Content Providers

Steven R. Bagley

Data

- Android stores data
  - In files (both internal and external storage)
  - In Databases
- Local to the app

Sharing Data

- Useful to be able to share data across apps
- Allows for standard locations to store Contacts etc…
- But we don't want to share access to files
- Android permits data to be shared by using `ContentProviderS`
- How accessible (read and/or write) is up to the app

Content Providers

- Are part of an app
- And are code
- Means that the app has full control of how its data is presented
- And how it is modified
- Called from other apps (other processes)
- So needs to be thread-safe

Content Providers

- Either create a new one (by sub-classing `ContentProvider`)
- Or add your data to an existing `ContentProvider`
- Android has default providers for common data, such as contacts, video, images, music, etc…

Content Provider Data Model

- `ContentProvider`s enforce a specific data model
- Very similar to a relational database table
- 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

Accessing a ContentProvider

- Don't instantiate a `ContentProvider` sub-class directly
- Instead access is mediated through a `ContentResolver`
- This is passed a URI to the content provider we want (which begins `content://`)

Content URIs

- Format of the Content URI:
  `content://com.example.provider/trains/123`
- `content://` — Standard prefix
- `com.example.provider` — The authority
  Identifies the `ContentProvider`, should be a fully-qualified classname to ensure uniqueness. Must match `<provider>` element `@android:authority` in manifest

Content URIs

- Format of the Content URI:
  `content://com.example.provider/trains/123`
- `trains` — path part
  Identifies the specific data being requested
  can be zero or more segments long
- `123` — the ID of a specific record, if necessary

Querying a Content Provider

- Either `getContentResolver()` and call `query()` method
- Or more simply use `Activity.managedQuery()` method
- This takes care of the query's lifetime automatically
- Returned a `Cursor` object as with SQLite

Querying a Content Provider

- `Cursor managedQuery (Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)`
- `query()` and `Activity.managedQuery()` methods parameters similar to SQLite queries
- `uri` is the `Uri` to the content provider we want to query

Accessing Contacts

- Android provides the Contacts as `ContentProvider`
- `Uri` is defined as a constant: `ContactsContract.Contacts.CONTENT_URI`
- Need to specify that our app has permission (`android.permission.READ_CONTACTS`) in manifest

Go access the contacts list from our SQLite program

Creating our own `ContentProvider`

- Create a system to store data (e.g. our student table)
- Subclass `ContentProvider`
- Declare the `ContentProvider` in the manifest
- Provide implementation…
- Must be thread-safe…

We'll go through what we need -- then look at the Notepad sample to see how its done in practice

`ContentProvider` Impl

- Six abstract methods:
  - `query()`
  - `insert()`
  - `update()`
  - `delete()`
  - `getType()`
  - `onCreate()`

URI processing

- All these methods (except `onCreate()`) take `Uri` as the first parameter
- The object will need to parse it to some extent to know what to return, insert or update
- Android provides `android.content.UriMatcher` to simplify this

`UriMatcher`

- Matches a portion of a URI to an integer
- Can then `switch` easily on the integer to handle the data

- Create a new `UriMatcher` object — set all the template `Uri`s and the integer they match too
- Then call `match`() on a `Uri` to get the `int`