# 提高UI的流畅度

Android中所有的界面绘制工作都是在UI线程中进行的，提高UI流畅度的最核心根本在于释放UI线程。即:不在主线程中做耗时的操作。

很多人都知道，耗时的操作要放到子线程中去做，比如访问网络，比如读写sd卡。像这类操作大家都会很自然的想到使用子线程来完成耗时的操作，等操作结束之后，再通过Handler通知主线程进行界面的更新。这是非常正确的方法。但是有一类方法，它必须得运行在在UI线程中，就是布局文件的加载。如果这类方法花的时间太多了，也是会对流畅度产生很大的影响。今天我们就来讲讲布局文件的优化。

加载布局文件，是必须在UI线程中完成的。我们通常是在onCreate方法中直调用setContentView，传入一个布局文件的id值，或者是通过LayoutInflater来将某一个布局文件转化成View对象。其实这两种方式的本质都是一样的，都是将xml文件转换成View对象。

我们现在要做的事，就是如何让xml文件转换成View对象所花的时间最少。做到了这点，就可以很大程度的提高UI的流畅度。

## 1、优化布局, 减少布局的嵌套层级

### a、使用drawableXXX属性



如果要实现这样一个效果，布局文件可以这样写

<LinearLayout orientation="vertical">

<ImageView/>

<TextView/>

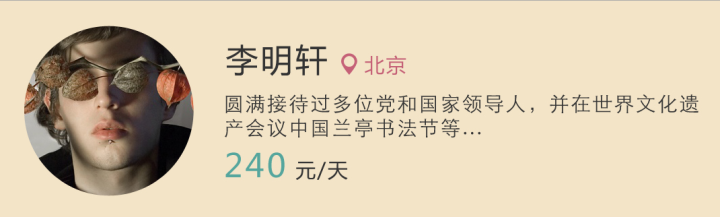
</LinearLayout>

优化后：

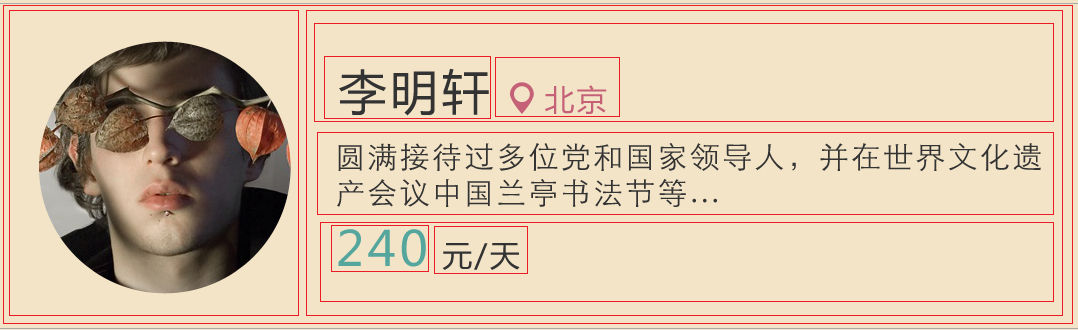
<TextView drawableBottom="@drawable/contact"/>

直接一个TextView就搞定，不需要在外面多一层LinearLayout

### b、多使用RelativeLayout，少使用LinearLayout



如果这样的布局使用LinearLayout来做的话，那么会是以下这个效果



<LinearLayout orientation="horizontal">

<ImageView/>

<LinearLayout orientation="vertical">

<LinearLayout orientation="horizontal">

<TextView/>

<TextView/>

</LinearLayout>

<TextView/>

<LinearLayout orientation="horizontal">

<TextView/>

<TextView/>

</LinearLayout>

</LinearLayout>

</LinearLayout>

这样就莫名其妙的多出了好多个LinearLayout.

这样过多的LinearLayout嵌套LinearLayout，会造成UI加载的非常慢。这样的布局完全可以使用一个RelativeLayout来完成，里面的子元素根据相对于其他控件的位置即可确定。

嵌套使用[LinearLayout](http://docs.eoeandroid.com/reference/android/widget/LinearLayout.html)很容易会导致视图层级过深。如果使用layout\_weight这个参数不断的进行嵌套，有可能会让各个子View付出计算两次的昂贵代价

优化后代码：

<RelativeLayout>

<Image id=avatar layout\_alignParentLeft=true />

<TextView id=name layout\_alignParentTop=true layout\_toRightOf=@id/avatar />

<TextView id=location layout\_alignParentTop=true layout\_toRightOf=@id/name />

<TextView id=desc layout\_below=@id/location layout\_toRightOf=@id/avatar />

....

</RelativeLayout>

### c、使用merge标签

使用merge标签也是能够减少一些布局的层次。merge标签经常会和include标签相联系。

那么什么时候使用merge标签呢？下面举例子说明。

<LinearLayout orientation="vertical">

......

<include layout="@layout/include\_view\_layout"/>

......

</LinearLayout>

而include\_view\_layout.xml 的代码如下：

<LinearLayout orientation="vertical">

<Button/>

<Button/>

</LinearLayout>

我们看到Button的父控件是LinearLayout，而include的父控件也是LinearLayout，这样子的布局最终的结果是

<LinearLayout orientation="vertical">

<LinearLayout orientation="vertical">

<Button/>

<Button/>

</LinearLayout>

</LinearLayout>

红色部分的LinearLayout完全是多余，于是这时候，我们就可以在include\_view\_layout.xml文件中使用merge标签了。如下：

<merge>

<Button/>

<Button/>

</merge>

这样，在加载这个include标签的时候，系统会忽略merge标签，直接将merge标签内的元素添加到外层的LinearLayout去了，达到减少层级的效果。

## 2、延迟加载

在开发某些功能时候，有时候需要动态的根据条件来判断显示哪一个View，不显示哪一个View。一般的做法是将所有的View都写在布局文件中去，然后根据条件再来设置他们的可见度Visibility为GONE或者VISIBLE。这种做法逻辑简单，便于理解。但是缺点就是那些不显示出来的View也占用了内存，消耗了inflate的时间。因为一个View，不论他的Visibility的值是什么，它都会被inflate出来，并占用内存空间。这时候其实就可以用到延迟加载的控件ViewStub了。

ViewStub是一个非常轻量级的控件，它占的资源非常小。注意，是ViewStub这个对象所占的资源小，但是你可以为ViewStub指定一个布局文件，这个布局文件被inflate的时候占的空间有可能很大。默认的情况下，ViewStub的所指定的布局文件是不被inflate的，只有当你调用了ViewStub的inflate方法时，ViewStub所指向的布局文件才会被inflate。所以ViewStub是一个延迟加载的控件。

<LinearLayout

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:gravity="center\_horizontal">

<ViewStub

android:id="@+id/viewstub1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout="@layout/viewstub\_layout1"/>

<ViewStub

android:id="@+id/viewstub2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout="@layout/viewstub\_layout2"/>

</LinearLayout>

在java代码中使用

ViewStub stub1 = (ViewStub) findViewById(R.id.viewstub1);

ViewStub stub2 = (ViewStub) findViewById(R.id.viewstub2);

if(isLogin()) {

stub1.inflate();

} else {

stub2.inflate();

}

这样就不会有浪费资源空间去加载没必要的控件了。

## ３、减少inflate的次数

这个的典型例子就是ListView的优化。我们说ListView的优化，实际上说的就是Adapter中getView方法的优化，我们来看一个没有优化过的getView方法。

@Override

public View getView(int position, View convertView, ViewGroup parent) {

MyItem product = list.get(position);

convertView = getLayoutInflater()

.inflate(R.layout.item\_record, null);

　　TextView tvDate = (TextView) convertView

.findViewById(R.id.tvDate);

TextView tvYongtu = (TextView) convertView

.findViewById(R.id.tvYongtu);

TextView tvMoney = (TextView) convertView

.findViewById(R.id.tvMoney);

tvDate.setText(product.detaildate);

tvYongtu.setText(product.auditmessage);

tvMoney.setText(product.detailmoney);

return convertView;

}

我们知道，ListView中的每一个Item被显示出来都要调用getView方法，这个Item如果滑出屏幕，又滑回来，重新显示在界面上的时候，又会再次调用getView方法。所以getView是不断的被调用的。而上面的代码，只要调用了getView方法，就一定会去inflate一个布局文件，真简直就是不敢想象的非常耗时的操作。于是，利用系统给我们的缓存convertView进行判断，可以大大减少inflate的次数。其实，findViewById也是一个很耗时的操作，我们可以利用ViewHolder来减少findViewById的次数。优化后的代码如下：

@Override

public View getView(int position, View convertView, ViewGroup parent) {

MyItem product = list.get(position);

ViewHolder holder;

if (convertView == null) {

convertView = getLayoutInflater().inflate(

R.layout.item\_record, null);

holder = new ViewHolder();

holder.tvDate = (TextView) convertView

.findViewById(R.id.tvDate);

holder.tvYongtu = (TextView) convertView

.findViewById(R.id.tvYongtu);

holder.tvMoney = (TextView) convertView

.findViewById(R.id.tvMoney);

convertView.setTag(holder);

} else {

holder = (ViewHolder) convertView.getTag();

}

holder.tvDate.setText(product.detaildate);

holder.tvYongtu.setText(product.auditmessage);

holder.tvMoney.setText(product.detailmoney);

return convertView;

}

static class ViewHolder {

TextView tvDate;

TextView tvYongtu;

TextView tvMoney;

}