|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 深圳市华曦达科技股份有限公司 | | 文 档 编 号 | 版本号 | 密级 |
| 文档编号 | V1.0 | 机密 |
| **文档名称** | Amlogic平台打包说明 | | 日期 | 2017-04-01 |

# Amlogic平台打包说明

dmc1

|  |  |  |  |
| --- | --- | --- | --- |
| **文档作者：** | **曾秋龙** | **日期：** | **2017-04-01** |
| **项目经理：** |  | **日期：** |  |
| **审 核：** |  | **日期：** |  |
| **批 准：** |  | **日期：** |  |

深圳市华曦达科技股份有限公司

文档历史发放及记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **变更（+/-）说明** | **作者** | **版本号** | **日期** | **批准** |
| 1 | 初稿 | 曾秋龙 | V1.0 | 2017-04-01 |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

文档简要功能及适用范围

### 文档的简要功能

该文档主要用于描述Amlogic平台克制化软件打包目录结构及其功能说明。

### 文档的适用范围

本文档适用于Amlogic平台打包维护人员及基础软件平台组人员。

目 录

[New Launcher概要设计文档 1](#_Toc462835630)

[文档的简要功能 3](#_Toc462835631)

[文档的适用范围 3](#_Toc462835632)

[一 引言 5](#_Toc462835633)

[1.1背景 5](#_Toc462835634)

[1.2术语和缩略语 5](#_Toc462835635)

[1.3参考资料 5](#_Toc462835636)

[二 主要功能及实现方案 6](#_Toc462835637)

[2.1 换肤功能 6](#_Toc462835638)

[2.1.1 实现方案概述 6](#_Toc462835639)

[2.2 应用界面 8](#_Toc462835640)

# 一 引言

### 1.1背景

随着公司业务迅速开展，客制化的平台越来越多，工程师参与平台维护，进行打包维护工作势不可免。该文档主要目的在于帮助工程师快速上手，掌握平台各方面定制的方法和技巧，提高客户平台维护的效率。

### 1.2术语和缩略语

[对文中使用的术语和缩略语进行说明。]

| **缩略语/术语** | **全 称** | **说 明** |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

### 1.3参考资料

《M8平台软件定制.docx》----------------------------------------------------------------柳先明

《Amlogic-mx平台打包说明》------------------------------------------------------------曾秋龙

# 二 主要目录结构及其结构说明

## 打包目录结构



如图，以M8为例，打包目录分为6个部分：Bootloader，kernel，platform，framework，tools，makefile，out目录和cach目录。

**值的一提的是，makefile，out，platform目录结构对打包维护来说非常重要，请重视这些目录的说明**。

### Bootloader

Bootloader目录，主要由基础平台同事提供，用于存放软件的Uboot启动程序，文件夹命名为bootloader（唯一），以M8普通平台为例，文件夹下面存放的是u-boot.bin，命名唯一。

根据平台的差异性除了u-boot.bin文件，bootloader下面也许还会有其它文件目录，如M8 secure平台，bootloader下存在Aml-Image文件夹，S912 平台，bootloader下存在upgrade文件夹。

无论如何，改目录的文件都是由基础平台的同事提供，再维护的时候，注意平台组相关的更新，同步及时更新即可

### Kernel

Kernel目录，主要又基础平台同事提供，用于存放软件的kernel和recovery程序，文件夹命名为kernel（唯一），文件夹下面存放的文件是：

kernel（命名唯一，内核文件）

ramdisk文件夹，文件夹包含文件root.tar.bz2和root-recovery.tar.bz2，命名都是唯一的

### framework

framework文件，指的是android原生的整个/system目录下所有文件，以zip压缩方式，由基础平台同事提供，以M8为例，在根目录下面，命名为m8-system.tar.bz2，名字唯一。

根据平台的差异性，命名也许不一样，但是必须要与后面所说的makefile文件一一对应

### Makefile文件

makefile文件，命名唯一，主要用于打包的时候一些脚本定义，在打包服务器上执行make的时候，首先执行的就是该文件。文件由基础软件同事提供，后续由打包人员，根据客制化需求，自己维护。

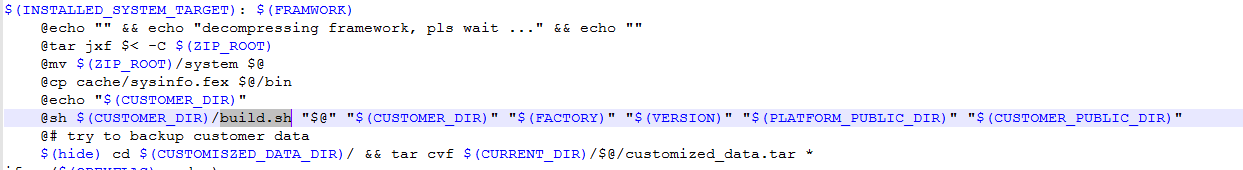
根据以往经验，makefile文件，一般来说有以下几点打包调试技巧：

#### 1）屏蔽build.sh执行脚本，打一个和基础平台类似的包

在整合过程中，难免会碰到有些功能基础平台的软件是好的，但是我们打出来的软件有问题，因此无法判断究竟是我们的打包环境有问题，还是我们的添加的文件有问题。

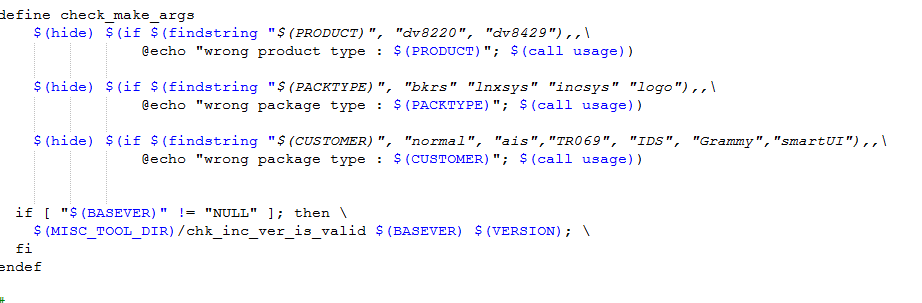
为了快速验证我们的打包工程，我们可以通过屏蔽build.sh脚本的方式，打一个类似基础平台的公版软件，来验证打包环境是否是好的。

如图，屏蔽带build.sh字样的那行执行命令即可



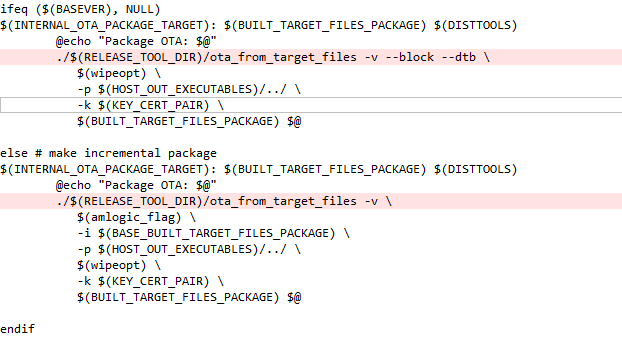
#### 2）添加平台信息，添加客户信息

如图，PRODUCT为平台信息，CUSTOMER为客户信息



#### 3）移除recovery编译时间校验

由于安卓平台升级的时候，自带recovery时间校验，新的recovery软件，无法升级到老的recovery软件上。有时候该功能不便于客户那边验证升级，可以更具客制化需要，把该部分校验拿掉，如图，将RELEASE\_TOOL\_DIR那两行脚按如图编写即可



### Tools目录

Tools目录主要由基础平台组整合，改tools目录的准确性，主要由基础平台的同事来保证，一般来说，只要新平台的打包环境没有问题了，tools目录是不会变更的。

### Cache 目录

Cache目录，只有执行了打包命令之后，才会生成。

Cache目录的作用在用存放打包过程中的解压文件，如bootloader，kernel下的文件，framework的解压文件都是存放在改目录下，而build.sh文件所处理的添加，删除，修改等，都是在该cache目录上实现。

最后，升级包的形成所依赖的所有文件，都在cache目录可以找的到。

#### Cache目录查看技巧：

我们可以查看一下cache目录，里面的所有文件或者文件夹，都是和android系统的文件管理系统一样的，因此，有时候打包的时候，想快速查看打包的准确性，可以直接到cache目录查看

**注意**：

由于cache目录用于存放打包过程中的解压文件，每次打包的时候，cache目录下的文件将会被最后打包的文件给覆盖。

### Out目录

Out目录也是打包的时候，由tools和makefile文件生成。

Out目录包含了一下几个目录：backup，base-otapackage，host，images，otapackages

#### Backup

Backup备份目录，一般来说作用不大，我们在维护的时候一般可以不用处理该目录

#### Base-otapackage（增量包“神器”）

该目录主要用于存放打包的时候生成的基础软件包，主要有两个用途：

1. 制作android增量包的时候，可以用该目录下的zip包来执行，具体的制作方式，可参考博客<http://blog.csdn.net/safsasasa/article/details/24134845>
2. 该目录下的zip包，解压后，可以查看每个版本对应的文件。

**注意**：

Base-otapackage目录，如果重复打同一个平台，客户，版本号的包，老的zip文件将会被新的zip包给覆盖

#### Host目录

Host目录，一般来说作用不大，我们在维护的时候一般可以不用处理该目录

#### Images目录

Image目录，主要生成了打包过程中的uboot文件，recovery镜像文件和system的镜像文件。

以M8以前的平台为例，我们在做启动卡过程中需要用的的uboot文件和recovery.img文件，都是在该目录下获取

**注意**：

打包过程中，该目录下的文件会被覆盖，最终存放的是最后一次打包生成的文件，因此我们在维护的过程中，重要的文件，要有意思的保留起来。

#### Otapackages

Otapackages目录，我们在打包过程中，生成的升级文件，就是在该目录下面。

我们可以查看一下该目录，可以找到对应的平台目录，客户目录，然后对应版本号的zip包和img包。

### Platform

Platform目录，对平台打包人员来说，非常重要，里面包含了所有的客制化信息，如添加了那些应用，删除了哪些应用，增加了哪些系统特性等等。

打包人员需要非常重视该目录的说明。

#### Platform目录结构图



如图，platform目录下，包含了不同的平台目录，如DV7819，DV7810…，也包含了平台的公共目录public。

**注意**：

根据makefile脚本的差异性，有时候public目录会放到平台目录下面，但整体来说，大同小异，只要makefile脚本定义好了目录结构，我们维护public目录基本没有区别

##### 客户目录：normal，customer1…

客户目录，主要存放了客户logo，开机动画，客户apk，客户定制apk等等，如图以normal为例。



###### Bootlogo

Boot-logo，主要用于存放客户开机第一张图片，即uboot图片

图片格式：1920\*1080， bmp， 位深24位

一般来说，这些图片，我们可以把图片要求发给客户，让客户自己提供

###### Init-logo

Init-logo，和bootlogo一样，放同样的图片即可

###### Private.system



Private.system目录，其目录结构和android system结构基本上是一样的，如app, etc, framework, preinstall, lib…等等，这些目录在android系统上都可以找的到，我们在打包的时候，通过build.sh脚本，把这些目录下存放的文件，拷到cache目录下的system目录下（注意：如果android原生存在相同的文件，会被private.system的文件给替换掉），最终生成的zip升级包，就包含了这个private.system目录下的文件。

build\_device.prop文件，主要用于添加一些客户定制化的系统默认属性，如下，会一一说明

App：一般用于存放客户定制化的apk，如客户的第三方apk，客户定制的launcher，和其它一些定制化的apk等等

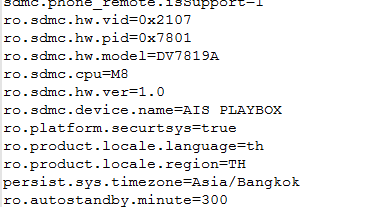
Etc：一般用于存放客户定制化的配置文件，如遥控器配置文件remote.conf，launcher的定制化配置文件precut\_app.xml文件

Framework：用于存放客制化的jar包，如客户对系统的某些功能做了定制化修改，期间生成的jar包，可以放到该目录下

Preinstall：主要用于存放安装的apk，有时候根据客户的需要，有些apk不是预置的，需要预安装，就可以通过该方式来预安装（主要由/system/bin/preinstall.sh脚本来启动安装，这个会在后面说明）。

Lib：用于存放客户定制化修改中生成的.so文件

Build\_device.prop:主要用于配置系统的一些默认设置，如语言，地区，是否支持某些功能，属性等等。如下图：



### 2.1.1 实现方案概述

针对于目前我们的需求，实现的主要途径为定义一套私有的素材加载协议，具体协议内容见文档《HotelUI素材加载协议》。协议主要定义了用户界面上每个元素的相关内容，包括坐标、宽高、加载路径、ID等一系列的内容，从而实现通过加载协议相关内容，实现应用页面内容的更换工作。

资源包的加载过程为：首先应用加载存储在机顶盒中的资源包，其次如果网络连接成功，从网络请求资源包，请求成功，则下载。资源包分为两个部分，其一为协议文件，其二为资源文件，应用程序需要进行资源解析。

#### 2.1.1.1资源下载与解析模块

资源的加载采用网络开源下载框架HttpUtil，实现资源包的断点下载。

协议的解析采用工厂方法模式，用于解析各个模块的资源文件，这样的好处是各个模块的解析相对来说比较独立，互相之间没有耦合性。为了保证协议解析的耗时在1S内，需要通过一个线程池，采用多线程的方式进行解析。

现阶段Json格式数据的解析主要分为以下几种：GJSON、JackSon、JSONNP等方法，下图是各个解析方法在小文件格式下的json数据解析速度[图1]，在小文件格式下面，Google的json解析方法明显比开源的JackJson和Java语言支持的JSONNP速度快，因此，为了保证协议解析的耗时，我们采用Google的GJSON方法。

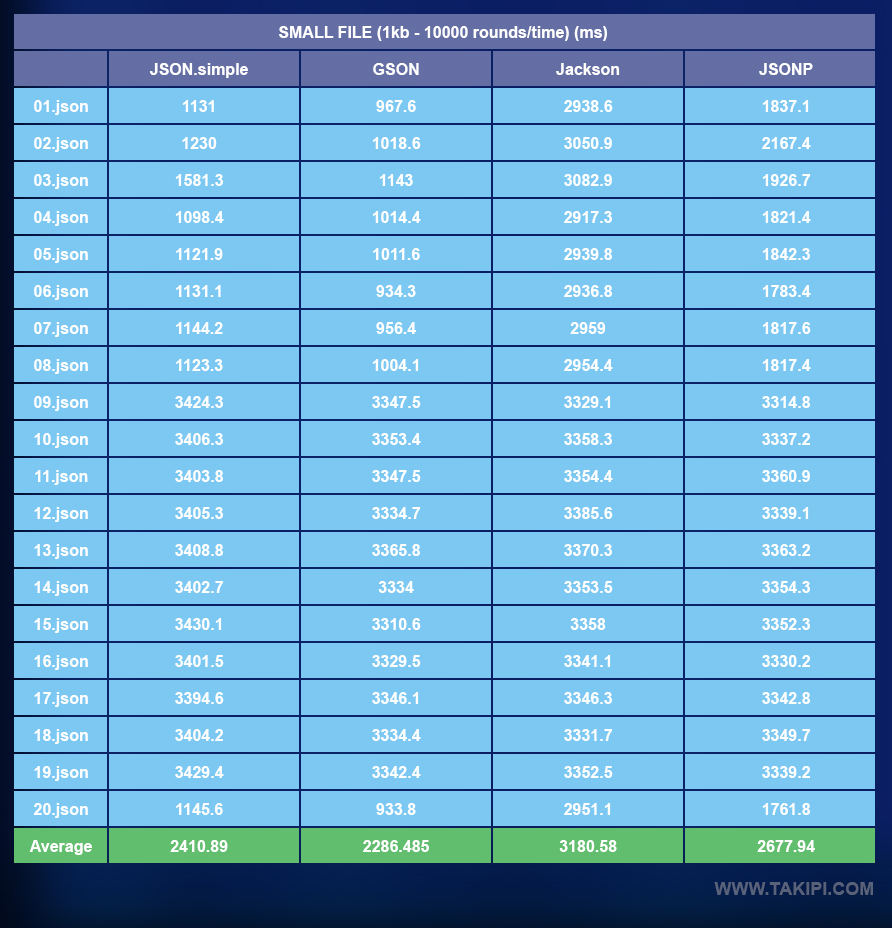


图1：json解析速度对比

协议文件采用Json数据格式，基于Json数据格式进行的解析框架图[图2]如下：



图2：协议解析类图

#### 2.1.1.2 资源加载模块

协议文件通过解析将各个模块的一些相关参数和图片等资源存放到各自的数据类当中，但是由于在Android应用中，图片资源占用的内存很多，所以我们需要对图片资源的加载进行缓冲处理，通常的做法是实现一个缓冲，防止图片资源的重复加载造成内存不断增加。

缓冲池采用Android推荐的LruCache，将图片动态缓存到Cache中，如果遇到大图片，则对大图片进行降采样的方式，缩减大图片占用的内存空间。

## 2.2 应用界面

应用界面整体可以分为三个模块，状态菜单栏、内容显示区（内容页）、内容分类菜单（导航栏）。

状态菜单栏：位于桌面最上方，显示时间日期、网络状态和一些跳转入口图标；

内容显示区：显示内容，包括视频、游戏、广告等；

内容分类菜单：显示分类，选择不同的分类会在内容显示区显示不同的内容界面；

整体模块图如下[图3]：



图3：应用界面整体模块图

#### 2.2.1 状态栏模块

状态栏主要分为三个部分：时间显示、供应商和机顶盒状态三个部分。考虑到这三个部分都具有一定的独立性，因此我们在加载View的时候，需要进行分别处理，类图设计如下[图4]：



图4：状态栏类图

TimeView是一个自定义的View，通过一个内部的线程不断的更新系统的时间。

StatusView是一个机顶盒状态信息显示的View，用于显示机顶盒的以太网、Wi-Fi、DLNA、OTA等信息，它组合了多个状态信息，通过注册的广播去监听系统状态的改变，并调用refresh去通过StatusHandler发送消息，更新状态栏。

#### 2.2.2 菜单栏

菜单栏主要用于处理客户提供的一些APK、URL等工具的跳转，其配置选项来自于资源包。针对于AIS客户，目前有的工具有：购物、账户、消息等。工具栏中各个工具没有任何相关性，因此，UML类图如下[图5]：



图5：菜单栏类图

MenuItem是一个抽象类，主要抽象了菜单跳转和生成菜单View的函数。各个菜单功能集成于MenuItem，用于实现具体的菜单跳转和生成具体的View。

#### 2.2.3 内容页

内容页的显示内容和布局风格大概分成三种：视频布局风格、游戏布局风格、广告布局风格，封装一个抽象类AbstractCommonHomeContent，让这三种风格的类来继承实现对应的接口，类图如下**[**图6**]**：

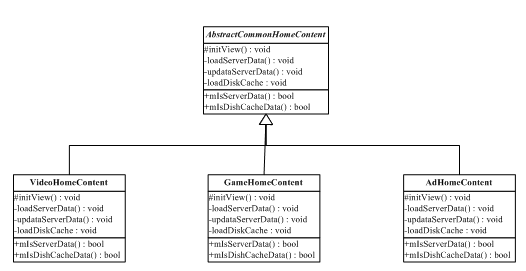


图6：内容页类图

#### 2.2.4 导航栏

导航栏是用于快速切换内容分类的部件，其个数来自于资源包和用户的手动添加两个部分。功能要求为焦点默认在最中间的位置，初始焦点在Home导航上，可以进行左右循环切换，可以进行用户手动添加功能。

实现的类图如下[图7]：



图7：导航栏类图

NavigationCreator负责根据资源包创建所有的子导航View(NavigationView)。

NavigationLayout负责显示所有的NavigationView。

##### 2.2.4.1 关键问题解决方案

###### 2.2.4.1.1 循环切换

循环切换的技术点在于导航数据的控制，我们在这里采用循环队列的方式，根据导航个数不同，组合不同的算法进行数据分配。

例如 固定导航数为11个。

情况1：前端配置导航个数小于11。则在此基础上面需要将队列头部元素复制一个到队列尾部，队列尾部元素复制一个到队列头部，在此基础上面取配置的个数个元素进行切换。

情况2：前端配置导航个数大于11。则在切换的过程中，取固定11个元素显示。

###### 2.2.4.1.2 焦点控制

循环切换要求焦点固定在中间位置，永远不改变，我们可以参考Float Launcher应用的焦点控制。

###### 2.2.4.1.3 高亮显示

根据效果图可知，拥有焦点的导航是高亮的，而没有焦点的导航是有一定灰度值的，在此基础上面有两个解决方案。方案1为设置View的灰度，方案2为编辑一张具有灰度但是中间没有灰度的图片，覆盖在导航栏上面，这样就不需要动态设置View的灰度值。

#### 2.2.5内容页的焦点控制

内容页的视频页页面包括左边的串流播放区和右边可以横向滑动的视频内容页，加起来一共两页，串流播放区自定义一个播放器类，用于管理播放逻辑和界面显示；视频页用一个自定义的HorizontalScrollLayout包装起来，实现左右的滑动翻页；HorizontalScrollLayout里面的每个子View都封装成一个自定义的View，方便管理；封装一个焦点控制管理类，专门控制子View获取到焦点和失去焦点后样式的变化。