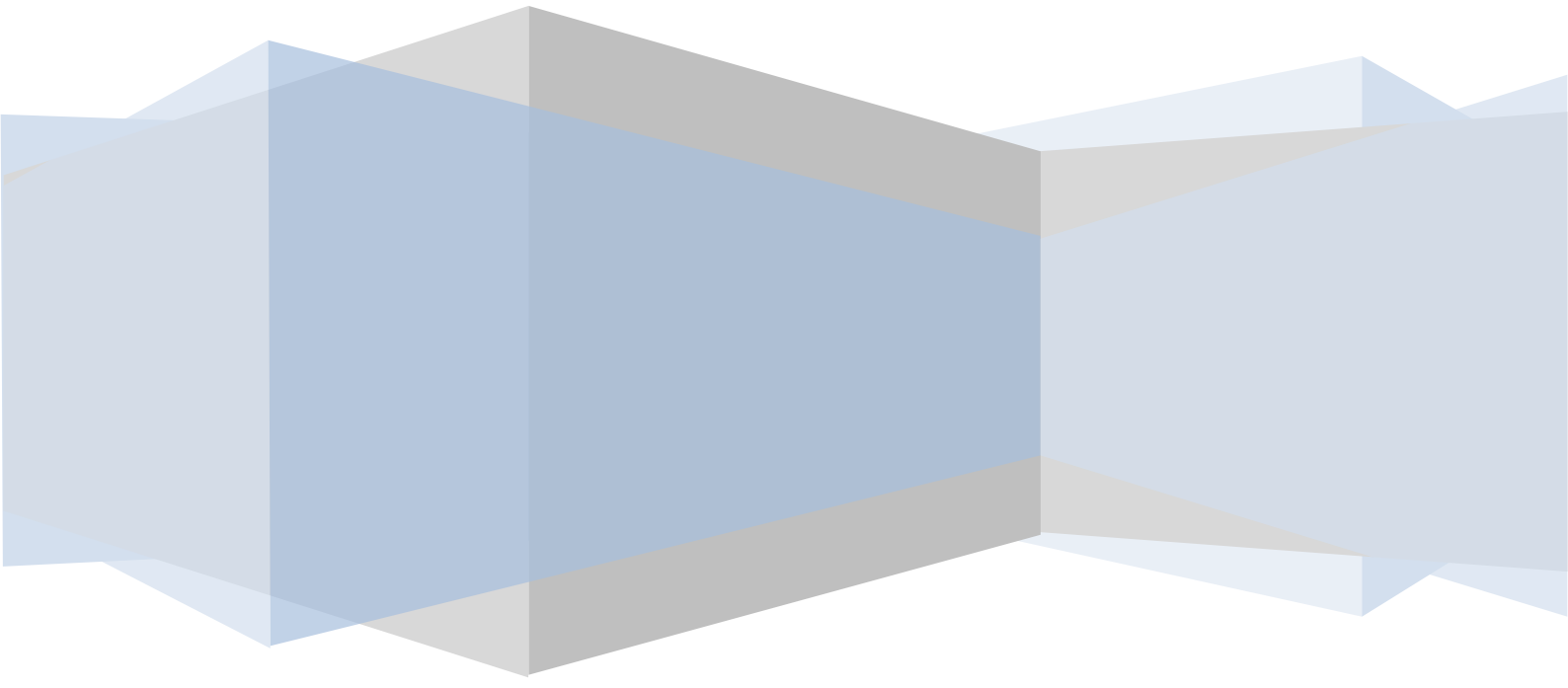

Fonts 学习分享

焉文臣

版本历史

版本	修订历史	修订人	修订日期
1.0	初始版本	焉文臣	2013.02.22

目录

1. 字体相关简介

1.1 什么是 ttf 文件？

TTF(TrueTypeFont)是 Apple 公司和 Microsoft 公司共同推出的字体文件格式,随着 windows 的流行,已经变成最常用的一种字体文件表示方式。是字库中的一种，应用范围非常广。

桌面出版系统使用的字库有两种标准：postscript 字库和 truetype 字库。android 系统中使用的是 truetype，这两种字体标准都是采用曲线方式描述字体轮廓，因此都可以输出很高质量的字形。常用的字库标准是 truetype 字库,truetype 字体是 windows 操作系统使用的唯一字体标准。truetype 字体的最大优点是可以很方便地把字体轮廓转换成曲线,可以对曲线进行填充,制成各种颜色和效果,它可以进一步变形,制作特殊效果字体,因此经常用来制作一些标题字或花样字。truetype 字库便宜,字款丰富。

1.2 有哪些字库设计公司？

Monotype Imaging、北大方正、华康科技、华文字库、北京汉仪等。

目前我们公司和 Monotype 公司合作，使用该司设计的中文字库及字体应用引擎。

1.3 相关专业名词

TTF：TrueTypeFont，一种字体文件

2. Android 字体

2.1 Android 字体简介

Android 系统默认支持三种字体，分别为：sans、serif、monospace，主要 java 类为 android.graphic.typeface

(1) 本类的常量静态定义，首先为字体类型（typeface）名称，如下：

① Typeface DEFAULT：默认常规字体

② Typeface DEFAULT_BOLD：默认常规粗体

③ Typeface SANS_SERIF：无衬线字体，相对于有衬线字型，其每个字母笔画粗细均匀。

例如：黑体、幼圆、雅黑

④ Typeface SERIF：衬线字体，在字的笔画开始、结束的地方有额外的装饰，而且笔画的粗细会有所不同。例如：Times New Roman

⑤ Typeface MONOSPACE：等宽字体。例如：Lucida Sans Typewriter

(2) 字体风格（style）名称，如下：

① int NORMAL：常规

② int BOLD：粗

③ int ITALIC：斜

④ int BOLD_ITALIC：粗、斜

2.2 Typeface 类用法简介

常用成员方法，如下：

① **staticTypeface createFromAsset(AssetManager mgr, String path);**

//↑静态方法，参数一为 AssetManager 对象，主要用于从 APK 的 assets 文件夹中取出字体，参数二为相对于 Android 工程下的 assets 文件夹中的外挂字体文件的路径。

② **staticTypeface createFromFile(File path);**

//↑静态方法，从文件系统构造一个字体，这里参数可以是 sdcard 中的某个字体文件

③ **staticTypeface createFromFile(String path);**

//↑静态方法，从指定路径中构造字体

下面用一个小程序来演示一下 typeface 类的用法。效果图如下：

2.3 Android 系统字体切换原理

内销的字库文件是由 monotype 公司提供的，所以字体移植包也来自 monotype。由于 monotype 公司提供泰文字库效果不理想，所以我们外销只采用 monotype 提供的字体移植包，字库由外销开发人员设计提取。

2.3.1 加载字体流程

以 12069 项目为例，其他项目中只是 flipfont 目录可能有所不同，功能大同小异。

OppoSettings\src\com\oppo\settings\flipfont\FontListPreference.java

用户点击字体设置选项后，FontListPreference 首先调用构造函数初始化，然后系统会回调 showDialog()函数和 onPrepareDialogBuilder()函数。showDialog()函数关键代码如下：

```
protected void showDialog(Bundle state) {
    if(mFontListAdapter == null || mNeedRescan) {
        mState = state;
        new LoadListTask().execute();
        mNeedRescan = false;
        return;
    }
    ……
    loadPreferences();
}
```

在 showDialog()函数中，使用 LoadListTask 继承 AsyncTask 类，进行异步任务更新 UI 线程。其中，关键代码如下：

```
private class LoadListTask extends AsyncTask<Void, Void, Void> {
    protected void onPreExecute() {
        ……
    }
    protected Void doInBackground(Void... params) {
        // load the font list - can take a few seconds
        mFontListAdapter = new FontListAdapter(context);
        mFontListAdapter.loadTypefaces();
        return null;
    }
    protected void onPostExecute(Void unused) {
        showDialog(mState);
    }
}
```

上面代码中，在 doInBackground 函数中进行 FontListAdapter 的初始化，主要是获取已安装的字体 apk 包和系统内置的字体文件。其中：

①安装的字体 apk 包名必须以"com.monotype.android.font."开头

②系统内置字体，实际上是先通过编译的 Android.mk 将 tiff 字体文件，拷贝到/system/fonts/下面，然后调用 Typeface.createFromFile(f)来创建字体

最后搜索的结果被赋值到了相应的变量中

```
mTypefaceFinder.getSansEntries(mPackageManager, mFontNames, mTypefaceFiles, mFontPackageNames);
```

如 mFontNames 就包含了内置和 apk 包安装的所有字体名称，mTypefaceFiles 则是对应字体 ttf

文件，mFontPackageNames 是对应 apk 包名。

2.3.2 选择字体操作

出现字体选择菜单后，点击选中某一字体后会执行 FontListPreference.java 的 onOkButtonPressed 函数，这里面有一个函数 boolean checkFont(String apkname)，用来检查 apk 的包名和 X.509 的签名信息，如果签名信息不是 Monotype Imanging Inc，则认为安装的字体不合法，会提示字体无效。下面就是根据点击列表索引判断得到选中字体的名称：

```
selectedFont = mFontListAdapter.mFontNames.elementAt(mClickedItem).toString();
```

然后再获取到对应的字体 ttf 文件，fontWriter.copyFontFile 这个函数将 ttf 文件复制一份到"/data/data/com.android.settings/app_fonts"下，其中 apk 包安装的字体是一个 for 循环，里面通过 apk 包名相关信息来获取对应的字体 ttf 文件。最后，调用

```
fontWriter.writeLoc(context, FontWriter.SANS_LOC_NAME, fontDir.getAbsolutePath());
```

writeLoc 函数将字体路径写入到"/data/data/com.android.settings/app_fonts/sans.loc"，同时将当前路径保存到系统属性中：

```
SystemProperties.set("persist.sys.flipfontpath", directory);
```

最后，执行更新系统配置操作：

```
am.updateConfiguration(config);
Intent font_chage_intent = new Intent("android.settings.ACTION_FLIPFONT_CHANGED");
this.context.sendBroadcast(font_chage_intent);
```

至此，所有 Activity 都会重新启动。非 Acitivty 形式的模块可以接受该广播"android.settings.ACTION_FLIPFONT_CHANGED"对字体进行处理。

2.3.3 更新系统字体

在 Activity 中的 onCreate 函数 onConfigurationChanged 函数中，会调用如下代码：

```
protected void onCreate(Bundle savedInstanceState) {
    ……
    Context context = this.getApplicationContext();
    if (context != null) {
        if(DEBUG)
        Log.v(TAG, "onConfigurationChanged():" + context.getPackageName());
        Typeface.setAppTypeFace(context.getPackageName());
    } else {
        if(DEBUG)
        Log.v(TAG, "onConfigurationChanged():null/android");
        Typeface.setAppTypeFace("android");
    }
    ……
}
```

上述代码中，主要实现了字体的更新操作。

Typeface.setAppTypeFace函数中，会对参数传入的包名进行判断，可以设置为对CTS认证相关的包，不更换其字体，以确保CTS认证通过。

在 setAppTypeFace 函数中调用了native底层C++代码，真正产生字体改变的地方是在Typeface.java里有android默认的字体类：

```
public static final Typeface DEFAULT;
```

public static final Typeface DEFAULT_BOLD;

这两个类是静态全局的，就是说android默认情况的字体都是来自这两个字体Typeface类，要使字体改变，在上层就想办法替换这两个字体类的相应方法和接口，所以在SetAppTypeFace里最终是调用private static void SetFlipFonts()，这个函数里面：

① 获取保存在系统属性里的"persist.sys.flipfontpath"的值，得到字体文件路径

② 将得到的ttf文件重新创建Typeface类实例，用于赋值替换掉默认的DEFAULT和DEFAULT_BOLD这两个字体类，代码在SetFlipFonts函数里面：

```
iNative = DEFAULT.native_instance;
if (strFontPath.isEmpty())
    DEFAULT.native_instance = nativeCreate(null, 0);
else
    DEFAULT.native_instance = nativeCreateFromFile(strFontPath);
    nativeUnref(iNative);
```

从而就改变了默认字体。

2.4 代码移植

主要涉及模块为 frameworks\base，packages\apps\Settings

frameworks\base\api\15.txt

frameworks\base\core\java\android\content\ContextWrapper.java

frameworks\base\core\java\android\content\pm\ActivityInfo.java

frameworks\base\core\java\android\content\res\Configuration.java

frameworks\base\core\java\android\app\Activity.java

frameworks\base\core\java\android\app\Application.java

frameworks\base\core\java\android\provider\Settings.java

frameworks\base\core\java\android\view\GLES20Canvas.java

frameworks\base\core\java\com\android\internal\app\LocalePicker.java

frameworks\base\core\res\res\values\attrs_manifest.xml

frameworks\base\graphics\java\android\graphics\Canvas.java

frameworks\base\graphics\java\android\graphics\Paint.java

frameworks\base\graphics\java\android\graphics\Typeface.java

frameworks\base\packages\SystemUI\com\android\systemui\statusbar\phone\PhoneStatusBar.java

frameworks\base\data\fonts\DroidSans_Subset.ttf

frameworks\base\data\fonts\fallback_fonts.xml

frameworks\base\data\fonts\fonts.mk

frameworks\base\data\fonts\OEM1.ttf

frameworks\base\data\fonts\OEM2.ttf

packages\apps\Settings\com\android\settings\flipfont\FontListAdapter.java

packages\apps\Settings\com\android\settings\flipfont\FontListPreference.java

packages\apps\Settings\com\android\settings\flipfont\FontWriter.java

packages\apps\Settings\com\android\settings\flipfont\Typeface.java

packages\apps\Settings\com\android\settings\flipfont\TypefaceFile.java

packages\apps\Settings\com\android\settings\flipfont\TypefaceFinder.java

packages\apps\Settings\com\android\settings\flipfont\TypefaceParser.java

packages\apps\Settings\res\drawable\flipfont_icon.png

packages\apps\Settings\res\values-zh-rTW\strings.xml

packages\apps\Settings\res\values\strings.xml

packages\apps\Settings\res\values-zh-rCN\strings.xml

packages\apps\Settings\res\xml\display_settings.xml

3．小结

待续…