1、管理内存

How Android Manages Memory

Sharing Memory：系统自身有很多共享内存的方式，每个进程都分裂字Zygote进程，代码共享

Allocating and Reclaiming App Memory：每个App开始分配的内存有限，可增长，但是仍有系统App限制

Restricting App Memory： `getMemoryClass()` .

How Your App Should Manage Memory

Use services sparingly：The best way to limit the lifespan of your service is to use an `IntentService`: Leaving a service running when it's not needed is **one of the worst memory-management mistakes** an Android app can make.

- Create an IntentService

To create an `IntentService` component for your app, define a class that extends `IntentService` , and within it, define a method that overrides `onHandleIntent()` . For example:

```
public class RSSPullService extends IntentService {
    @Override
    protected void onHandleIntent(Intent workIntent) {
        // Gets data from the incoming Intent
        String dataString = workIntent.getDataString();
        ...
        // Do work here, based on the contents of dataString
        ...
    }
}
```

- Sending Work Requests to the Background Service

```
/*
 * Creates a new Intent to start the RSSPullService
 * IntentService. Passes a URI in the
 * Intent's "data" field.
 */
mServiceIntent = new Intent(getActivity(), RSSPullService.class);
mServiceIntent.setData(Uri.parse(dataUrl));
```

这种做法比较低端，不如直接传递数据，定义个一个静态方法，不知道这样又没哟坏处，程序以外死亡类似

```
public static void startActivity(Context context){
        Intent intent = new Intent(context, ActivityAccountEntrance.class);
        //非activity的context启动activity会出错
        if(!(context instanceof Activity))

            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        context.startActivity(intent);}
```

Once you call `startService()`, the `IntentService` does the work defined in its `onHandleIntent()` method, and then stops itself.

- Reporting Work Status

The recommended way to send and receive status is to use a `LocalBroadcastManager`, which limits broadcast `Intent` objects to components in your own app.

```
        /*
         * Instantiates a new action filter.
         * No data filter is needed.
         */
        statusIntentFilter = new IntentFilter(Constants.ACTION_ZOOM_IMAGE);

        ...
```

```
        // Registers the receiver with the new filter
        LocalBroadcastManager.getInstance(getActivity()).registerReceiver
(
                mDownloadStateReceiver,
                mIntentFilter);
```

- Loading Data in the Background

Querying a `ContentProvider` for data you want to display takes time.----Learn how to run a query in the background, using a `CursorLoader` .

Running a Query with a CursorLoader：A `CursorLoader` runs an asynchronous query in the background against a `ContentProvider` `LoaderManager.LoaderCallbacks<Cursor>`

- Managing Device Awake State：Certain apps need to keep the screen turned on, such as games or movie apps. The best way to do this is to use the `FLAG_KEEP_SCREEN_ON` in your activity `public class MainActivity extends Activity {`

```
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_
SCREEN_ON);
  }
```

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:keepScreenOn="true">
    ...
</RelativeLayout>
```

But if you want to explicitly clear the flag and thereby allow the screen to turn off again, use `clearFlags()` : `getWindow().clearFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON)` .

- Keep the CPU On
- Add the receiver to your app's manifest file with an intent filter that filters on the `ACTION_BOOT_COMPLETED` action:

```
<receiver android:name=".SampleBootReceiver"
        android:enabled="false">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"></action>
    </intent-filter>
</receiver>
```

Notice that in the manifest, the boot receiver is set to `android:enabled="false"` . This means that the receiver will not be called unless the application explicitly enables it. This prevents the boot receiver from being called unnecessarily. You can enable a receiver (for example, if the user sets an alarm) as follows:

```
ComponentName receiver = new ComponentName(context, SampleBootReceiver.class);
PackageManager pm = context.getPackageManager();

pm.setComponentEnabledSetting(receiver,
        PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
        PackageManager.DONT_KILL_APP);
```