Access Contacts Data

CS443 – Mobile Applications
Instructor: Bo Sheng

Outline

• Use CursorLoader

• Retrieve Contact Data
  – Retrieve a list of contacts
  – Retrieve details for a contact

Introduction

• Contacts Provider in Android
  – Central repository of the user's contacts information, including data from contacts apps and social networking apps
  – You can access Contacts Provider information directly by calling ContentResolver methods or by sending intents to a contacts app

CursorLoader

• Query a ContentProvider in the background
  – Querying directly from an Activity may block the UI thread

• Use CursorLoader
  – Use the LoaderCallbacks interface
  – Initialize the Query
    • When user input is ready, or in onCreate() if you don't need user input
    • getLoaderManager().initLoader(URL_LOADER, null, this);

CursorLoader

• Start a Query
  – As soon as the background framework is initialized, it calls your implementation of onCreateLoader()
  – To start the query, return a CursorLoader from this method

CursorLoader

• Start a Query
  – When the query is started, the data is loaded, and the results are passed to your LoaderCallbacks interface

  ```java
  public Loader<Cursor> onCreateLoader(int id, Bundle arguments) {
      switch (id) {
      case SQLITE: // Default activity content
          mDb = mDbHelper.getWritableDatabase();
          return CursorLoader.getLoader(mDb, null, null, null, null, null);
      case URL_LOADER: // Optional content
          return CursorLoader.getLoader(mDb, null, null, null, null, null);
      case CONTENT: // Optional activity content
          return CursorLoader.getLoader(mDb, null, null, null, null, null);
      case CONTACTS: // Optional activity content
          return CursorLoader.getLoader(mDb, null, null, null, null, null);
      }
      return null;
  }
  ```
CursorLoader

- Start a Query
  - As soon as the background framework is initialized, it calls your implementation of `onCreateLoader()`
  - To start the query, return a `CursorLoader` from this method
  - Once the background framework has the object, it starts the query in the background
  - When the query is done, the background framework calls `onLoadFinished()`

CursorLoader

- Handle the Query Results
  - One of the incoming arguments of `onLoadFinished()` to this method is a `Cursor` containing the query results
  - Use a `View` class that implements `AdapterView` and provide the view with an adapter that implements `CursorAdapter`
    - You can set up the linkage between the view and adapter before you have any data to display
    - Then move a `Cursor` into the adapter in `onLoadFinished()`

CursorLoader

- Handle the Query Results

```java
// Define a handle to a Listview
ListView adapter = (ListView) findViewById(R.id.listview);
// Define a SimpleCursorAdapter for the ListView
SimpleCursorAdapter adapter = new SimpleCursorAdapter(
    MainActivity.this, // Current context
    R.layout.listitem, // Layout for a single row
    cursor, // Cursor to use
    new int[] { R.id.textview, // Columns to use in list
                R.id.flags } // Flags
);
// Bind the adapter for the view
adapter.setViewAdapter(adapter);
```

CursorLoader

- Handle the Query Results

```java
// Define the callback that CursorLoader calls when it's finished the query
private void onLoadFinishedLoader(CursorLoader cursorLoader, Cursor cursor) {
    // Move the query results into the adapter, causing the
    // ListView containing this adapter to re-display
    adapter.changeCursor(cursor);
}
```

CursorLoader

- Delete Old Cursor References
  - The `CursorLoader` is reset whenever its `Cursor` becomes invalid
  - Usually caused by data changes
  - Before re-running the query, the framework calls your implementation of `onLoaderReset()`
  - In this callback, you should delete all references to the current `Cursor`

```
@Override
public void onLoaderReset(@NonNull CursorLoader cursorLoader) {
    // Clear the adapter's reference to the Cursor
    // This prevents memory leaks
    // (Note: this assumes the CursorLoader has access to the View's Context)
    adapter.changeCursor(null);
}
```

Retrieving a List of Contacts

- Match contact names
  - Retrieve a list of contacts by matching the search string to all or part of the contact name data

- Match a specific type of data, such as a phone number
  - Retrieve a list of contacts by matching the search string to a particular type of detail data such as an email address.

- Match any type of data
  - Retrieve a list of contacts by matching the search string to any type of detail data, including name, phone number, street address, email address, and so forth
Retrieving a List of Contacts

- **Request Permission to Read the Provider**
  - your app must have `READ_CONTACTS` permission
  ```xml
  <uses-permission android:name="android.permission.READ_CONTACTS"/>
  ```

- **Settings**
  - Data table: `ContactsContract.Contacts`
  - Display the results in a `ListView`

  ```xml
  <xml version="1.0" encoding="utf-8"?>
  <data android:label="Contacts"/>
  <item android:key="android.permission.READ_CONTACTS" android:name="Contacts.Contract"/>
  <item android:key="android.permission.WRITE_CONTACTS" android:name="Contacts.Contract"/>
  </xml>
  ```

- **Structure**
  - `ListView` with `Adapter` and `CursorLoader`
  - Display columns:
    - `Contacts._ID`
    - `Contacts.DISPLAY_NAME`
    - `Contacts.TEL Numbers`

- **Prepare the CursorLoader**
  - Define a projection
    - Contains the columns you want to return from your query
  - The column `Contacts._ID` is used by the `SimpleCursorAdapter` binding process
Retrieving a List of Contacts

- Initialize the CursorLoader
  - getLoaderManager().initLoader(0, null, this);
- Implement onCreateLoader() and onLoadFinished()

Outline

- Use CursorLoader
  - Retrieve Contact Data
    - Retrieve a list of contacts
    - Retrieve details for a contact

Retrieve Details for a Contact

- Contacts Provider Organization
  - ContactsContract.Contacts table
    - Rows representing different people, based on aggregations of raw contact rows.
  - ContactsContract.RawContacts table
    - Rows containing a summary of a person's data, specific to a user account and type.
  - ContactsContract.Data table
    - Rows containing the details for raw contact, such as email addresses or phone numbers.

- Raw contact
  - A raw contact represents a person's data coming from a single account type and account name
  - The Contacts Provider allows multiple raw contacts for the same person

  Most of the data for a raw contact isn’t stored in the ContactsContract.RawContacts table. Instead, it’s stored in one or more rows in the ContactsContract.Data table
  - Each data row has a column Data.RAW_CONTACT_ID that contains the RawContacts._ID value of its parent ContactsContract.RawContacts row

Retrieve Details for a Contact

- Illustration of Contact, Raw contact, and Data
Retrieve Details for a Contact

• Important columns in ContactsContract.RawContacts

<table>
<thead>
<tr>
<th>Column</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNT_NAME</td>
<td>The account name for the account type that is the source of this raw contact. For example, the account name of a Google account is one of the device owner’s Gmail addresses. The format of this name is specific to its account type. It is not necessarily an email address. Always qualify your account type with a domain identifier for a domain you own or control. This will ensure that your account type is unique. An account type that offers contacts data usually has an associated sync adapter that synchronizes with the Contacts Provider. This flag allows the Contacts Provider to maintain the row internally until sync adapters are able to delete the row from their servers and then finally delete the row from the repository.</td>
</tr>
<tr>
<td>ACCOUNT_TYPE</td>
<td>The account type that’s the source of this raw contact. For example, the account type of a Google account is com.google. Always qualify your account type with a domain identifier for a domain you own or control. This will ensure that your account type is unique. An account type that offers contacts data usually has an associated sync adapter that synchronizes with the Contacts Provider.</td>
</tr>
<tr>
<td>DELETED</td>
<td>The “deleted” flag for a raw contact. Any example – Consider “Emily Dickinson” has the following three user accounts defined on her device: • <a href="mailto:emily.dickinson@gmail.com">emily.dickinson@gmail.com</a> • <a href="mailto:emilyd@gmail.com">emilyd@gmail.com</a> • Twitter account “belle_of_amherst” – She has enabled Sync Contacts for all three of these accounts – Suppose Emily Dickinson logs into Gmail as <a href="mailto:emily.dickinson@gmail.com">emily.dickinson@gmail.com</a>, opens Contacts, and adds “Thomas Higginson”. – Later on, she logs into Gmail as <a href="mailto:emilyd@gmail.com">emilyd@gmail.com</a> and sends an email to &quot;Thomas Higginson&quot;, which automatically adds him as a contact. – She also follows &quot;colonel_tom&quot; (Thomas Higginson’s Twitter ID) on Twitter.</td>
</tr>
</tbody>
</table>

Retrieve Details of a Contact

• Data of a raw contact

– Descriptive column names

  • RAW_CONTACT_ID: The value of the _ID column of the raw contact for this data.
  • MIMETYPE: The type of data stored in this row, expressed as a custom MIME type.
  • IS_PRIMARY: This column flags the data row that contains the primary data for the type.

– Generic column names

  • Another 4 generic columns SYNC1 through SYNC4 that should only be used by sync adapters
  • The DATA1 column is indexed

Data of a raw contact

– The data is stored in a ContactsContract.Data row that is linked to the raw contact’s _ID value

  • E.g., multiple emails of a person

  • Different types of data are stored in the same table

  • Display name, phone number, email, postal address, photo, and website detail rows are all found in the ContactsContract.Data table

  • To help manage this, the ContactsContract.Data table has some columns with descriptive names, and others with generic names.

  • The contents of a descriptive-name column have the same meaning regardless of the type of data in the row, while the contents of a generic-name column have different meanings depending on the type of data.

Data of a raw contact

– Generic column names

  • 15 generic columns named DATA1 through DATA15

  • Another 4 generic columns SYNC1 through SYNC4 that should only be used by sync adapters

  • The DATA1 column is indexed

  • E.g., in an email row, this column contains the actual email address.

  • By convention, the column DATA15 is reserved for storing Binary Large Object (BLOB) data such as photo thumbnails
Retrieve Details for a Contact

- Data of a raw contact
  - Type-specific column names, defined in subclasses of ContactsContract.CommonDataKinds
  - E.g., the ContactsContract.CommonDataKinds.Email class defines type-specific column name constants for a ContactsContract.Data row that has the MIME type Email.CONTENT_ITEM_TYPE
  - The class contains the constant ADDRESS for the email address column. The actual value of ADDRESS is "data1"

- The Contacts Provider combines the raw contact rows across all account types and account names to form a contact.
- The Contacts Provider maintains contacts automatically
  - Applications are not allowed to add contacts
  - If you try to add a contact to the Contacts Provider with an insert(), you'll get an UnsupportedOperationException exception
- The Contacts Provider links a contact row to its raw contact rows with the contact row’s _ID column in the Contacts table
- The ContactsContract.Contacts table also has the column LOOKUP_KEY that is a "permanent" link to the contact row

Retrieve Details for a Contact

- Set up a projection
  - Always retrieve Data._ID if you're binding the result Cursor to a ListView; otherwise, the binding won’t work.
  - Also retrieve Data.MIMETYPE so you can identify the data type of each row you retrieve

Define the selection criteria

- Use the Contacts.LOOKUP_KEY column to find the contact

- Define the sort order

Initialize the Loader
Retrieve Details for a Contact

• Implement onCreateLoader()

```java
public void onCreateLoader(int loaderId, Bundle args) {
    // Create the proper service
    Intent queryIntent = new Intent(Intent.ACTION_VIEW);
    // Replace the selection parameters
    queryIntent.setData(ContactsContract.CommonDataKinds.Email.CONTENT_URI);
    queryIntent.putExtra(ContactsContract.CommonDataKinds.Email.ADDRESS, args.getString(ADDRESS_KEY));
    queryIntent.putExtra(ContactsContract.CommonDataKinds.Email.TYPE, args.getInt(TYPE_KEY));
    queryIntent.putExtra(ContactsContract.CommonDataKinds.Email_LABELS, args.getInt(LABELS_KEY));
    queryIntent.putExtra(ContactsContract.CommonDataKinds.Email_ID, args.getInt(ID_KEY));
    Loader loader = LoaderFactory.createLoader(loaderId, contentResolver, queryIntent, null);

    // Handle the loader
    if (loader == null) {
        return;
    }

    // Load the data
    loader.loadInBackground();
}
```

Retrieve Specific Details (e.g., just emails)

– Specify a projection

```java
private String[] REQUEST_PROJECTIONS = {
    ContactsContract.CommonDataKinds.Email.ADDRESS,
    ContactsContract.CommonDataKinds.Email.TYPE
};
```

– Specify the matching type

```java
private String[] REQUEST_SELECTION = {
    ContactsContract.CommonDataKinds.Email.MIMETYPE
};
```

– Selection: Modify the selection text to search for the MIMETYPE value that's specific to your data type.

– Sort order: Since you're only selecting a single detail type, don't group the returned Cursor by Data.MIMETYPE.