­­­­­­­

**《移动平台开发技术》**

**分析与设计说明书**

**课 题：**  **DroidEye监控应用**

**院 （系）：**  **软件学院**

**专 业：**  **软件工程**

**小组成员 ：姓名 钟玉生 学号 201436615121**

**姓名 顾芳铭 学号 201430611105**

**指导老师** ： **程兴国老师**

**提交日期：**  **2016年12月26日**

1. 需求分析
   1. 任务概述

项目背景

作为如今市场上最大的移动端系统赢家，安卓无疑是目前最受关注和推崇的系统。但是跟IOS相比，仍然有不少用户觉得Android比IOS低级那么一点，有些人会觉Android应用没有iOS应用使用得流畅，越用越慢似乎成了Android甩不掉的骂名。但是，其实Android系统已经做得相当出色了，随着如今的不断的系统版本的迭代，Android已经逐渐成熟，归根结底，并不是Android系统的问题，而是Android系统设计的太随意，用户也使用的很随意。不同IOS有专门的应用商店和审查机制，安卓应用没有很官方的应用提供来源，虽然有GooglePlay提供较为正规的应用，但大多数应用程序随处可见，用户也可以随便安装，而且中国用户几乎无法接触到GooglePlay，这就导致了安卓手机为什么装上应用后的表现就大不如前，是应为安装的应用在设计方面有许多的问题。而且伴随着4G网络的推广，手机网速日益增加，但是用户流量却增加甚少，为了让用户更加有效的使用流量，对Android手机上应用程序的流量监控的要求就呼之欲出了。因此，一款监控Android手机上运行应用的程序就变得格外需要了。

实现目标

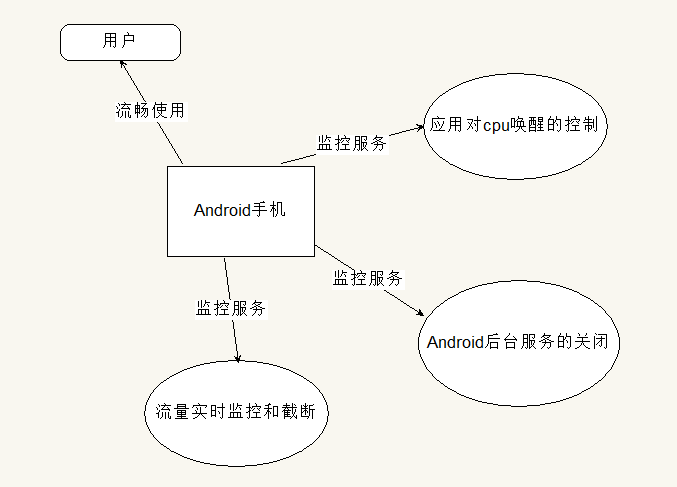
针对目前的应用存在的主要问题，以及用户的使用习惯，吸取了目前应用的不足之后，我们主要实现目标如下：

1. 全面监控Android应用对流量数据的访问，根据用户习惯，设置流量限额，根据通过测量实时网速，判断应用程序的行为，并告知用户，让用户对后台不想浪费流量的应用做到及时的处理，而不仅仅是一个流量的统计而已。
2. 获取Android系统后台正在运行的进程或者服务，在后台自动帮用户维护忘记关闭后者自动打开的应用程序，让用户不在为内存不足而担心，不再为忘记关闭应用而浪费的电而痛心。在后台实现的是强行关闭，而不是释放内存，用户可以添加白名单来允许后台长期允许的应用程序。
3. 监控应用程序对cpu的唤醒，Android手机的电量损耗很大程度是来源于手机锁屏后设备仍然处于唤醒状态而导致的浪费电量。我们从系统的底层出发，在native层监控应用程序的各种自唤醒操作，让手机设备能够在锁屏期间得到真正的节电，延长使用期限。

我们实现的功能不在多，而在于关键，只有直击问题的本质，才能从根源上解决问题。很多现有的应用说是实现了什么样的功能，但实则无效，或者浪费的更多资源。我们要做到是以最少的损耗实现最大的效果，而且切实有效。

* 1. 功能需求

功能划分



功能描述

Table 1

|  |  |  |
| --- | --- | --- |
| 功能 | 描述 | 预期 |
| 监控流量 | 当用使用数据网络时，所有的申请了网络连接权限的应用都可以在此时连接网络，因此，为了让用户了解流量到底是被那个用了，让该应用用多少，让不让该应用使用，这些都将得到控制，而不是仅仅的一个打开和关闭数据网络。 | 当用户的Android手机的流量使用超过预定值时，会向用户请求可以继续使用的流量只，当用户的Android手机突然流量变大，或者超过该应用设定的预期值，则提醒用户谁在使用，是否允许。从而全面监控流量的使用详情，让用户不在害怕在没有防备的情况下超过流量而损失金钱。 |
| 控制后台允许服务 | 根据8,2原理，推广到用户的使用情况上，80%的操作对20%的应用，所以，为了让用户在打开那另外80%不常操作而又没有及时关闭，直至填满手机内存发生之前，有后台服务自动强行关闭那些不常使用的应用及其服务，让常常使用的应用能够常驻内存，提高用户体验，同时为应用打开新应用提高打开速度。 | 当用户由于打开某项服务，比如晚上听歌，看视频而忘记关闭服务而导致第二天的没电甚至关机，后台服务保证手机的后台程序在无人响应时能即时关闭。防止用户的忘记而导致手机的表现降低，同时在日常使用过程中也能提高手机的使用流畅程度。 |
| 控制后台服务的唤醒锁 | 很多服务包括系统服务，都有可能频繁的唤醒cpu，为了达到应用的某个目的或者程序设计的缺陷，而导致手机在锁屏的情况下仍然耗电严重，控制应用程序对cpu的唤醒，能够让手机得到休眠，节省电量。 | 该情况特别适用在晚上，很多人都层试过手机一个晚上就变得几乎没电甚至关机，而对此却一无所知，甚至毫无办法。因此，控制应用程序的唤醒锁，能有效节省电量，在不干扰应用原本的服务的前提下，尽最大的可能提高电池续航能力。 |

* 1. 性能需求

性能要求

由于是Android的本地服务，所以，应用能够长时间在手机后台运行服务而不崩溃，拥有良好的稳定性，可靠性。

适应性

在不同的设备当中，包括手机，平板均能正常使用，在显示方面能够兼容不同设备。

* 1. 运行环境描述

硬件

适应Android系统的移动设备，拥有sd卡，内存512M以上，

软件

程序正式运行：Android 4.4以上的系统，

开发环境：Android studio

Android API 19

数据库：SQLlite

测试平台：Google nexus 5 Android 4.4 ，Google nexus 5 Android 5.0

开发语言：java

开发基础：Xposed框架

* 1. 其他需求

程序部分功能只针对获取了root权限的用户，或者这对基于原生安卓操作系统的系统。在Android手机上部分功能需要预先安装xposed框架。

* 1. 市场现状

如今市场对于Android手机的监控程序层出不穷，但大多功能不精，实现效果一般，有的甚至是指标不治本，而有的只是让用户安心，实则对于手机的管理并没有所说的那么好，也就是达不到用户要求。要么提供多余的，与系统提供了重复的功能。为此，列举了市场上常有的几款监控应用。

1. 猎豹管理大师

这款软件主要用于清理垃圾，软件管理，清理内存，安全隐私等方面，作为垃圾清理的软件，这款软件效果还可以，但是现如今系统一般都内置了垃圾清理的功能，与系统内置的功能冲撞了，再有就是软件管理，这也是系统自带了的功能，用户可以选择使用或者不使用，其次就是安全隐私方面，主要就是历史记录的清理，然而每款软件的历史记录都有自己的清理功能调用，所以，这款软件也只是聚集了一些功能，能够帮助管理手机的功能，但是实质上并没有帮助手机系统做到更加流畅，用户也没有很强烈的需求去使用。

1. 360手机助手

这款软件主要功能在手机杀毒，清理内存，骚扰拦截，软件管理等。对于手机杀毒，确实有一定的必要，但是这一项功能并不能优化手机体验。至于内存清理，只不过是暂时释放掉后台应用所占有的内存空间，应用并没有被关闭，可以重新获取内存空间，这就是为什么所谓的应用杀不死，清理了立刻就死灰复燃的样子，他的本质只是释放内存，并没有从根源上关闭应用。而骚扰拦截，这也是手机一般都会内置的功能，而且一般作用于打电话，发短信，于手机使用的体验没有太大关系。

1. LEB加速大师

这款软件的主要功能有清理内存，电池优化，等。同样的清理内存同其他软件一样，是治标不治本的，软件仍然驻留在后台，但是在此开启却像重新打开一样慢，并没有做到很好的优化，只是在那一小段时间增加了内存。至于电池优化，这是通过设置屏幕亮度，亮屏时间，蓝牙，WiFi，gps等设备的开关来达到省电的效果，而这一切用户都可以自行设置，而且有时候还会给用户带来不愉快的体验。

而关于流量的监控，市面上的程序也有一些，系统也可能自带了流量的管理，但是一般只有统计作用，这样不过形同虚设，好一点的话会在超出流量是提醒用户是否继续，但是用户一旦点击继续，则变得无法控制，系统不会再次提示，而是都默认为用户允许，而这时一般用户都会无视的一项设计，因为肯定会继续使用，而这时候就变得无法监控了。

总结了一下目前市面上其他普遍的管理Android手机的应用程序，一般为清理内存，清理垃圾，手机杀毒，电池优化等看上去很美，但是实现效果一般，用户可能使不使用该软件都不会有太大的体验差。

相比之下，我们设计的这款软件，DroidEye，功能主要集中于后台进程和服务的强行关闭，数据流量的每时每刻监控，都可以在用户设定下控制，而省电则是采用禁止无用唤醒cpu实现，不会影响用户体验的同时，大大提高省电能力。没有添加多余的，与系统重复的功能，而且这些功能都作为后台服务，无需用户常常打开设定，24小时保障用户手机的使用情况。

1. 功能介绍
2. 用户可以设定策略来管理后台的服务，或者打开应用立刻对某个应用做出操作，方便简单
3. 流量的监控程序会即时监控当前流量情况，用户不需担心在使用过程中超出预期，因为有后台的监控程序提醒，并要求用户重新设定限额来继续运行某应用使用数据流量。
4. 后台的应用唤醒得到监控，从而有效保证手机在锁屏期间得到真正的休眠，在提高手机续航能力的同时不影响用户体验。
5. 项目计划与进度

3.1 项目计划

Table 2

|  |  |
| --- | --- |
| 周数 | 计划 |
| 14周 | 完成程序初期有关功能的运行机制和管理机制，认识背后的原理  学习并了解Android系统的应用程序的运行，停止，暂停，管理等技术，  对于Android系统的设计，分层，函数的调用管理，一个应用的生命周期等做出正确的认识和学习。  对于安卓应用程序获取流量的方式，应用程序连接网络的请求，以及如何管理应用程序的流量开关等  学习安卓底层函数的调用关系和方式，学习Linux关于cpu的唤醒，休眠的底层的知识的学习。  学习Android有关权限的管理机制，认识每一个应用程序在获取权限后是如何管理和操作的。 |
| 15周 | 完成流量监控的实现，包括界面，后台服务，设置等   1. 统计当前安装应用的信息，获取每个应用是否拥有连接网络的权限 2. 统计每个应用程序的流量使用情况， 3. 监控实时的流量信息，获取当前正在使用流量的应用程序的信息 4. 根据用户的设置参数，设置后台服务监控监控当前流量的使用详情 5. 设计显示界面，有关的设置界面 6. 将界面与后台服务数据连接，做到能够实时改变设置数据 |
| 16周 | 完成后台应用程序的监控，包括界面，后台服务，设置等   1. 获取正在运行的应用程序及其相关信息 2. 区分运行程序中哪些是系统进程，哪些是用户程序 3. 统计每个应用程序的运行时长 4. 提高我们软件的权限：由于Android的设计师每个进程相当于一个用户，而用户之间是有隔离的，即所谓的沙箱机制，所以其他应用不能关闭除了自己以外的程序，故要提升权限，是之能成功关闭其他进程和服务 5. 在没有root的手机上提升权限：由于Android本身的设计因素，应用是不能随便提升权限的，如果没有root的话，而大多数用户没有获取root，因此，为了使更多的用户能使用我们的软件，我们利用系统签名的方式，将我们的程序强行提升为系统级别，因而有了更为高权限。 6. 根据用户的设置参数设置后台服务，即时关闭无用进程 7. 设计显示界面，有关的设置界面 8. 将界面与后台服务数据连接，做到能够实时改变设置数据 9. 在应用程序中能让用户即时关闭想关闭的应用 |
| 17周 | 完成应用程序对手机cpu的唤醒监控，包括界面，后台服务，设置等   1. 学习使用xposed框架，理解Android的native层和应用层的调用关系 2. 搭建xposed框架，包括开发和手机运行环境 3. 利用hook技术，将自己的监控代码注入到对系统函数唤醒cpu的函数，从而及时获取系统调用并采取对应策略，允许或者禁止。 4. 获取尝试唤醒cpu的应用程序信息 5. 统计每个尝试唤醒cpu的应用程序，并根据用户设置允许部分唤醒一提供服务 6. 将统计信息保存在本地，以便再次打开时获取 7. 设计显示界面，有关的设置界面 8. 将界面与后台服务数据连接，做到能够实时改变设置数据 9. 在应用程序中能让用户即时关闭想关闭的应用 |
| 18周 | 进行测试，包括模块测试，功能测试，接口测试，集成测试  完善相关文档，优化应用设计 |

3.2 实际进度

Table 3

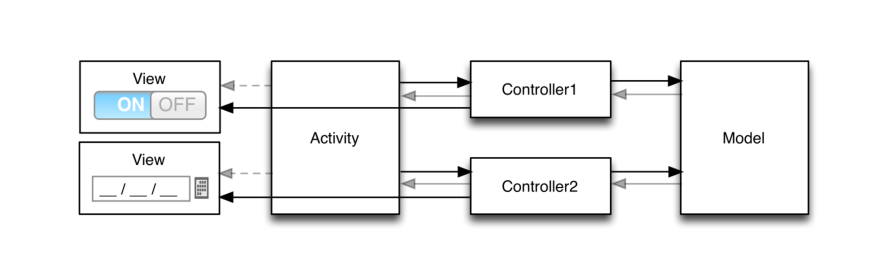
|  |  |
| --- | --- |
| 周数 | 执行情况 |
| 14周 | 通过google官方文档，以及Android开发文档，以及Linux调用机制的有关论文，加上Google学术和xda安卓开发社区有关流量，应用，唤醒，权限等关键字的文章，较为深入的学习了Android的相关技术，为后期的开发做出了一定的基础 |
| 15周 | 首先设计好了界面，采用列表的形式显示，流量的统计也已经实现，后台服务已经搭建好，将该服务作为后台服务的其中一个功能， |
| 16周 | 基本完成预定目标，程序成功在root和非root环境提权，并且能有效杀死进程，构建好了后台服务，界面也都设计完成 |
| 17周 | 在手机成功搭建起xposed框架和Androidstudio的xposed开发框架，成功利用hook技术将自己写的代码注入到相关系统函数调用的前面和后面，顺利监控了相关的唤醒调用。 |
| 18周 | 着手写文档，基本通过了功能测试，系统的相关代码也都得到了一定的优化。 |

1. 软件架构

4.1 软件框架

软件架构：

采用passive mvc架构，即为：



架构说明：

Activity：负责跟用户交互，接受用户输入，接受用户操作，创建View和Controller，并将View作为参数初始化Controller

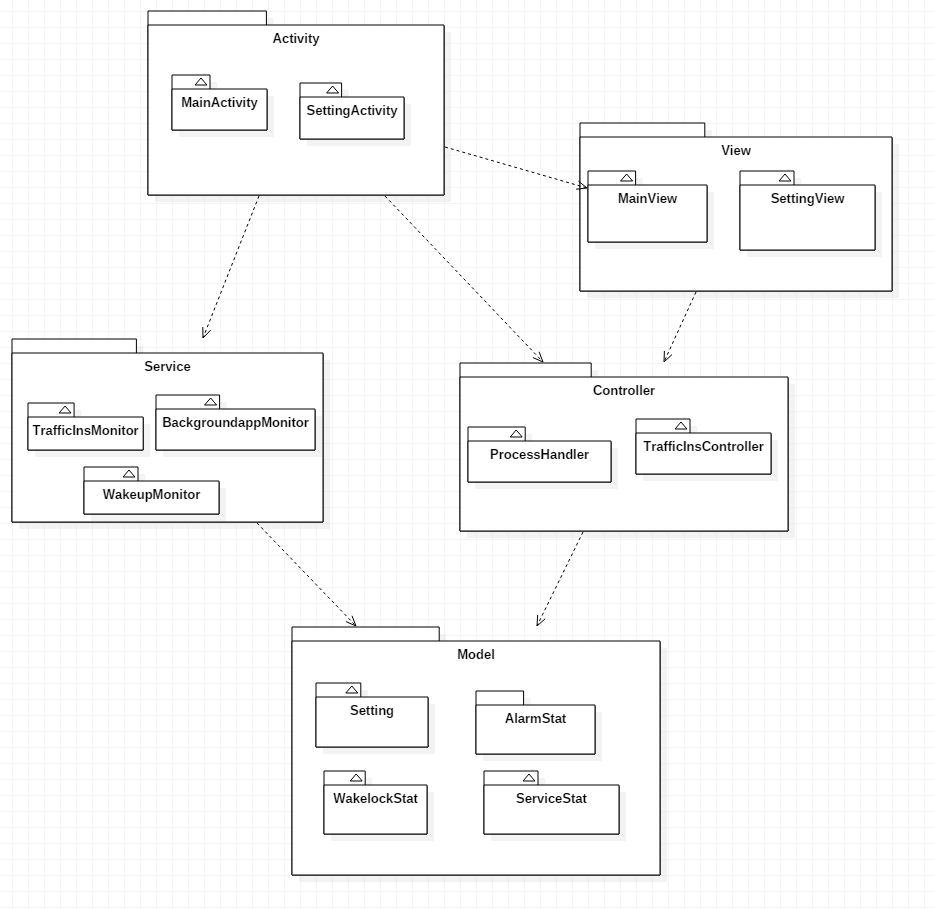
View：负责管理界面，包括每一个控件，提供么一个控件的设置值和获取值的方法，以及刷新界面的方法

Controller：负责控制View的显示数据，创建并拥有数据层Model的实例，将model的数据在View层显示出来

Model：负责管理数据，包括程序自定义的各种数据类型，提供获取每种数据的方法和设置方法

逻辑视图：

由于每个分层的类较多，此处为了清楚显示逻辑结构，只显示关键模块，具体在分层能够查看

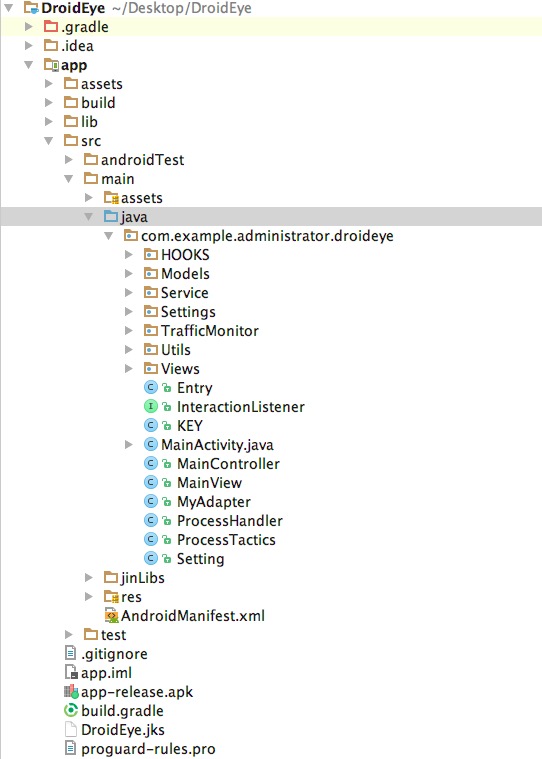


分层：

类图：

代码结构：

1.项目工程文件结构图：



1. 工程模块分解图



4.2 模块设计（类的设计）

4.2.1 流量监控

1、APPTrafficMonitor类

Table 4

|  |  |
| --- | --- |
| **类名** |  |
| **说明** |  |
| **方法名** | **功能说明** |
| public AppTrafficMonitor(TrafficInsListener listener) | 公共初始化构造函数，用于传递listener参数，为后续工作传递重要的Context变量。 |
| public List<HashMap<String,Object>> showApps() | 遍历当前手机系统中已存在的系统软件、应用软件，获取他们的Icon，程序名，UID，包名等基本信息。 |
| public static List<PackageInfo> requireNetPackages(PackageManager packageManager) | 遍历并返回一个列表，包含了当前系统中所有使用网络的App。 |
| public static long getTrafficIn(int uId) | 获取入口流量。 |
| public static long getTrafficOut(int uId) | 获取出口流量。 |
| public static int getUidFromInfo(PackageInfo info) | 获取APP-UID。 |
| public String formatprintTraffic(long traffic) | 格式化输出流量字符串。 |
| public String staticTraffic(PackageInfo appinfo) | 流量统计工具。 |

2、TrafficDetailActivity类

Table 5

|  |  |
| --- | --- |
| **类名** |  |
| **说明** |  |
| **方法名** | **功能说明** |
| public void onCreate(Bundle savedInstanceState) | 本类主要功能非常单一明确，即以一个独立的Activity展示各个app在当前系统环境下流量使用情况的统计值。 |

3、TrafficINSController类

Table 6

|  |  |
| --- | --- |
| **类名** |  |
| **说明** |  |
| **方法名** | **功能说明** |
| public TrafficInsController(TrafficInsView view , TrafficInsListener listener) | 公共构造函数，传入2个关键的构造参数，便于下一步工作的进行。 |
| public void initAdapter() | 初始化适配器 |
| public void initsValueOnView() | 初始化接口监听适配器 |
| public void onClick(View v) | 默认继承函数，在此处实现动画效果。 |
| public void ShowTrafficDetail(int position) | 与TrafficDetail类在此衔接。 |

4.2.2 后台应用程序监控

1、ProcessHandler类

Table 7

|  |  |
| --- | --- |
| **类名** |  |
| **说明** |  |
| **方法名** | **功能说明** |
| **public static** ProcessHandler getInstance() | 获取进程处理对象的一个实例，此类为一个单例类，有MainActivity初始化，其他对象可以通过该函数获取实例并调用方法 |
| **public static void** init(Context context) | 用来初始化ProcessHandler这个实例，有MainActivity传入Context环境变量来初始化 |
| **private** ProcessHandler(Context context) | 私有的初始化函数，从静态init函数初始化，确保单例的唯一性 |
| **public** Drawable getAppIcon | 获取进程对应的图标 |
| **public** List<Map<String,Object>> getInstalledAppWithKillingPermission | 获取所有安装的应用程序是否允许后台杀死了一个属性列表 |
| **private** Map<String,Object> generateMapItem(Drawable drawable,String name,**boolean** switch1,String processname) | 生成每一个应用程序是否允许后台杀死的一个项，包括：图标，进程名，应用名，是否允许值。 |
| **public** List<ApplicationInfo> getInstalledApps() | 获取Android手机里面所有安装了的应用程序 |
| **public void** forceStopProcess(String packagename) | 强行关闭应用程序 |
| **public** List<ActivityManager.RunningServiceInfo> getRunningServices() | 获取正在运行的服务 |
| **public long** getServiceUsedTime(ActivityManager.RunningServiceInfo service) | 获取服务已经使用时间 |
| **public** List<HashMap<String,Object>> getRunningApplications(**boolean** includeSystem) | 获取正在运行的应用程序 |
| **public** HashMap<String,UsedRecord> getAppUsedRecords() | 获取应用程序使用记录，主要提供给后台服务使用，不包括应用的图标，只有应用名，进程名，使用时长 |
| **public** HashMap<String,SimpleProcess> getRunningSimpleProcess() | 在命令行获取正在运行的应用程序，作为getRunningApplication的一个内置需求 |
| **public** ApplicationInfo getSpecifyAppInfo(String packageName) | 根据进程名，提供完整的应用程序的相关信息 |
| **public boolean** isInSystem(ApplicationInfo app) | 判断应用程序是否在系统内部 |
| **public** Long getAppBackgroundTime(String packageName) | 获取应用程序在后台的存活时间，此函数主要针对没有服务后台的应用进程 |
| **public void** getUsageStatsPermission() | 获取查看其他应用程序信息的权限 |
| **private boolean** hasUsageStatsOption() | 检查手机是否有设置允许查看其他应用的设置界面 |
| **private boolean** hasOpenUsageStats() | 检查当前应用有没有打开允许程序查看其他应用设置的开关 |
| **public class** UsedRecord **implements** Comparable<UsedRecord> | 作为该类的子类，主要记录应用程序使用记录，包括应用名，使用时间 |
| **class** SimpleProcess | 该类的子类，主要记录了从命令行获取到的进程，包括用户，进程名，pid，占用内存大小，是否在系统内部 |

2、MyAdapter类

Table 8

|  |  |
| --- | --- |
| **类名** |  |
| **说明** |  |
| **方法名** | **功能说明** |
| **public** MyAdapter( | 传入环境变量，资源文件的id，要显示的数据列表，以及数据跟控件的对应关系，从而生成一个adapter来在listview来显示有关数据 |
| **public int** getCount() | 重写BaseAdapter的方法，返回数据列表的大小 |
| **public** Object getItem(**int** position) | 重写BaseAdapter的方法，返回指定位置的一个数据块 |
| **public long** getItemId(**int** position) | 获取某个模块的id号，此处没有用到，但是继承BaseAdapter必须重写的方法 |
| **public** View getView(**final int** position, View convertView, ViewGroup parent) | 这个是Adapter最为重要的一个函数，负责显示每一项的显示逻辑，程序在内部指定对应数据块的值显示在指定的控件上，最后返回一个view。在此处，我们的程序还在里面对每一个项的view绑定了监听事件，从而实现点击用户操作的监听与相应的反应 |

4.2.3 后台唤醒

1、Wakelocks类

Table 9

|  |  |
| --- | --- |
| **类名** | Wakelocks |
| **说明** |  |
| **方法名** | **功能说明** |
| **private void** resetFilesIfNeeded(Context context) | 重新设计记录文件，此函数用来丢弃过时数据的记录文件，以减少手机内存的占用和rom空间的节省 |
| **private void** hookAmplifyClasses(LoadPackageParam lpparam) | 这个函数用来记录该类的调用情况以及一些基本信息。 |
| **private void** setupReceiver(XC\_MethodHook.MethodHookParam param) | 响应每一种监听函数的调用情况，来即时发出action来记录响应的数据文件 |
| **private void** hookAlarms(LoadPackageParam lpparam) | 监控对手机alarm的调用情况，其中针对每一种sdk编译版本有不同的实现方式 |
| **private void** hookWakeLocks(LoadPackageParam lpparam) | 监控唤醒锁的调用，同样也是针对每一种编译sdk版本的不同有不同的实现方式 |
| **private void** hookServices(LoadPackageParam lpparam) | 监控服务调用情况，同样是针对每一种sdk编译版本的不同而又不同的实现方式 |
| **private void** try23ServiceHook(LoadPackageParam lpparam) | 尝试对sdk版本为23的服务调用监控 |
| **private void** try17To20ServiceHook(LoadPackageParam lpparam) | 尝试对sdk版本为20的服务调用监控 |
| **private void** try14To16ServiceHook(LoadPackageParam lpparam) | 尝试对sdk版本为14到16的服务调用监控 |
| **private void** try21WakeLockHook(LoadPackageParam lpparam) | 尝试对sdk版本为21 的唤醒锁的调用监控 |
| **private void** try19To20WakeLockHook(LoadPackageParam lpparam) | 尝试对sdk版本为19到20的唤醒锁获取的调用的监控 |
| **private void** try17To18WakeLockHook(LoadPackageParam lpparam) | 尝试对sdk版本为17到18的唤醒锁获取的调用的监控 |
| **private void** try19To21AlarmHook(LoadPackageParam lpparam) | 尝试对sdk版本为19到21的alarm的调用的监控 |
| **private void** try15To16WakeLockHook(LoadPackageParam lpparam) | 尝试对sdk版本为15到16的唤醒锁的调用监控 |
| **private void** handleServiceStart(XC\_MethodHook.MethodHookParam param, Intent serviceIntent) | 处理一个服务的开始，以便后面的一个服务使用时间的统计做出计算结论 |
| **private void** handleWakeLockAcquire(XC\_MethodHook.MethodHookParam param, String wakeLockName, IBinder lock, **int** uId) | 处理当唤醒锁被获取是的操作，包括允许或者禁止，这依赖于用户的设置 |
| **private void** recordServiceStart(XC\_MethodHook.MethodHookParam param, String serviceName, **int** uId) | 记录一个服务的开始 |
| **private void** recordAcquire(String wakeLockName, IBinder lock, **int** uId) | 记录获取，获取一个应用程序的调用情况 |
| **private void** recordAlarmAcquire(Context context, String alarmName, String packageName) | 记录Alarm的获取情况 |
| **private void** handleWakeLockRelease(XC\_MethodHook.MethodHookParam param, IBinder lock) | 处理唤醒锁的释放，包括释放唤醒锁，处理唤醒锁的持有时间等 |
| **private void** updateStatsIfNeeded(Context context) | 更新所有记录的情况，用于用户的刷新服务 |
| **private void** handleAlarm(XC\_MethodHook.MethodHookParam param, ArrayList<Object> triggers) | 处理Alarm的获取情况，包括允许或者禁止等 |
| **private void** recordWakelockBlock(XC\_MethodHook.MethodHookParam param, String name, **int** uId) | 记录唤醒锁的禁止情况 |
| **private void** recordServiceBlock(XC\_MethodHook.MethodHookParam param, String name, **int** uId) | 记录服务被调用的禁止情况 |
| **private void** recordAlarmBlock(XC\_MethodHook.MethodHookParam param, String name, String packageName) | 记录Alarm被禁止的情况 |
| **public void** handleLoadPackage(LoadPackageParam loadPackageParam) **throws** Throwable | 这是一整个类的入口，所有的服务的设置的开始都是从这里开始，实现了xposed框架的调用服务，处理当应用程序打开时开始所有被监控的应用的情况都从这里开始调用 |

2、UnbounceStatsCollection类

Table 10

|  |  |
| --- | --- |
| **类名** |  |
| **说明** |  |
| **方法名** | **功能说明** |
| **public static** UnbounceStatsCollection getInstance() | 由于该类为单例类，故有此静态方法来获取单例 |
| **public static** String getFormattedTime(java.lang.Object... paramTimes) | 获取格式化输出的时间 |
| **public** ArrayList<BaseStats> toAlarmArrayList(Context context) | 获取Alarm的记录情况的列表，将记录提取出来并将每个应用的记录防止一个记录类里，在这里获取从而被View层的函数成功获取数据 |
| **public** ArrayList<BaseStats> toWakelockArrayList(Context context) | 类似于上一个函数，只不过这次只是对唤醒锁的调用情况的记录 |
| **public** ArrayList<BaseStats> toServiceArrayList(Context context) | 类似于上一个函数，只不过这次只是对服务的调用情况的记录 |
| **public** String getWakelockDurationAllowedFormatted(Context context, **int** statType) | 获取唤醒锁持续时间的格式化输出函数，也是用来对View层提供数据的显示 |
| **public** WakelockStats getWakelockStats(Context context, String wakelockName) | 获取某个单独的唤醒锁的调用的详细情况 |
| **public** ServiceStats getServiceStats(Context context, String serviceName) | 获取某个单独的服务的调用的详细情况 |
| **public** AlarmStats getAlarmStats(Context context, String alarmName) | 获取某个单独的Alarm的调用的详细情况 |
| **public long** getTotalAllowedWakelockCount(Context context, **int** statType) | 获取总共的允许唤醒锁被调用的次数 |
| **public long** getTotalAllowedServiceCount(Context context, **int** statType) | 获取总共的服务允许被调用的次数 |
| **public long** getTotalBlockWakelockCount(Context context, **int** statType) | 获取唤醒锁被禁止的次数 |
| **public long** getTotalBlockServiceCount(Context context, **int** statType) | 获取服务被禁止的次数 |
| **public long** getTotalAllowedAlarmCount(Context context, **int** statType) | 获取Alarm总共允许调用的次数 |
| **public long** getTotalBlockAlarmCount(Context context, **int** statType) | 获取总共Alarm被禁止调用的次数 |
| **public void** createFiles(Context context) | 创建记录文件的函数 |
| **public void** refreshPrefs(XSharedPreferences prefs) | 刷新用户设置的函数 |
| **private** HashMap loadStatsFromFile(String statFilename) | 从文件中读取调用记录 |
| **private boolean** saveStatsToFile(String statFilename, HashMap statChoice) | 保存记录数据值文件，以便在应用再次打开时能够接着以前的数据显示 |

3、AlarmStats类

Table 11

|  |  |
| --- | --- |
| **类名** |  |
| **说明** |  |
| **方法名** | **功能说明** |
| **public** String getDerivedPackageName(Context ctx) | 由于此类继承自BaseStats，所以实现了其中的抽象方法，返回独特的关于Alarm的调用起源包名 |

4、ServiceStats类

Table 12

|  |  |
| --- | --- |
| **类名** |  |
| **说明** |  |
| **方法名** | **功能说明** |
| **public** String getDerivedPackageName(Context ctx) | 由于此类继承自BaseStats，所以实现了其中的抽象方法，返回独特的关于Service的调用起源包名 |

5、WakelockStats类

Table 13

|  |  |
| --- | --- |
| **类名** |  |
| **说明** |  |
| **方法名** | **功能说明** |
| **public** String getDerivedPackageName(Context ctx) | 由于此类继承自BaseStats，所以实现了其中的抽象方法，返回独特的关于唤醒锁的调用起源包名 |

6、BaseStats类

Table 14

|  |  |
| --- | --- |
| **类名** |  |
| **说明** |  |
| **方法名** | **功能说明** |
| **public void** setName(String name) | 设置该状态记录的标识符 |
| **public boolean** getBlockingEnabled() | 获取是否被禁止 |
| **public long** getAllowedCount() | 获取调用的允许次数 |
| **public void** setAllowedCount(**long** count) | 设置调用的允许次数 |
| **public long** incrementAllowedCount() | 增加每次调用后的计数 |
| **public long** getBlockCount() | 获取调用的禁止次数 |
| **public void** setBlockCount(**long** count) | 设置禁止调用的次数 |
| **public long** incrementBlockCount() | 增加调用禁止后的加数操作 |
| **public int** getUid() | 获取该调用记录对应应用程序的uid |
| **public void** setUid(**int** uid) | 设置该应用程序的uid |
| **public abstract** String getDerivedPackageName(Context ctx) | 抽象方法，由各自的子类去实现 |
| **protected void** setDerivedPackageName(String mDerivedPackageName) | 设置该调用方法发自那个应用程序包 |

4.2.4 后台服务

1、MonitorService类

Table 15

|  |  |
| --- | --- |
| **类名** |  |
| **说明** |  |
| **方法名** | **功能说明** |
| public int onStartCommand(Intent intent, int flags, int startId) | Android Service默认启动方法之一，在每次onCreate()方法被调用后会紧接着调用此方法，作为自定义程序的起点。 |
| sendBroadcast() | 以BroadCast-Receiver形式向对应程序发送Alarm. |
| Private void Mission1() | 调用任务一，作为service的任务之一。 |
| Public void showNotice() | 使用Android Notification ， 将需要的消息以公告板的方式推送给用户。实现了上述方式。 |

2、StartUpReceiver类

Table 16

|  |  |
| --- | --- |
| **类名** |  |
| **说明** |  |
| **方法名** | **功能说明** |
| public StartUpReceiver() | Android BroadCast-Receiver的默认构造函数。 |
| public void onReceive(Context context, Intent intent) | 作出当收到消息时所需的响应动作。 |

4.2.5 设置属性

1、SettingsActivity类

Table 17

|  |  |
| --- | --- |
| **类名** |  |
| **说明** |  |
| **方法名** | **功能说明** |
| private MyAdapter generateMyAdapter() | 通过generate MyAdapter型的适配器，与项目的其他部分建立了联系。 |
| private SimpleAdapter generateAdapter(boolean includeSystem) | 产生一个Simple Adapter. |

2、SettingsAdapter类

Table 18

|  |  |
| --- | --- |
| **类名** |  |
| **说明** |  |
| **方法名** | **功能说明** |
| public SettingsAdapter(Context context) | 继承自BaseAdapter, 首先通过构造函数初始化List各段的头部。 |
| public void addSection(String section, Adapter adapter) | 对应每一段头部，添加一个section到里面去。 |
| 其余方法皆为BaseAdapter重载方法。 | 实现BaseAdapter的基本功能并在其基础上作出改进。 |

3、SettingsController类

Table 19

|  |  |
| --- | --- |
| **类名** |  |
| **说明** |  |
| **方法名** | **功能说明** |
| public void initsValueOnView() | 查看并管理运行进程项。 |
| 其余方法皆为OnItemClickListener重载方法。 | 实现基本的界面监听和相关功能。 |

4、SettingsView类

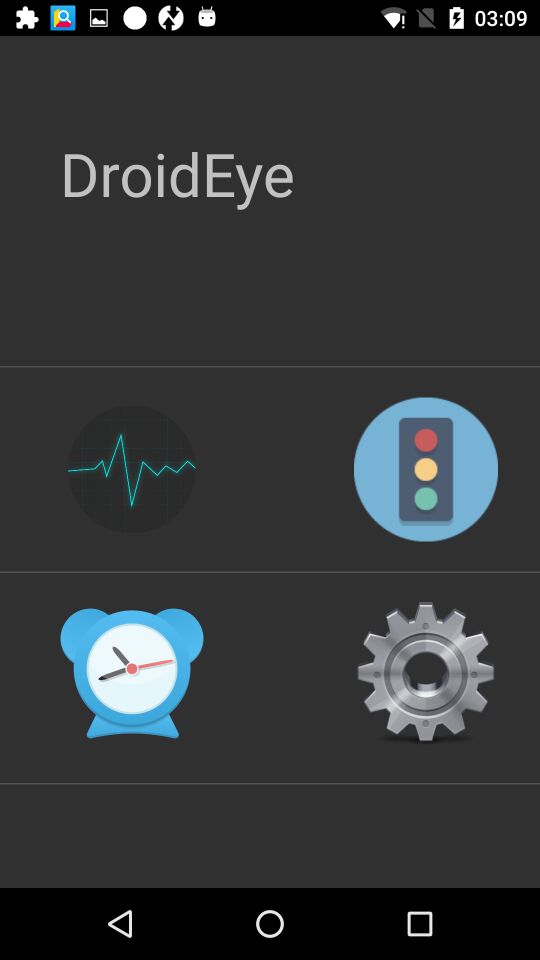
Table 20

|  |  |
| --- | --- |
| **类名** |  |
| **说明** |  |
| **方法名** | **功能说明** |
| public SettingsView(Context context, AttributeSet attributeSet) | 继承Relative Layout的一个相对布局的构造函数。 |
| public void init() | 初始化函数。 |
| 其他皆为设置监听器相关函数 | 确保监听器正确执行。 |

4.3 资源列表

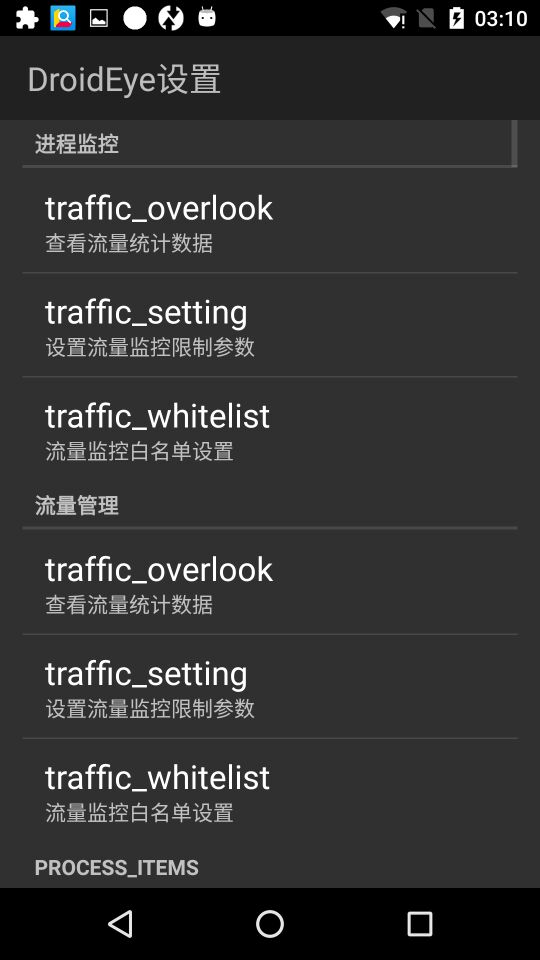
1. 系统实现

系统运行时效果截图效果：

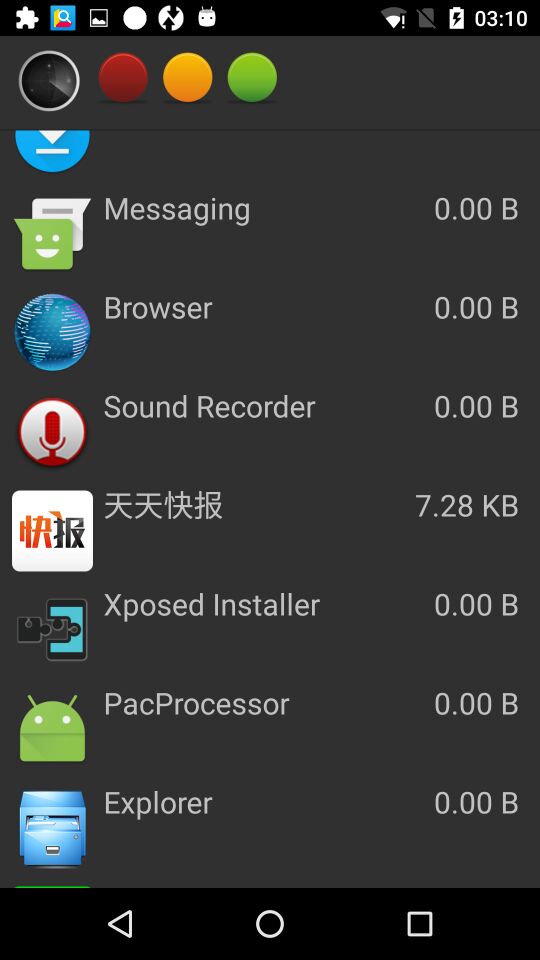


*主界面进入效果（简约风格）*

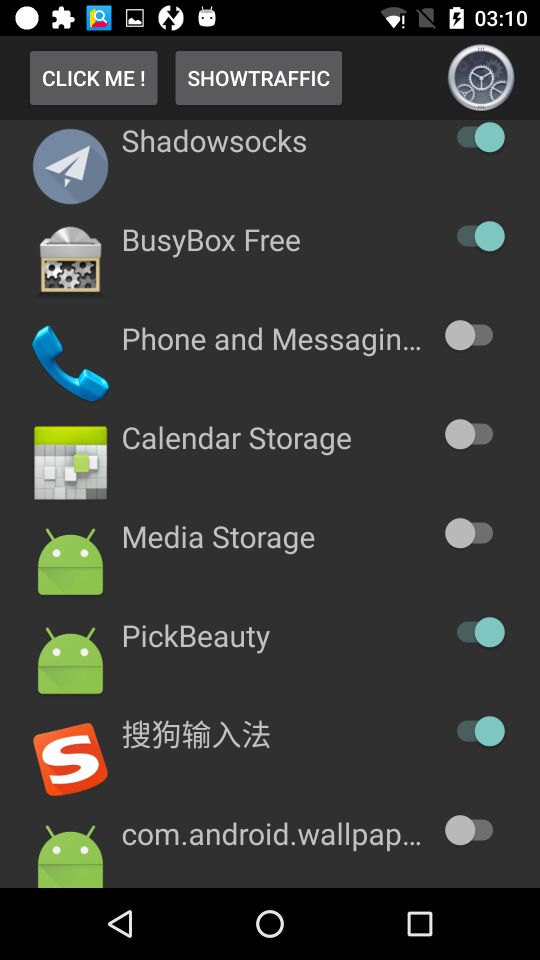
*系统设置界面*



*流量监管控制模块*



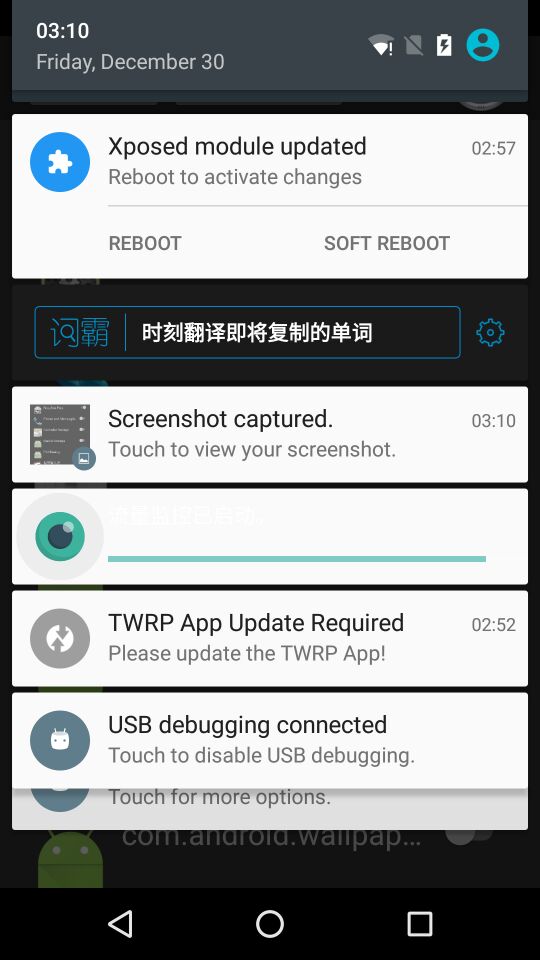
进程权限管理界面



流量细节界面



通知栏提醒界面



软件设置示意图



1. 系统测试说明

指导软件测试并给出结果表以填写测试结果

6.1 测试项目及说明

测试方案

Table 21

|  |  |
| --- | --- |
| 测试内容 | 测试时间 |
| 功能测试1：测试流量监控模块 | 16周-18周 |
| 功能测试2：测试后台程序关闭模块 | 17周-18周 |
| 功能测试3：测试应用程序唤醒cpu监控模块 | 18周-18周 |
| 性能测试1：测试流量数据是否在5s内实时刷新 | 16周-18周 |
| 性能测试2：测试显示数据列表是否在1s内全部加载 | 18周-18周 |

测试项目

Table 22

|  |  |  |  |
| --- | --- | --- | --- |
| 测试名称 | 测试内容 | 达标要求 | 目的 |
| 测试流量监控模块 | 1. 获取流量数据 2. 截断数据连接 3. 达到一定量时触发 4. 刷新时间 | 能够获取每个应用的流量数据，同时能够按应用切断数据连接，超过设定流量则触发切断数据连接，刷新在5s内实现 | 即时，定量的按照用户要求操作，防止流量超出预期使用户遭受可能的金钱损失 |
| 测试后台应用程序关闭模块 | 1. 强行关闭后台进程 2. 超过预定存活时间立即杀死进程 3. 测试白名单进程存活状态 | 后台进程能够被完全关闭，当到达时限则能杀死进程，在白名单的应用程序不会是被杀死 | 定时杀死无用进程，为用户节省资源，防止用户无禁止地浪费资源导致手机使用卡顿 |
| 测试应用程序唤醒cpu监控模块 | 1. 当有应用程序唤醒cpu时触发监控 2. 成功避免不该唤醒的时机 | 成功检测唤醒cpu，并能够有效拦截，获取统计数据 | 使用户在不用手机是保存电量，节省资源损耗 |

6.2 测试结果

6.3 评价

1. 用户手册

7.1 系统说明

运行环境：在Android 4.4 以上版本运行，流量监控都可成功运行，后台关闭程序需在Android原生系统或获取了root权限的系统运行，应用程序自唤醒监控需要手机安装xposed框架

7.2 系统安装

直接使用apk安装包安装，可运行在安卓真机或虚拟机中

7.3 常见问题处理

应用程序无法关闭：系统不是安卓原生，经过厂商修改，而且手机没有root权限，或者没有授予该应用程序权限

检测不到应用程序唤醒：手机没有安装xposed框架，或者没有在xposed运行该应用程序，或者没有root导致安装xposed失败

1. 附录

8.1 小组分工

Table 23

|  |  |
| --- | --- |
| 姓名 | 职责 |
| 钟玉生 | 负责后台应用程序关闭模块，和应用程序唤醒cpu监控模块，以及负责替换应用程序签名为系统签名，获取root权限， |
| 顾芳铭 | 负责界面的设计，实现，和流量监控模块的设计和实现，后台服务等，引导安装xposed模块等 |

8.2 版本记录

8.3 设计心得

Table 24

|  |
| --- |
| From姓名：钟玉生  这次的Android项目是最让我记忆深刻的一刻，与之前的设计相比，这次的项目功能显得略微单调，但是实现起来却远比我之前实现的功能难，以前的都是单纯的调用API，只是一些功能的堆叠，并没有深入到应用程序的运行流程，甚至是底层的调用。而这次的这个项目就是抛开外在的功能，让我深入的了解到Android，Linux，root，user，Java，native这些名词所代表的内在含义。  不仅如此，进过这次的项目经历，让我想起那样一种说法，功夫不在多，而在精。要是想真正做出一个能让用户使用，能让用户相信，能让用户依赖的功能就已经非常难了。而不是随便的功能的实现。借助这次的项目，我也更加发现自己对Android开发知识的认识是那么有限，还有很多问题等着我去探索，等着我去学习。  而在技术上面，我就更加发现，优化代码是多么的重要，了解运行环境 是多么的重要，我们的项目在开始的版本中，运行起来比较耗内存，但是功能不多，只是显示的项有点多，而这就耗了比较多大内存，后来在显示上面做了优化，才使得较之前有所好转。  这次的锻炼让我更加熟悉Android，让我坚信在Android开发的路上越走越远是可行的。 |
| From姓名：顾芳铭  通过本次安卓项目实践，发现了安卓系统非常多有趣的地方。之前的了解都偏于高级的应用和功能，如调用各种框架使用各种工具类去实现自己的编程目的，但是很少有机会能够亲眼看到和做一些针对安卓系统底层结构的实验，借助这次大作业的机会，我有幸一睹安卓系统底层的架构，同时，也非常锻炼自己的编程能力，由于时间紧任务重，我在承受着较大的项目压力的同时也不断地激励着自己提高自己的编程水平，以适应复杂项目的需要，同时，为了努力赶上项目进度也做了许多付出，我相信这些付出是值得的，因为它们的确教会了我许多。安卓是移动领域发展最为迅速的系统，我对它有较为浓厚的兴趣，这是一个C/C++和Java语言以前所未有的方式错综复杂的糅合在一起的世界，对于不了解它的人来说，这样一个编程环境可以是让人很容易产生受挫感的，这也是我努力涉足系统底层的理由，正所谓知己知彼者，百战不殆。  希望以后能有更多机会参与到安卓项目的实践当中，不断提升自己的编程水平的同时更加深对安卓这样一个系统的理解。 |