UNIVERSITY *of* WASHINGTON

Android 250 - Lecture 5

Content Provider

Margaret Maynard-Reid

April 27, 2015

Agenda

- Master/Detail flow
- SQLite Database (review)
- Content Provider
- Loaders
- Homework 2 Requirements

Sample Code

- SampleMasterDetail - how to create Split Pane UI with Fragments
- SampleDatabase (review) - how to create a Sqlite Db
- SampleLoader - access an existing Provider with a CursorLoader
- SampleContentProvider - how to write your own Content Provider

Android Stories

- Google’s Smartwatches Now Let You Leave Your Phone at Home
- Android Developers Blog - Android Support Library 22.1 ← ActionBarActivity has been deprecated in favor of the new AppCompatActivity.

Review from Last Week

- Why do we use the Android Support Library?
- What happens when you press the Back button on device?
- How do you add a fragment to the Back Stack?
- What is the difference between Up vs. Back?
- What is the method for drawing the Fragment UI?

Sample Code

- Walk through SampleMasterDetail
  - ListFragment ← make sure layout has a ListView with *android: id="@android:id/list"*
  - List item click to Details screen ← define interface in Fragment for communications
  - Split Pane for large screens ← alternative *activity_main.xlm* under *res/layout-sw600dp*

SQLite Database Review

- A way to persist data on device
- Local private data storage
- Content Provider Backing Storage
- You can use SQLite Browser (GUI) or SQLite3 (command line) to browse or query data directly
- To create a SQLite db -
    - Create a static schema class - *Contract.java
    - Create a class extend from SQLiteOpenHelper

SQLite Types

- NULL – A NULL value type
- INTEGER – A signed integer stored in 1, 2, 3, 4, 6, 8 bytes
- REAL – A floating point value stored as an 8-byte floating point number
- TEXT – A text string stored in UTF-8, UTF-16 BE or UTF-16 LE
- BLOB – A binary blob of data

Basic SQL

- SELECT – Select data from a table
- UPDATE – Update data from a table
- DELETE – Deletes data from a table
- INSERT – Inserts data into a table in a database
- CREATE {DATABASE | TABLE} – Creates a new element
- ALTER {DATABASE | TABLE} – Updates an element

*** https://www.sqlite.org/lang.html

Basic SQL Select

- SELECT {projection}
- FROM {table}
- WHERE {conditions}
- ORDERBY {argument}

Basic SQL

- SELECT * FROM books
- SELECT * FROM books WHERE _id=5
- SELECT _id, name FROM books ORDERBY name

SQLite Tools

- Sqlite3
- SQLite Browser – http://sqlitebrowser.org/
- SQLite Administrator – http://sqliteadmin.orbmu2k.de/
- Navicat
- Apps on Google Play
  - Access Database on SDCARD
  - Access via SuperUser on Rooted phone

SQLite Db Location

Where is the db located?

- Located on internal storage, associated with your app
- Open DDMS/File Explorer,
- Find db file under

  data/data/<package name>/databases

Sqlite3

- Open command line
- *adb shell*
- *cd data/data/<package name>/databases*
- ***sqlite3 dbname*** ← *invoke sqlite3 on database*
- ***.schema*** ← print the SQL CREATE statement for an existing table
- ***.tables*** ← list all the tables in db
- ***.exit*** ← exit sqlite3

Command line shell for SQLite - http://www.sqlite.org/cli.html

SQLite Browser

- Download SQLite Browser: http://sqlitebrowser.org/
- Get the db file from emulator

  adb pull <remote> <local>

- Open SQLite Browser and modify db
- Copy the updated db file back to emulator

  adb push <local> <remote>

SQLite Db Review - Sample Code

- Walk through SampleDatabase
  - Review how to create a SQLite Db
  - Review how to use SQLite3 to inspect db
  - Review how to use SQLite Browser
  - Best practice:
    - Use SQLite3 to make sure your db schema and data are created correctly
    - manually remove the db when changing schema

Break

Content Providers

- Part of an Android application
  - Defined in the application manifest as <provider>
  - Maps to a ContentProvider class in your project
- Provide managed and secured access to data
- They encapsulate the data with a consistent, standardized URI-based access
- Useful as a cross-process interface for data-sharing amongst running processes

Available System Content

- Browser
  - Bookmarks
- Calendar
  - Attendees
  - Events
  - Reminders
- CallLog
- ContactsContract
  - Name
  - Phones
  - Photos etc.

- MediaStore
  - Audio
    - Albums
    - Artists
    - Playlists
  - Images
  - Video
- Settings
- SyncState
- UserDictionary
- VoicemailContract

http://developer.android.com/reference/android/provider/package-summary.html

Access a Content Provider

- Get permission to the Content Provider

```
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

- Then access it via
  - ContentResolver or
  - CursorLoader

Access a Content Provider

android.content.ContentResolver (API 1)

- uri - The provider table used in resolution
- projection - The columns that should be returned
- selection - Criteria for selecting rows
- selectionArgs - Replace ? arguments from Selection
- sortOrder - The order of the rows returned

Access a Content Provider

android.content.CursorLoader (API 11)

- context - the current context
- uri - The provider table used in resolution
- projection - The columns that should be returned
- selection - Criteria for selecting rows
- selectionArgs - Replace ? arguments from Selection
- sortOrder - The order of the rows returned

What is a CursorLoader

An Android Loader built from AsyncTaskLoader

- Targets a ContentProvider
- Loads data asynchronously to prevent ANR (Application Not Responding)
- Handles the Cursor lifecycle

Loader

CursorLoader

Loader

Class for loading data

- If you subclass, you are responsible for
  - onStartLoading()
  - onStopLoading()
  - onForceLoad()
  - onReset()
- If you need asynchronous loading, subclass from AsyncTaskLoader

CursorLoader

- Subclass of AsyncTaskLoader used to query a ContentResolver and return a Cursor
- The most common Loader
- Built for offloading data work - run query in background thread
- Monitor and maintain a connection to your data through the lifecycle - i.e. automatically rerun the query when data associated with the query changes

LoaderManager

The manager of Loader instances

- Found in either an Activity or Fragment
- The controller in relation to the lifecycle
  - *initLoader(int id, Bundle args, LoaderCallbacks<D> callback)* ← call in onCreate() or onCreateView()
  - *restartLoader(int id, Bundle args, LoaderCallbacks<D> callback)*
  - *destroyLoader(int id)*
- LoaderCallbacks are fired when data is ready

LoaderCallbacks

Interface for handling LoaderManager updates

- Given the Loader ID, create a new instance
  Loader<D> onCreateLoader(int id, Bundle args)
- The data is loaded... do something with it
  void onLoadFinished(Loader<D> loader, D data)
- The data has become unavailable
  void onLoaderReset(Loader<D> loader)

Implementing a Loader

- Specialize the LoaderManager. LoaderCallbacks
- Get the LoaderManager
- Init the Loader as it makes sense for your app

Multiple Loaders

- The Loader instance is passed as the first param of onLoadFinished and onLoaderReset
- Use the ID to differentiate

Code Example

- It begins with initializing separate loaders with separate IDs

```
this.getLoaderManager().initLoader(1, null, this);
this.getLoaderManager().initLoader(2, null, this);
```

- The onCreateLoader callback needs to route based on ID

```
@Override
public Loader<String> onCreateLoader(int id, Bundle args) {
    switch (id) {
        case 1:
            return new OneLoader(this);
        case 2:
            return new TwoLoader(this);
    }
    return null;
}
```

Sample Code

- Walk through SampleLoader code:
  - Android Contacts Provider
  - Use a CursorLoader to load contacts data
  - Use a ListActivity (note we take care of empty list)
- Run SampleLoader app
  - Observe UI when there is an empty list
  - Add a contact on device/emulator, see change reflected in app
  - Delete a contact on device/emulator, see change reflected in app

Break

Content Provider

Providing Content

Providing Content

Why write your own Content Provider?

- Provide content to other apps
- Allow other apps to contribute to your content
- Include your content in the Search framework
- You want a RESTful interface to your own data

Providing Content

Define the following for a Content Provider

- **Content URI** - A URI that identifies data in a provider: Authority + Path
- **Content Authority** - Symbolic name of the provider
- **Content Path** - A name that points to a table or file (a path)
- **Content ID** - The optional id part points to an individual row in a table
- **Content Type** - What is actually stored in your provider

Provider Permissions

- In a Content Provider, by default, your data is public
- To change the permissions, set it on your <provider> declaration in the Manifest
    - Set for all the content of the provider
    - Set for the content of individual tables
    - Set for the content of individual records

<provider>

- android:permission
  A single permission for giving the ability for clients to read and write provider data
- android:readPermission
  Clients can read the provider data
- android:writePermission
  Clients can write to the provider data

Content Type

A Content Type is a defined MIME type

- Type
  - vnd.android.cursor.dir
    - Cursor may contain multiple items
  - vnd.android.cursor.item
    - Cursor should only contain 1 item
- Subtype
  - Potentially a MIME subtype
  - Usually something custom to your application

Content Type Examples

- text/plain
- text/html
- audio/mpeg
- video/quicktime
- vnd.android.cursor.item/phone
- vnd.android.cursor.dir/vnd.my.awesome.type

UriMatcher

- Convenience class
- Matches incoming Uri to integer values
  - * Matches a string of any valid characters of any length.
  - # Matches a string of numeric characters of any length.

```
private static final UriMatcher sUriMatcher = createUriMatcher();

private static UriMatcher createUriMatcher() {
    final UriMatcher uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    final String authority = AndroidContract.CONTENT_AUTHORITY;
    uriMatcher.addURI(authority, AndroidContract.PATH_VERSION, VERSION);
    uriMatcher.addURI(authority, AndroidContract.PATH_VERSION + "/#", VERSION_ID);

    return uriMatcher;
}
```

ContentProvider Class

These are the main methods to override:

- onCreate() - initialize the provider
- query() - return data to caller
- insert() - insert data to content provider
- update() - update existing data
- delete() - delete data from content provider
- getType() - return MIME type of the data in provider

Summary

To write your own Content Provider:

- Create a class that extends from ContentProvider
  - Should match the name attribute in the Manifest
  - Create Content Authority, Path, Content Uri, and Content Type
    - Add to Provider class If only one table in DB
    - Add to DbContract class if multiple tables in DB
  - Use UriMatcher to match Uri pattern to integers
  - Implement the required methods
- Define your ContentProvider in the Manifest

Sample Code

- Walk through SampleContentProvider
  - Create a Content Provider
  - Use Fragments for UI
  - Use SimpleCursorAdapter
  - Use CursorLoader to load data

Homework 2

- Due on May 18, 2015
- Go over homework 2 requirements
  - Use fragments for UI
  - Master-detail flow: split pane for large screens
  - Use SQLite DB to store data
  - Use Content Provider & CursorLoader