# 加载器

本文内容



关键类

- `LoaderManager`
- `Loader`

相关示例

- LoaderCursor
- LoaderThrottle

Android 3.0 中引入了加载器，支持轻松在 Activity 或片段中异步加载数据。 加载器具有以下特征：

- 可用于每个 `Activity` 和 `Fragment`。
- 支持异步加载数据。
- 监控其数据源并在内容变化时传递新结果。
- 在某一配置更改后重建加载器时，会自动重新连接上一个加载器的游标。 因此，它们无需重新查询其数据。

## Loader API 摘要

在应用中使用加载器时，可能会涉及到多个类和接口。 下表汇总了这些类和接口：

| 类/接口 | 说明 |
| --- | --- |
| `LoaderManager` | 一种与 `Activity` 或 `Fragment` 相关联的的抽象类，用于管理一个或多个 `Loader` 实例。 这有助于应用管理与 `Activity` 或 `Fragment` 生命周期相关联的、运行时间较长的操作。它最常见的用法是与 `CursorLoader` 一起使用，但应用可自由写入其自己的加载器，用于加载其他类型的数据。<br><br>每个 Activity 或片段中只有一个 `LoaderManager`。但一个 `LoaderManager` 可以有多个加载器。 |
| `LoaderManager.LoaderCallbacks` | 一种回调接口，用于客户端与 `LoaderManager` 进行交互。例如，您可使用 `onCreateLoader()` 回调方法创建新的加载器。 |
| `Loader` | 一种执行异步数据加载的抽象类。这是加载器的基类。 您通常会使用 `CursorLoader`，但您也可以实现自己的子类。加载器处于活动状态时，应监控其数据源并在内容变化时传递新结果。 |
| `AsyncTaskLoader` | 提供 `AsyncTask` 来执行工作的抽象加载器。 |
| `CursorLoader` | `AsyncTaskLoader` 的子类，它将查询 `ContentResolver` 并返回一个 `Cursor`。此类采用标准方 |

| | |
|---|---|
| | 式为查询游标实现 `Loader` 协议。它是以 `AsyncTaskLoader` 为基础而构建，在后台线程中执行游标查询，以免阻塞应用的 UI。使用此加载器是从 `ContentProvider` 异步加载数据的最佳方式，而不用通过片段或 Activity 的 API 来执行托管查询。 |

上表中的类和接口是您在应用中用于实现加载器的基本组件。 并非您创建的每个加载器都要用到上述所有类和接口。但是，为了初始化加载器以及实现一个 `Loader` 类（如 `CursorLoader`），您始终需要要引用 `LoaderManager`。 下文将为您展示如何在应用中使用这些类和接口。

# 在应用中使用加载器

此部分描述如何在 Android 应用中使用加载器。使用加载器的应用通常包括：

- `Activity` 或 `Fragment`。
- `LoaderManager` 的实例。
- 一个 `CursorLoader`，用于加载由 `ContentProvider` 支持的数据。您也可以实现自己的 `Loader` 或 `AsyncTaskLoader` 子类，从其他源中加载数据。
- 一个 `LoaderManager.LoaderCallbacks` 实现。您可以使用它来创建新加载器，并管理对现有加载器的引用。
- 一种显示加载器数据的方法，如 `SimpleCursorAdapter`。
- 使用 `CursorLoader` 时的数据源，如 `ContentProvider`。

## 启动加载器

`LoaderManager` 可在 `Activity` 或 `Fragment` 内管理一个或多个 `Loader` 实例。每个 Activity 或片段中只有一个 `LoaderManager`。

通常，您会在 Activity 的 `onCreate()` 方法或片段的`onActivityCreated()` 方法内初始化 `Loader`。您执行操作如下：

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

`initLoader()` 方法采用以下参数：

- 用于标识加载器的唯一 ID。在此示例中，ID 为 0。
- 在构建时提供给加载器的可选参数（在此示例中为 `null`）。
- `LoaderManager.LoaderCallbacks` 实现， `LoaderManager` 将调用此实现来报告加载器事件。在此示例中，本地类实现 `LoaderManager.LoaderCallbacks` 接口，因此它会传递对自身的引用 `this`。

`initLoader()` 调用确保加载器已初始化且处于活动状态。这可能会出现两种结果：

- 如果 ID 指定的加载器已存在，则将重复使用上次创建的加载器。
- 如果 ID 指定的加载器*不*存在，则 `initLoader()` 将触发 `LoaderManager.LoaderCallbacks` 方法 `onCreateLoader()`。在此方法中，您可以实现代码以实例化并返回新加载器。有关详细介绍，请参阅 onCreateLoader 部分。

无论何种情况，给定的 `LoaderManager.LoaderCallbacks` 实现均与加载器相关联，且将在加载器状态变化时调用。如果在调用时，调用程序处于启动状态，且请求的加载器已存在并生成了数据，则系统将立即调用 `onLoadFinished()`（在 `initLoader()` 期间），因此您必须为此做好准备。 有关此回调的详细介绍，请参阅 onLoadFinished。

请注意，`initLoader()` 方法将返回已创建的 `Loader`，但您不必捕获其引用。`LoaderManager` 将自动管理加载器的生命周期。`LoaderManager` 将根据需要启动和停止加载，并维护加载器的状态及其相关内容。 这意味着您很少直接与加载器进行交互（有关使用加载器方法调整加载器行为的示例，请参阅 LoaderThrottle 示例）。当特定事件发生时，您通常会使用 `LoaderManager.LoaderCallbacks` 方法干预加载进程。有关此主题的详细介绍，请参阅使用 LoaderManager 回调。

## 重启加载器

当您使用 `initLoader()` 时（如上所述），它将使用含有指定 ID 的现有加载器（如有）。如果没有，则它会创建一个。但有时，您想舍弃这些旧数据并重新开始。

要舍弃旧数据，请使用 `restartLoader()`。例如，当用户的查询更改时，此 `SearchView.OnQueryTextListener` 实现将重启加载器。 加载器需要重启，以便它能够使用修订后的搜索过滤器执行新查询：

```
public boolean onQueryTextChanged(String newText) {
    // Called when the action bar search text has changed.  Update
    // the search filter, and restart the loader to do a new query
    // with this filter.
    mCurFilter = !TextUtils.isEmpty(newText) ? newText : null;
    getLoaderManager().restartLoader(0, null, this);
    return true;
}
```

# 使用 LoaderManager 回调

`LoaderManager.LoaderCallbacks` 是一个支持客户端与 `LoaderManager` 交互的回调接口。

加载器（特别是 `CursorLoader`）在停止运行后，仍需保留其数据。这样，应用即可保留 Activity 或片段的 `onStop()` 和 `onStart()` 方法中的数据。当用户返回应用时，无需等待它重新加载这些数据。您可使用 `LoaderManager.LoaderCallbacks` 方法了解何时创建新加载器，并告知应用何时停止使用加载器的数据。

`LoaderManager.LoaderCallbacks` 包括以下方法：

- `onCreateLoader()`：针对指定的 ID 进行实例化并返回新的 `Loader`
- `onLoadFinished()` ：将在先前创建的加载器完成加载时调用
- `onLoaderReset()`：将在先前创建的加载器重置且其数据因此不可用时调用

下文更详细地描述了这些方法。

## onCreateLoader

当您尝试访问加载器时（例如，通过 `initLoader()`），该方法将检查是否已存在由该 ID 指定的加载器。 如果没有，它将触发 `LoaderManager.LoaderCallbacks` 方法 `onCreateLoader()`。在此方法中，您可以创建新加载器。 通常，这将是 `CursorLoader`，但您也可以实现自己的 `Loader` 子类。

在此示例中，`onCreateLoader()` 回调方法创建了 `CursorLoader`。您必须使用其构造函数方法来构建 `CursorLoader`。该方法需要对 `ContentProvider` 执行查询时所需的一系列完整信息。具体地说，它需要：

- *uri*：用于检索内容的 URI
- *projection*：要返回的列的列表。传递 `null` 时，将返回所有列，这样会导致效率低下
- *selection*：一种用于声明要返回哪些行的过滤器，采用 SQL WHERE 子句格式（WHERE 本身除外）。传递 `null` 时，将为指定的 URI 返回所有行
- *selectionArgs*：您可以在 selection 中包含 ?s，它将按照在 selection 中显示的顺序替换为 *selectionArgs* 中的值。该值将绑定为字串符
- *sortOrder*：行的排序依据，采用 SQL ORDER BY 子句格式（ORDER BY 自身除外）。传递 `null` 时，将使用默认排序顺序（可能并未排序）

例如：

```
 // If non-null, this is the current filter the user has provided.
String mCurFilter;
...
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.
    // First, pick the base URI to use depending on whether we are
    // currently filtering.
    Uri baseUri;
    if (mCurFilter != null) {
        baseUri = Uri.withAppendedPath(Contacts.CONTENT_FILTER_URI,
                  Uri.encode(mCurFilter));
    } else {
        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");
}
```

### onLoadFinished

当先前创建的加载器完成加载时，将调用此方法。该方法必须在为此加载器提供的最后一个数据释放之前调用。 此时，您应移除所有使用的旧数据（因为它们很快会被释放），但不要自行释放这些数据，因为这些数据归其加载器所有，其加载器会处理它们。

当加载器发现应用不再使用这些数据时，即会释放它们。 例如，如果数据是来自 `CursorLoader` 的一个游标，则您不应手动对其调用 `close()`。如果游标放置在 `CursorAdapter` 中，则应使用 `swapCursor()` 方法，使旧 `Cursor` 不会关闭。例如：

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

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);
}
```

### onLoaderReset

此方法将在先前创建的加载器重置且其数据因此不可用时调用。 通过此回调，您可以了解何时将释放数据，因而能够及时移除其引用。

此实现调用值为 `null` 的`swapCursor()`：

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

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);
}
```

## 示例

以下是一个 `Fragment` 完整实现示例。它展示了一个 `ListView`，其中包含针对联系人内容提供程序的查询结果。它使用 `CursorLoader` 管理提供程序的查询。

应用如需访问用户联系人（如此示例中所示），其清单文件必须包括权限 READ_CONTACTS。

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

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

    // If non-null, this is the current filter the user has provided.
    String mCurFilter;

    @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");

        // We have a menu item to show in action bar.
        setHasOptionsMenu(true);

        // 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("Search");
        item.setIcon(android.R.drawable.ic_menu_search);
        item.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
        SearchView sv = new SearchView(getActivity());
        sv.setOnQueryTextListener(this);
        item.setActionView(sv);
    }

    public boolean onQueryTextChange(String newText) {
        // Called when the action bar search text has changed.  Update
        // the search filter, and restart the loader to do a new query
        // with this filter.
        mCurFilter = !TextUtils.isEmpty(newText) ? newText : null;
        getLoaderManager().restartLoader(0, null, this);
        return true;
    }

    @Override public boolean onQueryTextSubmit(String query) {
        // Don't care about this.
        return true;
    }

    @Override public void onListItemClick(ListView l, View v, int position, long id) {
        // Insert desired behavior here.
        Log.i("FragmentComplexList", "Item clicked: " + id);
    }

    // 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.
        // First, pick the base URI to use depending on whether we are
        // currently filtering.
        Uri baseUri;
        if (mCurFilter != null) {
            baseUri = Uri.withAppendedPath(Contacts.CONTENT_FILTER_URI,
                    Uri.encode(mCurFilter));
        } else {
            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);
    }
}
```

## 更多示例

**ApiDemos** 中还提供了一些不同的示例，阐述如何使用加载器：

- LoaderCursor：上述代码段的完整版本
- LoaderThrottle：此示例显示当数据变化时，如何使用限制来减少内容提供程序的查询次数

有关下载和安装 SDK 示例的信息，请参阅获取示例。