor to a CursorAdapter and ListView

- SimpleCursorAdapter(...Cursor...)
- Map the projection to a View layout for a single item, populate a list of views
- Link resource IDs to projection columns. Requires each row to have an `_id` field
- Automatically generates a View for each row of the cursor

RecyclerView

- Optimised, flexible version of the above
- Only creates Views for visible data
- As the user scrolls, or more data is added
- Create new Views as appropriate. Re-bind old Views to new data
- Programmatically describe how to create View from data

4.5 CursorLoader

A query may last some time. Database may be large, may be in a different process. We Dont want to block the main UI thread

CursorLoader: Populates views asynchronously, Auto Updating, Monitors for notification that content has changed

4.6 Database Abstraction

Good software architecture. Separation of data model from presentation / views. Abstraction of database architecture:

- Easier to update storage code
- Expose column indices as static class variables: `c.getInt(0) -> c.getInt(DBHelper.NAME)`
- Helper methods keep database internals from leaking into other classes. Return a Collection of results rather than a Cursor. Use Cursor internally in DBHelper class
- SQL injection: Sanitise user input
- Important when thinking about the logical next step exposing data to other applications via a Component. Appropriate database schema

5 Sharing data

5.1 Content Provider

Access to data is restricted to the app that owns it

- Database is usually located in internal app-specific storage (Inaccessible by other applications)
- If we want other apps to access our data, or we want to access other apps data, or we want to be notified when data has changed

Provide or make use of a Content Provider

- Application component number 3
- Exposes data / content to other applications in a structured manner
- Fundamentally IPC via Binder (again) + ashmem with a well defined (database-like) interface

5.2 System Content Providers

Content Providers manage data for:

- Browser: Bookmarks, history
- Call log: Telephone usage
- Contacts: Contact data, WhatsApp?
- Media: Media metadata database
- UserDictionary: Database for predictive spelling
- And other common mobile capabilities

Good practice even when making data available only within the application

- Either create a new one (by sub-classing ContentProvider)
- Or add / query data via an existing / native Content Provider

Assuming that spawning an Activity via Intent is not sufficient

- Querying complex data
- Task flow
- Requiring close coupling of application to data. c.f. Binding to Services

5.3 Data Model

Content Providers implement a specific data model. Very similar to a relational database table:

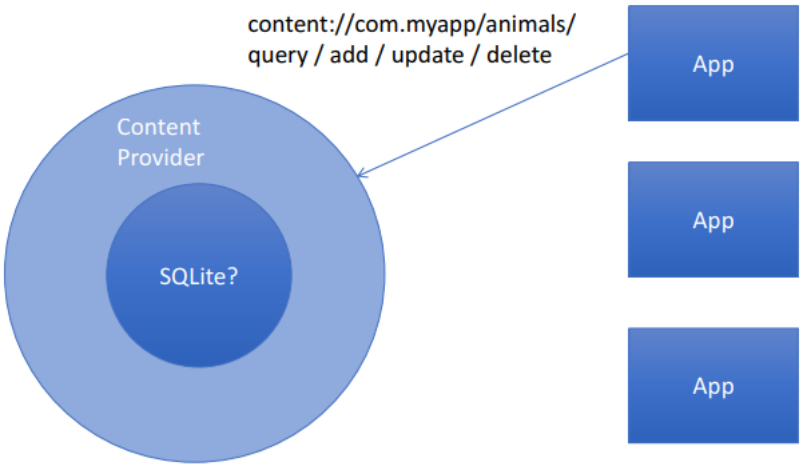
- A collection of records
- Support for reading and writing
- Support typical database operations: CRUD

Records are stored in rows, with each column providing different data fields

- Each record has a numeric id (in the field `_ID`) that uniquely identifies it

Tables exposed via URI

- Abstraction again, can be close to or distant from underlying storage
- Most of the work is specifying the abstraction / linkage



5.4 Querying a Content Provider

ContentResolver

- Manages and supports Content Provider access
- How to service a request for content (Similar to ServiceManager)
- Enables Content Providers to be used across multiple applications (Provides additional services such as change notification)
- Can observe a Content Provider to be informed of real-time modifications
  - A new MP3 has been added to the library
  - ContentObserver

Content Providers identify data sets through URIs

- **content://authority/path/id**
- **content**: Data managed by a Content Provider
- **authority**: ID for the Content Provider (i.e. fully qualified class name, com.example.martindata)
- **path**: 0 or more segments indicating the subset of data to be requested e.g. table name, or something more readable / abstracted. RESTful resource philosophy
- **id**: Specific record (row) being accessed

5.5 Querying a Content Provider

URI for searching Contacts.
`ContactsContract.Contacts.CONTENT_URI = "content://com.android.contacts/contacts/"`
`ContentResolver.query(...)`:

- Returns a Cursor instance for accessing results
- *Cursor* is a pointer to a *CursorWindow*
- A read-only reference to shared memory allocated by ashmem, retrieved via Binder
- Has to be closed and **max CursorWindow size is 2MB**. (Because it's inside shared memory)

```
Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
```

5.5.1 Accessing content

To access / modify Contacts, requires a Permission

- `android.permission.READ_CONTACTS`
- `android.permission.WRITE_CONTACTS`

Contacts has three components

- Data: Rows (mime-typed) that can hold personal information
- RawContacts: A contact for a given person from a given system. Gmail contact, Facebook contact etc, associated with Data entries
- Contacts: Aggregated RawContacts, single view of a person

5.6 Modifying a Content Provider

Done by using `update` function. We pass **Uri** and **ContentValues**, which is key value pairs, where key is name of the column.

5.7 Creating a Content Provider

Implement a storage system for the data:

- Structured data / SQLite (Values, binary blobs up to 64k, database)
- Large binary blobs (Files)
- Photos / media manager

Implement a ContentProvider

- `query, add, update, insert` etc
- `onCreate`
- `getType`. What type of data are we providing? Single item, multiple items, mime types
- ParcelFileDescripter `openFile()`

6 Contract

- Defines metadata pertaining to the provider
- Constant definitions that are exposed to developers via a compiled .jar file
  - Authority (Which app is responsible for this data)
  - URI schema
  - Meta-data types
  - Column names

7 URI Matching

All of these methods take a URI as the first parameter

- (except for onCreate)
- The object will need to parse it to some extent to know what to return, insert or update

`android.content.UriMatcher`: Provides mapping between abstraction of contract class to concrete db implementation

8 Network

Can use cloud storage (e.g. Google drive) by utilising network to push data off the device. To do this we implement `SyncAdapter`

- Synchronizes a local database / content provider with a remote server
- Needs a content provider access to store data locally
- Needs an authenticator to access the data
- Wrapped in a service, so external processes can bind to it, triggering the synchronisation

Make use of a Service to push data in the background

Reference section

placeholder