Loaders

Contents

1 Loaders

1.1 What's a Loader?

Loaders make it easy to load data asynchronously in an activity or fragment. Loaders have these characteristics:

- They are available to every Activity and Fragment.
- They provide asynchronous loading of data.
- They monitor the source of their data and deliver new results when the content changes.
- They automatically reconnect to the last loader's cursor when being recreated after a configuration change. Thus, they don't need to re-query their data.

Loaders were introduced in Honeycomb (API 11).

- The Android Support Package includes support for loaders. By including the support package in your application, you can use loaders even if your application for a `minSdkVersion` of 4 or later.

1.2 The Loader Classes and Interfaces

There are several classes and interfaces that implement the loader functionality:

`LoaderManager`
An abstract class associated with an `Activity` or `Fragment` for managing one or more `Loader` instances. There is only one `LoaderManager` per activity or fragment. But a `LoaderManager` can have multiple loaders.

`LoaderManager.LoaderCallbacks`
A callback interface for a client to interact with the `LoaderManager`.

`Loader`
An abstract class that performs asynchronous loading of data.

`AsyncTaskLoader`
Abstract loader that provides an `AsyncTask` to do the work.

`CursorLoader`
A subclass of `AsyncTaskLoader` that queries the `ContentResolver` and returns a `Cursor`. This class implements the `Loader` protocol in a standard way for querying cursors, building on `AsyncTaskLoader` to perform the cursor query on a background thread so that it does not block the application's UI.

As of API 15, the only concrete `Loader` implementation is the `CursorLoader`, which can query only a content provider; it cannot run a query directly against a SQLite database.

1.3 Using Loaders in an Application

An application that uses loaders typically includes the following:

- An `Activity` or `Fragment`.
- An instance of the `LoaderManager`.
- A `CursorLoader` to load data backed by a `ContentProvider`. Alternatively, you can implement your own subclass of `Loader` or `AsyncTaskLoader` to load data from some other source.

- A data source, such as a `ContentProvider`, when using a `CursorLoader`.
- An implementation for `LoaderManager.LoaderCallbacks`. This is where you create new loader instances and manage your references to existing loaders.
- A way of displaying the loader's data, such as a `SimpleCursorAdapter`.

1.4 Accessing the `LoaderManager`

To use loaders in an activity or fragment, you need an instance of `LoaderManager`.

- There is only one `LoaderManager` per activity or fragment.

If you're using the standard APIs, invoke `getLoaderManager()` as provided in the `Activity` and `Fragment` classes.

If you're using the Support Package, you must use `android.support.v4.app.FragmentActivity` as the base class for your activity.

- Invoke `FragmentActivity.getSupportLoaderManager()` to obtain a `LoaderManager` for the activity.
- Invoke `android.support.v4.app.Fragment.getLoaderManager()` to obtain a `LoaderManager` for a fragment.

1.5 Starting a Loader

Invoke `FragmentManager.initLoader()` to initialize a specified `Loader`:

+

```
// Prepare the loader.  Either re-connect with an existing one,
// or start a new one.
getLoaderManager().initLoader(0, null, this);
```

The `initLoader()` method takes the following parameters:

- A unique integer ID that identifies the loader. In this example, the ID is `0`.
- Optional arguments in the form of a `Bundle` to supply to the loader at construction (`null` in this example).
- A `LoaderManager.LoaderCallbacks` implementation, which the `LoaderManager` calls to report loader events. In this example, the local class implements the `LoaderManager.LoaderCallbacks` interface, so it passes a reference to itself, `this`.

The `initLoader()` call ensures that a loader is initialized and active. It has two possible outcomes:

- If the loader specified by the ID already exists, the last created loader is reused.
- If the loader specified by the ID does not exist, `initLoader()` triggers the `LoaderManager.LoaderCallbacks` method `onCreateLoader()`. This is where you implement the code to instantiate and return a new loader.

You typically initialize a `Loader` within an activity's `onCreate()` method, or within a fragment's `onActivityCreated()` method.

1.6 Restarting a Loader

If you want to discard the the old data returned by a `Loader` and have it load fresh data, invoke the `FragmentManager.resetLoad`
method.

```
// Refresh the loader with new data
getLoaderManager().resetLoader(0, null, this);
```

The `resetLoader()` method accepts the same arguments as `initLoader()`:

- The integer ID of a `Loader`
- Optional arguments in the form of a `Bundle`
- A `LoaderManager.LoaderCallbacks` implementation

1.7 The `LoaderManager.LoaderCallbacks` Listener

The `LoaderManager.LoaderCallbacks` listener must implement the following interface:

```
public interface LoaderCallbacks<DataType> {
        public Loader<DataType> onCreateLoader(int id, Bundle args);
        public void onLoadFinished(Loader<DataType> loader, DataType data);
        public void onLoaderReset(Loader<DataType> loader);
}
```

`onCreateLoader()`
The `LoaderManager` invokes this method if it needs to create the `Loader` with the specified `id`. The `LoaderManager` also passes along the `Bundle` argument it received in the `LoaderManager.initLoader()` method.

The listener must instantiate the specified loader and return a reference to it. The `LoaderManager` then manages interaction with the `Loader` as required.

`onLoadFinished()`
The `LoaderManager` invokes this method when the loader has finished loading the requested data. It provides a reference to the `data` and the `loader` that generated it.

The listener should then use the data in whatever way the fragment or activity requires. For example, if a `CursorLoader` provides a `Cursor` as a result of a query, the `onLoadFinished()` method might hook up the `Cursor` to a `CursorAdapter`
to display the data.

`onLoaderReset()`
The `LoaderManager` invokes this method when a loader is being reset. It indicates that the data provided by the loader is becoming unavailable. At this point, the listener should remove any references it has to the loader's data.

Important
The loader "owns" the data is provides. The listener should not attempt to release or delete the data. For example, a `CursorLoader` provides a `Cursor`, the consumer of the `Cursor` should not attempt to close it; it should simply release its reference to the `Cursor` in response to the `onLoaderReset()` method call.

1.8 The `CursorLoader`

The `CursorLoader` is a concrete loader implementation that queries the `ContentResolver` and returns a `Cursor`.

- Typically, the only interaction your listener implementation needs to have with a `CursorLoader` is to instantiate it in response to an `onCreateLoader()` call.

- The `CursorLoader` constructor take a `Context` followed by the same arguments as `ContentResolver.query()`:

  `uri`
  The URI, using the `content://` scheme, for the content to retrieve.

  `projection`
  A list of which columns to return. Passing `null` returns all columns, which is inefficient.

  `selection`
  A filter declaring which rows to return, formatted as an SQL `WHERE` clause (excluding the `WHERE` itself). Passing `null` returns all rows for the given URI.

`selectionArgs`: You may include `?`s in `selection`, which will be replaced by the values from `selectionArgs`, in the order that they appear in the selection.

`sortOrder`
How to order the rows, formatted as an SQL `ORDER BY` clause (excluding the `ORDER BY` itself). Passing `null` use the default sort order.

1.9 A Simple Example

```
public static class CursorLoaderListFragment extends ListFragment
        implements MenuItem.OnMenuItemClickListener,
                LoaderManager.LoaderCallbacks<Cursor> {

    // This is the Adapter being used to display the list's data.
    SimpleCursorAdapter mAdapter;

    @Override public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);

        // Give some text to display if there is no data.  In a real
        // application this would come from a resource.
        setEmptyText("No phone numbers");

        // Create an empty adapter we will use to display the loaded data.
        mAdapter = new SimpleCursorAdapter(getActivity(),
                android.R.layout.simple_list_item_2, null,
                new String[] { Contacts.DISPLAY_NAME, Contacts.CONTACT_STATUS },
                new int[] { android.R.id.text1, android.R.id.text2 }, 0);
        setListAdapter(mAdapter);

        // Prepare the loader.  Either re-connect with an existing one,
        // or start a new one.
        getLoaderManager().initLoader(0, null, this);
    }

    @Override public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        // Place an action bar item for searching.
        MenuItem item = menu.add("Refresh");
        item.setIcon(android.R.drawable.ic_menu_rotate);
        item.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
                item.setOnMenuItemClickListener(this);
    }

    public boolean onMenuItemClick(String newText) {
        // Called when the user clicks the refresh button.
        getLoaderManager().restartLoader(0, null, this);
        return true;
    }
    // These are the Contacts rows that we will retrieve.
    static final String[] CONTACTS_SUMMARY_PROJECTION = new String[] {
        Contacts._ID,
        Contacts.DISPLAY_NAME,
        Contacts.CONTACT_STATUS,
        Contacts.CONTACT_PRESENCE,
        Contacts.PHOTO_ID,
        Contacts.LOOKUP_KEY,
    };

    public Loader<Cursor> onCreateLoader(int id, Bundle args) {
        // This is called when a new Loader needs to be created.  This
        // sample only has one Loader, so we don’t care about the ID.
        Uri baseUri = Contacts.CONTENT_URI;

        // Now create and return a CursorLoader that will take care of
        // creating a Cursor for the data being displayed.
        String select = "((" + Contacts.DISPLAY_NAME + " NOTNULL) AND ("
                + Contacts.HAS_PHONE_NUMBER + "=1) AND ("
                + Contacts.DISPLAY_NAME + " != ’’ ))";
        return new CursorLoader(getActivity(), baseUri,
                CONTACTS_SUMMARY_PROJECTION, select, null,
                Contacts.DISPLAY_NAME + " COLLATE LOCALIZED ASC");
    }

    public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
        // Swap the new cursor in.  (The framework will take care of closing the
        // old cursor once we return.)
        mAdapter.swapCursor(data);
    }

    public void onLoaderReset(Loader<Cursor> loader) {
        // This is called when the last Cursor provided to onLoadFinished()
        // above is about to be closed.  We need to make sure we are no
        // longer using it.
        mAdapter.swapCursor(null);
    }
}
```