Keywords:miniui

# Minui

源码位置；bootable/recovery/minui/

## minui部分接口的说明

zdfg

1. **int** gr\_init(**void**);             /\* 初始化图形显示,主要是打开设备、分配内存、初始化一些参数 \*/
2. **void** gr\_exit(**void**);            /\* 注销图形显示,关闭设备并释放内存 \*/
4. **int** gr\_fb\_width(**void**);         /\* 获取屏幕的宽度 \*/
5. **int** gr\_fb\_height(**void**);        /\* 获取屏幕的高度 \*/
6. gr\_pixel \*gr\_fb\_data(**void**);    /\* 获取显示数据缓存的地址 \*/
7. **void** gr\_flip(**void**);            /\* 刷新显示内容 \*/
8. **void** gr\_fb\_blank(bool blank);  /\* 清屏 \*/
10. **void** gr\_color(unsigned **char** r, unsigned **char** g, unsigned **char** b, unsigned **char** a);  /\* 设置字体颜色 \*/
11. **void** gr\_fill(**int** x, **int** y, **int** w, **int** h);  /\* 填充矩形区域,参数分别代表起始坐标、矩形区域大小 \*/
12. **int** gr\_text(**int** x, **int** y, **const** **char** \*s);  /\* 显示字符串 \*/
13. **int** gr\_measure(**const** **char** \*s);             /\* 获取字符串在默认字库中占用的像素长度 \*/
14. **void** gr\_font\_size(**int** \*x, **int** \*y);         /\* 获取当前字库一个字符所占的长宽 \*/
16. **void** gr\_blit(gr\_surface source, **int** sx, **int** sy, **int** w, **int** h, **int** dx, **int** dy);  /\* 填充由source指定的图片 \*/
17. unsigned **int** gr\_get\_width(gr\_surface surface);   /\* 获取图片宽度 \*/
18. unsigned **int** gr\_get\_height(gr\_surface surface);  /\* 获取图片高度 \*/
19. /\* 根据图片创建显示资源数据,name为图片在mk文件指定的相对路径 \*/
20. **int** res\_create\_surface(**const** **char**\* name, gr\_surface\* pSurface);
21. **void** res\_free\_surface(gr\_surface surface);       /\* 释放资源数据 \*/

ASD

<http://blog.csdn.net/u010753159/article/details/51325500>

## gr

### gr\_init

int gr\_init(void)

{

gr\_init\_font();

**gr\_vt\_fd = open("/dev/tty0", O\_RDWR | O\_SYNC);**

if (gr\_vt\_fd < 0) {

// This is non-fatal; post-Cupcake kernels don't have tty0.

perror("can't open /dev/tty0");

} else if (ioctl(gr\_vt\_fd, KDSETMODE, (void\*) KD\_GRAPHICS)) {

// However, if we do open tty0, we expect the ioctl to work.

perror("failed KDSETMODE to KD\_GRAPHICS on tty0");

gr\_exit();

return -1;

}

gr\_backend = open\_adf();

if (gr\_backend) {

gr\_draw = gr\_backend->init(gr\_backend);

if (!gr\_draw) {

gr\_backend->exit(gr\_backend);

}

}

if (!gr\_draw) {

gr\_backend = open\_fbdev();

gr\_draw = gr\_backend->init(gr\_backend);

if (gr\_draw == NULL) {

return -1;

}

}

overscan\_offset\_x = gr\_draw->width \* overscan\_percent / 100;

overscan\_offset\_y = gr\_draw->height \* overscan\_percent / 100;

gr\_flip();

gr\_flip();

return 0;

}

### gr\_fill

填充矩形区域,参数分别代表起始坐标、矩形区域大小

### dpad:gr\_circle?

手动画：内存缓存机制

用图像:存储换为

画几何图形吧？

## events.c

### ev\_init

ev\_init():minui/events.c[open /dev/input/event\*]打开 /dev/input/event\*

这部分是在，充电状态下，按键操作的初始化，比如：短按显示充电logo，长按开机，初始化代码如下。

**[cpp]** [view plain](http://blog.csdn.net/bi511304183/article/details/9303043) [copy](http://blog.csdn.net/bi511304183/article/details/9303043)

1. **int** ev\_init(ev\_callback input\_cb, **void** \*data)
2. {
3. DIR \*dir;
4. **struct** dirent \*de;
5. **int** fd;
6. dir = opendir("/dev/input");//打开驱动结点；
7. **if**(dir != 0) {
8. **while**((de = readdir(dir))) {
9. unsigned **long** ev\_bits[BITS\_TO\_LONGS(EV\_MAX)];
10. //            fprintf(stderr,"/dev/input/%s\n", de->d\_name);
11. **if**(strncmp(de->d\_name,"event",5)) **continue**;
12. fd = openat(dirfd(dir), de->d\_name, O\_RDONLY);
13. **if**(fd < 0) **continue**;
14. /\* read the evbits of the input device \*/
15. **if** (ioctl(fd, EVIOCGBIT(0, **sizeof**(ev\_bits)), ev\_bits) < 0) {
16. close(fd);
17. **continue**;
18. }
19. /\* TODO: add ability to specify event masks. For now, just assume
20. \* that only EV\_KEY and EV\_REL event types are ever needed. \*/
21. **if** (!test\_bit(EV\_KEY, ev\_bits) && !test\_bit(EV\_REL, ev\_bits)) {
22. close(fd);
23. **continue**;
24. }
25. ev\_fds[ev\_count].fd = fd;
26. ev\_fds[ev\_count].events = POLLIN;
27. ev\_fdinfo[ev\_count].cb = input\_cb;
28. ev\_fdinfo[ev\_count].data = data;
29. ev\_count++;
30. ev\_dev\_count++;
31. **if**(ev\_dev\_count == MAX\_DEVICES) **break**;
32. }
33. }
34. **return** 0;
35. }

### uevent\_open\_socket

int uevent\_open\_socket(int buf\_sz, bool passcred)

{

struct sockaddr\_nl addr;

int on = passcred;

int s;

memset(&addr, 0, sizeof(addr));

addr.nl\_family = AF\_NETLINK;

addr.nl\_pid = getpid();

addr.nl\_groups = 0xffffffff;

**s = socket(PF\_NETLINK, SOCK\_DGRAM, NETLINK\_KOBJECT\_UEVENT);**

if(s < 0)

return -1;

setsockopt(s, SOL\_SOCKET, SO\_RCVBUFFORCE, &buf\_sz, sizeof(buf\_sz));

setsockopt(s, SOL\_SOCKET, SO\_PASSCRED, &on, sizeof(on));

if(bind(s, (struct sockaddr \*) &addr, sizeof(addr)) < 0) {

close(s);

return -1;

}

return s;

}

uevent\_open\_socket这个函数是通过kobject\_uevent的方式通知的应用层，就是往一个socket广播一个消息，只需要在应用层打开socket监听NETLINK\_KOBJECT\_UEVENT组的消息，就可以收到了,主要是创建了socket接口获得uevent的文件描述符，然后触发/sys/class/power\_supply目录及其子目录下的uevent，然后接受并创建设备节点，至此设备节点才算创建。

## resources

### res\_create\_multi\_display\_surface

int res\_create\_multi\_display\_surface(const char\* name, int\* frames, GRSurface\*\*\* pSurface) {

GRSurface\*\* surface = NULL;

int result = 0;

png\_structp png\_ptr = NULL;

png\_infop info\_ptr = NULL;

png\_uint\_32 width, height;

png\_byte channels;

int i;

png\_textp text;

int num\_text;

unsigned char\* p\_row;

unsigned int y;

\*pSurface = NULL;

\*frames = -1;

result = open\_png(name, &png\_ptr, &info\_ptr, &width, &height, &channels);

if (result < 0) return result;

\*frames = 1;

if (png\_get\_text(png\_ptr, info\_ptr, &text, &num\_text)) {

for (i = 0; i < num\_text; ++i) {

if (text[i].key && strcmp(text[i].key, "Frames") == 0 && text[i].text) {

\*frames = atoi(text[i].text);

break;

}

}

printf(" found frames = %d\n", \*frames);

}

if (height % \*frames != 0) {

printf("bad height (%d) for frame count (%d)\n", height, \*frames);

result = -9;

goto exit;

}

surface = reinterpret\_cast<GRSurface\*\*>(malloc(\*frames \* sizeof(GRSurface\*)));

if (surface == NULL) {

result = -8;

goto exit;

}

for (i = 0; i < \*frames; ++i) {

surface[i] = init\_display\_surface(width, height / \*frames);

if (surface[i] == NULL) {

result = -8;

goto exit;

}

}

#if defined(RECOVERY\_ABGR) || defined(RECOVERY\_BGRA)

png\_set\_bgr(png\_ptr);

#endif

p\_row = reinterpret\_cast<unsigned char\*>(malloc(width \* 4));

for (y = 0; y < height; ++y) {

png\_read\_row(png\_ptr, p\_row, NULL);

int frame = y % \*frames;

unsigned char\* out\_row = surface[frame]->data +

(y / \*frames) \* surface[frame]->row\_bytes;

transform\_rgb\_to\_draw(p\_row, out\_row, channels, width);

}

free(p\_row);

\*pSurface = reinterpret\_cast<GRSurface\*\*>(surface);

exit:

png\_destroy\_read\_struct(&png\_ptr, &info\_ptr, NULL);

if (result < 0) {

if (surface) {

for (i = 0; i < \*frames; ++i) {

if (surface[i]) free(surface[i]);

}

free(surface);

}

}

return result;

}

## trwp

<https://github.com/TeamWin/Team-Win-Recovery-Project>

pixelflinger？

# RecoveryUI

个简单的基于framebuffer的ui系统，叫miniui 主要建立了图像部分（gglInit、gr\_init\_font、framebuffer）及进度条和事件处理（input\_callback）

rm500:/dev/graphics $ ls -al | grep fb0

crw-rw---- 1 root graphics 29, 0 2013-01-18 08:50 fb0

在用手机app通过framebuffer截屏的时候，手机已经root过了，但是执行代码：

fb->fd = open("/dev/graphics/fb0", O\_RDONLY);

还是返回失败

当我们通过按键或者应用进入recovery模式，实质是kernel后加载recovery.img,kernel起来后执行的第一个进程就是init，此进程会读入init.rc启动相应的服务。在recovery模式中，启动的服务是执行recovery可执行文件，此文件是bootable/recovery/recovery.cpp文件生成，我们就从recovery.cpp文件开始分析

## PNG支持的种类

并非所有的PNG图片，在recovery下面都可以显示，这也是很多人会遇到的问题，明明是png图片，怎么还是不能显示呢，recovery原生对png图片的支持如下：

bit\_depth channels color\_type

8 3 2

<=8 1 0

<=8 1 3

其中bit\_depth是位深度, channels是指支持的颜色的种类，color\_type是值颜色类型。我们需要记住的是：

bit\_depth不能大于8。

color\_type值有 PNG\_COLOR\_TYPE\_GRAY（0）

PNG\_COLOR\_TYPE\_RGB（2）

PNG\_COLOR\_TYPE\_PALETTE（3）

channels值，与color\_type有关，PNG\_COLOR\_TYPE\_GRAY与PNG\_COLOR\_TYPE\_PALETTE时，只有一种颜色，PNG\_COLOR\_TYPE\_RGB时有三种颜色

当然，也不是什么图都可以的，在recovery中，所有的png图片必须是RGB且不带且不能带alhpa通道信息。

关于这一点，我们可以看open\_png这个函数:

if (bit\_depth == 8 && \*channels == 3 && color\_type == PNG\_COLOR\_TYPE\_RGB) {

// 8-bit RGB images: great, nothing to do.

} else if (bit\_depth <= 8 && \*channels == 1 && color\_type == PNG\_COLOR\_TYPE\_GRAY) {

// 1-, 2-, 4-, or 8-bit gray images: expand to 8-bit gray.

png\_set\_expand\_gray\_1\_2\_4\_to\_8(\*png\_ptr);

} else if (bit\_depth <= 8 && \*channels == 1 && color\_type == PNG\_COLOR\_TYPE\_PALETTE) {

// paletted images: expand to 8-bit RGB. Note that we DON'T

// currently expand the tRNS chunk (if any) to an alpha

// channel, because minui doesn't support alpha channels in

// general.

png\_set\_palette\_to\_rgb(\*png\_ptr);

\*channels = 3;

} else {

fprintf(stderr, "minui doesn't support PNG depth %d channels %d color\_type %d\n",

bit\_depth, \*channels, color\_type);

result = -7;

goto exit;

}

这个函数将图片文件的数据读取到内存，我在其中输出了一些调试信息，输出图片的 color\_type, channels 等信息。

查看LOG发现，android原生的图片 channels == 3，channels 即色彩通道个数，等于 3 的话，意味着只有 R,G,B 三个通道的信息，

没有 ALPHA 通道信息！这段代码的逻辑是如果channels 不等于3， 则按channels = 1 来处理，即灰度图。

美工给的图片是带 alpha通道信息的，即channels = 4，被当成灰度图像来处理了，怪不得显示的效果是灰度图像。

我一直以为 png 图像就只有一种格式，都是带有 alpha通道的。。。

使用图像处理工具(photoshop 或者 gimp)，将美工给的图片去掉 alpha 通道信息，再替换recovery 的图片，编译，替换recovery.img ，

reboot -r 。图片终于正常显示啦。

### 判断PNG属性方法

<http://blog.csdn.net/csdn66_2016/article/details/72303400>

### 6.4 png转换

    如果我拿到的png图片，不支持怎么办呢，我们可以使用PS工具，把png图片转换为recovery可支持的格式即可，如果又不太会使用PS工具呢，还有个更简单的方法，采用QQ的截图功能，然后另存为png图片即可，QQ截图保存的png是标准的RGB格式，即bit\_depth（8）channels（3）color\_type（2）

Android recovery和android本质上是两个独立的rootfs, 只是recovery这个rootfs存在的意义就是为android这个rootfs服务，因此被解释为Android系统的一部分。 recovery作为一个简单的rootfs, 提供了非常有限的几个功能，只包含了几个简单的库，UI的显示采用的是直接刷framebuffer的形式，作为android framework及app层的码农，对这种形式相对陌生，特抽点时间梳理了一番。 首先，浏览一下reocvery的main函数代码中UI相关的语句

1. main(**int** argc, **char** \*\*argv) {
3. ......
5. Device\* device = make\_device();
6. ui = device->GetUI();
7. gCurrentUI = ui;
9. ui->Init();
10. ui->SetLocale(locale);
11. ui->SetBackground(RecoveryUI::NONE);
12. **if** (show\_text) ui->ShowText(**true**);
14. ......
16. **if** (status != INSTALL\_SUCCESS || ui->IsTextVisible()) {
17. prompt\_and\_wait(device, status);
18. }
20. ......
21. }

1、首先新建了一个Device类的对象， Device类封装了一些操作，包括UI的操作

2、调用Device类的GetUI()返回一个DefaultUI对象，recovery中涉及到三个UI类，三个类之间为继承关系，分别为DefaultUI、 ScreenRecoveryUI、RecoveryUI

3、调用DefaultUI类的Init(), DefaultUI类没有Init()方法，因此将调用它的父类ScreenRecoveryUI的Init()

4、同理，调用ScreenRecoveryUI类的SetLocale()来标识几个比较特别的区域

5、同理，调用ScreenRecoveryUI类的SetBackground()设置初始状态的背景图

6、显示recovery的主界面，即一个选择菜单

## ScreenRecoveryUI::Init

Zcv

1. **void** ScreenRecoveryUI::Init()
2. {
3. gr\_init();
5. gr\_font\_size(&char\_width, &char\_height);
7. text\_col = text\_row = 0;
8. text\_rows = gr\_fb\_height() / char\_height;
9. **if** (text\_rows > kMaxRows) text\_rows = kMaxRows;
10. text\_top = 1;
12. text\_cols = gr\_fb\_width() / char\_width;
13. **if** (text\_cols > kMaxCols - 1) text\_cols = kMaxCols - 1;
15. LoadBitmap("icon\_installing", &backgroundIcon[INSTALLING\_UPDATE]);
16. backgroundIcon[ERASING] = backgroundIcon[INSTALLING\_UPDATE];
17. LoadBitmap("icon\_error", &backgroundIcon[ERROR]);
18. backgroundIcon[NO\_COMMAND] = backgroundIcon[ERROR];
20. LoadBitmap("progress\_empty", &progressBarEmpty);
21. LoadBitmap("progress\_fill", &progressBarFill);
23. LoadLocalizedBitmap("installing\_text", &backgroundText[INSTALLING\_UPDATE]);
24. LoadLocalizedBitmap("erasing\_text", &backgroundText[ERASING]);
25. LoadLocalizedBitmap("no\_command\_text", &backgroundText[NO\_COMMAND]);
26. LoadLocalizedBitmap("error\_text", &backgroundText[ERROR]);
28. **int** i;
30. progressBarIndeterminate = (gr\_surface\*)malloc(indeterminate\_frames \*
31. sizeof(gr\_surface));
32. **for** (i = 0; i  0) {
33. installationOverlay = (gr\_surface\*)malloc(installing\_frames \*
34. sizeof(gr\_surface));
35. **for** (i = 0; i

fgh

1、gr\_init() 初始化图形设备，分配Pixelflinger库渲染的内存

2、gr\_font\_size() 将字体对应的surface长宽赋值给char\_width和char\_height

3、LoadBitmap() 将png生成surface, 每个png图片对应一个surface, 所有surface存放在一个数组中

4、LoadLocalizedBitmap() 将区域文字所在的图片中的text信息根据当前的locale提取出来，生成对应的surface, 所以 surface也存放在一个数组中

6、pthread\_create(&progress\_t, NULL, progress\_thread, NULL) 创建一个线程，该线程的任务是一个死循环，在该循环中不停 地检测currentIcon以及progressBarType来决定是不是要更新进度条。

7、调用RecoveryUI的Init()，初始化输入事件处理。

### LoadBitmap

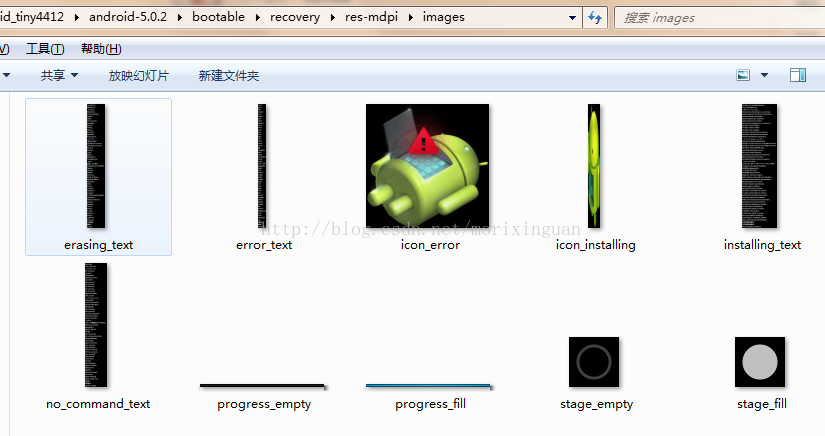
装载图片了，会调用到LoadBitmapArray和LoadBitmap这两个函数。其中，我们会看到这些函数里图片的名称:

"icon\_installing"、"icon\_error"、"progress\_empty"、"progress\_fill"、

"stage\_empty"、"stage\_fill"、"installing\_text"、"erasing\_text"、

"no\_command\_text"、"error\_text"

将上面的字符串与下面的图片一一对应：



那么这些分别是怎么显示的？其中erasing\_text是用来显示做清除的时候显示的文字,放大后如下：

这上面有许许多多的语言版本，我们可以根据需要来选择，这些主要要看接下来初始化文字的代码逻辑。

其余的图片中，后缀带text的，也和这些是类似的，有出现错误显示的字体error\_text，更新系统显示的字体installing\_text，没有命令的时候显示的字体no\_command\_text。

除了文字显示，我们最关心的就是icon\_installing这张图片了，在做系统更新的时候，这个机器人会转动。这不是动画吗？怎么只有一张图片呢？我们找到Android官方网站看看是为什么？

替换Android原生态中的图片，换成我们自己的图片，

### 机器人动画实现

Android5.x以上的版本，机器人的动画是PNG图片和帧动画组成的，  
//我们可以使用recovery目录下的interlace-frames.py这个python脚本来进行合成，  
//具体的合成方法需要使用recovery代码中的一个python合成工具。  
//我就可以把android原生态的动画给换了，因为这个机器人实在是丑。

#### LoadBitmapArray

void ScreenRecoveryUI::**LoadBitmapArray**(const char\* filename, int\* frames, gr\_surface\*\* surface) {

int result = **res\_create\_multi\_display\_surface**(filename, frames, surface);

if (result < 0) {

LOGE("missing bitmap %s\n(Code %d)\n", filename, result);

}

}

调完这个函数后会调用resources.cpp中的res\_create\_multi\_display\_surface函数用于显示，源码如下：

## ScreenRecoveryUI::SetLocale

Dgh

1. **void** ScreenRecoveryUI::SetLocale(**const** **char**\* locale) {
2. **if** (locale) {
3. **char**\* lang = strdup(locale);
4. **for** (**char**\* p = lang; \*p; ++p) {
5. **if** (\*p == '\_') {
6. \*p = '\0';
7. **break**;
8. }
9. }
11. // A bit cheesy: keep an explicit list of supported languages
12. // that are RTL.
13. **if** (strcmp(lang, "ar") == 0 ||   // Arabic
14. strcmp(lang, "fa") == 0 ||   // Persian (Farsi)
15. strcmp(lang, "he") == 0 ||   // Hebrew (new language code)
16. strcmp(lang, "iw") == 0 ||   // Hebrew (old language code)
17. strcmp(lang, "ur") == 0) {   // Urdu
18. rtl\_locale = **true**;
19. }
20. free(lang);
21. }
22. }

ScreenRecoveryUI类的SetLocale， 该函数根据locale判断所用的字体是否属于阿拉伯语系，阿拉伯语的书写习惯是从右到左，如果是阿拉伯语系的话，就设置一个标志，后面根据这个标志决定从右到左显示文字或进度条。SetLocale的参数locale赋值逻辑是这样的，先从command文件中读取, command文件中设置locale的命令如"--locale=zh\_CN“，如果没有传入locale,初始化过程中会尝试从/cache/recovery/last\_locale中读取locale, 如果该文件也没有，则locale不会被赋值，就默认用English.

## ScreenRecoveryUI::SetBackground

Dfhg

1. **void** ScreenRecoveryUI::SetBackground(Icon icon)
2. {
3. pthread\_mutex\_lock(&updateMutex);
5. // Adjust the offset to account for the positioning of the
6. // base image on the screen.
7. **if** (backgroundIcon[icon] != NULL) {
8. gr\_surface bg = backgroundIcon[icon];
9. gr\_surface text = backgroundText[icon];
10. overlay\_offset\_x = install\_overlay\_offset\_x + (gr\_fb\_width() - gr\_get\_width(bg)) / 2;
11. overlay\_offset\_y = install\_overlay\_offset\_y +
12. (gr\_fb\_height() - (gr\_get\_height(bg) + gr\_get\_height(text) + 40)) / 2;
13. }
15. currentIcon = icon;
16. update\_screen\_locked();
18. pthread\_mutex\_unlock(&updateMutex);
19. }

S

SetBackground函数比较简洁，关键部分在update\_screen\_locked，下面我们重点分析一下。 update\_screen\_locked和update\_progress\_locked是recovery的UI部分的关键函数，update\_screen\_locked用来更新背景, update\_progress\_locked用来更新进度条，因为显示的画面会一直在更新，所以这两个函数会在不同的地方被反复调用

## ScreenRecoveryUI::update\_screen\_locked

1. **void** ScreenRecoveryUI::update\_screen\_locked()
2. {
3. draw\_screen\_locked();
4. gr\_flip();
5. }

update\_screen\_locked包含两个操作，一是更新screen, 二是切换前后buffer。

## ScreenRecoveryUI::draw\_screen\_locked()

Tr

1. **void** ScreenRecoveryUI::draw\_screen\_locked()
2. {
3. draw\_background\_locked(currentIcon);
4. draw\_progress\_locked();
6. **if** (show\_text) {
7. SetColor(TEXT\_FILL);
8. gr\_fill(0, 0, gr\_fb\_width(), gr\_fb\_height());
10. **int** y = 0;
11. **int** i = 0;
12. **if** (show\_menu) {
13. SetColor(HEADER);
15. **for** (; i  y+2 && count
17. draw\_background\_locked函数的实现代码中又出现了几个以gr\_开头的函数，以gr\_开头的函数来自minui库，minui库的代码在recovery源码下的minui目录下，minui提供的接口实现了图形的描绘以及固定大小的文字显示。


21. gr\_color(unsigned **char** r, unsigned **char** g, unsigned **char** b, unsigned **char** a);  /\* 设置字体颜色 \*/
22. gr\_fill(**int** x, **int** y, **int** w, **int** h);  /\* 填充矩形区域,参数分别代表起始坐标、矩形区域大小 \*/
23. gr\_blit(gr\_surface source, **int** sx, **int** sy, **int** w, **int** h, **int** dx, **int** dy);  /\* 填充由source指定的图片 \*/

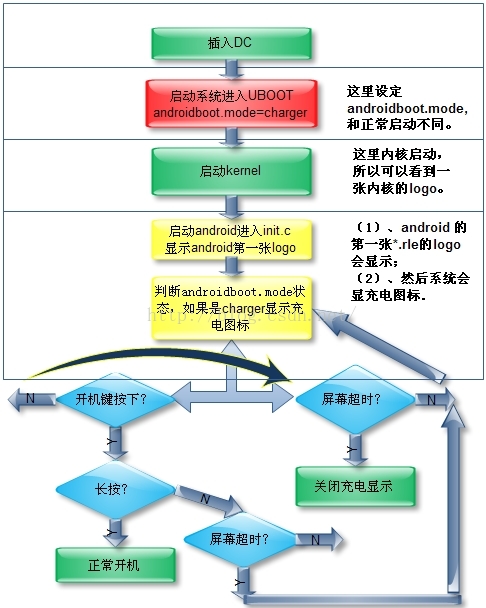

27. draw\_background\_locked函数先将整个渲染buffer填充为黑色，然后计算背景surface的长宽，文字surface的长宽，再结合fb的长宽计算出背景surface以及文字surface显示的坐标，有长宽和坐标就可以调用Pixelflinger的接口在渲染buffer上进行渲染。

## ScreenRecoveryUI::draw\_progress\_locked()

1. **void** ScreenRecoveryUI::draw\_progress\_locked()
2. {
3. **if** (currentIcon == ERROR) **return**;
5. **if** (currentIcon == INSTALLING\_UPDATE || currentIcon == ERASING) {
6. draw\_install\_overlay\_locked(installingFrame);
7. }
9. **if** (progressBarType != EMPTY) {
10. **int** iconHeight = gr\_get\_height(backgroundIcon[INSTALLING\_UPDATE]);
11. **int** width = gr\_get\_width(progressBarEmpty);
12. **int** height = gr\_get\_height(progressBarEmpty);
14. **int** dx = (gr\_fb\_width() - width)/2;
15. **int** dy = (3\*gr\_fb\_height() + iconHeight - 2\*height)/4;
17. // Erase behind the progress bar (in case this was a progress-only update)
18. gr\_color(0, 0, 0, 255);
19. gr\_fill(dx, dy, width, height);
21. **if** (progressBarType == DETERMINATE) {
22. **float** p = progressScopeStart + progress \* progressScopeSize;
23. **int** pos = (**int**) (p \* width);
25. **if** (rtl\_locale) {
26. // Fill the progress bar from right to left.
27. **if** (pos > 0) {
28. gr\_blit(progressBarFill, width-pos, 0, pos, height, dx+width-pos, dy);
29. }
30. **if** (pos  0) {
31. gr\_blit(progressBarFill, 0, 0, pos, height, dx, dy);
32. }
33. **if** (pos
35. draw\_progress\_locked函数的原理与 update\_screen\_locked函数类似， 最终是将进度条的surface输出到渲染buffer,
36. recovery中各个场景的画面，就是由背景、文字、进度条的重叠，不同的是所用的surface 以及surface的坐标。
38. recovery main函数中的UI代码基本上已经分析过了，最后一点主菜单的显示，就是通过上面介绍的这些接口将文字图片显示出来，因此就不再多讲。总的来说，recovery的UI显示部分难度不大，应用层调用minui库实现了图形的描绘以及固定大小的文字显示，minui库调用了Pixelflinger库来进行渲染。

# android 关机充电动画

与正常开机过程相比，这个流程只走到init这一部分，就没有往后走了，这部分我们会在后面的代码中分析。如下图

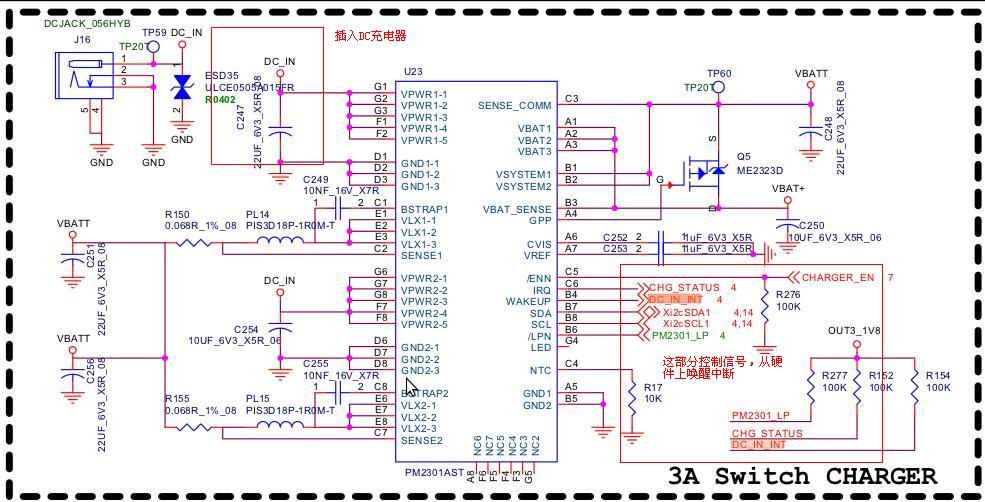


我们主要是修改上层的UI界面，所以底层的流程有个大概的了解就行。

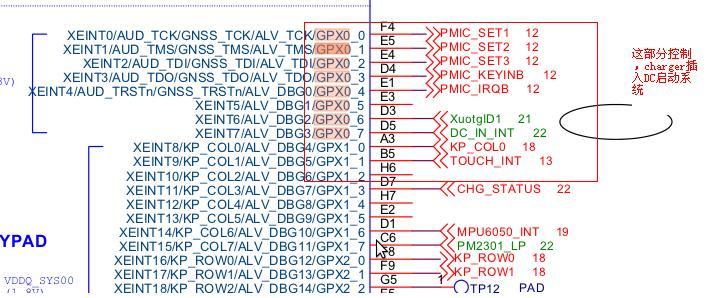
好了，下面我们就在Android源码上把关机充电的样式改为LG手机充电的风格。就那本人之前修改过做一个总结。

## 硬件逻辑

插入DC，charger IC从硬件上唤醒系统，相当于长按开机键开机。



下面这部分是charger IC连接系统的控制部分。



## 软件逻辑，UBOOT？

DC插入，其实相当于关机状态下“按开机键”开机。第一步要走UBOOT、kernel 、android init这一流程。

1、UBOOT：UBOOT启动代码我们不在这里详细分析，这里我们只要注意二个问题：

a：如何判断是DC插入；

b：设定setenv("bootargs", "androidboot.mode=charger")，androidboot.mode这个参数相当重要，这个参数决定系统是正常启动、还是关机充电状态。

Uboot/board/samsung/smdk4212/smkd4212.c

1. **int** board\_late\_init (**void**)
2. {
3. **int** keystate = 0;
4. printf("check start mode\n");
5. **if** ((\*(**int** \*)0x10020800==0x19721212) || (\*(**int** \*)0x10020804==0x19721212)
6. || (\*(**int** \*)0x10020808==0x19721212)) //（1）、检查是否有DC插入；
7. {
8. setenv ("bootargs", "");//（2）、没有DC插入；
9. } **else**  {//DC插入
10. **int** tmp=\*(**int** \*)0x11000c08;
11. \*(**int** \*)0x10020800=\*(**int** \*)0x10020804=0x19721212;
12. \*(**int** \*)0x11000c08=(tmp&(~0xc000))|0xc000;
13. udelay(10000);
14. **if** ((\*(**int** \*)0x11000c04 & 0x80)!=0x80 && INF\_REG4\_REG != 0xf) {
15. setenv ("bootargs", "androidboot.mode=charger");//（3）、设定bootargs为charger状态
16. printf("charger mode\n");
17. } **else** {
18. setenv ("bootargs", "");
19. }
20. \*(**int** \*)0x11000c08=tmp;
21. }
22. #ifdef CONFIG\_CPU\_EXYNOS4X12
23. **int** charge\_status=CheckBatteryLow();//（4）、检查电池电量；
24. keystate=board\_key\_check();//（5）、检查按键状态；
25. // fuse bootloader
26. **if**(second\_boot\_info != 0) {
27. boot\_symbol=1;
28. INF\_REG2\_REG =0x8;
29. run\_command(CONFIG\_BOOTCMD\_FUSE\_BOOTLOADER, NULL);
30. }
31. **if**((INF\_REG4\_REG == 0xd)) {
32. // reboot default
33. **char** buf[10];
34. sprintf(buf, "%d", CONFIG\_BOOTDELAY);
35. setenv ("bootdelay", buf);
36. setenv ("reserved", NULL);
37. saveenv();
38. } **else** **if**((INF\_REG4\_REG == 0xe) || keystate == (0x1 | 0x2)) {//（6）、按键进入fastboot模式；
39. // reboot bootloader
40. boot\_symbol=1;
41. INF\_REG2\_REG =0x8;
42. printf("BOOTLOADER - FASTBOOT\n");
43. setenv ("reserved", "fastboot");
44. setenv ("bootdelay", "0");
45. } **else** **if**((INF\_REG4\_REG == 0xf) || keystate == (0x1 | 0x2 | 0x4)) {//（7）、按键进入recovery模式；
46. // reboot recovery
47. printf("BOOTLOADER - RECOVERY\n");
48. boot\_symbol=1;
49. INF\_REG2\_REG =0x8;
50. setenv ("reserved", CONFIG\_BOOTCMD\_RECOVERY);
51. setenv ("bootdelay", "0");
52. } **else**
53. **if**(keystate == (0x1 | 0x4) || second\_boot\_info != 0 || partition\_check()) {//（8）、按键进入卡升级模式；
54. // 2nd boot
55. printf("BOOTLOADER - 2ND BOOT DEVICE\n");
56. boot\_symbol=1;
57. INF\_REG2\_REG =0x8;
58. setenv ("bootcmd", CONFIG\_BOOTCOMMAND);
59. setenv ("reserved", CONFIG\_BOOTCMD\_FUSE\_RELEASE);
60. setenv ("bootdelay", "0");
61. } **else** {//（9）、正常启动；
62. // normal case
63. **char** buf[10];
64. sprintf(buf, "%d", CONFIG\_BOOTDELAY);
65. setenv ("bootdelay", buf);
66. }
67. INF\_REG4\_REG = 0;
68. **return** 0;
69. }

### 检查是否有DC插入；

**[cpp]** [view plaincopy](http://blog.csdn.net/xubin341719/article/details/8498580)

1. **if** ((\*(**int** \*)0x10020800==0x19721212) || (\*(**int** \*)0x10020804==0x19721212)
2. (\*(**int** \*)0x10020808==0x19721212))

这部分检查寄存器的值

### 没有DC插入

### 设定bootargs为charger状态

H

**[cpp]** [view plaincopy](http://blog.csdn.net/xubin341719/article/details/8498580)

1. **if** ((\*(**int** \*)0x11000c04 & 0x80)!=0x80 && INF\_REG4\_REG != 0xf) {
2. setenv ("bootargs", "androidboot.mode=charger");

**[cpp]** [view plain](http://blog.csdn.net/bi511304183/article/details/9303043) [copy](http://blog.csdn.net/bi511304183/article/details/9303043)

1. **if** ((\*(**int** \*)0x11000c04 & 0x80)!=0x80 && INF\_REG4\_REG != 0xf) {
2. setenv ("bootargs", "androidboot.mode=charger");

**这是这部分的重点，如果能过寄存器判断是DC插入，把androidboot.mode设定为charger状态。**

（4）、检查电池电量；

这个在正常开机状态下，如果检测电量太低，则不开机，这部分代码就不做分析。

（5）、检查按键状态；

我们这个平台有几种模式:fastboot\recovery\卡升级等……

（6）、按键进入fastboot模式；

（7）、按键进入recovery模式；

（8）、按键进入卡升级模式

（9）、正常启动；

## Kernel

同正常流程

## init

前面所有的描述其实只有一点和正常启动不太一样，那就是在UBOOT中把androidboot.mode设定为charger状态，内核正常流程启动，然后到init时要对charger这种状态处理。

system\core\init\init.c

**[cpp]** [view plain](http://blog.csdn.net/bi511304183/article/details/9303043) [copy](http://blog.csdn.net/bi511304183/article/details/9303043)

1. **int** main(**int** argc, **char** \*\*argv)
2. {
3. ………………
4. action\_for\_each\_trigger("early-init", action\_add\_queue\_tail);
6. queue\_builtin\_action(wait\_for\_coldboot\_done\_action, "wait\_for\_coldboot\_done");
7. queue\_builtin\_action(property\_init\_action, "property\_init");
8. queue\_builtin\_action(keychord\_init\_action, "keychord\_init");
9. queue\_builtin\_action(console\_init\_action, "console\_init");  //(1)、显示initlogo.rle，也就是android第二张图片;
10. queue\_builtin\_action(set\_init\_properties\_action, "set\_init\_properties");
12. /\* execute all the boot actions to get us started \*/
13. action\_for\_each\_trigger("init", action\_add\_queue\_tail);
15. /\* skip mounting filesystems in charger mode \*/
16. **if** (strcmp(bootmode, "charger") != 0) {//(2)、这里就是UBOOT中设定的bootmode，如果是charger模式，跳过下面初始化；
17. action\_for\_each\_trigger("early-fs", action\_add\_queue\_tail);
18. action\_for\_each\_trigger("fs", action\_add\_queue\_tail);
19. action\_for\_each\_trigger("post-fs", action\_add\_queue\_tail);
20. action\_for\_each\_trigger("post-fs-data", action\_add\_queue\_tail);
21. }
23. queue\_builtin\_action(property\_service\_init\_action, "property\_service\_init");
24. queue\_builtin\_action(signal\_init\_action, "signal\_init");
25. queue\_builtin\_action(check\_startup\_action, "check\_startup");
27. **if** (!strcmp(bootmode, "charger")) {//(3)、如果为charger，则调用charger.c。
28. action\_for\_each\_trigger("charger", action\_add\_queue\_tail);
29. } **else** {
30. action\_for\_each\_trigger("early-boot", action\_add\_queue\_tail);
31. action\_for\_each\_trigger("boot", action\_add\_queue\_tail);
32. }
33. ……………………
34. }

### charger模式

这里就是UBOOT中设定的bootmode，

如果是charger模式，跳过下面初始化；

**[cpp]** [view plaincopy](http://blog.csdn.net/xubin341719/article/details/8498580)

1. /\* skip mounting filesystems in charger mode \*/
2. **if** (strcmp(bootmode, "charger") != 0) {
3. action\_for\_each\_trigger("early-fs", action\_add\_queue\_tail);
4. action\_for\_each\_trigger("fs", action\_add\_queue\_tail);
5. action\_for\_each\_trigger("post-fs", action\_add\_queue\_tail);
6. action\_for\_each\_trigger("post-fs-data", action\_add\_queue\_tail);
7. }

调用charger.c

**[cpp]** [view plaincopy](http://blog.csdn.net/xubin341719/article/details/8498580)

1. action\_for\_each\_trigger("charger", action\_add\_queue\_tail);

## charger

旧版是system\core\charger\charger.c 。Android5.0源码位置统一放在healthd？中，

Healthd的主函数调用被healthd\_mode\_charger重载

**[cpp]** [view plain](http://blog.csdn.net/bi511304183/article/details/9303043) [copy](http://blog.csdn.net/bi511304183/article/details/9303043)

1. **int** main(**int** argc, **char** \*\*argv)
2. {
3. ………………
4. klog\_set\_level(CHARGER\_KLOG\_LEVEL);
5. dump\_last\_kmsg();
6. LOGI("--------------- STARTING CHARGER MODE ---------------\n");
8. gr\_init();
9. gr\_font\_size(&char\_width, &char\_height); //(1)、初始化graphics,包括buf大小；
11. ev\_init(input\_callback, charger);//(2)初始化按键；
13. fd = uevent\_open\_socket(64\*1024, **true**);
14. **if** (fd >= 0) {
15. fcntl(fd, F\_SETFL, O\_NONBLOCK);
16. ev\_add\_fd(fd, uevent\_callback, charger);
17. }
19. charger->uevent\_fd = fd;
20. coldboot(charger, "/sys/class/power\_supply", "add");//(3)、创建/sys/class/power\_supply结点，把socket信息通知应用层；
22. ret = res\_create\_surface("charger/battery\_fail", &charger->surf\_unknown);
23. **if** (ret < 0) {
24. LOGE("Cannot load image\n");
25. charger->surf\_unknown = NULL;
26. }
27. **for** (i = 0; i < charger->batt\_anim->num\_frames; i++) {//(4)、这里是显示charger logo，res\_create\_surface显示图片函数；
28. **struct** frame \*frame = &charger->batt\_anim->frames[i];
29. ret = res\_create\_surface(frame->name, &frame->surface);
30. **if** (ret < 0) {
31. LOGE("Cannot load image %s\n", frame->name);
32. /\* TODO: free the already allocated surfaces... \*/
33. charger->batt\_anim->num\_frames = 0;
34. charger->batt\_anim->num\_cycles = 1;
35. **break**;
36. }
37. }
38. ev\_sync\_key\_state(set\_key\_callback, charger);
39. gr\_fb\_blank(**true**);
41. charger->next\_screen\_transition = now - 1;
42. charger->next\_key\_check = -1;
43. charger->next\_pwr\_check = -1;
44. reset\_animation(charger->batt\_anim);
45. kick\_animation(charger->batt\_anim);
46. event\_loop(charger);//（5）、event\_loop循环，电池状态，检测按键是否按下；
47. **return** 0;
49. }

### 创建/sys/class/power\_supply结点，把socket信息通知应用层

创建了socket接口获得uevent的文件描述符，然后触发/sys/class/power\_supply目录及其子目录下的uevent，然后接受并创建设备节点，至此设备节点才算创建。

### 这里显示charger logo，res\_create\_surface显示图片函数；

res\_create\_surface:minui/resource.c[create surfaces for all bitmaps used later, include icons, bmps]

创建surface为所以的位图，包括图标、位图。 这些图片的位置为：system\core\charger\images

### event\_loop

，检测按键是否按下；

这个函数判断按键状态，DC是否插拔。如果长按开机：执行android\_reboot(ANDROID\_RB\_RESTART,0, 0);如果拔出DC:执行android\_reboot(ANDROID\_RB\_POWEROFF,0, 0);

**[cpp]** [view plaincopy](http://blog.csdn.net/xubin341719/article/details/8498580)

1. **static** **void** event\_loop(**struct** charger \*charger)
2. {
3. **int** ret;
4. **while** (**true**) {
5. int64\_t now = curr\_time\_ms();//（1）、获得当前时间；
6. LOGV("[%lld] event\_loop()\n", now);
7. handle\_input\_state(charger, now);//（2）、检查按键状态；
8. handle\_power\_supply\_state(charger, now);// (3)、检查DC是否拔出；
9. /\* do screen update last in case any of the above want to start
10. \* screen transitions (animations, etc)
11. \*/
12. update\_screen\_state(charger, now);//(4)、对按键时间状态标志位的判断，显示不同电量的充电logo;
13. wait\_next\_event(charger, now);
14. }
15. }

F

（1）、获得当前时间；

   int64\_t now = curr\_time\_ms();

       这个时间来判断，有没有屏幕超时，如果超时关闭屏幕充电logo显示。

（2）、检查按键状态；

**[cpp]** [view plaincopy](http://blog.csdn.net/xubin341719/article/details/8498580)

1. **static** **void** handle\_input\_state(**struct** charger \*charger, int64\_t now)
2. {
3. process\_key(charger, KEY\_POWER, now);
4. **if** (charger->next\_key\_check != -1 && now > charger->next\_key\_check)
5. charger->next\_key\_check = -1;
6. }
7. 我们再看下：process\_key(charger, KEY\_POWER, now);
8. **static** **void** process\_key(**struct** charger \*charger, **int** code, int64\_t now)
9. {
10. ………………
11. **if** (code == KEY\_POWER) {
12. **if** (key->down) {
13. int64\_t reboot\_timeout = key->timestamp + POWER\_ON\_KEY\_TIME;
14. **if** (now >= reboot\_timeout) {//如果长按power键，就重新启动，也就是重启开机；
15. LOGI("[%lld] rebooting\n", now);
16. android\_reboot(ANDROID\_RB\_RESTART, 0, 0);//重启命令；
17. }
18. ………………
19. }
21. key->pending = **false**;
22. }

D

（3）、检查DC是否拔出；

handle\_power\_supply\_state(charger, now);

**[cpp]** [view plaincopy](http://blog.csdn.net/xubin341719/article/details/8498580)

1. **static** **void** handle\_power\_supply\_state(**struct** charger \*charger, int64\_t now)
2. {
3. **if** (charger->num\_supplies\_online == 0) {
4. **if** (charger->next\_pwr\_check == -1) {
5. charger->next\_pwr\_check = now + UNPLUGGED\_SHUTDOWN\_TIME;
6. LOGI("[%lld] device unplugged: shutting down in %lld (@ %lld)\n",
7. now, UNPLUGGED\_SHUTDOWN\_TIME, charger->next\_pwr\_check);
8. } **else** **if** (now >= charger->next\_pwr\_check) {
9. LOGI("[%lld] shutting down\n", now);
10. android\_reboot(ANDROID\_RB\_POWEROFF, 0, 0);//如果DC拔出，则关机；
11. }
12. ………………
13. }

### 显示不同电量的充电logo;

对按键时间状态标志位的判断，显示不同电量的充电logo;

update\_screen\_state(charger, now);

这个函数比较长了，其实做用就是：我们在状态的过程中,充电logo的电量是要增加的，比如电量是20%时，要从第一格开始闪烁；如果是80%时，则要从第三格开始闪烁，电量显示就是通过这个函数来计算实现的。

**[cpp]** [view plaincopy](http://blog.csdn.net/xubin341719/article/details/8498580)

1. **static** **void** update\_screen\_state(**struct** charger \*charger, int64\_t now)
2. {
3. **struct** animation \*batt\_anim = charger->batt\_anim;
4. **int** cur\_frame;
5. **int** disp\_time;
7. **if** (!batt\_anim->run || now < charger->next\_screen\_transition)
8. **return**;
10. /\* animation is over, blank screen and leave \*/
11. **if** (batt\_anim->cur\_cycle == batt\_anim->num\_cycles) {
12. reset\_animation(batt\_anim);
13. charger->next\_screen\_transition = -1;
14. gr\_fb\_blank(**true**);
15. LOGV("[%lld] animation done\n", now);
16. **return**;
17. }
19. disp\_time = batt\_anim->frames[batt\_anim->cur\_frame].disp\_time;
21. /\* animation starting, set up the animation \*/
22. **if** (batt\_anim->cur\_frame == 0) {
23. **int** batt\_cap;
24. **int** ret;
26. LOGV("[%lld] animation starting\n", now);
27. batt\_cap = get\_battery\_capacity(charger);
28. **if** (batt\_cap >= 0 && batt\_anim->num\_frames != 0) {
29. **int** i;
31. /\* find first frame given current capacity \*/
32. **for** (i = 1; i < batt\_anim->num\_frames; i++) {
33. **if** (batt\_cap < batt\_anim->frames[i].min\_capacity)
34. **break**;
35. }
36. batt\_anim->cur\_frame = i - 1;
38. /\* show the first frame for twice as long \*/
39. disp\_time = batt\_anim->frames[batt\_anim->cur\_frame].disp\_time \* 2;
40. }
42. batt\_anim->capacity = batt\_cap;
43. }
45. /\* unblank the screen  on first cycle \*/
46. **if** (batt\_anim->cur\_cycle == 0)
47. gr\_fb\_blank(**false**);
49. /\* draw the new frame (@ cur\_frame) \*/
50. redraw\_screen(charger);
52. /\* if we don't have anim frames, we only have one image, so just bump
53. \* the cycle counter and exit
54. \*/
55. **if** (batt\_anim->num\_frames == 0 || batt\_anim->capacity < 0) {
56. LOGV("[%lld] animation missing or unknown battery status\n", now);
57. charger->next\_screen\_transition = now + BATTERY\_UNKNOWN\_TIME;
58. batt\_anim->cur\_cycle++;
59. **return**;
60. }
62. /\* schedule next screen transition \*/
63. charger->next\_screen\_transition = now + disp\_time;
65. /\* advance frame cntr to the next valid frame
66. \* if necessary, advance cycle cntr, and reset frame cntr
67. \*/
68. batt\_anim->cur\_frame++;
70. /\* if the frame is used for level-only, that is only show it when it's
71. \* the current level, skip it during the animation.
72. \*/
73. **while** (batt\_anim->cur\_frame < batt\_anim->num\_frames &&
74. batt\_anim->frames[batt\_anim->cur\_frame].level\_only)
75. batt\_anim->cur\_frame++;
76. **if** (batt\_anim->cur\_frame >= batt\_anim->num\_frames) {
77. batt\_anim->cur\_cycle++;
78. batt\_anim->cur\_frame = 0;
80. /\* don't reset the cycle counter, since we use that as a signal
81. \* in a test above to check if animation is over
82. \*/
83. }
84. }

**[cpp]** [view plain](http://blog.csdn.net/bi511304183/article/details/9303043) [copy](http://blog.csdn.net/bi511304183/article/details/9303043)

1. **static** **void** update\_screen\_state(**struct** charger \*charger, int64\_t now)
2. {
3. **struct** animation \*batt\_anim = charger->batt\_anim;
4. **int** cur\_frame;
5. **int** disp\_time;
7. **if** (!batt\_anim->run || now < charger->next\_screen\_transition)
8. **return**;
10. /\* animation is over, blank screen and leave \*/
11. **if** (batt\_anim->cur\_cycle == batt\_anim->num\_cycles) {
12. reset\_animation(batt\_anim);
13. charger->next\_screen\_transition = -1;
14. gr\_fb\_blank(**true**);
15. LOGV("[%lld] animation done\n", now);
16. **return**;
17. }
19. disp\_time = batt\_anim->frames[batt\_anim->cur\_frame].disp\_time;
21. /\* animation starting, set up the animation \*/
22. **if** (batt\_anim->cur\_frame == 0) {
23. **int** batt\_cap;
24. **int** ret;
26. LOGV("[%lld] animation starting\n", now);
27. batt\_cap = get\_battery\_capacity(charger);
28. **if** (batt\_cap >= 0 && batt\_anim->num\_frames != 0) {
29. **int** i;
31. /\* find first frame given current capacity \*/
32. **for** (i = 1; i < batt\_anim->num\_frames; i++) {
33. **if** (batt\_cap < batt\_anim->frames[i].min\_capacity)
34. **break**;
35. }
36. batt\_anim->cur\_frame = i - 1;
38. /\* show the first frame for twice as long \*/
39. disp\_time = batt\_anim->frames[batt\_anim->cur\_frame].disp\_time \* 2;
40. }
42. batt\_anim->capacity = batt\_cap;
43. }
45. /\* unblank the screen  on first cycle \*/
46. **if** (batt\_anim->cur\_cycle == 0)
47. gr\_fb\_blank(**false**);
49. /\* draw the new frame (@ cur\_frame) \*/
50. redraw\_screen(charger);
52. /\* if we don't have anim frames, we only have one image, so just bump
53. \* the cycle counter and exit
54. \*/
55. **if** (batt\_anim->num\_frames == 0 || batt\_anim->capacity < 0) {
56. LOGV("[%lld] animation missing or unknown battery status\n", now);
57. charger->next\_screen\_transition = now + BATTERY\_UNKNOWN\_TIME;
58. batt\_anim->cur\_cycle++;
59. **return**;
60. }
62. /\* schedule next screen transition \*/
63. charger->next\_screen\_transition = now + disp\_time;
65. /\* advance frame cntr to the next valid frame
66. \* if necessary, advance cycle cntr, and reset frame cntr
67. \*/
68. batt\_anim->cur\_frame++;
70. /\* if the frame is used for level-only, that is only show it when it's
71. \* the current level, skip it during the animation.
72. \*/
73. **while** (batt\_anim->cur\_frame < batt\_anim->num\_frames &&
74. batt\_anim->frames[batt\_anim->cur\_frame].level\_only)
75. batt\_anim->cur\_frame++;
76. **if** (batt\_anim->cur\_frame >= batt\_anim->num\_frames) {
77. batt\_anim->cur\_cycle++;
78. batt\_anim->cur\_frame = 0;
80. /\* don't reset the cycle counter, since we use that as a signal
81. \* in a test above to check if animation is over
82. \*/
83. }
84. }

下面是不能容量时显示logo的函数：

**[cpp]** [view plaincopy](http://blog.csdn.net/xubin341719/article/details/8498580)

1. **static** **struct** frame batt\_anim\_frames[] = {
2. {
3. .name = "charger/battery\_0",
4. .disp\_time = 750,
5. .min\_capacity = 0,
6. },
7. {
8. .name = "charger/battery\_1",
9. .disp\_time = 750,
10. .min\_capacity = 20,
11. },
12. {
13. .name = "charger/battery\_2",
14. .disp\_time = 750,
15. .min\_capacity = 40,
16. },
17. {
18. .name = "charger/battery\_3",
19. .disp\_time = 750,
20. .min\_capacity = 60,
21. },
22. {
23. .name = "charger/battery\_4",
24. .disp\_time = 750,
25. .min\_capacity = 80,
26. .level\_only = **true**,
27. },
28. {
29. .name = "charger/battery\_5",
30. .disp\_time = 750,
31. .min\_capacity = BATTERY\_FULL\_THRESH,
32. },
33. };

draw\_surface\_centered完成了最终的绘制

# Pixelflinger

Pixelflinger的源代码位于system/core/libpixelflinger。

头文件位于system/core/include/libpixelflinger和system/core/include/private/pixelflinger。

http://blog.csdn.net/jing\_huanhuan/article/details/6556990

# REF

[Android关机充电流程](http://blog.csdn.net/bi511304183/article/details/9303043)

这部分和正常启动是一样的。

一：首先要找到关机充电显示图片所在的位置：mediatek/custom/common/lk/logo

在这个目录下找到你的源码工程配置信息向对应的目录，我的工程对应的文件是cu\_hd720 ，然后把需要的图片给替换掉