## 深入浅出 - Android系统移植与平台开发（五）－ 定制手机模拟器ROM

一、 修改化定制Android4.0系统

Android系统启动时，先加载Linux内核，在Linux的framebuffer驱动里可以定制开机界面，Linux内核启动成功后，挂载根文件系统，启动Android系统，这个时候设备屏幕上开始出现滚动的Android动画，等全部的Android服务启动完毕之后，开始启动Android的HOME界面，也就是桌面。

而在这个过程中，我们可以将开机界面和Android动画全部定制为自己需要的效果。

在整个开机过程中，屏幕上会出现三次内容：

Ø  Linux启动时画面，通常是个黄嘴的小企鹅

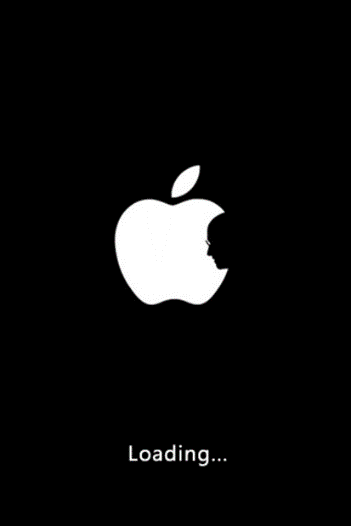
Ø  Android系统本地启动阶段画面，是“ANDROID”文字字样

Ø  Android系统显示系统启动阶段动画，是滚动的ANDROID动画

我们下面将三个过程中的屏幕内容都进行个性化设置。

1. 定制手机开机界面

根据前面文章介绍，我们要在Linux的framebuffer驱动里修改一些代码，让屏幕上出现我们自己的一个开机图片，如下图所示效果。



在Linux系统中，LCD显示设备的驱动都是基于framebuffer的，framebuffer我们可以看成是Android的显存，只要我们向该显存里写入数据，这些数据就可以显示在LCD上。

根据不同的LCD所支持的颜色可以分为：16位色，24位色，32位色。

学过初中物理都知道，色彩由三元色：红（R），绿（G），蓝（B）组成。

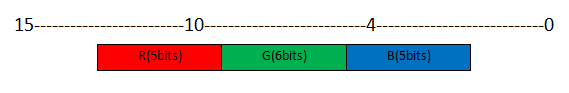
Ø  16位色：一个像素点由16bit表示，占两个字节，RGB组成分为：565或555二种

Ø  24位色：一个像素点由24bit表示，占三个字节，RGB每个颜色由8位组成。

Ø  32位色：一个像素点由32bit表示，占四个字节，除了RGB每个颜色8位外，还有8位的Alpha的透明度，共组成32位。

很明显，位数越高，可显示的色彩越丰富，相同像素的LCD占用的显存越大，现在手机和平板基本上都使用32位色的LCD。

Android模拟器里使用16位565格式显示驱动。



既然如此，那么出现在framebuffer里的应该是16位的具体颜色值，我们从怎么知道一个图片的RGB的值呢？

还好，我们使用一款叫Image2Lcd的软件，可以将一个指定的bmp位图格式图片转换成指定的RGB格式数据的数组中。

如下图所示：



选择好图片，设置好宽度和高度及颜色位数，保存成一个头文件mylogo.h，打开里面内容如下：

1. const unsigned char bmp[307200] = { /\* 0X00,0X10,0X40,0X01,0XE0,0X01,0X01,0X1B, \*/
2. 0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,
3. …
4. };

那么我们可以直接将这个头文件拷贝到内核目录中，包含到驱动里，然后直接通过bmp数组名访问图片内容。

Ø  修改goldfish的fb驱动文件：

drivers/video/goldfishfb.c

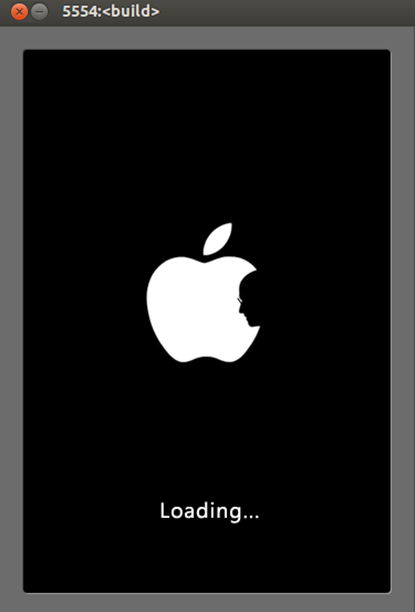
在里面添加一个绘制Logo图片函数draw\_logo，如下所示：

1. 191 // MichaelTang add for bootlogo
2. 192 #include "mylogo.h"
3. 193 static int draw\_logo(struct goldfish\_fb \*fb)
4. 194 {
5. 195 int height = fb->fb.var.yres;
6. 196 int width = fb->fb.var.xres;
7. 197
8. 198 printk("---------> h = %d, w = %d\n", height, width);
9. 199 memcpy(fb->fb.screen\_base, bmp, height\*width\*2);
10. 200 return 0;
11. 201 }

Ø  在goldfish\_fb\_probe函数里调用我们刚才添加的draw\_logo函数：

1. 204 static int goldfish\_fb\_probe(struct platform\_device \*pdev)
2. {
3. …
4. 312 // MichaelTang add for bootlogo
5. 313 draw\_logo(fb);
6. ...
7. }

重新编译goldfish内核，然后将模拟器的内核指定为新编译的内核，启动后效果如下图所示：



当然，这儿的实验只是在Android的模拟器上实现的，如果在真实设备上，其步骤和上述一样，只不过驱动文件可能不一样，修改的位置不太一样，其原理是一样的。不过最后苹果和三星专利大战再次敲响，希望苹果有一天别来找我，否则，卖肾都赔不起，现在肾也不值钱了。

2.定制Android启动字样

当Linux启动完毕之后，开始挂载根文件系统ramdisk.img，通过命令行指定Linux运行Linux系统里的第一个用户进程init：

 init程序由system/core/init/目录下的源码编译而成，其入口文件为：init.c，console\_init\_action函数就是用来打开console终端，然后在屏幕上打印“A N D R O I D”字样的，如果想修改这个值，则直接将其内容修改了，重新编译init程序，然后重新生成system.img即可，不过，一般是将其内容注释掉。

1. 538 static int console\_init\_action(int nargs, char \*\*args)
2. 539 {
3. …
4. 548 fd = open(console\_name, O\_RDWR);
5. 549 if (fd>= 0)
6. 550 have\_console = 1;
7. 551 close(fd);
8. 552
9. 553 if( load\_565rle\_image(INIT\_IMAGE\_FILE) ) {
10. 554 fd = open("/dev/tty0", O\_WRONLY);
11. 555 if (fd>= 0) {
12. 556 const char \*msg;
13. 557 msg = "\n"
14. 558 "\n"
15. 559 "\n"
16. 560 "\n"
17. 561 "\n"
18. 562 "\n"
19. 563 "\n" // console is 40 cols x 30 lines
20. 564 "\n"
21. 565 "\n"
22. 566 "\n"
23. 567 "\n"
24. 568 "\n"
25. 569 "\n"
26. 570 "\n"
27. 571 " A N D R O I D ";
28. 572 write(fd, msg, strlen(msg));
29. 573 close(fd);
30. 574 }
31. 575 }
32. 576 return 0;
33. 577 }

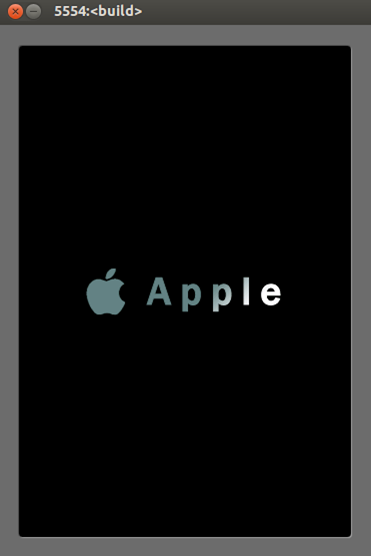
修改完之后，操作步骤如下：

1. $ source build/envsetup.sh
2. $ mmm system/core/init
3. $ make snod

重新启动模拟器，可以发现ANDROID字样发生了相应的改变。

3. 定制Android动画

Android系统过程中会滚动Android字样的一个动画，我们可以根据自己的需要，定制这个开机动画，如下图所示：

Android的开机动画是由Linux本地程序bootanimation控制实现的，其代码在：frameworks/base/cmds/bootanimation/，通过分析源码可知，修改Android开机动画有两种方式：

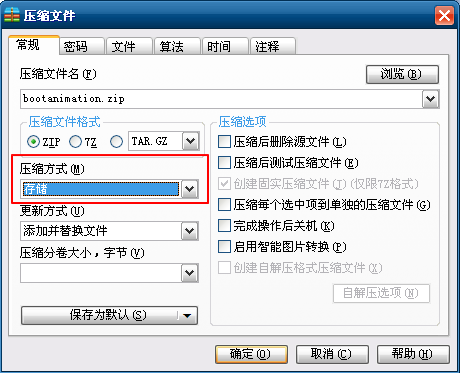
Ø  替换frameworks/base/core/res/assets/images/目录下的两个图片文件：android-logo-mask.png和android-logo-shine.png，android-logo-mask.png是镂空蒙板png图片，android-logo-shine.png是镂空蒙板后面的闪光png图片

Ø  在/data/local/或/system/media/目录创建bootanimation.zip文件

bootanimation.zip文件打包前的结构为：

1. desc.txt 动画属性描述文件
2. part0/ 第一阶段动画图片的目录（动画是由一帧帧图片组成的）
3. part1/ 第二阶段动画图片的目录

bootanimation.zip文件是直接由这几个文件打包的，打包的格式是ZIP，并且要指定用压缩打包方式（就是在打包时的压缩方式选择为存储）。



desc.txt文件的格式为：

1. 480 250 15
2. p 1 0 part0
3. p 0 10 part1

其中各个参数的意义为：

|  |  |  |  |
| --- | --- | --- | --- |
| 480 | 250 | 15 |  |
| 图片的宽 | 图片的高 | 每秒显示帧数 |  |
| p | 1 | 0 | part0 |
| 标识符 | 循环的次数 | 阶段切换间隔时间 | 对应图片目录 |
| p | 0 | 10 | part1 |
| 标识符 | 循环的次数 | 阶段切换间隔时间 | 对应图片目录 |

*注：*

*标识符：p 是必须的。*

*循环次数：指该目录中图片循环显示的次数，0表示本阶段无限循环。*

*每秒显示帧数：就是每秒显示的图片数量，决定每张图片显示的时间。*

*阶段切换间隔时间：指的是该阶段结束后间隔多长时间显示下一阶段的图片，其单位是每张图片显示的时间。*

*对应图片目录：就是该阶段动画的系列图片，以图片文件目录的顺序显示动画，而且图片的格式必须要为PNG。*