# STK 模块架构\_apk

|  |  |  |
| --- | --- | --- |
| Date  日期 | Version  版本 | Comments  备注 |
| 2018-12-20 | 0.1 | First version |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Function  职位 | Name  姓名 | Date  日期 | Signature  签名 |
| Written by  拟定 | SW | 曾灿炫 | 2018-12-20 | 曾灿炫 |
| Verified by  审核 |  |  |  |  |
| Verified by  审核 |  |  |  |  |
| Approved by  批准 |  |  |  |  |

目录

[STK 模块架构\_apk 1](#_Toc533599380)

[1. 概述 3](#_Toc533599381)

[**1.1.** **本文档的目的** 3](#_Toc533599382)

[**1.2.** **背景** 3](#_Toc533599383)

[**1.3.** **缩略语清单** 3](#_Toc533599384)

[**1.4.** **参考文献** 3](#_Toc533599385)

[2. Stk基本框架 3](#_Toc533599386)

[2.1. 简介 4](#_Toc533599387)

[2.2. Stk应用层框架 5](#_Toc533599388)

[2.2.1 框架图 5](#_Toc533599389)

[2.2.2 关键类解析 5](#_Toc533599390)

[2.2.3 STK COMMAND与对应类 6](#_Toc533599391)

[3. StkAppService启动流程 6](#_Toc533599392)

[3.1 基本流程 6](#_Toc533599393)

[3.2 由BootCompletedReceiver 启动(Stk开机流程) 7](#_Toc533599394)

[3.3 由StkCmdReceiver启动(StkCommand处理流程) 7](#_Toc533599395)

[3.4 由StkLauncherActivity等启动(用户) 7](#_Toc533599396)

[4. 与Framework层的消息传递 8](#_Toc533599397)

[4.1 接收消息 8](#_Toc533599398)

[4.2 发送Resopnse 8](#_Toc533599399)

[5. Android.mk 9](#_Toc533599400)

[6. 总结 10](#_Toc533599401)

# 概述

* 1. **本文档的目的**

*本文档作为一个学习输出，总结STK模块在应用层的架构与运作流程。*

* 1. **背景**

*STK即SIM Tool Kit，它提供一系列用于移动设备与SIM卡间交互的机制。通过这些机制，支持STK的手机可以操作SIM卡里的应用。其Framework层负责消息的转化和传递，实现RIL层与应用层间的交互。*

* 1. **缩略语清单**

*列出文中使用的术语的定义和缩略（语）词的英文全名和中文解释。*

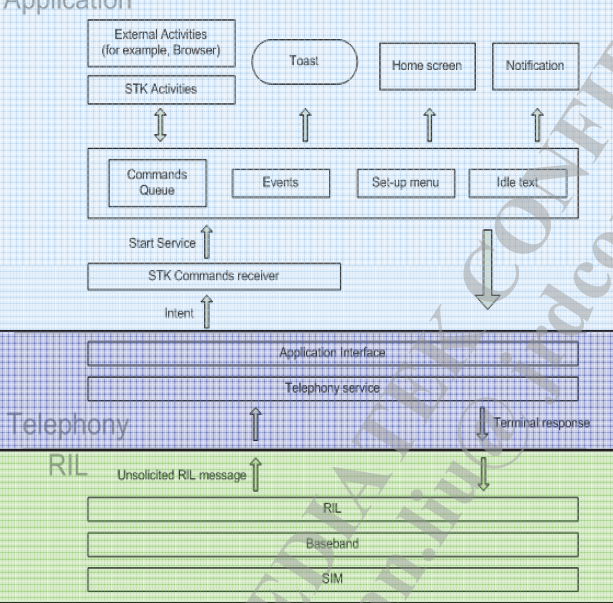
| Term | Explanation |
| --- | --- |
|  |  |
|  |  |
|  |  |

* 1. **参考文献**

| Document | Explanation |
| --- | --- |
| Android中的Uicc框架 | https://blog.csdn.net/u010961631/article/category/2430627 |
| AndroidL-STK架构与概要设计 | 叶文平 |
| 理解Android Build系统 | https://www.ibm.com/developerworks/cn/opensource/os-cn-android-build/ |

# Stk基本框架

## 简介



该图为Stk在AP端的基本框架，包含了应用层(蓝底部分)和Telephony层和RIL层，这里主要讲应用层部分的框架。

## Stk应用层框架

### 2.2.1 框架图



上图为Stk在应用的框架组成，图采用至文档《AndroidL-STK架构与概要设计》，

在现在的源生代码中，底下的StkService应该是CatService。

### 2.2.2 关键类解析

**BootCompletedReceiver**: 广播接收器，用于接收Intent.ACTION\_BOOT\_COMPLETED并启动StkAppService.

**StkAppService**:Stk的核心类，继承了Serivce并实现Runnable接口，用于处理各主动式命令，可由StkLauncherActivity、BootConpletedReceiver、StkCmdReceiver启动。

**StkLauncherActivity**: Stk应用显示类，当前设备存在两张SIM卡，且两张SIM都含Stk功能，则点击进入STK则会进入该类。

**StkMain**:STK应用的启动类，启动后会根据当前SIM卡数量决定是启动StkAppService还是StkLauncherActivity类。

**StkDialogActivity**:Activity类并实现OnClickListener接口监听按钮的事件，该类主要用于显示Display Text的信息，同时，将用户的操作响应信息如确认操作，返回操作，无响应等发送到StkAppService进行其他处理。

**StkInputActivity** : 显示Get Inkey/Input的输入框格式，如输入字符类型（字符或数字）、隐藏模式（no echo）、Yes/No的单选模式、输入字符的长度限制等，同时，将用户的操作响应信息如输入内容确定操作，返回操作，无响应等发送到StkAppService进行其他处理。

**StkMenuActivity**: 继承ListActivity类，显示STK的Set Up Menu与Select Item的菜单以及界面的定制，同时，将用户的操作响应信息如输入内容确定操作，返回操作，无响应等发送到StkAppService进行其他处理。

**ToneDialog**:定制播放STK Play Tone的对话框如播放的铃声类型，Icon的显示等，同时，将最终的执行结果发送给StkAppService进行其他的处理。

### 2.2.3 STK COMMAND与对应类

|  |  |
| --- | --- |
| Function | Class |
| DISPLAY\_TEXT | com.android.stk.StkDialogActivity |
| SET\_UP\_CALL |
| SELECT\_ITEM | com.android.stk.StkMenuActivity |
| GET\_INKEY | com.android.stk.StkInputActivity |
| GET\_INPUT |
| SET\_UP\_IDLE\_MODE\_TEXT | com.android.stk.NotificationAlertActivity |
| SET\_UP\_MENU | com.android.stk.StkAppService |
| SEND\_DTMF |
| SEND\_SMS |
| SEND\_SS |
| SEND\_USSD |
| PLAY\_TONE |
| RUN\_AT\_COMMAND |
| OPEN\_CHANNEL |
| CLOSE\_CHANNEL |
| RECEIVE\_DATA |
| SEND\_DATA |
| GET\_CHANNEL\_STATUS |
| CALLCTRL\_RSP\_MSG |

# StkAppService启动流程

## 3.1 基本流程

从Benz/Gauss的源码来看，StkAppService的大致启动流程留下：

1. onCreate()：获取CatService实例，开启新线程(StkAppService自身实现了Runnalbe接口)，在新线程中实例化内部类ServiceHandler(继承了Handler)
2. onStart()：根据启动StkAppService时传入的opcode设置Message的obj并发至MessageQuene
3. ServiceHandler->HandleMessage:根据cmd类型做对应的处理。

## 3.2 由BootCompletedReceiver 启动(Stk开机流程)

1. 在**BootCompletedReceiver**接收到来自UserController的开机广播Intent.ACTION\_BOOT\_COMPLETED后，启动StkAppService并传递OP\_BOOT\_COMPLETED，用于后续接受到Set Up Menu后建立 STK菜单。 然后调用StkAppInstaller.install(context)进行apk的安装。

1. 进入**StkAppService**，在onCreate()当中，注册监听卡状态改变的广播TelephonyIntents.ACTION\_SIM\_STATE\_CHANGED，并开启新线程。
2. 进入onStart()，新建一个message并传入opcode和卡id，发送至内部类ServiceHandler处理。
3. 进入**ServiceHandler**的消息处理函数，当op为OP\_BOOT\_COMPLETED时遍历所有卡实例查看是否有mMainCmd，即set\_up\_menu，没有则打log提示当前无Stk。

以上便是大致的Stk开机流程。

## 3.3 由StkCmdReceiver启动(StkCommand处理流程)

StkCmdReceive是静态注册的广播，在对应的AndroidManifest.xml文件中可看到其注册的接收事件，所有的事件都会启动StkAppService。

StkAppService的启动过程同上，直接进入ServiceHandler的HandleMessage，这里以SET\_UP\_MENU为例：

1. 当cmd为SET\_UP\_MENU时，设置当前卡实例的mMainCmd为mCurrentCmd（当用户点击STK图标且mMainCmdw为null时无法进入stk）
2. 调用removeMenu(slotId)查看当前是否所有卡状态都是removed，是则移除stk应用。不是则安装stk。

## 3.4 由StkLauncherActivity等启动(用户)

1. 在Benz/Gauss中，点击stk应用时，进入的是StkMain，程序会检测当前可用的stk数。单个时直接启动StkAppService，多个则启动StkLauncherActivity由用户选择想使用的stk。

这两种启动方式都传了一个OP\_LAUNCH\_APP。

1. 进入ServiceHandler的HandleMessage，当opcode为OP\_LAUNCH\_APP时调用cleanUpInstanceStackBySlot获取栈顶的第一个cmd类型，当类型为SET\_UP\_MENU或SELECT\_ITEM时调用launchMenuActivity进入stk的菜单显示。
2. 进入launchMenuActivity，该activity继承了ListActivity，直接看onListItemClick。根据用户点击的项，调用sendResponse并传入对应的itemId.
3. 进入sendResponse,可看到该函数启动了StkAppService并传入了OP\_RESPONSE。
4. 进入ServiceHandler的HandleMessage，当opcode为OP\_RESPONSE时调用handleCmdResponse根据ResId执行对应的处理，最终通过调用StkService的onCmdResponse(resMsg将消息发至Framework层;

# 与Framework层的消息传递

## 4.1 接收消息

应用通过StkCmdReceiver接收来自Framework层的消息并传递给StkAppService，该部分已在3.3说明。

## 4.2 发送Resopnse

进入handleCmdResponse，可看到：

if (mStkContext[slotId].mCurrentCmd == null) {

return;

}

if (mStkService[slotId] == null) {

mStkService[slotId] = CatService.getInstance(slotId);

if (mStkService[slotId] == null) {

// This should never happen (we should be responding only to a message

// that arrived from StkService). It has to exist by this time

CatLog.d(LOG\_TAG, "Exception! mStkService is null when we need to send response.");

throw new RuntimeException("mStkService is null when we need to send response");

}

}

第一个if用于判定当前是否存在指令，每一个response都需要对应存在的command。

第二个if用于获得CatService的实例。

获取CatService的实例后，新建一个CatResponseMessage,这个是由Framework层定义的消息结构体。

跳过swicth语句里对msg的具体操作，可看到：

mStkService[slotId].onCmdResponse(resMsg);

这一句是将消息发至CatService的MessageQuene。

# Android.mk

在STK的项目路径下可看到Android.mk，具体内容如下：

LOCAL\_PATH:= $(call my-dir)

include $(CLEAR\_VARS)

LOCAL\_MODULE\_TAGS := optional

LOCAL\_JAVA\_LIBRARIES := telephony-common

LOCAL\_SRC\_FILES := $(call all-subdir-java-files)

LOCAL\_PACKAGE\_NAME := Stk

LOCAL\_CERTIFICATE := platform

include $(BUILD\_PACKAGE)

做一个简单的解释。

前两行是Android.mk的标配，**LOCAL\_PATH := $(call my-dir)**，这句代码LOCAL\_PATH是变量名，$(call my-dir)的意思是调用my-dir这个函数，这个函数的返回值是Android.mk这个文件所在的位置。LOCAL\_PATH这个变量是一定要定义的，它告诉编译系统当前模块的位置。

**include $(CLEAR\_VARS)**这句代码的作用是清楚除了LOCAL\_PATH变量之外的LOCAL\_XXX变量，为什么要清楚这些变量呢？因为在编译当前模块之前可能编译过别的模块，此时可能存在别的项目保留的变量，会被系统误认为是当前模块的变量这样就会产生不可预知的错误。

**LOCAL\_MODULE\_TAGS := optional**：指明编译标签，当我们编译源码的时候需要调用lunch函数，lunch函数会打印出来源码包含的定制版本，可以分成user，userdebug，eng三种,该选可填user,eng,debug,optional，最后一个指所以的版本下都会编译。

**LOCAL\_JAVA\_LIBRARIES := telephony-common**：引入包，这个是存放系统中的包，当需要引用不存在与系统的包使用“LOCAL\_STATIC\_JAVA\_LIBRARIES”

**LOCAL\_SRC\_FILES :=$(call all-subdir-java-files)**：LOCAL\_SRC\_FILES变量代表需要编译的文件，这里调用了all-subdir-java-files函数，这个函数的返回值是LOCAL\_PATH子目录的所有java文件

**LOCAL\_PACKAGE\_NAME := Stk：**生成包名(相同包名会覆盖)

**LOCAL\_CERTIFICATE := platform**：指定使用了模块使用的签名文件，这里是使用system级别的签名文件。

**include $(BUILD\_PACKAGE)：**指明编译成APK

# 总结

以上便是STK在应用层的基本架构与核心服务类的启动流程，这里概要总结了整体的框架和部分流程，还需要熟悉各部分代码的具体作用，更新框架图。