sugar 定制化

V0.4

2013-3-30

Revision History

Version	Date	Section	Changes compared to previous issue
V0.6	2013-03-30		创建文档

目录

- 编写目的.....6
- 定义.....6
- 参考资料.....6
- 注意事项.....6

1 添加定制的方案板编译打包环境.....7

1.1 软件层定制.....7
1.1.1 sugar-ref001.mk.....7
1.1.2 AndroidProducts.mk.....8
1.1.3 BoardConfig.mk.....8
1.1.4 init.sun7i.rc.....8
1.1.5 initlogo.rle.....9
1.1.6 recovery.fstab.....10
1.1.7 Sun7i-ir.kl.....10
1.1.8 vendorsetup.sh.....10
1.1.9 vold.fstab.....10
1.1.10 package.sh.....10
1.2 硬件层定制.....10

2 配置自己的遥控器.....12

2.1 修改红外遥控的地址码：.....12
2.2 修改按键映射.....12
2.3 修改“软鼠标模式”下使用的键值.....12

3 自产板子 USB 设备挂载配置.....13

3.1 硬件层.....13
3.2 Android 层.....13

4 添加编译官方 APK.....15

5 NandFlash 分区配置说明.....16

5.1 NandFlash 分区配置文件.....16
5.2 常用的针对 NandFlash 分区的操作.....16
5.2.1 调整 System 分区大小,以适应 system.img 文件大小.....16
5.2.2 调整 Data 分区容量,以装更多的软件和游戏.....16
5.2.3 挂载内部盘.....16

6 修改显示输出设置 18

6.1 修改遥控器快捷操作中的显示列表 18
6.2 修改 Settings 里面具体的显示列表 18
6.3 修改显示策略 18

7 如何添加新的 product 20

7.1 创建 procuct 目录 20
7.2 创建此 product 对应的配置文件 20
7.3 验证新 product 的正确性 20
7.4 创建此 product 对应的 git 仓库 20
7.5 将此仓库添加至 repo 中 21
7.6 将 tools 下针对此 product 新添加的配置文件提交至服务器 21

8 如何控制 GPIO 22

8.1 定义需要控制的 GPIO 22
8.2 控制 GPIO 的接口 22
8.2.1 java 层的接口 22
8.2.2 c++层的接口 22

9 Private 分区的配置与读写操作 24

9.1 配置 Private 分区 24
9.1.1 硬件层配置 24
9.1.2 软件层配置 24
9.2 Private 分区读写 25

10 音频动态管理 27

10.1 概要 27
10.2 音频管理策略 27
10.3 上层提供的接口如下 28
10.4 音频相关接口使用例子 29
10.4.1 监听 USB 音频设备热插拔 29
10.4.2 音频输出/输入模式 30

11 支持 spdif 33

11.1 使能 spdif 模块 33
11.2 安装 spdif 驱动 33
11.3 切换声道至 spdif 33

12 配置出厂时的默认 launcher.....34

12.1 配置默认 launcher.....34
12.2 保留原生做法.....34

13 关机.....35

13.1 关机时不再出现对话框.....35
13.2 短按 power 键关机.....35

14 替换鼠标图标.....36

15 配置 boot 阶段初始化的 gpio.....37

16 全屏显示.....38

16.1 总是隐藏状态栏.....38
16.2 使能 windowFullscreen.....38

17 隐藏软键盘.....39

18 多屏互动设备名称.....40

19 配置流媒体缓冲策略.....41

20 SystemMix 可扩展接口说明.....42

20.1 提供该接口的目的.....42
20.2 原理.....42
20.3 接口使用.....42

引言

◆ 编写目的

本文档介绍 sugar 方案中常见的定制

◆ 定义

homlet：面向客厅设备的产品线的名称。
sugar：基于全志 A20 芯片的家庭娱乐产品 sdk 标号.

◆ 参考资料

◆ 注意事项

1 添加定制的方案板编译打包环境

1.1软件层定制

在 android4.2\device\softwinner\目录下添加自己的方案目录 xxx（如 sugar-ref001）,软件上的配置全部放在这目录下.每个方案目录下的文件名字和结构基本相同。下面以 sugar-ref001 为例说明。:

1.1.1 sugar-ref001.mk

文件内部定义了需要定制的信息,如需要预装的 apk,需要编译的 apk 源码,产品名字等等.应该把这个文件名改为自己的方案名,如:sugar-ref001.mk

文件中有几个比较重要变量的值可能要改的,如下:

1.1.1.1 PRODUCT_PACKAGES(用于添加需要编译的产品 APK 或库)

这里定义了需要添加的产品包或库,添加上去后,会编译该源码,打包之后固件里就会有该 apk 或库文件,需要添加生成的 apk 或.so 文件时,应该把它的.mk 文件中定义的 PACKAGENAME 的值加上去， TVD 盒子使用的应用程序包有针对于 Homlet 的 TvdSettings,TvdLauncher,TvdVideo,TvdFileManager.

1.1.1.2 PRODUCT_COPY_FILES

编译时把该环境变量中定的东西拷贝到指定的路径,如在 sugar-ref001 方案目录下定制了一个用于红外遥控的按键映射文件,想在编译时把它拷贝到 system/usr/keylayout 目录,则可以这么写:

```
PRODUCT_COPY_FILES += \

    device/softwinner/sugar-ref001/sun7i-ir.kl:system/usr/keylayout/sun7i-ir.
kl \
```

1.1.1.3 PRODUCT_PROPERTY_OVERRIDES(定义 Property 环境参数)

1.1.1.4 该文件中所有和方案名称有关的文字都需要改成自己方案的

1.1.2 AndroidProducts.mk

这里只有一句话

```
PRODUCT_MAKEFILES := \

    $(LOCAL_DIR)/sugar-ref001.mk
```

其中 sugar-ref001.mk 就是上个小节中说的文件,所有也要改成自己方案的该文件名.

1.1.3 BoardConfig.mk

这里一般定义了 Wifi 和其他一些的配置变量,.Wifi 的配置变量值可参考该文档的“Wifi 配置”小节。

同理该文件下的和方案名有关的文字都要改成自己方案的.

1.1.4 init.sun7i.rc

该脚本会在 android 系统启动时被调用,功能是做与方案有关的软件上的初始化.一般需要注意的是：

1.如果改动过 nandflash 分区的划分，需要根据自定制的 nand 分区来挂载上对应的目录。nand 的分区在 lichee\tools\pack\chips\sun7i\configs\android\sugar-ref001 下的 sys_partition.fex 文件中有定义。相应的脚本为：

```
# try to mount /data

    wait /dev/block/nande

    mount ext4 /dev/block/nande /data wait noatime nosuid nodev barrier=0

    setupfs /dev/block/nande

    umount /data

    exec /system/bin/logwrapper /system/bin/e2fsck -y /dev/block/nande

    exec        /system/bin/busybox        mount        -t        ext4        -o
noatime,nosuid,nodev,barrier=0,nodiratime,noauto_da_alloc    /dev/block/nande
```

```
/data


# try to mount /cache
    mount ext4 /dev/block/nandh /cache wait noatime nosuid nodev barrier=0
     setupfs /dev/block/nandh
     umount /cache
     exec /system/bin/logwrapper /system/bin/e2fsck -y /dev/block/nandh
    mount ext4 /dev/block/nandh /cache noatime nosuid nodev barrier=0


    format_userdata /dev/block/nandk mars


# try to mount /private
    export PRIVATE_STORAGE /mnt/private
    format_userdata /dev/block/nandi PRIVATE
    mkdir /mnt/private 0000 system system
    mount            vfat            /dev/block/nandi            /mnt/private
gid=1019,uid=1019,fmask=0007,dmask=0007
```

2.如果需要关闭网络 adb 调试,只使用 USB adb 调试，则把如下脚本中的几行注释掉

```
on boot
setprop service.adb.tcp.port 5555
stop adbd
start adbd
```

1.1.5 initlogo.rle

自定义开机 log 图片.目前默认的是 720p 的图片，图片采用工具 LogoGen 生成。

1.1.6 recovery.fstab

该文件中有定义 recovery 阶段各个分区或设备对应的挂载点.一般默认使用公版的分区形式.

1.1.7 Sun7i-ir.kl

针对遥控器按键的映射值配置.自定义遥控器按键的配置可参考”配置自己的遥控器”小节

1.1.8 vendorsetup.sh

只有一句话:

```
add_lunch_combo sugar_ref001-user
```

修改”sugar_ref001-user”为自己方案的名称”XXX-user”,然后在编译时”lunch”时就会看到该方案.这里之所以改成 user 模式，是为了增强系统的稳定性。

1.1.9 vold.fstab

这里是关于存储设备(如 SD 卡,nandflash,usb,sata)的定义,在刚创建定制的方案目录时可以先不用理,等到需要定制自产板子的存储设备的挂载时再修改

1.1.10 package.sh

打包时会被调用的脚本,里面有一句话,例如

```
./pack -c sun7i -p android -b sugar-ref001
```

把 sugar-ref001 改为自己方案的名称,如 homlet,那么打包时就会调用 1.2 节”硬件参数配置文件”中讲的两个硬件参数配置文件.

1.2硬件层定制

拿 sugar-ref001 为例,需要添加的文件有:

1. 硬件参数配置文件

在 lichee\tools\pack\chips\sun7i\configs\android\sugar-ref001 中定义了各个方案的硬件参数配置,每个方案都由两个文件:sys_config.fex 和 sys_partition.fex 来定义.建议在 sugar-ref001 基础上修改,对于每个模块中譬如 XXX_used 这个参数模块是表示该模块是否用到,当设置为 0(不可用)时其他参数可以不用配置,如模块[ps2_0_para]中的 ps2_used 设置为 0 时,ps2_scl 和 ps2_sda 可以不用配置.

同时需要修改 1.1.12 小节中说的 package.sh 文件,该文件与这两个硬件参数配置文件一一对应,

2 配置自己的遥控器

2.1修改红外遥控的地址码：

在文件 lichee\linux-3.0\drivers\input\keyboard\ir-keymap.h 中，根据自己的遥控器的地址码修改如下代码

#define IR_ADDR_CODE (0x7f80)

譬如说，如果地址码为 0x7f80，则修改成

#define IR_ADDR_CODE (0x7f80)

如果发现无效，则将两个字节的值交换一下位置，修改成

#define IR_ADDR_CODE (0x807f)

2.2修改按键映射

在文件 android4.2\device\softwinner\sugar-ref001\sun7i-ir.kl 中，重新建立按键扫描码与系统中定义的按键名称的映射关系

按键扫描码可以通过在串口中输入 getevent,然后点击按键时看打印出来的键值来确定,

注意：扫描码不能重复，否则此文件将失效

2.3修改“软鼠标模式”下使用的键值

(待续)

3 自产板子 USB 设备挂载配置

3.1硬件层

修改lichee\tools\pack\chips\sun7i\configs\android\sugar-ref001目录下的sys_config.fex文件的[usbc0][usbc1][usbc2]三组模块的参数,USB控制标志配置项几个名称的定义为:

配置项	配置项含义
usb_used =xx	USB 使能标志(xx=1 or 0)。置 1，表示系统中 USB 模块可用，置 0，则表示系统 USB 禁用。此标志只对具体的 USB 控制器模块有效。
usb_port_type =xx	USB 端口的使用情况。(xx=0/1/2) 0：device only 1：host only 2：OTG
usb_detect_type=xx	USB 端口的检查方式。 0：无检查方式 1：vbus/id 检查
usb_id_gpio=xx	USB ID pin 脚配置。具体请参考 gpio 配置说明。《配置与 GPIO 管理.doc》
usb_det_vbus_gpio＝xx	USB DET_VBUS pin 脚配置。
usb_drv_vbus_gpio=xx	USB DRY_VBUS pin 脚配置。
usb_host_init_state=xx	host only 模式下，Host 端口初始化状态。 0：初始化后 USB 不工作 1：初始化后 USB 工作

3.2Android 层

a.修改 android4.2/device/softwinner/<方案>/vold.fstab 文件.

该文件定义了每个存储设备的挂载点,其中每一行代表一个存储设备,它的格式为:

dev_mount <设备标签> <挂载点> <分区个数(一般设为 auto)> <存储设备在文件系统上的路径>, 中间用 tab 制符号隔开

如要定义 SATA 设备,它挂载到/mnt/sata 目录下,设备路径为/devices/platform/sw_ahci.0,那就添加这么一句

dev_mount sata /mnt/sata auto /devices/platform/sw_ahci.0

假设我现在要在 vold.fstab 中添加一个 USB1 的设备的定义,就先打开板子的打印,输入 logcat,然后在该 USB1 的插口插入一个 U 盘,这时会看到一个类似于如下的打印:

I/USB3G (90): event { 'add', '/devices/platform/sw-ehci.1/usb1/1-1/1-1.3', 'usb', '', 189, 3 }

I/USB3G (90): path : '/sys/devices/platform/sw-ehci.1/usb1/1-1/1-1.3'

I/USB3G (90): VID :size 5,vid_path '/sys/devices/platform/sw-ehci.1/usb1/1-1/1-1.3/idVendor',VID '17ef

其中第一行的"/devices/platform/sw-ehci.1/usb1/1-1/1-1.3"就是该接口设备在文件系统上的路径,这样就可以在 vold.fstab 里添加该设备的定义,如下:

dev_mount usbhost1 /mnt/usbhost1 auto /devices/platform/sw-ehci.1/usb1/1-1/1-1.3

这样在 USB1 设备插入时,会把它挂载到/mnt/usbhost1 路径下.

b.修改 android4.2\device\softwinner\<方案>\overlay\frameworks\base\core\res\res\xml 中的 storage_list.xml 文件,在这里添加自己的在 vold.fstab 中定义的 USB 设备,可以仿照其他的来写,其中每个参数的含义在该文件头有说明.该文件定义了上层应用读取的设备列表.

c.修改 android4.2\device\softwinner\<方案>\目录下的 init.sun7i.rc 文件,在

"on early-init"的地方添加创建这些挂载目录的文件,如:

mkdir /mnt/usbhost0 0000 system system

mkdir /mnt/usbhost1 0000 system system

4 添加编译官方 APK

官方应用程序源码在 android/package/apps 目录下,如果需要在生成固件时包含某个 apk(如音乐播放器,超清播放器),需要把其包名添加到需要编译的列表中.比如要添加超清播放器,则在 android4.2\device\softwinner\wing-common\ProductCommon.mk 文件内，给变量 PRODUCT_PACKAGES 添加一个新的值“Gallery3D” （注意，可能需要添加“ \”做分隔）。Gallery3D 就是该应用的包名(PACKAGE_NAME).包名可以在每个应用程序源码的 Android.mk 文件中找到” LOCAL_PACKAGE_NAME”的值.

5 NandFlash 分区配置说明

5.1 NandFlash 分区配置文件

lichee\tools\pack\chips\sun7i\configs\android\<方案>\目录下的 sys_partition.fex 文件定义了 nandflash 的分区划分,每个分区有不同作用,如分区 bootloader 存放 boot 启动代码,boot 存放 linux 编译的内核代码,system 存放 android 编译时生成的那个 system.img。data 存放安装的 apk 数据文件，如游戏，应用软件资源等等。recovery 存放 recovery 模式启动代码.

假设该文件中定义了 partition0--partition7 这 8 个分区,那它在 android 层的名字就分别叫做 nanda,nandb,nandc,nandd,nande,...,nandh, 然后剩下的空间就为 nandi,做内部存储用,也就是文件管理器中显示的”本地磁盘”.

5.2 常用的针对 NandFlash 分区的操作

5.2.1 调整 System 分区大小,以适应 system.img 文件大小

由于不同方案软件上的定制都集中在 android 层,编出来的 system.img 文件有大有小,因此要注意 nandFlash 划分的 system 分区大小要比 system.img 文件略大.修改 sys_config.fex 中的 system 分区的 size_lo 参数来适应 system.img 大小.

system 分区划分容量比 system.img 容量小时,常见的出错现象是:烧录固件时出错.

5.2.2 调整 Data 分区容量,以装更多的软件和游戏

Data 分区存放第三方下载的软件和游戏,如果想装更多的游戏和软件,应该修改该分区容量.

修改 sys_config.fex 中的 data 分区中的 size_lo 参数,目前的单位为扇区，即 512M

5.2.3 挂载内部盘

这个针对于解决定制后发现内部 nandflash 盘挂不上的问题.由 4.1 里面说的,如果在该文

件中定义了 n 个分区,那剩下的第 n+1 个就是作为本地盘挂载到板子上给外部使用的.

假设目前定义了 8 个分区,分别是从 nanda~nandh,那剩下的 nandi 就是本地盘.需要修改几个文件来让它挂载上去:

1. 修改 android4.2\device\softwinner\<方案名>\目录下的 recovery.fstab 文件.recovery.fstab 文件定义了每个分区的挂载路径.它会在系统启动,进入 recovery 模式时被读取.这里面应该把那几个 nanda~nandi 和他们的挂载目录对应起来

2. 修改 android4.2\device\softwinner\<方案名>\目录下的 vold.fstab 文件.把 dev_mount sdcard /mnt/sdcard auto /devices/virtual/block/nandi 这句话的 /devices/virtual/block/nandi 改为自己定制的本地盘的名字.该文件是在管理 android 存储设备挂载的 vold 启动后被读取.

3. 修改 android4.2\device\softwinner\<方案名>\目录下的 init.sun6i.rc 文件,把 format_userdata /dev/block/nandk mars 中的 nandk 改为 nandi

6 修改显示输出设置

6.1修改遥控器快捷操作中的显示列表

在文件 DispList.java（路径 android4.2\frameworks\base\core\java\android\view）中，有一个列表变量 mShortCutArray，根据自己的需要来增、删其中表项。

6.2修改 Settings 里面具体的显示列表

在文件 DispList.java（路径 android4.2\frameworks\base\core\java\android\view）中，有一个列表变量 mItemArray，根据自己的需要来增、删其中的表项。

6.3修改显示策略

当前的策略是：

1）在 boot 阶段，系统检测设备当前各个显示端口上电缆的连接情况，根据连接情况决定输出模式。

相关代码在文件 android4.2\system\core\init\init_disp.c 中的函数 init_initdisplay()内。

2）在系统进入 launcher 之前，系统将根据当前输出模式、当前电缆连接情况和上次保存的输出模式这 3 个因素，重新决定输出模式。

相关代码在文件 android4.2\frameworks\base\services\java\com\android\server\SystemServer.java 中函数 run()内的最后 100 多行。

3) 系统进入 launcher 的时候,会在系统 APP—SystemUi 中去注册对显示设备热插拔的检测,在接收到某种显示设备插拔信息时,会重新切换显示输出模式。

相关代码在文件 android4.2\frameworks\base\packages\SystemUI\src\com\android\systemui\statusbar\policy\DisplayController.java 中。

4) 对于遥控器操作,Homlet 定义了一个叫”Tv System”的快捷键操作,点击遥控器上该快捷键时,会按顺序切换当前的显示模式。

相关代码在文件

android\frameworks\base\policy\src\com\android\internal\policy\impl\PhoneWindowManager.java 中 _interceptKeyBeforeDispatching()方法的 keyCode == KeyEvent.KEYCODE_TV_SYSTEM 时的处理中和子类 MyDispList 的 onHide()方法的实现中。

5）recovery 阶段也牵扯到显示问题，系统也会检测设备当前各个显示端口上电缆的连接情况，根据连接情况决定输出模式。

相关代码在文件 android4.2\bootable\recovery\minui\graphics.c 中的函数 gr_init()内

7 如何添加新的 product

此处，我们将以添加一个新的 product（命名为 sugar-ref002）为例来说明。

7.1创建 procuct 目录

1）clone 一个新的 sugar-ref001 仓库到本地。之所以需要一个新的仓库，是因为在编译过程中此目录下面会生成一些临时文件。

2）将此 sugar-ref001 文件夹整体复制，并重新命名为“sugar-ref002”。

3）删除此目录下的“.git”文件夹

4）利用某些工具（比如 UltraEdit），将此目录下的所有文件内的“sugar-ref001”替换为“sugar-ref002”。

5）将“sugar-ref001.mk”重命名为“sugar-ref002.mk”

7.2创建此 product 对应的配置文件

1）在目录 lichee\tools\pack\chips\sun7i\configs\android\下，复制文件夹“sugar-ref001”。

2）将复制后的文件夹重命名为“sugar-ref002”。

3）根据 product 的具体情况，修改“sugar-ref002”目录下的 sys_config.fex 和 sys_partition.fex 文件内的配置信息

7.3验证新 product 的正确性

在验证之前，务必将 mars-demo 目录整体备份一下，因为验证时会在此目录下产生一些临时文件。

7.4创建此 product 对应的 git 仓库

如果验证无误，将备份的 sugar-ref002 目录恢复出来。

1）本地创建 git 仓库。假设分支名为“sugar-dev”，在 sugar-ref002 目录下

```
$git init
```

```
$git add .

$git commit -m "create a new product ‘sugar-ref002'."

$git branch sugar-dev        ;// sugar-dev 是主分支名称，请根据自己的实际情况决定

$cd ..

$git clone --bare sugar-ref002 sugar-ref002.git
```

2）将生成的 sugar-ref002.git 放至服务器上

3）删除本地的 sugar-ref002 和 sugar-ref002.git 文件夹

7.5将此仓库添加至 repo 中

1）在文件 android4.2\.repo\manifests\sugar-android.xml 内，根据创建的仓库名称添加如下信息：

```
<project path="device/softwinner/ sugar-ref002" name="device/softwinner/ sugar-ref002" />
```

2)将修改提交至服务器

7.6将 tools 下针对此 product 新添加的配置文件提交至服务器

8 如何控制 GPIO

8.1定义需要控制的 GPIO

在 lichee\tools\pack\chips\sun7i\configs\android\sugar-ref001\sys_config.fex 文件中，添加类似如下的配置信息：

```
;gpio configuration
;------------------------------------------------------------
 [gpio_para]
gpio_used               = 1
gpio_pin_1              = port:PH10<1><default><default><0>
gpio_pin_2              = port:PH20<1><default><default><1>
gpio_pin_3              = port:PB03<0><default>
```

在这个范例中，变量 gpio_used 置为“1”表示此配置将起作用。其他的就是各个 GPIO 的配置信息。这些 GPIO 的编码必须从“1”开始依次递增。

8.2控制 GPIO 的接口

8.2.1 java 层的接口

java 控制 GPIO 的接口定义在文件 Gpio.java 中，其路径为 android4.2\frameworks\base\swextend\gpio\java\Gpio.java。

8.2.2 c++层的接口

系统启动后，在/sys/class/gpio_sw/目录下看到各个 GPIO 节点的子目录

如图：

```
# pwd
/sys/class/gpio_sw
# ls
PH1   PH11  PH2
```

```
# pwd
/sys/class/gpio_sw/PH1
# ls
data       drv_level  power      subsystem  uevent
device     mul_sel    pull       trigger
```

在 GPIO 节点的子目录中可以看到 data ,drv_level, mul_sel,pull 等我们可以对 data ,drv_level, mul_sel,pull 这 4 个文件进行读写操作来控制 GPIO 口。

mul_sel： 值代表端口目前的功能

pull： 值代表端口目前的电阻状态

drv_level： 值代表端口目前的驱动等级

data： 值代表端口目前的电平状态

在 C 语言中可以用 read 和 write 函数直接操作这 4 个文件。具体的范例可参考文件 android4.2\frameworks\base\swextend\gpio\libgpio\GpioService.cpp 中的代码。

9 Private 分区的配置与读写操作

以 sugar-ref001 的 Private 分区为例

9.1配置 Private 分区

9.1.1 硬件层配置

修改 lichee\tools\pack\chips\sun7i\configs\android\sugar-ref001 目录下的分区信息,如 sugar-ref001 的 nandi 是作为 private 分区的.

```
;------------------------------>nandi, private partition
[partition]
    name        = private
    size        = 16384
    keydata      = 1
```

9.1.2 软件层配置

修改 android4.2\device\softwinner\sugar-ref001 下的 init.sun7i.rc 文件中的如下脚本:

```
# try to mount /private
    export PRIVATE_STORAGE /mnt/private          //这里自己定制private分区的挂载路径
    format_userdata /dev/block/nandi PRIVATE     //该行是用来启动时如果还没格式化该分区则格式化一下,nandi是在硬件层配置时得到的private分区的名,如mele的是nandi
    mkdir /mnt/private 0000 system system        //创建挂载目录,该挂载目录是第一行中自己定制的那个目录
    mount         vfat         /dev/block/nandi         /mnt/private gid=1019,uid=1019,fmask=0007,dmask=0007 //挂载private分区
```

9.2Private 分区读写

homlet 提供一个扩展接口来读写 Private 分区，该接口位于 android/framework/base/swextend/securefile/java/SecureFile.java

> 注:构造 SecureFile 实例时,如果传入的是相对路径,则该文件位于定制的 private 分区内,如 SecureFile file = new SecureFile(“abc.rc”).

SecureFile 对 private 分区中的文件操作和 File 对文件的操作类似,对文件读写的操作有如下四个接口:

```
/**把源文件内容写入本文件中
*@param srcFilePath  源文件路径,传入相对路径是表示该文件的根路径是private分区
*@param append  是否以添加的方式把数据写入文件末尾
*@return 返回真表示成功
*/
public boolean write(String srcFilePath, boolean append)

/**把源数据写入文件
*@param srcData  数据流,最大为1MB
*@param append  同上
*@return 同上
*/
public boolean write(byte[] srcData, boolean append)

/**把本文件的内容读到目标文件中,数据会覆盖掉目标文件,即append为false
*@param destFilePath  目标文件路径,可以为相对路径
*@return  同上
*/
public boolean read(String destFilePath)
```

```
/**把本文件内容读到目的数据流中
*@param destData  目的数据流,最大只能读入1MB
*@return  同上
*/
public boolean read(byte[] destData)
```

10 音频动态管理

10.1概要

音频动态管理机制主要有三个特点:

1.音频支持多通道同时输出,

2.支持单通道输入

3.支持 USB 音频设备热插拔检测

这三个功能都有相应的接口.

10.2音频管理策略

目前公版对于音频动态管理机制的策略如下:

a.上电后,一开始默认 HDMI 输出,

b.然后注册检测显示设备的热插拔广播信息,显示输出设备热插拔,音频输出做相应改变,相关代码在

frameworks\base\packages\SystemUI\src\com\android\systemui\statusbar\policy\DisplayController.java 中

c.在系统 APK:SystemUI 起来后注册监听 USB 音频设备和耳机的热插拔信息.目前公版的做法是:

对于输入:启动时,先检查是否有 USB 音频输入设备接入,有则切换至该输入设备

对于输出:启动时,先检查耳机是否已插入,有则切换至该输出设备,否则再检查 USB 设备,有多个输出设备接入时,只选择第一个.

然后开始监听热插拔广播.

每次有新的 USB 设备插入则切换至该输入通道,拔出时切换至默认的 codec 音频输入通道.

每次有新的USB输出设备插入,或耳机插入,则在状态栏中弹出"音频输出设备插入"的通知,用户点击该通知会弹出关于"选择音频输出模式"的设置项,可多选.代码位于

frameworks\base\packages\SystemUI\src\com\android\systemui\statusbar\policy\SoundController.java

d.针对遥控器操作,由于定义了一个快捷键用于点击时切换显示输出模式,所以音频输出部分也要做相应的切换.代码位于

frameworks\base\policy\src\com\android\internal\policy\impl\PhoneWindowManager.java 的截获到 key 时对于 keyCode == KeyEvent.KEYCODE_TV_SYSTEM 情况的处理中.

由于设置中有手动切换显示输出模式的设置项,因此音频在显示模式切换时也要做相对应的切换,代码位于 android4.2\device\softwinner\fiber-common\prebuild\packages\TvdSettings\src\com\android\settings\DisplaySetting.java 的最后十几行

e.用户可自己设置选择哪些音频输出模式(设置->声音->选择音频输出模式),该代码位 android4.2\device\softwinner\fiber-common\prebuild\packages\TvdSettings\src\com\android\settings\SoundSetting.java 和 AudioChannelsSelect.java

10.3上层提供的接口如下

frameworks\base\media\java\android\media\AudioManager.java

```
/* 定义三种默认音频设备的名称,如果是 USB 音频设备,则名字叫做 "AUDIO_USB0","AUDIO_USB1",... */
    public static final String AUDIO_NAME_CODEC          = "AUDIO_CODEC";
    public static final String AUDIO_NAME_HDMI       = "AUDIO_HDMI";
    public static final String AUDIO_NAME_SPDIF      = "AUDIO_SPDIF";

    /* define type of device */
    public static final String AUDIO_INPUT_TYPE      = "audio_devices_in";
    public static final String AUDIO_OUTPUT_TYPE     = "audio_devices_out";
    public static final String AUDIO_INPUT_ACTIVE    = "audio_devices_in_active";
    public static final String AUDIO_OUTPUT_ACTIVE = "audio_devices_out_active";

/** 获取当前可用的音频设备
```

* @param devType 值为 AudioManager.AUDIO_INPUT_TYPE 是,返回当前可用的的音频输入设备列表;或者为 AudioManager.AUDIO_OUTPUT_TYPE 时返回当前可用的音频输出设备列表,参数为其他值时返回 null

*/

public ArrayList<String> getAudioDevices(String devType)

/** 获取当前被使用的音频设备 */

* @param devType 值为 AudioManager.AUDIO_INPUT_ACTIVE 是返回当前被使用的音频输入设备列表;或者为 AudioManager.AUDIO_OUTPUT_ACTIVE 时返回当前被使用的音频输出设备列表,参数为其他值时返回 null

*/

public ArrayList<String> getActiveAudioDevices(String devType)

/** 把传进来的音频设备设置为当前被使用的音频输出/输入设备,每次调用该方法会更新当前被使用的设备列表

* @param devices 音频设备列表,必须是 getAudioDevices()得到的设备列表中的某些设备

*@param state 状态,值只能为 AUDIO_INPUT_ACTIVE 和 AUDIO_OUTPUT_ACTIVE

* 调用该方法,会把传入的设备列表中的设备全部设置为被使用状态,这会覆盖之前的被使用设备列表,*/

public void setAudioDeviceActive(ArrayList<String> devices, String state)

10.4音频相关接口使用例子

可以参考 frameworks\base\packages\SystemUI\src\com\android\systemui\statusbar\policy\SoundController.java 中的代码

10.4.1 监听 USB 音频设备热插拔

```
//注册接收USB音频设备热插拔的信息

IntentFilter filter = new IntentFilter();
```

```
filter.addAction(Intent.ACTION_AUDIO_PLUG_IN_OUT);

mContext.registerReceiver(mBroadcastReceiver, filter);



//在mBroadcastReceiver的onReceive()方法中,关于设备热插拔信息如下

public void onReceive(Context context, Intent intent) {

Bundle bundle = intent.getExtras();

final int state = bundle.getInt(AudioDeviceManagerObserver.AUDIO_STATE);

final String name = bundle.getString(AudioDeviceManagerObserver.AUDIO_NAME);

final int type = bundle.getInt(AudioDeviceManagerObserver.AUDIO_TYPE);

//state有两个值:AudioDeviceManagerObserver的PLUG_IN和PLUG_OUT分别表示设备插入或移
除;name是该USB音频设备名;type有两个值AudioDeviceManagerObserver的AUDIO_INPUT_TYPE
和AUDIO_OUTPUT_TYPE分别表示该设备是音频输入设备还是输出设备

}
```

10.4.2 音频输出/输入模式

(1)操作

```
mAudioManager=(AudioManager)context.getSystemService(Context.AUDIO_SERVICE);

ArrayList<String> lst = new ArrayList<String>();

//设置HDMI音频输出

lst.add(AudioManager.AUDIO_NAME_HDMI);

mAudioManager.setAudioDeviceActive(lst, AudioManager.AUDIO_OUTPUT_ACTIVE);

//设置SPDIF音频输出

lst.clear();

lst.add(AudioManager.AUDIO_NAME_SPDIF);

mAudioManager.setAudioDeviceActive(lst, AudioManager.AUDIO_OUTPUT_ACTIVE);

//设置CODEC音频输出(注:如果切换显示设备到CVBS输出时,音频应该同时切换到CODEC输出)
```

```
lst.clear();

lst.add(AudioManager.AUDIO_NAME_CODEC)

mAudioManager.setAudioDeviceActive(lst, AudioManager.AUDIO_OUTPUT_ACTIVE);

//设置某个USB音频输出，比如"AUDIO_NAME_USB0"，如果有usb音频设备插入时，通过getAudioDevices(...)接口得到的可用音频设备列表中就可以看到该USB音频设备的名字

lst.clear();

lst.add("AUDIO_NAME_USB0");

mAudioManager.setAudioDeviceActive(lst, AudioManager.AUDIO_OUTPUT_ACTIVE);

以上关于设置音频设备生效的操作中,lst可以为任意个设备的组合,以实现多设备同时输出


//设置音频输入设备生效,目前只支持使用单设备做输入,因此lst中的元素只有一个,比如设置使用CODEC输入,则:

lst.clear();

lst.add(AudioManager.AUDIO_NAME_CODEC);

mAudioManager.setAudioDeviceActive(lst, AudioManager.AUDIO_INPUT_ACTIVE);


//获取当前可用的输入设备列表

lst = mAudioManager.getAudioDevices(AudioManager.AUDIO_INPUT_TYPE);

//获取当前可用的输出设备列表

lst = mAudioManager.getAudioDevices(AudioManager.AUDIO_OUTPUT_TYPE);
```

11 支持 spdif

11.1 使能 spdif 模块

在配置文件 lichee\tools\pack\chips\sun7i\configs\android\<方案>\sys_config.fex 中，将参数 spdif_used 配置为 1：

```
spdif_used = 1
```

同时，根据方案的原理图配置 spdif 所需的 pin 脚参数“spdif_dout”。注意，需要在此文件内排查此 pin 脚是否存在使用冲突问题。

11.2 安装 spdif 驱动

修改 android4.2\device\softwinner\<方案>\init.sun7i..rc 文件,在其中加入加载 spdif 驱动的脚本.

```
#spdif

    insmod /system/vendor/modules/sun7i_spdif.ko

    insmod /system/vendor/modules/sun7i_spdma.ko

    insmod /system/vendor/modules/sndspdif.ko

    insmod /system/vendor/modules/sun7i_sndspdif.ko
```

11.3 切换声道至 spdif

详情请见"音频动态管理"章节中,"音频输出模式"小节的代码例子

12 配置出厂时的默认 launcher

在原生的 android 系统中，如果系统中存在多个 launcher，系统初次启动时将会弹出一个对话框，上面列出了当前系统中所有的 launcher。然后用户从列表中选择一个作为当前使用的 launcher。

很多用户并不懂这个列表是什么意思，从而就产生了困扰。对于许多厂家来讲，他们也希望用户第一眼看到的就是他们定制化的 launcher。

基于这种实际需求，我们新增加了一种机制，允许方案客户针对不同的方案配置出厂时的默认 launcher。

12.1配置默认 launcher

在文件 android4.2\device\softwinner\sugar-ref001\sugar-ref001.mk 中，在变量“PRODUCT_PROPERTY_OVERRIDES”中增加两个配置项 ro.sw.defaultlauncherpackage 和 ro.sw.defaultlaunchercla ss。 这两个配置项分别对应所选 launcher 的 package name 和 class name。

譬如，如果想把 TvdLauncher 作为出厂时的默认 launcher，可以如下添加

```
ro.sw.defaultlauncherpackage=com.softwinner.launcher \

ro.sw.defaultlaunchercla ss=com.softwinner.launcher.Launcher
```

12.2保留原生做法

当然，某些方案可能仍然希望保留原生做法，那么只要不添加上述两个字段即可。

13 关机

13.1关机时不再出现对话框

在文件 device\softwinner\sugar-ref001\sugar-ref001.mk 中 PRODUCT_PROPERTY_OVERRIDES 字段后面追加一个属性值，如下所示：

```
ro.statusbar.directlypoweroff = true;
```

就可以在关机时不再出现对话框，而是直接进入关机流程，界面上仅仅出现一个提示。

如果希望保留原生做法，那么可以将其值设为“false”或者删除此字段。

此值默认为“false”。

13.2短按 power 键关机

Android 原生的做法是：长按 power 键关机，短按 power 键进入 standby。

某些方案可能希望短按遥控器的 power 键进入关机，为此，在文件 device\softwinner\sugar-ref001\sugar-ref001.mk 中 PRODUCT_PROPERTY_OVERRIDES 字段后面追加一个属性值，如下所示：

```
ro.sw.shortpressleadshut = true;
```

此值默认为“false”。

14 替换鼠标图标

替换 3 个图片文件：

android4.2\frameworks\base\core\res\res\drawable-mdpi\pointer_arrow.png

android4.2\frameworks\base\core\res\res\drawable-hdpi\pointer_arrow.png

android4.2\frameworks\base\core\res\res\drawable-xhdpi\pointer_arrow.png

15 配置 boot 阶段初始化的 gpio

在配置文件 sys_config.fex 中配置[gpio_init]参数。

范例：

```
[gpio_init]
pin_1                   = port:PH10<1><default><default><0>
pin_2                   = port:PH20<1><default><default><1>
```

以上配置表示：在 boot 阶段，设置 ph10 输出低电平，设置 ph20 输出高电平。

16 全屏显示

在原生的 android4.2 系统中，界面底部的状态栏始终存在。但对于在客厅使用的设备，隐藏状态栏是有必要的。为此，homlet SDK 中提供了两种机制用来隐藏状态栏。

16.1总是隐藏状态栏

系统新添加了一个属性“ro.statusbar.alwayshide”，如果此属性的值为“true”，那么状态栏总是隐藏。

此属性值默认为“false”。

16.2使能 windowFullscreen

上诉机制一刀切，要么始终有状态栏，要么始终没有。很多客户希望由应用程序自身确定是否全屏，也就说希望原先在 2.3 之前的 android 系统上能够全屏显示的 activity 在 421 的系统上仍然能够全屏显示。

为此，在 homlet 4.2 的 SDK 中，我们必须做改动，使得原先在 2.3 及之前版本的 android 上用于全屏显示的标签或代码在 4.2 上仍然起作用。譬如如下代码：

```
<activity android:name=".ActivityDemoActivity"
          android:label="@string/app_name"
android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
         >
```

以上代码在 2.3 及之前的系统中，能够使 activity 全屏显示，但在 4.2 上平板模式下就不起作用了。在我们改造后，这段代码仍然起作用。

目前系统中，默认使能这一机制。如果希望遵循原生的系统，可在文件 device\softwinner\sugar-ref001\sugar-ref001.mk 中 PRODUCT_PROPERTY_OVERRIDES 字段后面追加一个属性值，如下所示：

```
ro.statusbar.inheritfullscn = false;
```

注意：“ro.statusbar.inheritfullscn”的配置只有在“ro.statusbar.alwayshide”为“false”的情况下才起作用。

17 隐藏软键盘

关于软键盘和物理键盘的共生关系，android 原生的策略是这样的：

在需要调用输入法时，如果没有物理键盘插入，则弹出软键盘供用户输入。如果系统检测到有物理键盘输入，则隐藏软键盘，希望用户直接通过物理键盘输入。此时，状态栏上会出现一个键盘图标，点击此图标，弹出输入法列表界面，在此界面的最上部有一个开关项：“使用物理键盘”，并且此开关的状态为“开”。如果将此开关的状态切换为“关”，则软键盘将会重新弹出。

但是某些用户希望：在插入物理键盘后，软键盘仍然存在。为了满足这一要求，系统做了修改，默认情况下，如果插入物理键盘，软件盘仍然存在。

如果希望系统在这方面仍然保持 android 原生的做法，可在文件 device/softwinner/sugar-ref001/sugar-ref001.mk 文件中，在变量“PRODUCT_PROPERTY_OVERRIDES”中增加一个新的配置项：

```
ro.sw.hidesoftkbwhenhardkbin=1
```

18 多屏互动设备名称

在多屏互动场景中，为了统一多个服务向客户端展示的设备名称，现在系统中增加了一个属性“persist.sys.device_name”，在文件 android/device/softwinner/sugar-ref001/sugar-ref001.mk 文件中定义，默认值为“MiniMax”。

方案商可以在出厂前修改此属性的值，也可以在 Settings 应用程序中添加一个设置项，使得消费者可以自定义设备名称。

19 配置流媒体缓冲策略

在 framework\av\media\CedarX-Projects/CedarXAndroid/IcecreamSanwitch/CedarXPlayer.h 文件中声明了

bool setCacheParams(int nMaxCacheSize, int nStartPlaySize, int nMinCacheSize, int nCacheTime, int bUseDefaultCachePolicy);

各参数作用如下：

nMaxCacheSize: 缓冲区最大值，当下载的数据量超过 nMaxCacheSize 字节时，缓冲线程停止继续下载数据；

nStartPlaySize: 启动播放的数据量大小，当播放器处于自动缓冲状态，播放器检测到数据量大于 nStartPlaySize 时，从缓冲状态恢复

到播放状态；

nMinCacheSize: 最小缓冲数据量，当播放器处于播放状态，播放器检测到数据量小于 nMinCacheSize 字节时，从播放状态切换到缓冲

状态；

nCacheTime: 当使用播放器默认的 cache 策略时，播放器根据 nCacheTime 计算 nStartPlaySize 的大小，此时 nMinCacheSize 为 64KB，

nMaxCacheSize 为 100MB. nCacheTime 单位为秒，nStartPlaySize = nCacheTime*Bitrate.

bUseDefaultCachePolicy: 是否使用播放器默认策略；

该函数需要在播放器的 Prepare()函数被调用后才能被调用，播放过程中随时可设。

20 SystemMix 可扩展接口说明

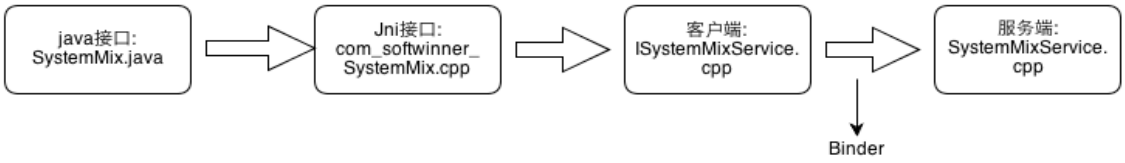
在 android/frameworks/base/swextend/systemmix 目录,我们提供了一套用于访问底层高权限信息的接口,客户可参照里面的做法来扩展该 SystemMix 类的功能

20.1提供该接口的目的

目前有些信息,如 mac 地址,序列号等,是保存在底层文件中,一般为 root,system 等这些高权限,因此客户自己开发的 apk 没权限去访问该文件的信息.systemmix 机制给 apk 以 root 权限去访问这些信息.

20.2原理

该机制使用了 android 上使用广泛的客户端<--->服务端机制去实现,调度流程为:



20.3接口使用

该 java 类目前提供了三个接口:

```
/** 获取某个property属性,传入属性的key值,返回其对应的value,如果key值不存
* 在,返回null
*/
public static String getProperty(String key);

/** 设置某个property属性的值,传入属性的key值和value值,如果该属性key不
*存在,则新建该属性并赋值为value
*/
```

```
public static String setProperty(String key, String value);


/** 获取系统启动参数(即系统启动后的/proc/cmdline文件中的参数)
*/
public static String getCmdPara(String name)
如获取MAC地址:String mac = SystemMix.getCmdPara(“mac_addr”);


客户在扩展该SystemMix的接口时,可参考getCmdPara()方法的调度流程,对java端,jni端,客户端,服务端都要做相应的接口扩展.
```