hwui_skia浅析

Last edited by **caoquanli** 1 month ago

(一).hwui概述

Android3.0以前，几乎所有的图形绘制都是由Skia完成，Skia是一个向量绘图库，使用CPU来进行运算；所以从Android3.0 开始，Google用hwui取代了Skia，准确的说，是推荐取代，因为Opengl的支持不完全，有少量图形api仍由Skia完成，多数view的绘制通过HWUI模块使用openGL的函数来实现

实例：textview显示字符

android 的TextView控件如何显示特殊字符？如果直接将特殊字符写入TextView是会报错的，需要对特殊字符进行ASCII转换。 ASCII转换表如下：

序号	特殊符号	在 strings.xml 写入	备注
1	空格	&#032;	
2	!	&#033;	感叹号
3	@	&#064;	艾特符号
4	#	&#035;	井号
5	$	&#036;	美元符号
6	%	&#037;	百分号
7	^	&#094;	异或符号
8	&	&#038;	与符号
9	*	&#042;	米字符号
10	(	&#040;	左小括号
11	)	&#041;	右小括号
12	+	&#043;	加号
13	-	&#045;	减号
14	_	&#095;	下划线
15	{	&#123;	左大括号
16	}	&#125;	右大括号
17	“	&#034;	需要在前面加入\
18	‘	&#039;	单引号
19	,	&#044;	逗号
20	.	&#046;	点
21	/	&#047;	左斜线
22	:	&#058;	冒号
23	;	&#059;	分号
24	<	&#060;	小于号
25	=	&#061;	等号
26	>	&#062;	大于号
27	?	&#063;	问号
28	[	&#093;	左中括号
29	]	&#091;	右中括号
30	`	&#096;	
31	|	&#124;	或符号
32	~	&#126;	

具体的文字和表情也是通过unicode来进行显示，表情的Unicode如下：

Back to top

1. Emoticons (1F601 - 1F64F)

Native [1]	Apple [2]	Android [3]	Android [3]	Symbola [4]	Twitter [5]	Unicode	Bytes (UTF-8)	Description
😁	😁	😁	😁	😁	😁	U+1F601	\xF0\x9F\x98\x81	grinning face with smiling eyes
😂	😂	😂	😂	😂	😂	U+1F602	\xF0\x9F\x98\x82	face with tears of joy
😃	😃	😃	😃	😃	😃	U+1F603	\xF0\x9F\x98\x83	smiling face with open mouth
😄	😄	😄	😄	😄	😄	U+1F604	\xF0\x9F\x98\x84	smiling face with open mouth and smiling eyes
😅	😅	😅	😅	😅	😅	U+1F605	\xF0\x9F\x98\x85	smiling face with open mouth and cold sweat
😆	😆	😆	😆	😆	😆	U+1F606	\xF0\x9F\x98\x86	smiling face with open mouth and tightly-closed eyes
😉	😉	😉	😉	😉	😉	U+1F609	\xF0\x9F\x98\x89	winking face
😊	😊	😊	😊	😊	😊	U+1F60A	\xF0\x9F\x98\x8A	smiling face with smiling eyes
😋	😋	😋	😋	😋	😋	U+1F60B	\xF0\x9F\x98\x8B	face savouring delicious food
😌	😌	😌	😌	😌	😌	U+1F60C	\xF0\x9F\x98\x8C	relieved face
😍	😍	😍	😍	😍	😍	U+1F60D	\xF0\x9F\x98\x8D	smiling face with heart-shaped eyes
😏	😏	😏	😏	😏	😏	U+1F60F	\xF0\x9F\x98\x8F	smirking face
😒	😒	😒	😒	😒	😒	U+1F612	\xF0\x9F\x98\x92	unamused face
😓	😓	😓	😓	😓	😓	U+1F613	\xF0\x9F\x98\x93	face with cold sweat
😔	😔	😔	😔	😔	😔	U+1F614	\xF0\x9F\x98\x94	pensive face

使用实例：在TextView显示 <用户协议>

```
<TextView  android:layout_width="wrap_content"
 android:layout_height="wrap_content" 、
 android:text="&#060;用户协议&#062;"    />
```

记住ASCII码值的后面一定要带上;号，否则会报错！

(二).Android绘制相关的概念

Canvas：

画布是应用程序用来绘制Widget或图形等元素的地方。Froyo和Gingerbread上，画布通过Skia来绘Honeycomb及以

在 Ice Cream Sandwich及以后的版本上，HWUI缺省用于图形的绘制

Skia：

Skia是一组2D绘图的API，它完全通过软件实现。由于性能方面的原因，Skia逐渐被HWUI所替代。(skia：软件实现的

HWUI：

HWUI 可以使UI组件使用GPU加速。HWUI是在Honeycomb中引入进来的，目的是使交互更加快速，及时响应，流畅。在

SkBlitter

不是单独的一个类，指代了一系列根据图像格式、是否包含Shader等区分出来的一系列子类。这一族类执行大块头的渲

HarfBuzz

是用于文字成型的软件开发库，用以转换Unicode文字到字形指标及方位的过程。最近的HarfBuzz（New HarfBuzz）

字库ttf/otf

字库文件，其中存放了字形的轮廓信息，还会有更高级的Hintting和抗锯齿信息

Freetype

将字体光栅化。将uicode代表的字形码从ttf或者otf字库中取出字的轮廓，输出的是一个位图。其中Hinting和抗锯齿

Cmap

由于存在不同的系统和编码集，cmap表内涵多个子表，每个子表包含的基本信息有系统ID、编码集以及在表的偏移量。

字符影射表(charmap)

字符对应的字体数据称为glyph，字体文件中通常带有一个字符映射表，用来把字符映射到对应glyph的索引值。因为字

(三).Android 字库

4.x以后最显著的变化是新增了Roboto家族，有Regular/Bold/Italic/BoldItalic四种变体，相比原来的DroidSans多了两个斜体。原来的DroidSans家族还在，但是已经被指向到Roboto家族，再修改原来的文件是无效的。

```
Roboto-Regular.ttf
Roboto-Italic.ttf
Roboto-Bold.ttf[BB
Roboto-BoldItalic.ttf
```

如何修改默认字库

进入/system/etc目录下可以找到fallback_fonts.xml这个文件。这是4.x后对新增语种文字的配置文件，可以用记事本打开进行修改： 在DroidSansFallback.ttf后面另起一行加入：xxx.ttf 系统字模必须放在/system/fonts/下面，需要使用的字模要配置到/system/etc/system_fonts.xml和/system/etc/fallback_fonts.xml中；系统优先从system_fonts.xml中列出的字模中查找字模，其次才是fallback_fonts.xml。[Android系统源码的字体配置文件位于frameworks/base/data/fonts/文件夹下]

(四) icu4c

ICU4C提供了C/C++平台强大的国际化开发能力，软件开发者几乎可以使用ICU4C解决任何国际化的问题，根据各地的风俗和语言习惯，实现对数字、货币、时间、日期、和消息的格式化、解析，对字符串进行大小写转换、整理、搜索和排序等功能，必须一提的是，ICU4C提供了强大的BIDI算法，对阿拉伯语等BIDI语言提供了完善的支持。 ICU4C有效地增强了C/C++平台的软件国际化能力，它使得开发者可以写出独立于风俗和语言的C/C++代码，然后这些代码可以通过利用相关资源组成语言和风俗相关软件。ICU4C除了具有对Unicode强大的支持能力以外，针对国际化中的各种问题，ICU4C提供了强大的国际化问题解决能力

Unicode

Unicode 标准有三种编码形式，UTF-8,UTF-16,UTF-32，分别用 1 个、2 个、4 个字节作为编码单元。UTF-16 编码平衡了文本的执行效率和内存的占用，目前使用 Unicode 作为内码的应用程序大都采用 UTF-16 编码，Symphony 就使用 UTF-16 编码 UTF-16 编码对于 Unicode 标准中定义的 BMP (Basic Multilingual Plane) 定义的字符用两个字节表示，这部分字符基本上涵盖了所有语言的常用字符。对于超出 BMP 平面的字符 UTF-16 用 4 个字节表示，叫做 surrogate

pairs。 字符到 UTF-16 编码的映射

简单实例： 解析unicode:

```
String getStrFromUniCode(String unicode){
        String str = "";
        for(int i=0;i<unicode.length();i+=4){
            String s = "";
            for(int j=i;j<i+4;j++){
```

```
                s+=String.valueOf(unicode.charAt(j));
            }
            str+=String.valueOf((char)Integer.valueOf(s, 16).intValue());
        }
        return str;
    }
```

（五） skia库

skia库在Android中的使用： 规定2D绘制API 规定图像数据结构 承担编解码调度和软件渲染职责 Android系统使用skia的场景：

bitmap的绘制

界面绘制

lockCanvas——draw——unlockCanvas流程

- UI控件未开启硬件加速。由performTravelsals——drawSoftware调入。 Android显示系统是最复杂的Android最复杂的子系统，没有之一，这里只说明软件绘制时的一个基本过程：
- a、计算布局，看是否需要更新View。
- b、计算所有需要更新的View的区域，计算其最小外包矩形，即dirtyRect
- c、由Surface去Lock一个Canvas，Lock时指定dirtyRect，这里涉及gui的buffer轮换机制，会去获取一块未在显示的Buffer，由于换了buffer，在gui模块会去拷贝上一帧非dirtyRect的部分。
- d、执行根View的draw方法，递归调用所有子View的onDraw方法。由于Canvas对应的是软件绘制的canvas，所有绘制操作经过Canvas——SkiaCanvas——SkCanvas的流程，由skia引擎执行。（AndroidL上在jni（libandroid_runtime）处作了一层SkiaCanvas的封装，4.4及以前是Canvas——SkCanvas）
- e、完成绘制，让Surface去unlockCanvas，将绘制好的Buffer送显。
- SurfaceView，这个是由应用显式实现lockCanvas——draw——unlockCanvas流程。

skia相关库

- libandroid_runtime：framework.jar的jni实现，链接framework和lib库的桥梁
- hwui：2D硬件加速库，使用skia的数据格式
- libchromeview：浏览器引擎，webview相关
- freetype 字体解析

（六） skia文字绘制

文字绘制主要包括编码转换（主要是中文）、字形解析（点线或image）和实际渲染三个步骤。在这个过程中，字形解析和实际渲染均是耗时步骤。Skia对文字解析的结果做了一套缓存机制 文字绘制流程：

- SkCanvas： 绘制文字和下划线
- SkDraw： 将文字解析为路径，然后绘制路径，缓存路径 将文字解析为Mask（32*32的A8图片），然后绘制模板，缓存模板
- SkGlyphCache：字形解析的结果缓存
- SkScalerContext： 负责字形的解析，有多种实现。Android中是用FreeType：SkScalerContext_FreeType
-

（七） hwui处理

- 硬件绘制，或硬件加速，就是通过hwui，将2D的绘图操纵转换为3D的绘图
- 每一个绘制采用一个RecordedOp进行描述，复杂的绘图将被拆分成简单的基本绘图，并利用RecordingCanvas进行录制。
- 每个View都对应RenderNode，而每个界面有一个DisplayList，用以保存录制的Ops。
- 每个进程只有一个RenderThread，所有的绘图都在RenderThread中完成，因此，其他线程的操纵都通过Task或WorkItem的形式post到RenderThread中完成。DrawFrameTask是RenderThread中比较特殊的一个task，是用以绘制整个界面的，跟随Vync而触发。
- OpenGL是单线程的，所以每个RenderThread都有各自的上下文，CanvasContext，通过Preparetree，将DisplayList中Ops都同步到CanvasContext的layerUpdateQueue中，准备好绘制帧的数据。
- 绘制是由具体的Pipeline完成的，目前有3中类型的Pipeline，OpenGLPipeline是默认的Pipeline。
- OpenGLPipeline绘制时，通过FrameBuilder和LayerBuilder，将DisplayList的数据进一步封装。在replayBakedOps时，将Opo的操纵转换为具体的绘制操纵，通过BakedOpDispatcher分发给BakedOpRenderer进行渲染。而真正的渲染是在mRenderState完成，直接调用OpenGL的接口。

经典hwui流程：

view绘制批处理

（八） 硬件加速