1. Improving Layout Performance

Optimizing Layout Hierarchies

> For example, using nested instances of `LinearLayout` can lead to an excessively deep view hierarchy.
> that use the `layout_weight` parameter can be especially expensive as each child needs to be measured twice.

Inspect Your Layout

Re-using Layouts with <include/>

```
<include android:id=”@+id/news_title”
         android:layout_width=”match_parent”
         android:layout_height=”match_parent”
         layout=”@layout/title”/>
```

However, if you want to override layout attributes using the `<include>` tag, you must override both `android:layout_height` and `android:layout_width` in order for other layout attributes to take effect.

Use the <merge> Tag Merge更像是剥了皮插入 include是带着套插入

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

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/add"/>

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/delete"/>

</merge>
```

Now, when you include this layout in another layout (using the `<include/>` tag), the system ignores the `<merge>` element and places the two buttons directly in the layout, in place of the `<include/>` tag.

Define a ViewStub

```
<ViewStub
    android:id="@+id/stub_import"
    android:inflatedId="@+id/panel_import"
    android:layout="@layout/progress_overlay"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom" />
```

Load the ViewStub Layout

When you want to load the layout specified by the `ViewStub`, either set it visible by calling `setVisibility(View.VISIBLE)` or call `inflate()`

inflate() .

```
((ViewStub) findViewById(R.id.stub_import)).setVisibility(View.VISIBLE);
// or
View importPanel = ((ViewStub) findViewById(R.id.stub_import)).inflate();
```

Once visible/inflated, the `ViewStub` element is no longer part of the view hierarchy. It is replaced by the inflated layout and the ID for the root view of that layout is the one specified by the `android:inflatedId` attribute of the ViewStub. (The ID `android:id` specified for the `ViewStub` is valid only until the `ViewStub` layout is visible/inflated.)**Note:** One drawback of `ViewStub` is that it d oesn't currently support the `<merge/>` tag in the layouts to be inflated.

Making ListView Scrolling SmoothThe key to a smoothly scrolling `ListView` is to keep the application's main thread (the UI thread) free from heavy processing. Ensure you do any disk access, network access, or SQL access in a separate thread. To test the status of your app, you can enable `StrictMode` .

Beginning with Android 3.0 (API level 11), an extra feature is available in `AsyncTask` so you can enable it to run across multiple processor cores. Instead of calling `execute()` you can specify `executeOnExecutor()` and multiple requests can be executed at the same time depending on the number of cores available.多核处理器

Use a Background Thread
A way around repeated use of `findViewById()` is to use the "view holder" design pattern.
A `ViewHolder` object stores each of the component views inside the tag field of the Layout, so you can immediately access them without the need to look them up repeatedly. First, you need to create a class to hold your exact set of views. 实际上，VIewHolder的作用仅仅限于不用重复的获取资源而已。