**KaiOS 笔记**

代码中使用dump("XXX");来添加日志

1. 应用间的跳转

自定义一个MozActivity步骤：

首先需要去某个application下得manifest.webapp定义一个activity

"activities":{

"calendar":{ //name

"data":{

"from":"clock" //从哪跳转

},

"disposition": "window"

}

}

还有一些参数可以参考https://developer.mozilla.org/en-US/docs/Archive/B2G\_OS/API/Web\_Activities#Starting\_an\_activity

然后在某个地方去调用这个activity

new MozActivity({

name:'calendar',

data:{

//一些需要传递的数据

from:"clock"

}

})

这样就能够成功从某个地方跳转到想要去的calendar应用程序了

2. KaiOS系统使用的存储方式之一：dataStore

2.1 数据存储所有者的清单必须包含datastores-owned声明所有权的字段，如：

"datastores-owned": {

"myData": {

"access": "readwrite",

"description": "my data store"

}

}

您可以包含多个属性来表示不同的数据存储，每个属性可以使用readonly/ readwrite指定是否可以由其他应用程序读取/修改数据存储。还包括描述以描述数据存储的目的。

2.2 其他想要访问（不拥有）数据存储的应用程序必须包含该datastores-access字段，如：

"datastores-access": {

"myData": {

"access": "readonly",

"description": "Read and modify my data store"

}

}

如果未指定此字段，则默认行为为“无访问权限”。同样，如果要访问多个数据存储，可以包含多个属性，并且可以设置访问权限readonly或readwrite可以设置为声明应用程序需要哪种访问类型

2.3 包含的api接口有：

DataStore：此接口表示检索的数据集，并包括用于访问商店名称，所有者等的标准属性，用于读取，修改和同步数据的方法，以及onchange用于对数据的更改作出反应的事件处理程序。

DataStoreCursor：此接口允许应用程序遍历DataStoreTask表示数据存储的更改历史记录的对象列表，以便在同步数据时使用。

DataStoreChangeEvent：此接口表示与数据存储中更改的记录相关的事件，即，一旦进行更改并change触发事件（请参阅DataStore.onchange处理程序），就会返回此事件，以便在同步各个更改时使用。

DataStoreTask：此接口表示当a DataStoreCursor用于遍历数据存储的更改历史记录时，数据存储中更改的记录

使用案例：

在emergency-call调用contact中的存储数据（在system中调用contact中的紧急联系人数据）

在system中的清单文件（manifest.webapp）中配置：

"datastores-access": {

"ice\_contacts": {

"readonly": true

}

}

获取ICE联系人：

ICEStore.getContacts().then((result) =>{

var iceContacts = [];

if (result.length>2) {

for (var i = 0; i <2 ; i++) {

iceContacts.push(result[i])

}

}else{

iceContacts = result;

}

iceContacts.forEach((iceContact) =>{

//联系人查找规则，我们是通过id进行查找的

var contactFilter = {

filterBy: ['id'],

filterValue: iceContact,

filterOp: 'equals',

filterLimit: 1

};

//使用联系人的查找api去查找

var contactRequest = navigator.mozContacts.find(contactFilter);

contactRequest.onsuccess = function() {

var contact = this.result[0];

var telNumber = [];

for (var i = 0; i < contact.tel.length; i++) {

telNumber.push(contact.tel[i].value);

}

var phoneNumber= String(telNumber).replace(/(\s|-|\.|\(|\))/g, '');

//对象用于异步和安全地访问和修改所有设置

var lock = navigator.mozSettings.createLock();

//用于获取gaia/build/config目录下的common-settings.json文件中的相关字串的内容，设置的就使用lock.set(key,value)

var req = lock.get('default.iceMessage');

req.onsuccess = function () {

var message = req.result['default.iceMessage'];

navigator.mozMobileMessage.send(phoneNumber, message);//使用短信的发送api发送信息

}

req.onerror = function () {

dump('wang-tag An error occured: ' + req.error);

}

};

});

});

至此，我们就通过使用dataStore来进行了一次数据访问

3. 旋转项目的横竖屏：

修改gecko/widget/gonk/nsScreenManagerGonk.cpp中的ro.sf.hwrotation 的值为90或者270

4. 修改app的横竖屏显示：

修改清单文件(manifest.webapp)：

"orientation": "landscape-primary",//横屏

"orientation": "portrait-primary",//竖屏

5. 查找页面的某个dom元素的方法:

document.getElementByClassName();

document.getElementById();

document.querySelector();

document.querySelectorAll();

document.getElementsByName();

document.getElementsByTagName();

1. 在能识别到端口的情况下进行强制刷机:adb reboot edl

7. KaiOS系统的一些app详情：

ftu:刷机后的引导界面

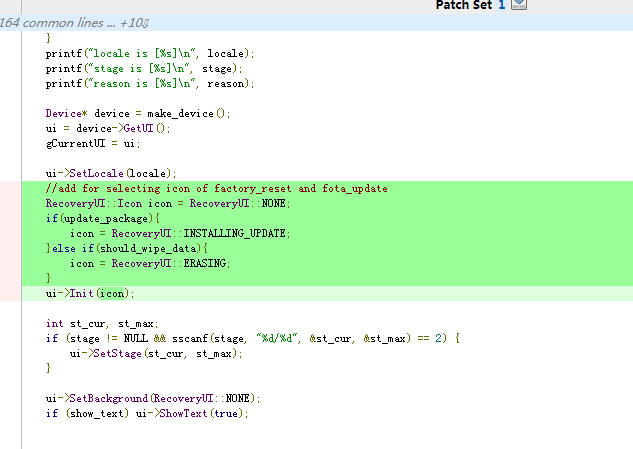
fota:在settings下的software update点击的时候用于检测系统

8. 制作recovery动画并替换

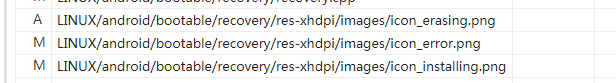
**(1)安装环境，根据<https://blog.csdn.net/cursem/article/details/53132797>**

**(2)向客户请求需要尺寸的一套FOTA动画，如320X240图片，然后按照（1）提供的方法进行动画合成**

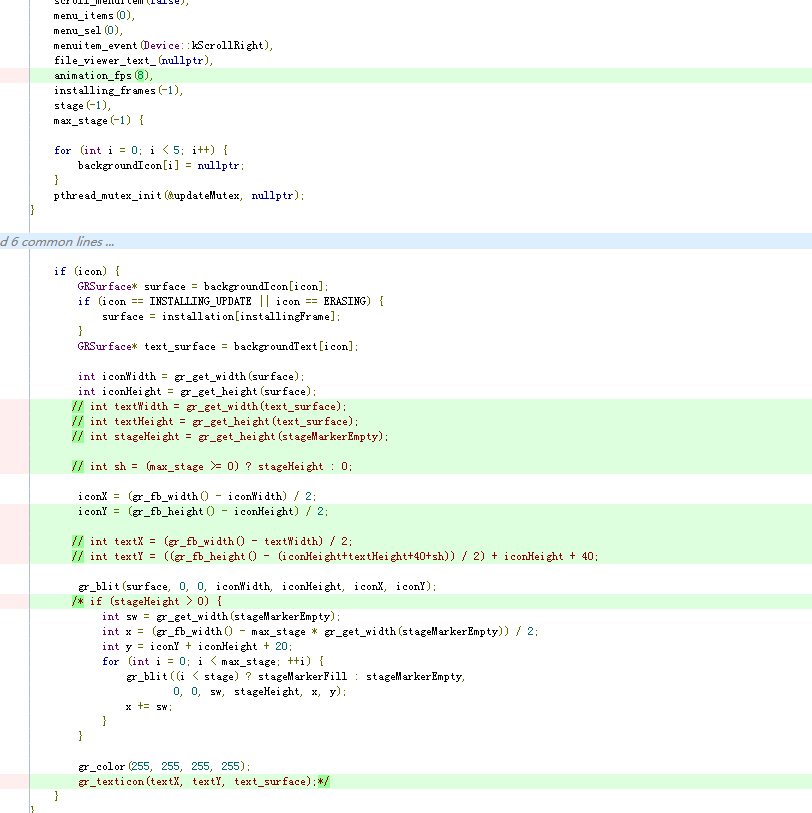
**(3)** 修改文件LINUX/android/bootable/recovery/recovery.cpp

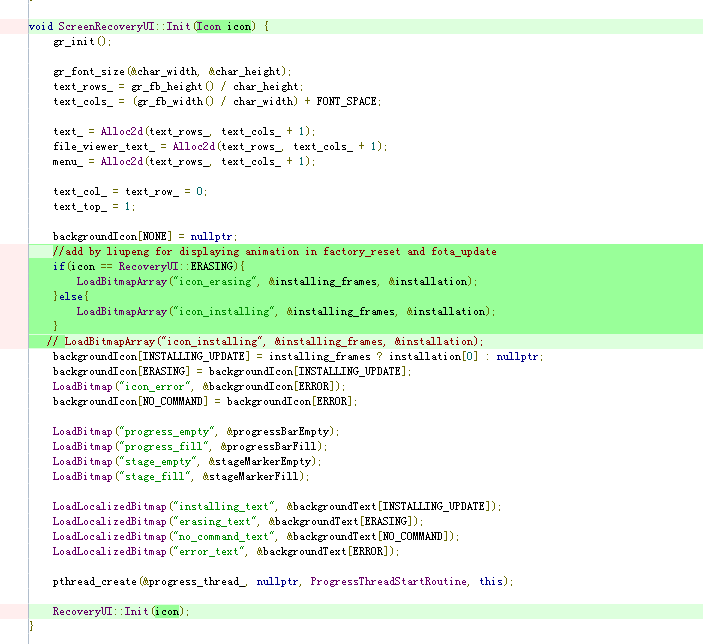
****

**替换图片文件**

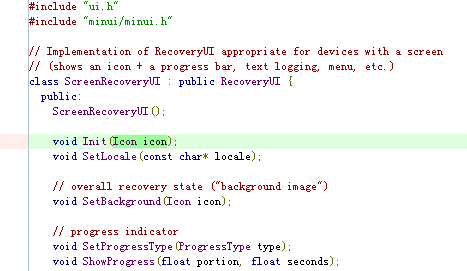
****

**修改文件**LINUX/android/bootable/recovery/screen\_ui.cpp

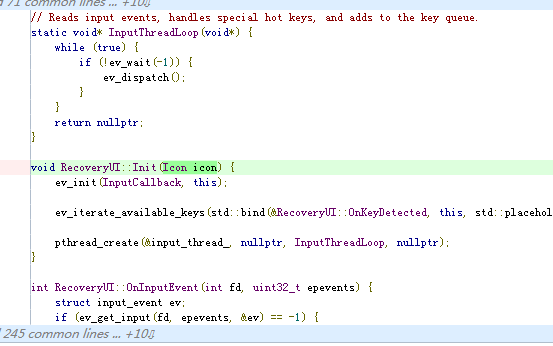




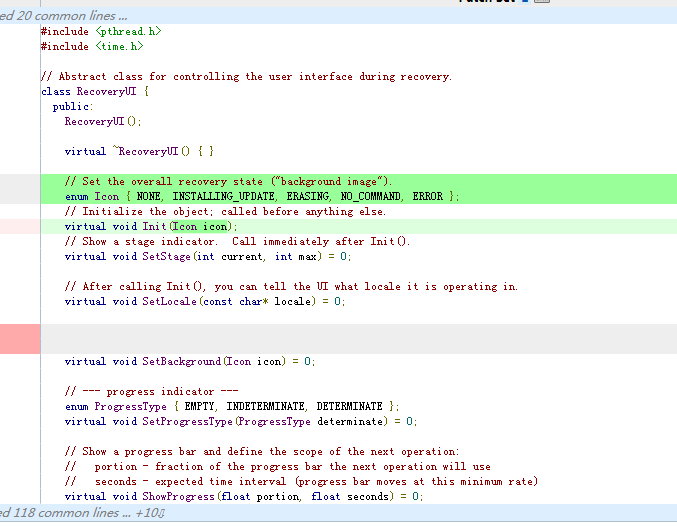
修改LINUX/android/bootable/recovery/screen\_ui.h



修改LINUX/android/bootable/recovery/ui.cpp



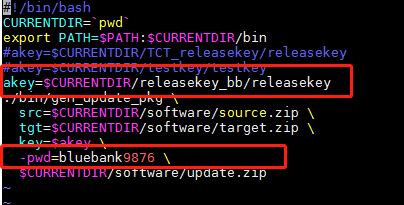
修改LINUX/android/bootable/recovery/ui.h



1. KaiOS fota制作

FOTA环境搭建:

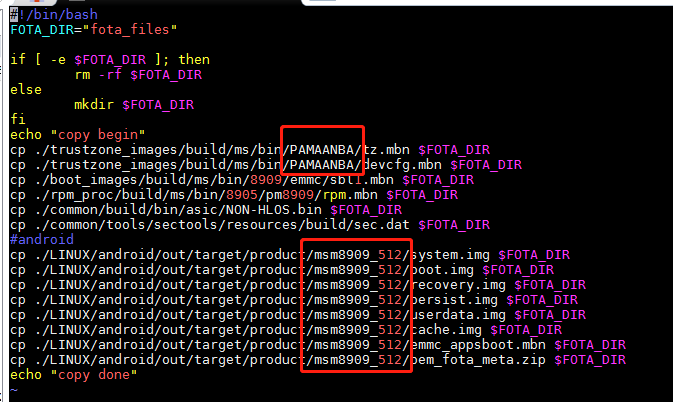
1. 获取KAIOS的FOTA工具:如FOTA\_1.5.6
2. 由于项目采用block的方式，所以”cd FOTA\_1.5.6/fota\_tools/block”,执行./Generat\_tools.sh,可在上一层目录生成KaiOS-QC-Block-V1.5.6目录
3. 切换到KaiOS-QC-Block-V1.5.6目录，在文件gen\_update\_pkg\_start.sh中修改项目的release\_key和pwd,(releasekey\_bb的文件夹需要手动添加到当前目录)

****

1. 执行make,再执行make links,FOTA环境才算搭建成功

Fota 制作：

1. 保证制作FOTA的工具及其环境配置正确
2. 保证FOTA工具中的release\_key与版本里的一致
3. 编译基版本和目标版本 （他们之间的差异可以只改一个版本号测试）
4. 在LINUX环境下，执行2个版本下的get\_all\_fota\_files.sh,生成文件夹fota\_files,(其中圈出的地方可能因项目不同做相应目录名的修改和增删)



1. 将基础版本生成的fota\_files下的文件全部放入FOTA工具下的software/src/下，将目标版本生成的fota\_files下的文件全部放入FOTA工具下的software/tgt/下
2. 回到FOTA主目录，执行prepare\_pkg\_start.sh，再执行gen\_update\_pkg\_start.sh
3. 等待执行完成，即可在software/下生成update.zip文件，此包为FOTA升级包

执行adb sideload update.zip进行测试，查看版本号是否从基版本变为目标版本

1. 开机动画制作并替换

A. Splash image/第一帧图像/开机logo(有很多叫法，但比较官方的一般叫bootloader logo或者LK display logo)

替换LK display (bootloader) logo有两种方式

a. 利用平台自带的logo\_gen.py生成splash.img镜像，可以使用fastboot重新刷splash.img分区

b. 利用三方软件将.png转为bootable/bootloader/lk/platform/msm\_shared/include/splash.h头文件中的buffer并替换

方法b只适用于分辨率较低的图片（经过测试发现貌似只支持分辨率不超过320\*200的图片，但不是官方数据），方法a适用于各种分辨率图片（.png）

看源码bootable/bootloader/lk/dev/fbcon/fbcon.c中的逻辑

系统会先通过fetch\_image\_from\_partition()获取.img格式镜像，如失败

会去splash.h头文件获取数组，如果数组无效或者获取失败，会显示default fbimg

先说下生成splash.img的步骤

1.需要符合屏幕分辨率的.png图像

2.将图像放到目录device/qcom/common/display/logo，并确认该目录下有logo\_gen.py脚本

3. python ./logo\_gen.py logo.png

如果遇到以下错误：

IMG_256

的话请安装pip install Pillow 组件

4.在partition.xml添加splash分区

<partition label="splash" size\_in\_kb="11264" type="20117f86-E985-4357-B9EE-374BC1D8487D" bootable="false" readonly="false" filename="splash.img" sparse="true"/>

5.在contents.xml添加splash.img

<download\_file minimized="true" fastboot="true">

        <file\_name>splash.img</file\_name>

        <file\_path>LINUX/android/out/target/product/msm8937\_32go/</file\_path>

</download\_file>

Fastboot一定要为true否则无法fastboot烧录

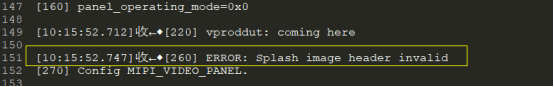
6.在编译脚本中把splash.img copy到代码out目录

PRODUCT\_COPY\_FILES += device/qcom/common/display/logo/splash.img:splash.img

7.在打包脚本中把splash.img copy到pub out目录

cp -f LA.UM.7.6.2/LINUX/android/device/qcom/common/display/logo/splash.img    $RELEASE\_DIR/$TAR\_VER/LA.UM.7.6.2/LINUX/android/out/target/product/${TARGET\_PRODUCT}/

1. 如果按上面方法操作了还不生效，请打串口log看下是否有splash image相关报错，我在改的过程中碰到了header invalid的错误，最后发现是logo\_gen.py中header起始/终止地址有误。修改文件读取指针的位置后就ok了，一般有splash image报错都是由于生成脚本本身有问题



下面说下方法b,

1. 找到bootable/bootloader/lk/platform/msm\_shared/include/splash.h，会发现里面有三个buffer: imageBuffer[], imageBuffer\_rgb888[]和image\_batt888[]。生效的逻辑也在fbcon.c中：

#if DISPLAY\_TYPE\_MIPI

        fbimg->image = (unsigned char \*)imageBuffer\_rgb888;

#else

        fbimg->image = (unsigned char \*)imageBuffer;

#endif

2. 用三方工具将.png转为buffer, 我用的是Image2Lcd，转出来的buffer大小如果大于源码中buffer的大小，会有异常，可以通过降低输出图像的色彩位数减少buffer size(不影响效果的前提下)。如果需要修改buffer size上限，方法如下

a. 找到项目.dtsi, arch/arm/boot/dts/qcom/msmxxxx.dtsi

b. 在64位的版本中buffer地址为4M,

reserved-memory {

…

cont\_splash\_mem: splash\_region@83000000 {

   reg = <0x0 0x90000000 0x0 0x1400000>;

  };

}

debug可以通过kernal log看是否有类似报错：

[ 0.000000] memblock\_reserve: [0x0000008e000000-0x00000090800000] dma\_contiguous\_reserve\_area+0xdc/0x1e4

[    0.000000] cma: CMA: reserved 40 MiB at 0x000000008e000000 for cont\_splash\_mem

[    0.000000]  reserved[0x3] [0x0000008e000000-0x000000907fffff], 0x2800000 bytes

Kernal log添加方法如下：

a. 在kernal 配置文件中扩大kernal log buffer的大小:

// msm-perf\_defconfig

CONFIG\_LOG\_BUF\_SHIFT=18

b. 打开memblock debug

//arch/arm/boot/dts/qcom/msmxxxx.dtsi

chosen {

// bootargs = "boot\_cpus=0,1,2,3,4 sched\_enable\_hmp=1";

bootargs = "boot\_cpus=0,1,2,3,4 sched\_enable\_hmp=1 memblock=debug";  };

3. 如果想删除第一帧的话，找到 arch/arm/boot/dts/qcom/msmxxxx-mdss.dtsi ,注掉以下标黄行代码

mdss\_fb0: qcom,mdss\_fb\_primary {

   cell-index = <0>;

   compatible = "qcom,mdss-fb";

   qcom,mdss-fb-splash-logo-enabled;

   qcom,cont-splash-memory {

    linux,contiguous-region = <&cont\_splash\_mem>;

   };

  };

B. Boot animation

这个比较简单，甚至都没有什么需要debug的点

1. bootanimation.zip由两部分组成（配置文件和图片文件夹）

a. desc.txt - 动画配置文件，帧数循环次数，文件名称, 需注意文件格式需要改为ANSI

480 960 1 - “480” 为动画播放宽度，”960”为动画播放高度，不确定的话可以用adb shell wm size或者adb shell dumpsys window displays看一下，”1”为fps值

p 1 0 part0 - “p”是一个标志位, 说明当service.bootanim.exit属性（想了解这个属性的使用场景的，可以去精读一下SurfaceFlinger进程的代码）为true时，不用等待所有图片播放完毕就跳出播放，第一位为”c”时，说明当service.bootanim.exit为true时，必须要等播放完成后才会跳出这部分。”1” 为循环次数，如果这个值为”0”的话，说明此部分动画会一直循环，直到service.bootanim.exit为true。”0”为播放完这部分的暂停，为0代表无暂停。”part0”为存放动画图片的文件夹名

p 0 0 part1

b. part0, part1个文件夹 - 也可以手动根据需求添加part3,或者改文件夹名字

2. Zip/unzip bootaimation.zip的所有动作最好都在Linux环境下完成

特别需要注意的是，压缩bootanimation.zip的时候压缩模式必须为存储模式：

zip -r -0 bootanimation.zip part0 part1 desc.txt

压缩完成后请用：

Unzip -t bootanimation检查压缩包是否完成

3. 本地验证可以直接把存储模式的bootanimation.zip push到device的/system/media目录下验证

4. 编译的话需要在.mk中将bootanimation.zip拷贝到system/media目录：

PRODUCT\_COPY\_FILES += device/qcom/msm8937\_64/bootanimation.zip:system/media/bootanimation.zip

1. 获取文件节点（系统属性）

通过navigator.engmodeExtension去获取，对应的api方法在gecko/proprietary/engmodeEx/engmodeExtension.js文件中

比如getSysInfo，fileReadLE，filewriteLE

1. 相关编译指令

01 编译kernel：make kernel -j32

02 单编app：在gaia目录下,执行命令：make APP=system

03 将app刷入手机：

adb root

adb remount

adb push gaia\profile\webapps\system.gaiamobile.org\application.zip system/b2g/webapps/system.gaiamobile.org

adb reboot

04 adb查看输入设备（按键）使用的哪个kl文件

dumpsys input

05 adb查看按键输入对应设备名称

cat /proc/bus/input/devices

06 adb查看输入设备相关信息（例如按键对应的kl文件，仅限android）

adb shell dumpsys input

07 获取按键的event事件

adb shell getevent

08 打日志

adb logcat > log.txt

09 离线打日志

nohup adb logcat > log.txt

10 编译bootimage（生成在out目录下）

在android目录下：make -j32 bootimage

11 刷bootimage

adb reboot bootloader

fastboot flash boot boot.img

pause

fastboot reboot