



Copyright@2017

征战手机

智能终端期末课程设计

小组成员	学号	联系电话	邮箱
杨明亮	3140104873	17816862322	453055051@qq.com
董泰佑	3140104431	17816878432	1349498382@qq.com

目录

一、项目概述.....	4
二、UI 设计.....	5
三、功能划分.....	6
1.应用详细使用信息统计	6
2.应用统计图谱绘制	6
3.应用管理	7
4.登陆注册	7
四、各功能模块设计	8
1.启动页及引导页	8
2.主模块及菜单	9
3.应用详细使用信息统计	12
4.应用信息图谱绘制	12
5.应用管理	14
五、代码架构.....	17
1.启动及引导页活动	18
2.主菜单栏模块	18
3.应用信息的使用情况及获取	20
4.应用信息列表展示	24
5.应用信息图表展示	26
6.应用管理列表	32
7.IO 模块	35

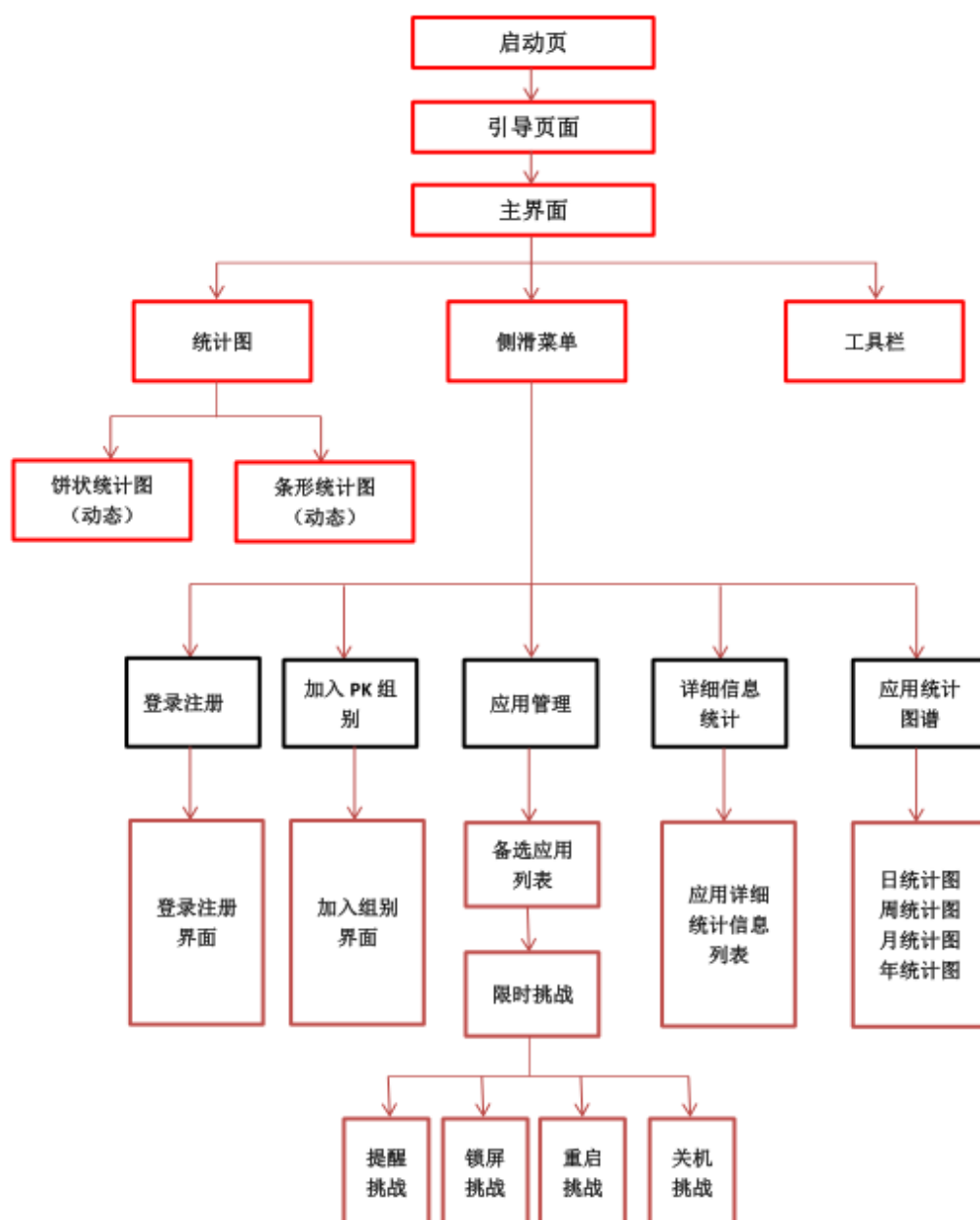
8.Notice_Toast 模块	36
9.设备管理模块	38
六、服务器部署	43
七、工程种遇到的问题.....	45
八、完成度	49
项目分工：	50
难点参考网站：	51

一、项目概述

在当今社会中，手机已经成为了我们不可缺少的工具，我们每天也将大量的时间花费在移动设备的使用中。不知不觉中我们似乎已经离不开这个满足我们需求的工具，但是慢慢更多的人意识到我们应该做手机的主人而不是被手机“奴役”。针对这个现象，我们小组决定制定一款 APP 帮助人们限制手机的使用，帮助手机使用者统计并限制手机应用的使用，甚至可以自行设置手机和 APP 的使用时间。

二、UI 设计

本产品的 UI 设计全部采用安卓原生组件，未采用第三方库或第三方代码，所有设计均为原创。UI 的架构如下：点击应用图标，弹出启动页面显示征战手机的 Logo。滑动三张引导页，进入主菜单。整体的 UI 架构可以用如下图表示：



三、功能划分

除去登录显示界面和引导界面外，我们将主要功能模块分为 4 部分，分别是应用详细信息统计、应用统计图谱绘制、应用管理和登录注册。其中前三个模块是本应用的核心功能模块。

1.应用详细信息统计

应用详细信息列表主要是将手机内每个应用的信息读取出来，列出的信息有应用图标、应用名称、使用时间和操作次数。同时为用户提供了 4 个按钮：DAY、WEEK、MONTH、YEAR。分别用来显示用户在当天、近一周、一月和一年的应用的使用情况列表。在每个列表的顶部会显示出用户在相应时间内的总的使用时间和操作次数。统计的时间精确到秒。同时统计的次数为用户对一款应用的操作次数，即每当用户在一个应用中打开或切换一个 activity，则这个应用的相应的操作次数就增加 1。

2.应用统计图谱绘制

应用信息图谱的绘制是将用户的应用使用情况通过图表的形式表现出来，主要是通过扇形图的模式。扇形图会标出显示应用的名称和应用使用时间所占的相应的比重，用户可以直观的得到其每个应用占手机使用时间的比例。在列出应用时，会选取占比重较大的 6 个应用进行显示，剩下的应用将其归类到“其他应用”模块中。在扇形图的上方会显示出手机的使用时间。

同时为用户提供了 4 个按钮：DAY、WEEK、MONTH、YEAR。分别用来显示用户在当天、一周内、一月和一年的使用情况。用户可以根据自己的喜好对扇形图进行旋转

操作。

3.应用管理

应用管理模块主要提供对应用使用情况的设定和限制功能。对应用的限定主要分为 4 个方式：通知栏提醒，锁屏，重启和关机。用户通过设定每个应用的限制时间和使用限制方式来达到限制应用使用的目的。例如设定 QQ 的限制使用时间为 2 个小时，限制方式为通知栏提醒，则当 APP 检测到用户 QQ 的使用时间超过 2 个小时后，则会在通知栏发布一个通知告诉用户 QQ 的使用情况超过时间设定。

应用管理模块会列出每个应用的图标，名称和用户相应的设定方式。用户通过点击每个应用进入相应的设定页面。设定页面会要求用户选择设定的类型和设定的时间或者取消对改应用的设定。设定完成后应用的设定情况会在应用管理列表中显示。

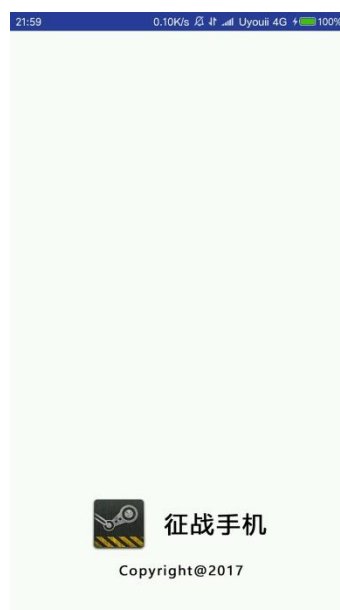
4.登陆注册

用户可以通过邮箱进行注册登录，登陆后 ID 默认为邮箱地址（同一邮箱只能注册一个账户）。用户登录后在主菜单中显示对应账号信息，同时可以进行账户间切换，重新登录其他账户。

四、各功能模块设计

1.启动页及引导页

点击 APP 图标启动 APP 后进入启动界面，根据启动情况自行显示本 APP 设计的 logo：



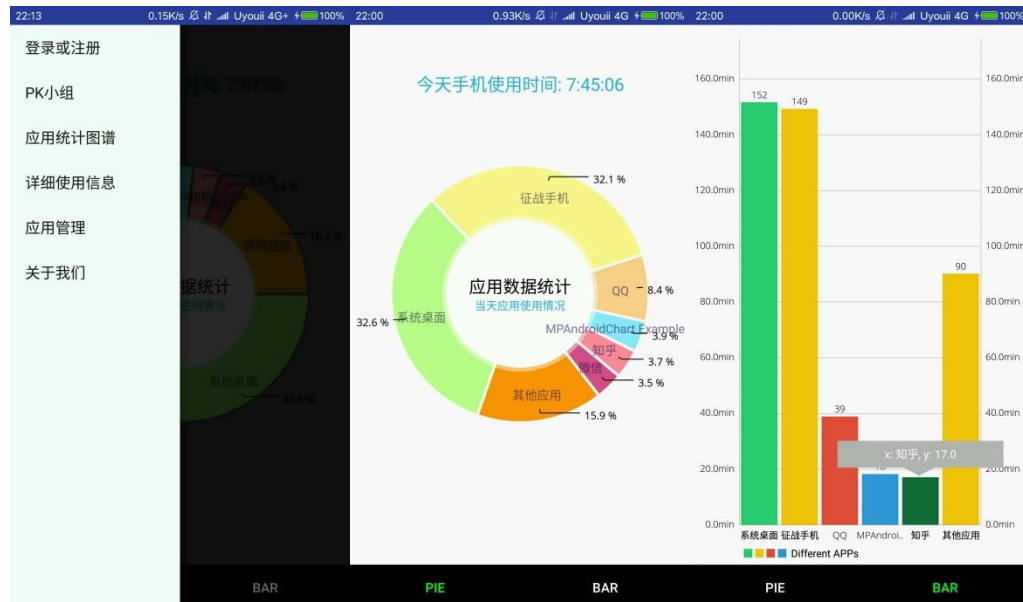
启动页面显示 3 秒钟，进入引导页面，轮播三张引导页：



2.主模块及菜单

主菜单由三部分组成,分别是侧滑菜单,toolbar 和主界面(统计图表),实现效果图

如下:

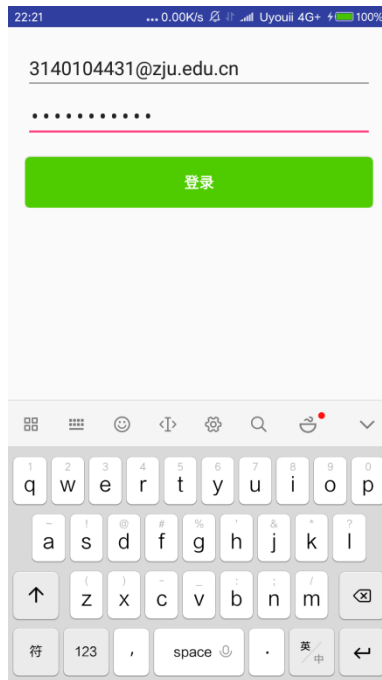


主菜单我们选择使用两个统计图来填充,这两个动态的统计图之间可以通过下边的按钮转换,统计图统计的是今天手机所有应用的使用情况。每次加载统计图都会动态显示,条形统计图还可以进行缩放。

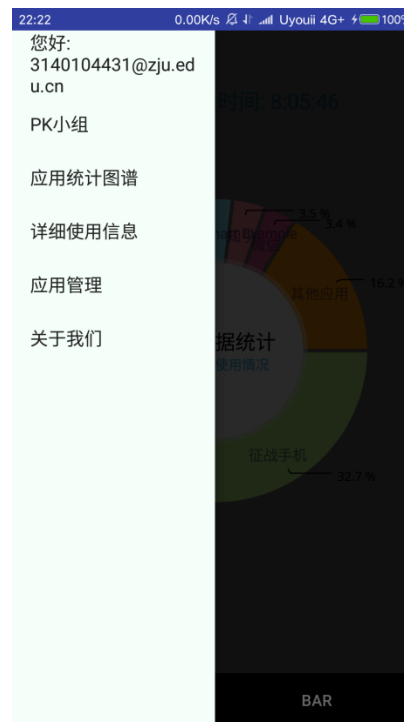
其次是侧滑菜单的设计,我们设计了侧滑菜单作为主要功能的菜单,包括登录注册功能、加入 PK 小组功能、应用统计图谱功能、应用详细信息统计功能和应用管理功能。

(1) 登录功能

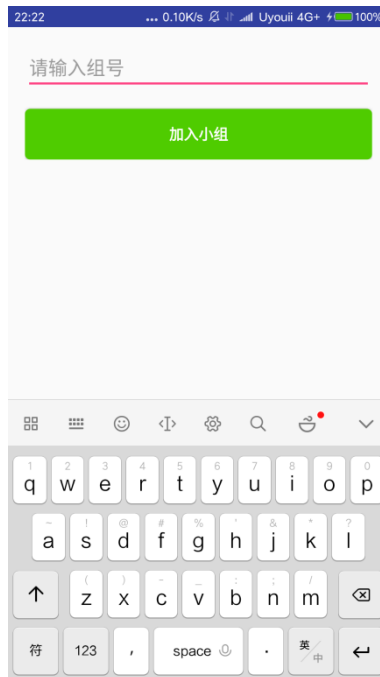
登录功能界面比较简单,就是输入已注册的(服务器数据库中已有的)用户账号和密码进行登录,这里账号我们限制只能使用邮箱,即用户的账号 id 即用户邮箱,不需要提供其他用户 id。



同时在界面上我们也做了优化，即当提示某用户成功登录的时候，登录一栏将会被更新的提示信息取代，变成：“您好，某某某”。登录前后对比图如下：



(2) 加入 PK 小组



加入 PK 小组功能是为了更好的在线互动。点击加入 PK 小组，输入 0-1000 的任意数字即可创建或加入 PK 小组。加入 PK 小组后组内的人可以进行互相监督，更好的限制了手机的使用。同时加入 PK 小组后提示信息会自动从服务器更改，但是我们要求每个人最多只能加入一个 PK 小组，即加入 PK 小组后不允许加入其它 PK 小组。加入 PK 小组前后提示如下：



3.应用详细使用信息统计

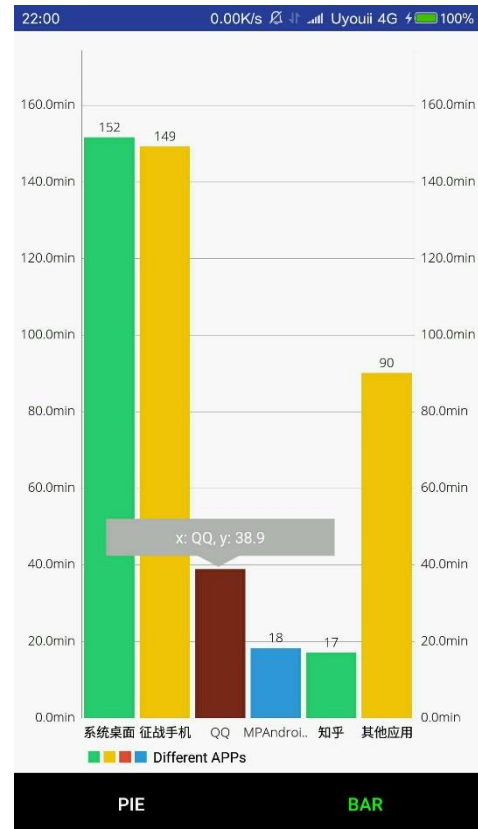
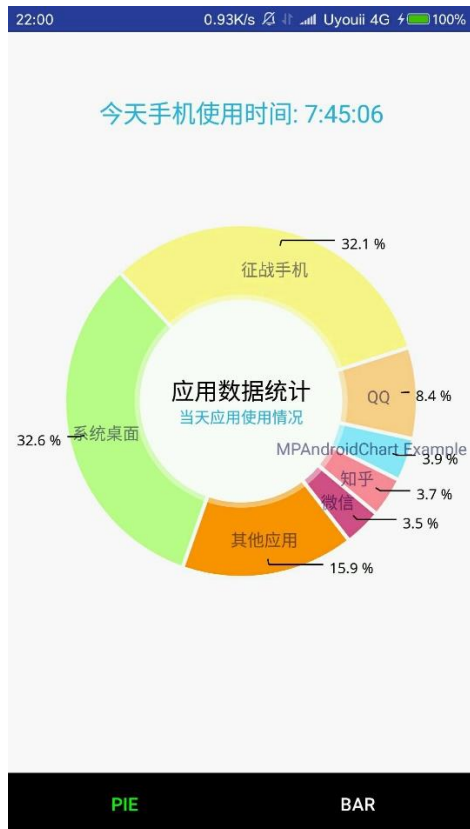


在应用详细信息列表中会列出每个应用的图标、名称、运行时间和本次开机后的操作次数。顺序会按照使用时间的长短进行排序，同时在列表的顶部计算出所有应用的运行时间和总的操作次数。

在下方有 4 个按钮：DAY、WEEK、MONTH、YEAR。用户通过点击相应的按钮实现相应的时间段的选择，其对应的时间段分别为当天，一周，一月和一年的时间范围。用户可以详细的了解到其每个应用在相应时间段内的使用情况并做出合理的判断。

4.应用信息图谱绘制

在应用的主界面绘制图表显示了用户当天应用信息的使用情况：



在主界面的下方提供了两个按钮：PIE 和 BAR。通过点击按钮来切换图表的显示，PIE 对应扇形统计图，BAR 对应条形统计图。

在扇形统计图中会根据每个应用当天的使用时间计算出其相应的比重并在统计图中显示出来。扇形统计图会列出占比最大的 6 个应用并在图中显示出来，将剩下的应用分类到“其他应用”选项中。在扇形统计图的上方会给出手机当天的使用时间。点击或选中某个扇区会将改扇区突出显示出来。

在条形统计图中，每个应用的高为相应应用当天使用的分钟数，应用的横坐标为应用的名称。期中整个统计图的高会根据应用的最大的分钟数变化。条形统计图会绘制出使用时间最多的前 5 款应用并将剩下的应用归类到“其他应用”模块中。点击或选中某个应用会将其 x，y 坐标突出显示出来。

同时在侧边栏还提供了一个分时段图表统计的功能。



在功能页面的下方同样提供了四个按钮，会分别将用户在当天，一周，一月和一年的使用情况通过扇形图的方式表现出来。并且在屏幕的上方会列出在相应时间段内的应用使用的总时间，用户可以通过页面下方的按钮选择自己想要观察的时间段进行显示，便于更加详细的掌握自己在各个时间段内的应用的使用情况。

5.应用管理



在应用管理的界面中会列出每个应用的图标、名称、用户设定的使用时间的上限和

用户设定的限定方式。如果应用未设置相应的使用限制则显示未设定。

用户点击相应的栏目后则进入应用设定的界面。在设定界面中用户需要选择应用使用限制的类型和输入该应用的限制时间。例如上述示例中选中 QQ 后选择限定方式为友情提示，输入挑战时间为 120 分钟，点击确定后 QQ 的设定时间更改为 2 个小时，限定方式为通知提醒。

我们设计的此款 APP 提供给用户有 4 个限定功能：

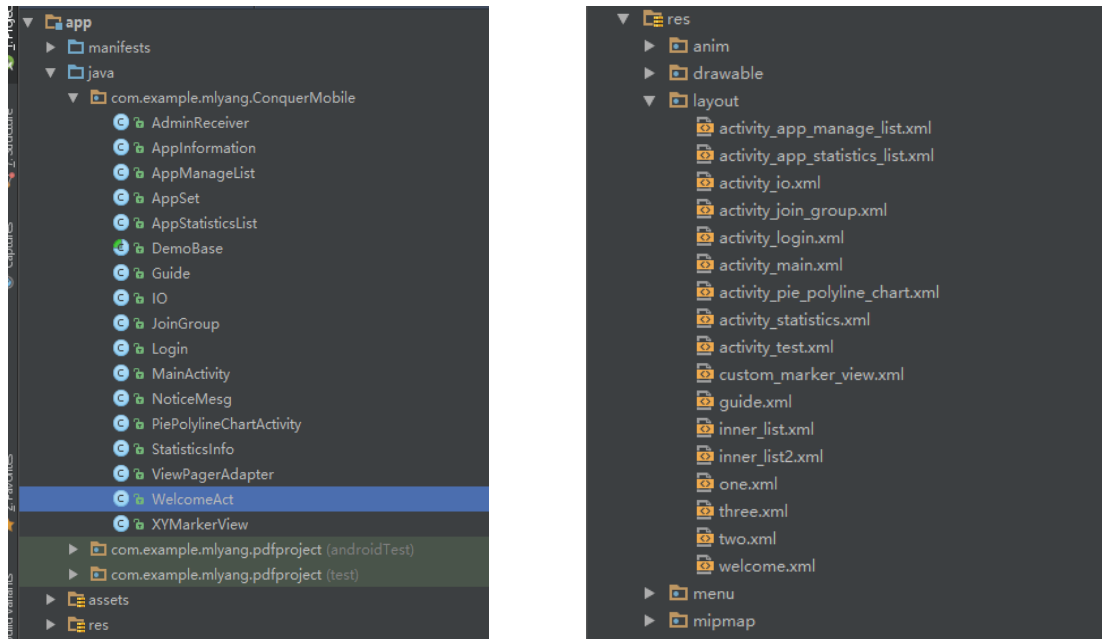
- ① 友情提示：当该应用的使用超出用户的设定的使用时间的限制后，则在应用的通知栏中提醒用户使用时间超过限制。
- ② 睡眠（锁屏）：当该应用的使用超出用户的设定的使用时间的限制后，则提醒用户使用时间超过限制，随后手机自动锁屏，并刷新用户对此应用的设定。
- ③ 重启：当该应用的使用超出用户的设定的使用时间的限制后，提醒用户某款应用使用时间超过限制，并在 5 秒后将手机自动重启，并刷新用户对此应用的设定（确保不会反复重启）。
- ④ 关机：与重启类似，将惩罚操作变为 5 秒后自动关机，并且刷新用户对此应用的设定。



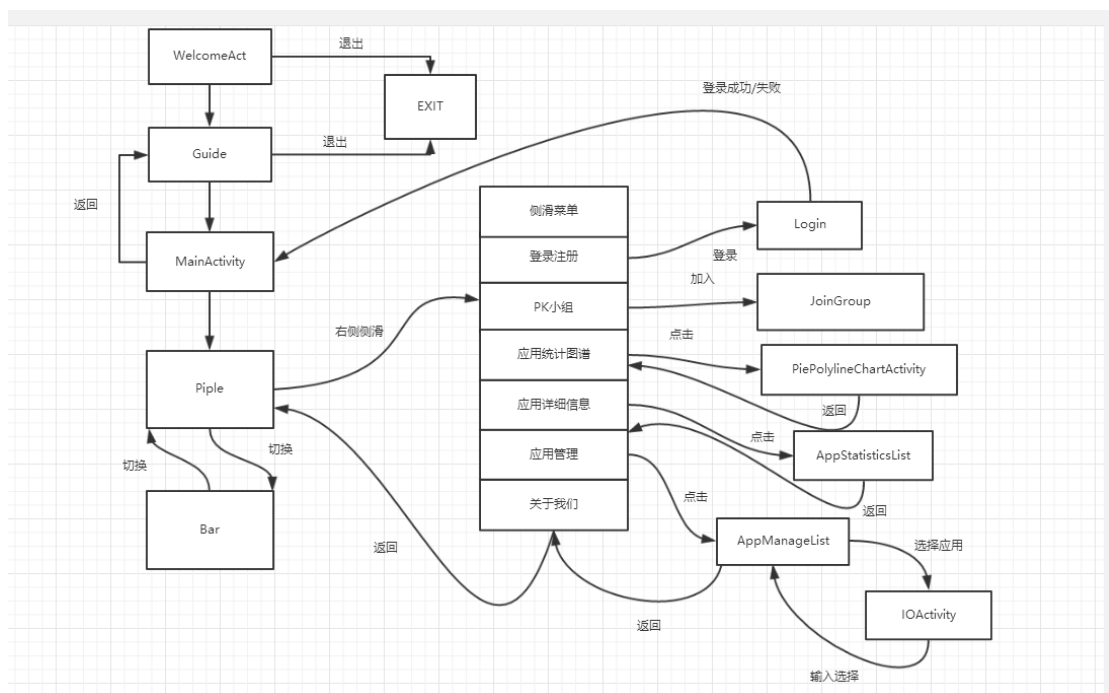
例如将征战手机(本款 APP)的限定时间设置为 12min ,限定操作为睡眠(锁屏), 操作完成后则会提示用户手机会在 5 秒后睡眠 (锁屏)。随后手机进入睡眠状态。对手机关机和睡眠操作的实现则需要获取手机的 root 权限才可以实现。

五、代码架构

本应用征战手机由以下 17 个 Java 类文件组成，其中有 9 个 Activity。工程的具体架构如下：



各个 activity 间的调用关系可以形象的表述为下图：



1.启动及引导页活动

WelcomeAct 为启动页活动，显示 3 秒 logo，其中根据启动信息启动引导界面

Guide 或主界面 MainActivity，该活动核心代码如下：

```
private Handler mHandler = new Handler(){
    public void handleMessage(android.os.Message msg) {
        switch (msg.what) {
            case GO_HOME:
                goHome();
                break;
            case GO_GUIDE:
                goGuide();
                break;
        }
    };
};
```

启动引导页活动 Guide 活动，引导页的开发使用高版本安卓提供的组件 PageViewer，设置 3 张引导页滑动轮播，在使用 PageViewer 时需要额外定义对应的适配器，也就是 ViewPagerAdapter。Guide 活动的实现主要有以下几步，一旦活动 onCreate 之后，对整个引导页进行初始化，然后特别的绘制引导页的“点”记录框。即先调用 initViews()，对各个页面和组件进行初始化，对 PageChange 进行监听，即页面间转换进行监听，通过 setOnPageChangeListener()函数监听。其中需要自行设计 dots，这里使用下面函数 initDots 巧妙实现，使用三张图片即可。

```
private void initDots() {
    dots = new ImageView[views.size()];
    for (int i = 0; i < views.size(); i++) {
        dots[i] = (ImageView) findViewById(ids[i]);
    }
}
```

2.主菜单栏模块

主菜单活动为 MainActivity，活动中声明侧滑菜单抽屉组件、ListView 等，侧滑菜

单中列表为任务可选项 ,对其设置监听 ,根据监听所传入的参数 ,判断监听事件的位置 ,根据不同的监听事件相应的响应不同活动。具体地 ,需要先在主活动的 onCreate 函数中初始化各个组件变量 ,menulist 中设置任务选项 ,然后对 menulist 设置适配器并使其生效 :

```
adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, menuLists);
mDrawerList.setAdapter(adapter);
mDrawerList.setOnItemClickListener(this);
```

其中最主要的功能即监听函数 ,通过判断是否是侧滑菜单的 ViewList ,高版本安卓提供对应的函数 ,第一个参数即 ViewList 的对象 ,在这里用于判断是哪一个 ListView ,这里即 DrawerLayout 里的侧滑 List ,对应不同的监听事件使用 switch 语句进行不同的启动活动即可 ,核心的监听函数如下 :

```
public void onItemClick(AdapterView<?> arg0, View arg1, int
position, long arg3){

    if(arg0==mDrawerList) {
        switch (position) {
            case 0:
                Intent i =new Intent(MainActivity.this,
Login.class);
                startActivityForResult(i,0);

                break;
            case 1://启动 activity 不保留抽屉
                Intent ig =new
Intent(MainActivity.this,JoinGroup.class);
                startActivityForResult(ig,1);
                break;
            case 2:
                Intent intent2 = new Intent(MainActivity.this,
PiePolylineChartActivity.class);
                startActivity(intent2);
                break;
            case 3:
                Intent intent3 = new Intent(MainActivity.this,
AppStatisticsList.class);
```

```

        startActivity(intent3);
        break;
    case 4:
        Intent intent4 = new Intent(MainActivity.this,
AppManageList.class);
        startActivity(intent4);
        break;
    case 5:

        break;
    default:
        exit(0);
    }
}
}
}

```

3.应用信息的使用情况及获取

Google 官方在 android API 21 版本以后提供了一个 android.app.usage 的 API 用于用户获取手机中各个应用的使用情况以及使用时间。

android.app.usage

Added in API level 21

Classes

ConfigurationStats	Represents the usage statistics of a device Configuration for a specific time range.
NetworkStats	Class providing enumeration over buckets of network usage statistics.
NetworkStats.Bucket	Buckets are the smallest elements of a query result.
NetworkStatsManager	Provides access to network usage history and statistics.
NetworkStatsManager.UsageCallback	Base class for usage callbacks.
UsageEvents	A result returned from queryEvents(long, long) from which to read UsageEvents.Event objects.
UsageEvents.Event	An event representing a state change for a component.
UsageStats	Contains usage statistics for an app package for a specific time range.
UsageStatsManager	Provides access to device usage history and statistics.

其中用于获取手机中应用时间统计的类主要是 UsageStats 和 UsageStatsManager

在 UsageStatsManager 类中为用户提供了一些获取应用使用信息统计列表的接口。

Public methods	
<code>boolean</code>	<code>isAppInactive(String packageName)</code> Returns whether the specified app is currently considered inactive.
<code>Map<String, UsageStats></code>	<code>queryAndAggregateUsageStats(long beginTime, long endTime)</code> A convenience method that queries for all stats in the given range (using the best interval for that range), merges the resulting data, and keys it by package name.
<code>List<ConfigurationStats></code>	<code>queryConfigurations(int intervalType, long beginTime, long endTime)</code> Gets the hardware configurations the device was in for the given time range, aggregated by the specified interval.
<code>UsageEvents</code>	<code>queryEvents(long beginTime, long endTime)</code> Query for events in the given time range.
<code>List<UsageStats></code>	<code>queryUsageStats(int intervalType, long beginTime, long endTime)</code> Gets application usage stats for the given time range, aggregated by the specified interval.
Inherited methods	

UsageStatsManager 接口

List<UsageStats> queryUsageStats(int intervalType, long beginTime, long endTime)

函数在函数 queryUsageStats 输入时间跨度类型，统计的起始时间和统计的结束时间来获取手机中全部应用的 UsageStats 的统计列表

Public methods	
<code>void</code>	<code>add(UsageStats right)</code> Add the statistics from the right <code>UsageStats</code> to the left.
<code>int</code>	<code>describeContents()</code> Describe the kinds of special objects contained in this Parcelable instance's marshaled representation.
<code>long</code>	<code>getFirstTimeStamp()</code> Get the beginning of the time range this <code>UsageStats</code> represents, measured in milliseconds since the epoch.
<code>long</code>	<code>getLastTimeStamp()</code> Get the end of the time range this <code>UsageStats</code> represents, measured in milliseconds since the epoch.
<code>long</code>	<code>getLastTimeUsed()</code> Get the last time this package was used, measured in milliseconds since the epoch.
<code>String</code>	<code>getPackageName()</code>
<code>long</code>	<code>getTotalTimeInForeground()</code> Get the total time this package spent in the foreground, measured in milliseconds.
<code>void</code>	<code>writeToParcel(Parcel dest, int flags)</code> Flatten this object in to a Parcel.

UsageStats 接口

在 UsageStats 中用户可以通过函数 getTotalTimeForeground()得到每个应用在前台的使用时间，通过 getPackageName () 方法得到应用的包名。

通过计算出应用开始统计的起始时间和到目前为止的结束时间得到 UsageStats 的列表储存在 result 中。通过获取时间的类型选择不同的启示时间和时间跨度类型。

```
private void setResultList(Context context) {
    UsageStatsManager m =
```

```

(UsageStatsManager)context.getSystemService(Context.USAGE_STATS_SERVICE
);
this.AppInfoList = new ArrayList<>();
if(m != null) {
    Calendar calendar = Calendar.getInstance();
    long now = calendar.getTimeInMillis();
    long begintime = getBeginTime();
    if(style == DAY)
        this.result =
m.queryUsageStats(UsageStatsManager.INTERVAL_BEST, begintime, now);
    else if(style == WEEK)
        this.result =
m.queryUsageStats(UsageStatsManager.INTERVAL_WEEKLY, begintime, now);
    else if(style == MONTH)
        this.result =
m.queryUsageStats(UsageStatsManager.INTERVAL_MONTHLY, begintime, now);
    else if(style == YEAR)
        this.result =
m.queryUsageStats(UsageStatsManager.INTERVAL_YEARLY, begintime, now);
    else {
        this.result =
m.queryUsageStats(UsageStatsManager.INTERVAL_BEST, begintime, now);
    }
}
}

```

在这里要注意的是通过 UsageStatsManager 中的 queryUsageStats 获取相应应用的使用信息时会获取到重复的应用使用情况。这是由于 google 提供的 app.usage 的 API 统计出的应用使用时间时分段统计的，因此获取时间也是分段获取的。在获取应用的使用时间时，例如统计当天的数据可能会获取到同一个应用的两个或三个结果。这时需要判断其中某些信息是否是我们想要获取的相应时间段内的信息。

```

private List<UsageStats> MergeList( List<UsageStats> result) {
    List<UsageStats> Mergeresult = new ArrayList<>();

    for(int i=0;i<result.size();i++) {

        long begintime;
        begintime = getBeginTime();

        if(result.get(i).getFirstTimeStamp() > begintime) {

```

```

        int num = FoundUsageStats(Mergeresult, result.get(i));
        if (num >= 0) {
            UsageStats u = Mergeresult.get(num);
            u.add(result.get(i));
            Mergeresult.set(num, u);
        } else Mergeresult.add(result.get(i));
    }
}
return Mergeresult;
}

```

我们可以看到 UsageStats 中提供了两个方法，add 和 getFirstTimeStamp()，即在包名相同的情况下，add 方法可以合并两个 UsageStats 中的数据，而 getFirstTimeStamp 方法则是获取到统计出的这个 UsageStats。我们通过 getFirstTimeStamp 判断这个 UsageStats 是否在我们需要的时间范围内，通过 add 方法将符合条件的统计数据合并。在真正显示时排除掉使用时间为 0 的数据即可。

由于显示时需要获取应用的其他信息，例如图标，label，总的使用次数等等，所以封装了一个 AppInformation 类，用于获取应用的各个详细信息。

私有变量：

```

public class AppInformation {
    private UsageStats usageStats;
    private String packageName;
    private String label;
    private Drawable icon;
    private long UsedTimebyDay; //milliseconds
    private Context context;
    private int times;
}

```

获取应用信息的方法：

```

private void GenerateInfo() throws
PackageManager.NameNotFoundException, NoSuchFieldException,
IllegalAccessException {
    PackageManager packageManager = context.getPackageManager();
    this.packageName = usageStats.getPackageName();
    ApplicationInfo applicationInfo =
packageManager.getApplicationInfo(this.packageName, 0);
    this.Label = (String)
}

```

```

packageManager.getApplicationLabel(applicationInfo);
    this.UsedTimebyDay = usageStats.getTotalTimeInForeground();
    this.times =
(Integer)usageStats.getClass().getDeclaredField("mLaunchCount").get(usageStats);

    if(this.UsedTimebyDay > 0) {
        this.Icon = applicationInfo.loadIcon(packageManager);
    }
}

```

通过 `getPackageManager()` 可以获取到手机中的包管理器。通过包管理器和在 `usageStats` 中得到的应用的包名可以得到储存该应用信息的 `ApplicationInfo` 类。通过 `applicationInfo` 可以得到该应用的图标和 `label`（应用名称）。通过之前统计的 `usageStats` 可以得到应用在前台的使用时间。

在 android API 21 以后，获取应用使用次数的接口被隐藏，因此只能用映射的方法获取该应用的操作次数：

```

this.times      =      (Integer)usageStats.getClass().getDeclaredField
("mLaunchCount").get(usageStats);

```

到目前为止我们希望得到的一个应用所有信息获取完毕。

4.应用信息列表展示

应用信息的列表展示通过 `listView` 实现。由于在显示用户列表的时候要显示应用图标和分层显示应用信息，所以这里的 `adapter` 用 `SimpleAdapter`

```

StatisticsInfo statisticsInfo = new StatisticsInfo(this,this.style);
totalTime = statisticsInfo.getTotalTime();
totalTimes = statisticsInfo.getTotalTimes();
datalist = getDataList(statisticsInfo.getShowList());

ListView listView = (ListView)findViewById(R.id.AppStatisticsList);
SimpleAdapter adapter = new
SimpleAdapter(this,datalist,R.layout.inner_list,
    new String[]{"label","info","times","icon"},
    new int[]{R.id.Label,R.id.info,R.id.times,R.id.icon});

```



```

listView.setAdapter(adapter);

adapter.setViewBinder(new SimpleAdapter.ViewBinder() {
    @Override
    public boolean setViewValue(View view, Object o, String s) {
        if(view instanceof ImageView && o instanceof Drawable){

            ImageView iv=(ImageView)view;
            iv.setImageDrawable((Drawable)o);
            return true;
        }
        else return false;
    }
});

```

这里在 SimpleAdapter 内部指定了一个 xml 文件用户规范每个 list 中内部的格式。

Inner_list.xml 文件可以在 layout 文件夹中得到。用户展示的信息由 getDataList 函数获取。这里要注意需要给 adapter 设置一个 setViewBinder, 否则无法正常显示应用的图标图片。这是由于获取应用图标是一个 Drawable 类型的数据, 这里需要先将其转化为 ImageView 才能够正常显示。

getDataList 将统计得到的应用数绑定到一个 List<Map<String, Object>> 中, 交给 listView 显示。Showlist 是调用 statisticsInfo.getShowList() 得到用于显示的应用数据的列表, 已按照使用时间的大小顺序排序。方法在 StatisticsInfo 类中实现。

```

private List<Map<String, Object>>
getDataList(ArrayList<AppInformation> ShowList) {
    List<Map<String, Object>> dataList = new ArrayList<Map<String,
Object>>();

    Map<String, Object> map = new HashMap<String, Object>();
    map.put("label", "全部应用");
    map.put("info", "运行时间: " + DateUtils.formatElapsedTime(totalTime /
1000));
    map.put("times", "本次开机操作次数: " + totalTimes);
    map.put("icon", R.drawable.use);
    dataList.add(map);

    for(AppInformation appInformation : ShowList) {

```

```

        map = new HashMap<String, Object>();
        map.put("label", appInformation.getLabel());
        map.put("info", "运行时间: " +
DateUtils.formatElapsedTime(appInformation.getUsedTimebyDay() / 1000));
        map.put("times", "本次开机操作次数: " + appInformation.getTimes());
        map.put("icon", appInformation.getIcon());
        dataList.add(map);
    }

    return dataList;
}

```

5.应用信息图表展示

在我们的应用中主要涉及两种图表：扇形统计图和条形统计图。

绘制图表调用的库来自于开源的 MPAndroidChart 图表项目，其项目位置：

<https://github.com/PhilJay/MPAndroidChart>。

扇形统计图：

统计图的基本显示设置：

```

mChart = (PieChart) findViewById(R.id.chart1);
mChart.setUsePercentValues(true);
mChart.getDescription().setEnabled(false);
mChart.setExtraOffsets(5, 10, 5, 5);

mChart.setDragDecelerationFrictionCoef(0.95f);

tf = Typeface.createFromAsset(getAssets(), "OpenSans-Regular.ttf");

mChart.setCenterTextTypeface(Typeface.createFromAsset(getAssets(),
"OpenSans-Light.ttf"));
mChart.setCenterText(generateCenterSpannableText(style));

mChart.setExtraOffsets(20.f, 0.f, 20.f, 0.f);

mChart.setDrawHoleEnabled(true);
mChart.setHoleColor(Color.WHITE);

```

```

mChart.setTransparentCircleColor(Color.WHITE);
mChart.setTransparentCircleAlpha(110);

mChart.setEntryLabelColor(R.color.dimgrey);

//设置内圈半径的角度
mChart.setHoleRadius(58f);
mChart.setTransparentCircleRadius(61f);

mChart.setDrawCenterText(true);

mChart.setRotationAngle(0);
// enable rotation of the chart by touch
mChart.setRotationEnabled(true);
mChart.setHighlightPerTapEnabled(true);

// mChart.setUnit(" €");
// mChart.setDrawUnitsInChart(true);

// add a selection listener
mChart.setOnChartValueSelectedListener(this);

setData(style);

mChart.animateY(1400, Easing.EasingOption.EaseInOutQuad);
// mChart.spin(2000, 0, 360);

Legend l = mChart.getLegend();
l.setVerticalAlignment(Legend.LegendVerticalAlignment.TOP);
l.setHorizontalAlignment(Legend.LegendHorizontalAlignment.RIGHT);
l.setOrientation(Legend.LegendOrientation.VERTICAL);
l.setDrawInside(false);
l.setEnabled(false);

```

通过 setCenterText 设置在扇形统计图中的中心显示的字体。

setDrawHoleEnabled(true);设置通过拖动扇形统计图中间的空洞也可以旋转扇形统计图。

setHoleColor(Color.WHITE);设置中心区域的颜色。

setTransparentCircleColor(Color.WHITE);设置内部圆环的颜色。

mChart.setHoleRadius(58f);

mChart.setTransparentCircleRadius(61f);设置内圈圆的内径和外径。

setDrawCenterText(true);设置是否显示中心区域文本。

setRotationAngle(0);设置初始旋转角度

setRotationEnabled(true);设置是否可以旋转统计图。

setOnChartValueSelectedListener(this);设置扇形统计图的选择相应监听。

setData(style);设置扇形统计图中的数据和扇形统计图中的颜色设置。

```
private void setData(int style) {
    StatisticsInfo statisticsInfo = new StatisticsInfo(this, style);
    ArrayList<AppInformation> ShowList = statisticsInfo.getShowList();

    totaltime = statisticsInfo.getTotalTime();
    TextView textView = (TextView) findViewById(R.id.textViewchart);

    SpannableString sp = new SpannableString("已使用总时间: " +
    DateUtils.formatElapsedTime(totaltime / 1000));
    sp.setSpan(new RelativeSizeSpan(1.35f), 0, sp.length(), 0);
    sp.setSpan(new ForegroundColorSpan(ColorTemplate.getHoloBlue()), 0,
    sp.length(), 0);
    textView.setText(sp);

    ArrayList<PieEntry> entries = new ArrayList<PieEntry>();

    // NOTE: The order of the entries when being added to the entries
    array determines their position around the center of
    // the chart.

    if(ShowList.size() < 6) {
        for (int i = 0; i < ShowList.size(); i++) {
            float apptime = (float)ShowList.get(i).getUsedTimebyDay() /
1000;
            if(apptime / totaltime * 1000 >= 0.001)
                entries.add(new PieEntry(apptime,
ShowList.get(i).getLabel()));
        }
    }
    else {
        for(int i = 0; i < 6; i++) {
            float apptime = (float)ShowList.get(i).getUsedTimebyDay() /
```

```

1000;
        if(apptime / totaltime * 1000 >= 0.001)
            entries.add(new PieEntry(apptime,
ShowList.get(i).getLabel()));
    }
    long otherTime = 0;
    for(int i=6;i<ShowList.size();i++) {
        otherTime += ShowList.get(i).getUsedTimebyDay() / 1000;
    }
    if(1.0 * otherTime / totaltime * 1000 >= 0.001)
        entries.add(new PieEntry((float)otherTime, "其他应用"));
    }

```

sp 为扇形统计图显示中其上方的手机使用时间的显示。

扇形统计图中显示的数据同样是从 `StatisticsInfo` 类中获取。

在设置数据时,如果需要显示的数据数量小于 7 个,则将其都显示出来。若需要显示的数据数量大于 7 个,则显示前六个数据(已按照时间大小顺序排序),将剩下的数据归类到“其他应用”中显示。

在用扇形统计图显示数据时,需要注意的是如果这个需要显示的数据如果占显示总数据的显示比重过小(例如小于 0.01%),扇形统计图的显示会出现问题。所以在设置数据时需要排除掉特别小的数据(统计应用时有些应用的使用时间接近 0s)。

条形统计图：

条形统计图的基本设置：

```

private void DrawBarChart() {
    StatisticsInfo statisticsInfo = new StatisticsInfo(this,style);
    ShowList = statisticsInfo.getShowList();

    bChart = (BarChart) findViewById(R.id.barchartmain);
    bChart.setOnChartValueSelectedListener(this);

    bChart.setDrawBarShadow(false);

```

```

bChart.setDrawValueAboveBar(true);

bChart.getDescription().setEnabled(false);

// drawn
bChart.setMaxVisibleValueCount(60);

// scaling can now only be done on x- and y-axis separately
bChart.setPinchZoom(false);

bChart.setDrawGridBackground(false);

IAxisValueFormatter xAxisFormatter = new IAxisValueFormatter() {
    @Override
    public String getFormattedValue(float value, AxisBase axis) {
        int i = (int)value;
        if(ShowList.size() > i) {
            if(i >= 5)
                return "其他应用";
            else {
                String str = ShowList.get(i).getLabel();
                if (str.length() < 8)
                    return str;
                else return (str.substring(0, 8) + "..");
            }
        }
        else return "";
    }
};

XAxis xAxis = bChart.getXAxis();
xAxis.setPosition(XAxis.XAxisPosition.BOTTOM);
xAxis.setTypeface(mTfLight);
xAxis.setDrawGridLines(false);
xAxis.setGranularity(1f); // only intervals of 1 day
xAxis.setLabelCount(7);
xAxis.setValueFormatter(xAxisFormatter);

IAxisValueFormatter custom = new IAxisValueFormatter() {
    @Override
    public String getFormattedValue(float value, AxisBase axis) {
        return value + "min";
    }
};

```

```

    ○ ○ ○ ○ ○ ○

    setDataBarChart();

    // setting data

    // mChart.setDrawLegend(false);
}

```

条形统计图的绘制在设置数据方面要比扇形统计图更复杂一点，因为要涉及到横纵坐标的数据和横纵坐标的格式。在初始化条形图时就需要指定好横纵坐标的显示格式。

横坐标的设置在 `xAxisFormatter` 指定：在制定横坐标时，我将 `showlist` 中的坐标赋值给相应的位置。这里需要判断一下，如果一个应用的名字过长，则需要将其截断，否则显示时会占据其他应用的显示位置。并且将位置大于 5 的应用名字设置为“其他应用”。

在设置纵坐标的显示格式时直接返回纵坐标后加 `min` 即可，即显示的分钟数。

```

private void setDataBarChart() {

    ArrayList<BarEntry> yVals1 = new ArrayList<BarEntry>();
    if(ShowList.size() < 5) {
        for (int i = 0; i < ShowList.size(); i++) {
            yVals1.add(new BarEntry(i, (float)(1.0 *
ShowList.get(i).getUsedTimebyDay() / 1000 / 60)));
        }
    }
    else {
        for(int i = 0;i < 5;i++) {
            yVals1.add(new BarEntry(i, (float)(1.0 *
ShowList.get(i).getUsedTimebyDay() / 1000 / 60)));
        }
        long otherTime = 0;
        for(int i=5;i<ShowList.size();i++) {
            otherTime += ShowList.get(i).getUsedTimebyDay();
        }
        yVals1.add(new BarEntry(5,(float)(1.0 * otherTime / 1000 /
60)));
    }
}

```

```

    }
    BarDataSet set1;
    ...

    bChart.setData(data);
}
}

```

同样在给条形统计图赋值时首先要判断一下显示个数是否大于 5，如果大于 5 则将剩下的应用归类为“其他应用”。在显示时需要将从 showlist 中得到的数据转化为分钟。

6.应用管理列表

应用管理列表与之前的应用信息显示列表相似，但不同的是要为相应的 listView 设置相应操作。即点击相应的 listView 即打开 IO Activity 让用户进行相应的操作的输入和设置。

在 IO Activity 中，用户对选中应用的设定信息格式化后储存在文件当中。

```

public void writeFileData(String fileName,String label,int type, String
time) throws IOException {
    if (Integer.parseInt(time) != 0) {
        File path = new File(getExternalCacheDir(), fileName);
        NoticeMesg noticeMesg = new NoticeMesg(this, "");
        boolean found = false;
        String content = "";
        if (path.exists()) {
            BufferedReader in = new BufferedReader(new FileReader(path));
            String line = in.readLine();
            while (line != null) {
                if (line.contains(label)) {
                    found = true;
                    if (type != 4) {
                        content += label + "#" + type + "#" + time + "\n";
                        Toast.makeText(this,getContext(), "设定
成功!", Toast.LENGTH_SHORT).show();
                    } else {
                        noticeMesg.Toast("4");
                    }
                } else {
                    content += line + "\n";
                }
            }
        }
    }
}

```



```

        }
        line = in.readLine();
    }
}
BufferedWriter bw;
if (found) {
    bw = new BufferedWriter(new
FileWriter(path.getAbsolutePath()));
    bw.write(content);
    bw.flush();
    bw.close();
} else {
    if (type != 4) {
        String str = label + "#" + type + "#" + time + "\n";
        bw = new BufferedWriter(new
FileWriter(path.getAbsolutePath(), true));
        bw.write(str);
        Toast.makeText(this.getContext(), "设定成功!",
Toast.LENGTH_SHORT).show();
        bw.flush();
        bw.close();
    } else {
        noticeMesg.Toast("4");
    }
}
}
}
}

```

根据设定，用户对应用的设定选项总共有 5 总类型：通知提醒、睡眠（锁屏）、关机、重启和取消设定。前四种操作在用户输入后，修改或添加相应的文件。如果是取消设定，则删除当前行。

文件路径通过 `getExternalCacheDir()` 获取。在用户执行在 IO 界面点击确定后，则会搜索相应的文件。搜先判断文件是否存在，如果文件不存在则直接进行文件的创建和写入操作。

首先判断用户输入的时间，如果时间为 0，则什么操作也不进行。如果文件存在，则判断是否存在用户输入的 label，如果不存在，则直接添加，如果存在相应的 label，如果用户选择的是取消设定，则删除当前行，如果是其他操作，则修改当前行。经过上

述操作则将用户的输入粗存在文件中。

在用户进入应用管理列表界面时，征战手机 APP 会先从文件中读取用户对各个应用的设定并显示在 listView 中。

```
private void getSetList() throws IOException {
    SetList = new ArrayList<AppSet>();
    File path = new File(getExternalCacheDir(), "TEST.txt");
    if(path.exists()) {
        BufferedReader in = new BufferedReader(new FileReader(path));
        String line = in.readLine();
        while (line != null) {
            int n1 = line.indexOf("#");
            int n2 = line.indexOf("#", n1 + 1);
            String label = line.substring(0, n1);
            String type = line.substring(n1 + 1, n2);
            String time = line.substring(n2 + 1);

            AppSet appSet = new
AppSet(label, Integer.parseInt(type), time);
            SetList.add(appSet);
            line = in.readLine();
        }
    }
}
```

将用户对文件的各个设定操作存在一个 ArrayList 当中，在显示应用列表时，判断储存设定的 listView 中是否包含关于此应用的设定，如果存在则格式化之后显示，如果不存在则显示“未设定”。

```
@Override
public void onItemClick(AdapterView<?> adapterView, View view, int i,
long l) {

    String label = ShowList.get(i).getLabel();

    Intent a= new Intent(this, IO.class);
    Bundle bundle=new Bundle();
```

```

        bundle.putString("name", label);
        a.putExtras(bundle);
        startActivity(a);
    }

```

为 listView 设定点击响应的回掉函数。实现 AdapterView.OnItemClickListener 提供的接口 onItemClick，当用户典型响应的栏目时则打开 IO Activity 并传入相应应用的 label（名称），之后便进入了应用的设定页面。

7.IO 模块

IO 模块是本应用交互操作的主要模块，获取用户输入并将数据格式化，输入到对应文件中，为后面判断关机、重启、锁屏以及提示服务做数据准备。首先通过 Android TextView 以及 EditText 的 getText()方法获取部分用户输入—限制时间。然后通过 intent 机制，获取参数应用名字：

```

Bundle bundle=getIntent().getExtras();
name= bundle.getString("name");

```

之后是 Type 参数的获取，type 参数并不能通过 EditText 那样简单的获得。type 参数是通过 RadioButton 选择的，将四个 RadioButton 设置为一个 RadioGroup，这样组内的 RadioButton 只能有一个被选中。参数的获取是通过 getCheckedRadioButtonId()方法获取 RadioButton 的 ID,根据 ID 信息找到对应的 RadioButton，获取 RadioButton 的 name 信息，根据名字进行判断赋值 type，核心代码如下：

```

group.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(RadioGroup arg0, int arg1) {
        //获取变更后的选中项的 ID
        int radioButtonId = arg0.getCheckedRadioButtonId();
        //根据 ID 获取 RadioButton 的实例
        rb = (RadioButton) IO.this.findViewById(radioButtonId);
    }
}

```

```

        //更新文本内容，以符合选中项
        String s=rb.getText().toString();
        tv.setText("您选择的类型: " + s);
        String s1="友情提示"+"";
        String s2="睡眠"+"";
        String s3="重启"+"";
        String s4="关机"+"";
        String s5="取消设定"+"";
        if(s.equals(s1))type=0;
        else if(s.equals(s2))type=1;
        else if(s.equals(s3))type=2;
        else if(s.equals(s4))type=3;
        else if(s.equals(s5))type=4;
        else exit(0);
    }
});

```

之后三个参数进行格式化读写到文件中，分别是应用名称，惩罚类型，限制时间，数据之间用‘#’ 隔开。

```
content += label + "#" + type + "#" + time + "\n";
```

8.Notice_Toast 模块

在用户使用应用超出自己的设定之后，将会弹出对应的提示信息，而且需要在消息栏中进行应用提示。为了提高代码的重用率，我们封装了 NoticeMesg 这样一个类，用来处理所有的通知和消息。

消息类的构造函数需要获取对应的上下文和应用的名字，类内对外提供两个 public 接口，分别是 Toast 和 Time_Notice，分别用于 Tosat 提示和消息提示。

Toast 接口的实现主要是对 type 进行判断及格式化，然后调用 Toast 的 makeText()接口以及 show()接口。核心代码如下：

```

public void Toast(String type){
    String s;
    switch (type){
        case "0":
            s="友情提示";
            Toast.makeText(context.getApplicationContext(), label

```

```

+ "使用已超时!", Toast.LENGTH_SHORT).show();
        break;
        case "1":
            s="睡眠";
            Toast.makeText(context.getApplicationContext(), label
+ "使用已超时, 将于 10 秒钟后自动"+s+"!", Toast.LENGTH_SHORT).show();
            break;
        case "2":
            s="重启";
            Toast.makeText(context.getApplicationContext(), label
+ "使用已超时, 将于 10 秒钟后自动"+s+"!", Toast.LENGTH_SHORT).show();
            break;
        case "3":
            s="关机";
            Toast.makeText(context.getApplicationContext(), label
+ "使用已超时, 将于 10 秒钟后自动"+s+"!", Toast.LENGTH_SHORT).show();
            case "4":
                Toast.makeText(context.getApplicationContext(), label
+ "设定已取消!", Toast.LENGTH_SHORT).show();
                break;
            default:break;
        }
    }
}

```

Time_Notice 接口用于发送应用消息，会在手机的通知栏中输出。与 Toast 类似
的首先需要针对本业务对输入信息进行格式化判断以及转换，核心的发送功能用

Notice_launcher()保护接口实现，不对外开放。格式化处理代码如下：

```

public void Time_Notice(String limit,String time, String type){

    if (type.equals("4"))return;
    switch (type){
        case "0":
            type="友情提示";
            break;
        case "1":
            type="睡眠";
            break;
        case "2":
            type="重启";
            break;
        case "3":
            type="关机";

```

```

        break;
    default:
        return;
    }

    String notice_title= "应用使用超时提醒";
    String notice_content= "您的" + Label + "超时: "+
DateUtils.formatElapsedTime(Integer.parseInt(time) -
Integer.parseInt(limit))+
        "\n 惩罚类型: "+type;
    Notice_launcher(notice_title, notice_content);
}

```

消息发送的实现使用了 Android 21 版本以上提供的 Notification()类，这个类较好的封装了消息发送功能。在 setSmallIcon()中设置发送消息的图标，在 setContentTitle()中设置提示的标题，在 setContentText()中设置提示的信息，具体核心代码如下：

```

protected void Notice_launcher(String notice_title, String
notice_content){
    Notification notifation= new Notification.Builder(context)
        .setContentTitle(notice_title)
        .setContentText(notice_content)
        .setSmallIcon(R.drawable.ic_launcher)
        .setLargeIcon(BitmapFactory.decodeResource(context.getResources(), R.drawable.ic_launcher))
        .build();
    NotificationManager manger= (NotificationManager)
context.getSystemService(NOTIFICATION_SERVICE);
    manger.notify(0, notifation);
}

```

9.设备管理模块

设备管理模块即在用户使用某款应用的时间超出自己的预定时间时，设备管理模块将会根据用户原来的设定进行判罚，即执行对应的关机、重启、提醒或者是锁屏操作。在设备管理模块，权限的注册是必要的，因为执行的大部分是系统级指令，所以需要 Manifest 中注册 APP 权限：

```

<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
android:name="android.permission.PACKAGE_USAGE_STATS" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.DEVICE_POWER"
/>
<uses-permission
android:name="android.permission.DISABLE_KEYGUARD" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.REBOOT"></uses-
permission>
<uses-permission
android:name="android.permission.SHUTDOWN"></uses-permission>

```

对于关机管理，由于 API 21 版本以上的安卓出于安全考虑，不再提供关机的接口，这样就需要使用其他的方式来控制关机，我们先后尝试了广播、映射等方式，最后选择使用系统映射的方式来管理设备关机。在判断惩罚是 SHUTDOWN 之后，先删除文件中本条设定，保证惩罚只进行一次，开机后能够正常运行。另一方面，如果使用时间超出限制，需要延时执行关机，这里我们使用 `Handler().postDelayed()` 方法，设置延时时间为 5 秒之后，执行关机操作。利用映射的方式需要先获取 `ServiceManager()` 类，这是一个隐藏接口，建立一个远程服务对象，然后获得 `IPowerManager.Stub()` 类，通过这个实例化的类执行 `asInterface()` 方法，这个方法可以通过 `invoke()` 接口得到我们熟悉的 `IPowerManager()` 对象，可以实现 shutdown 操作。经过这样一个过程就通过系统映射的方式实现了 shutdown 的操作。具体代码如下：

```

else if (appSet.getType() == AppSet.SHUTDOWN) {
    Toast.makeText(this.getApplicationContext(), "应用" +
appInformation.getLabel() + "超时，手机将于 5 秒钟后关机",
Toast.LENGTH_SHORT).show();

```

```

    NoticeMesg noticeMesg = new NoticeMesg(MainActivity.this,
appInformation.getLabel());
    noticeMesg.Time_Notice(appSet.getTime(), String.valueOf(time),
String.valueOf(appSet.getType()));
    //执行惩罚之后删除之前的设定
    try {
        delete_label(appInformation.getLabel());
    } catch (IOException e) {
        e.printStackTrace();
    }
    new Handler().postDelayed(new Runnable() {
        public void run() {
            try {
                //获得 ServiceManager 类
                Class<?> ServiceManager = Class
                    .forName("android.os.ServiceManager");
                //获得 ServiceManager 的 getService 方法
                Method getService =
ServiceManager.getMethod("getService", java.lang.String.class);
                //调用 getService 获取 RemoteService
                Object oRemoteService =
getService.invoke(null, Context.POWER_SERVICE);
                //获得 IPowerManager.Stub 类
                Class<?> cStub = Class
                    .forName("android.os.IPowerManager$Stub");
                //获得 asInterface 方法
                Method asInterface = cStub.getMethod("asInterface",
android.os.IBinder.class);
                //调用 asInterface 方法获取 IPowerManager 对象
                Object oIPowerManager = asInterface.invoke(null,
oRemoteService);
                //获得 shutdown()方法
                Method shutdown =
oIPowerManager.getClass().getMethod("shutdown", boolean.class, boolean.class);
                shutdown.invoke(oIPowerManager, false, true);
            } catch (Exception e) {
            }
        }
    }, 5000);

```


相比于关机的系统映射实现，重启功能就可以调用 `PowerManager()` 来实现，在注册了相应的 `REBOOT` 权限之后，类似的调用 `Handler().postDelayed()` 延时执行重启操作，调用 `PowerManager.reboot()` 接口，具体实现如下：

```
else if (appSet.getType() == AppSet.REBOOT) {
    Toast.makeText(this.getApplicationContext(), "应用" +
appInformation.getLabel() + "超时，手机将于 5 秒钟后重启",
Toast.LENGTH_SHORT).show();
    NoticeMesg noticeMesg = new NoticeMesg(MainActivity.this,
appInformation.getLabel());
    noticeMesg.Time_Notice(appSet.getTime(), String.valueOf(time),
String.valueOf(appSet.getType()));
    //执行惩罚之后删除之前的设定
    try {
        delete_label(appInformation.getLabel());
    } catch (IOException e) {
        e.printStackTrace();
    }
    new Handler().postDelayed(new Runnable(){
        public void run() {

            PowerManager pManager=(PowerManager)
getSystemService(Context.POWER_SERVICE);
            pManager.reboot(null); //重启
            try {
                delete_label(appInformation.getLabel());
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }, 5000);
}
```

最后对于锁屏操作，先判断操作类型，然后发出 `Toast` 和 `Notice`，和前两种操作类似的，需要判断文件中是否已经存在这个应用的设置，如果存在需要将其清除，避免反复锁屏。然后调用 `Sleep` 调用，锁屏 `sleep` 的实现调用了 `DvicePolicManager()` 中的服务，与关机类似的，`Android 21` 以后的版本本身没有提供 `API` 供调用，所以使用了类似于系统反射的方式实现锁屏。核心的代码如下：

```

else if (appSet.getType() == AppSet.SLEEP) {
    Toast.makeText(this.getApplicationContext(), "应用" +
appInformation.getLabel() + "超时，手机将于 5 秒钟后睡眠",
Toast.LENGTH_SHORT).show();
    NoticeMesg noticeMesg = new NoticeMesg(MainActivity.this,
appInformation.getLabel());
    noticeMesg.Time_Notice(appSet.getTime(), String.valueOf(time),
String.valueOf(appSet.getType()));
    new Handler().postDelayed(new Runnable() {
        public void run() {

            //执行惩罚之后删除之前的设定
            try {
                delete_label(appInformation.getLabel());
            } catch (IOException e) {
                e.printStackTrace();
            }

            devicePolicyManager = (DevicePolicyManager)
getSystemService(DEVICE_POLICY_SERVICE);
            componentName = new ComponentName(MainActivity.this,
AdminReceiver.class);

            if (devicePolicyManager.isAdminActive(componentName))
                devicePolicyManager.lockNow();
            else {
                Registration();
            }
        }
    }, 5000);
}

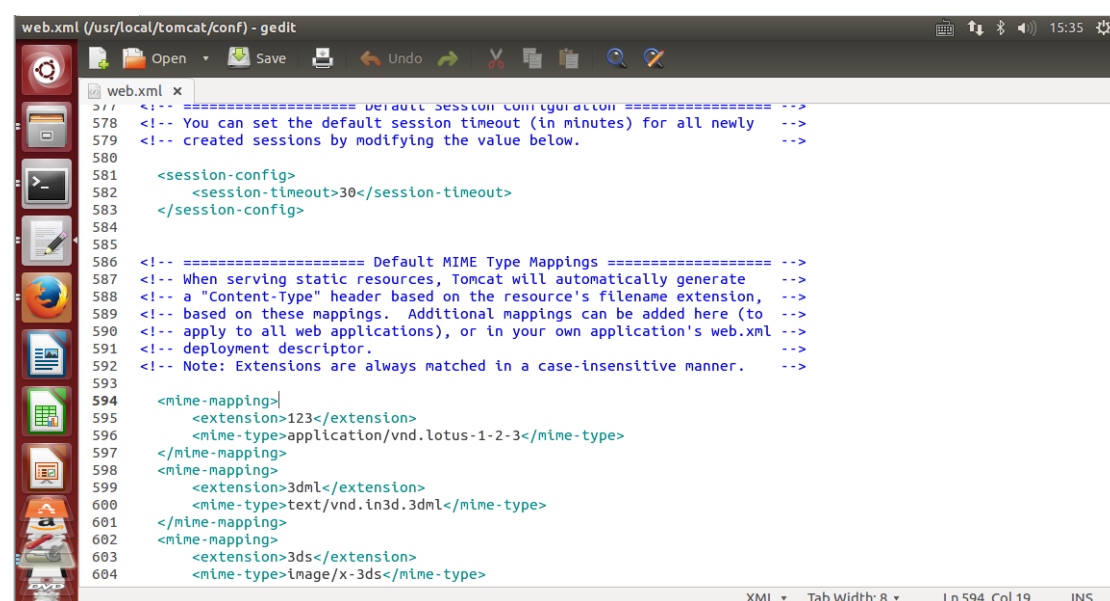
```

六、服务器部署

我们使用的服务器环境是：部署在本机 Linux Ubuntu14.041 虚拟机上的 Tomcat 环境，后端编译环境为集成 Tomcat 的 eclipse，在虚拟机上的文件结构如下：

```
mlyang@mlyang-virtual-machine:/usr$ cd local/  
mlyang@mlyang-virtual-machine:/usr/local$ ls  
apache-tomcat-8.5.8    bin  games  lib  sbin  src  
apache-tomcat-9.0.0.M13  etc  include  man  share  tomcat  
mlyang@mlyang-virtual-machine:/usr/local$ cd tomcat  
mlyang@mlyang-virtual-machine:/usr/local/tomcat$ ls  
apache-tomcat-9.0.0.M13  conf  LICENSE  NOTICE  RUNNING.txt  webapps  
bin                      lib  logs     RELEASE-NOTES  temp          work  
mlyang@mlyang-virtual-machine:/usr/local/tomcat$
```

Tomcat 的 web 端配置文件在 conf 文件夹中，部署在 webapps 中，Tomcat 会自动配置。由于我们的服务器只是应用于安卓端的特定应用 ZjuPdf 阅读器，所以不需要部署服务器前端界面。我们的服务器 session 部署文件如下：



```
web.xml (/usr/local/tomcat/conf) - gedit  
577 <!-- ===== Default Session Configuration ===== -->  
578 <!-- You can set the default session timeout (in minutes) for all newly -->  
579 <!-- created sessions by modifying the value below. -->  
580  
581 <session-config>  
582 <session-timeout>30</session-timeout>  
583 </session-config>  
584  
585  
586 <!-- ===== Default MIME Type Mappings ===== -->  
587 <!-- When serving static resources, Tomcat will automatically generate -->  
588 <!-- a "Content-Type" header based on the resource's filename extension, -->  
589 <!-- based on these mappings. Additional mappings can be added here (to -->  
590 <!-- apply to all web applications), or in your own application's web.xml -->  
591 <!-- deployment descriptor. -->  
592 <!-- Note: Extensions are always matched in a case-insensitive manner. -->  
593  
594 <mime-mapping>  
595 <extension>123</extension>  
596 <mime-type>application/vnd.lotus-1-2-3</mime-type>  
597 </mime-mapping>  
598 <mime-mapping>  
599 <extension>3dml</extension>  
600 <mime-type>text/vnd.in3d.3dml</mime-type>  
601 </mime-mapping>  
602 <mime-mapping>  
603 <extension>3ds</extension>  
604 <mime-type>image/x-3ds</mime-type>
```

session 的部署内容比较多，这里不全部截取了。之后是 webserver 的内容部署，sever 的部署在 tomcat 对应的 config 配置文件下的 server.xml 中配置即可，编写好后 tomcat 会自动识别并且构建，具体地文件内容如下：

```
server.xml (/usr/local/tomcat/conf) - gedit
server.xml x
53 <Service name="Catalina">
54
55 <!--The connectors can use a shared executor, you can define one or more named thread pools-->
56 <!--
57 <Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
58     maxThreads="150" minSpareThreads="4"/>
59 -->
60
61
62 <!-- A "Connector" represents an endpoint by which requests are received
63 and responses are returned. Documentation at :
64 Java HTTP Connector: /docs/config/http.html
65 Java AJP Connector: /docs/config/ajp.html
66 APR (HTTP/AJP) Connector: /docs/apr.html
67 Define a non-SSL/TLS HTTP/1.1 Connector on port 8080
68 -->
69 <Connector port="8080" protocol="HTTP/1.1"
70     connectionTimeout="20000"
71     redirectPort="8443" />
72 <!-- A "Connector" using the shared thread pool-->
73
74 <Connector executor="tomcatThreadPool"
75     port="8080" protocol="HTTP/1.1"
76     connectionTimeout="20000"
77     redirectPort="8443" />
78
79 <!-- Define a SSL/TLS HTTP/1.1 Connector on port 8443
80     This connector uses the NIO implementation with the JSSE engine. When
```

最后是对用户的部署，添加一个用户即可：

```
*tomcat-users.xml (/usr/local/tomcat/conf) - gedit
*tomcat-users.xml x
1 <?xml version="1.0" encoding="UTF-8"?>
2
3
4
5 <tomcat-users>
6 <user xmlns="http://tomcat.apache.org/xml"
7     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
8     xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
9     version="1.0"/>
10
11 <!--
12 <role rolename="tomcat"/>
13 <role rolename="role1"/>
14 <user username="tomcat" password="<must-be-changed>" roles="tomcat"/>
15 <user username="both" password="<must-be-changed>" roles="tomcat,role1"/>
16 <user username="role1" password="<must-be-changed>" roles="role1"/>
17 -->
18
19
20
21 </tomcat-users>
22
23
24 <tomcat-users>
25     <user name="admin" password="admin" roles="admin-gui,manager-gui" />
26 </tomcat-users>
```

七、工程中遇到的问题

本次实验中遇到的问题还是很多的，在应用管理部分涉及到很多的 Android 系统级的操作，比如关机重启锁屏等操作，以及定时器的设置、权限的管理等等。这些操作在 Android 早期的版本中还是有对应的接口的，但是随着 Android 版本的升级，在 Android 21 版本之后出于安全因素的考虑，这些接口被删除掉了。需要以各种其他方式对其实现。

Android 权限问题：

首先是权限问题，我们的 APP 定位在用户级别，尽管在 AndroidManifest.xml 中的显式声明权限，但是一些 ROOT 权限下的命令还是不能直接被允许的。我们到网上查阅了大量的相关资料，尝试了注册系统权限，发送系统广播、Runtime 调用 Linux 脚本以及反射调用 PowerManagerService 的隐藏接口等等方式。在实践的过程中，发现反射调用和系统广播的方式可以很好的实现这些 ROOT 权限的功能的。其中反射调用的方式可以调用一些 API 隐藏的接口，比如 ServiceManager 这个类是隐藏的，通过两次反射调用就可以调用 Power 服务了。另外在注册权限的时候，最开始对于 Android 的权限并不是很清楚，只好按照网上的各种博客以及官方文档去按部就班的实现，但是发现一些方式在本机上根本不能运行，比如共享系统进程 ID 的方式等。所以我们又重新从 Android 官方文档权限说明入手，在理解了 Android 权限机理的基础上重新配置了应用的权限，这次主要从 UserID 的进程级别考虑，修改 sharedUserId 的 userID，另外又显示声明了一些权限。通过不断的查阅资料和尝试，最后终于能够实现本应用所需的 system_call 层面的功能。

获取应用使用信息：

在实现应用信息获取功能时，查阅资料会发现国内的网站上基本没有现成的代码或者例子可以使用，了解到 Android SDK 在 21 level 以后将很多接口隐藏了起来或者改变，所以网上很多的方法并不适用。查阅资料发现 Android SDK 在 21 level 以后提供了一个查阅应用信息的接口，所以便从官方文档提供的函数和接口开始了尝试，所以应用信息获取这方面的功能基本上是自己摸索出来的。

首先便是应用信息权限的获取，获得这个权限并不需要 root 身份，但是除了在 manifests 配置相应的权限外，还需要用户在首次打开 app 时手动设置相应的权限，否则无法通过 SDK 提供的接口获取应用的使用时间的统计。

在获取应用信息时，刚开始会发现统计的应用信息并不准确，或者说不精确。因为会发现利用 SDK 获取的应用时，同一个应用可能会出现多个，或者获取得到的应用所在的时间段与调用 API 设置的时间段并不相符。后来经过查阅资料和自己的推理发现其内部储存应用使用时间并不是连续储存的，而是分段储存的。在本次开机时间内应用的使用情况会储存在一个 UsageStats 中。当用户将手机关机或者重启以后会重新用一个 UsageStats 储存同一个应用在本次开机的使用情况。所以如果说统计当天的数据，由于连个 UsageStats 虽然时间同一个应用，但是由于时间范围仍符合，所以在得到的 list 中便会显示出同一个名称的应用数据有多个。所以如果不加处理，会得到同一个应用的多个或者三个使用时间。即使手机不关机，在同一天内，其在统计数据时仍会将应用的数据分为多个 UsageStats，所以要得到应用当天真正的使用数据，需要首先判断一个 UsageStats 中储存数据的起始时间是否符合要求，随后将各个 UsageStats 中包名相同的合并，并排除掉使用时间为 0 的数据，才是我们真正想要的结果。得到一周内，一月，甚至一年的使用时间的数据方法同一天类似。

在获取应用的使用次数时又遇到了新的问题，发现在 Android SDK 21 level 后将

很多接口都隐藏了起来，并不能直接得到，这其中就包括应用使用次数获取的功能。后来在翻墙在国外的网站上发现可以通过“反射”的方法获取一个应用的使用次数，但是获取的使用次数也仅限于本次开机后的操作次数。

图表的绘制：

SDK 并没有提供相应的图表绘制功能，但是对于我们的应用数的直观性和可读性异常重要，便引用了 github 一个开源的图表绘制项目：MPAndroidChart，主要绘制了扇形图和条形统计图。接下来花了很长一段时间通过阅读 MPAndroidChart 提供的文档将这两种图表的绘制添加到我们的应用当中，这段期间的工作主要是对文档的阅读和学习。

服务器端问题：

在服务器端也遇到一些问题，起初我们准备搭建 windows 下的服务器，后来发现 windows 服务器开发 java 等后端不太合适，而且配置也比较麻烦，所以我们选择了在 Linux 下配置 tomcat 后端环境，结合 eclipse 编译环境，使用 java 进行开发。最开始出现服务器不能正常运行的问题，在外网不能正常访问服务器，只能通过本地访问。但是由于我们开发的应用需要使用外网访问服务器，所以必须使虚拟机上的服务器能够通过外网访问。最后我们通过镜像端口转发的方式，将物理主机上的 socket IP+8080 端口直接镜像转发到虚拟机的局域网 socket 即局域网 IP+8080 端口，这个问题才得以解决。

实际细节处理：

实际开发中遇到的问题还有很多，比如在设计 app 整体架构的时候，由于所有的代码都是我们自己原创的，没有参考任何网络上的资源，所以很多细节必须开动脑筋来处理。就比如引导页面的提示，为了提示引导页而且又不以数字的形式简单的表示处理，

我们就使用了“图片”资源，不断刷新“透明白点”这样一张图片，然后很好的表示出了当前的页面范围。而且这个做法在后边的很多模块也同样的用到了，比如需要显示按钮的按下变化等等。

八、完成度

经过了为期近两周的集中式开发，我们如期的按照开题报告中的功能模块完成了开发。我们将开题报告中的各个功能模块进行了整合，实现了三个大的功能模块，分别是应用统计图谱、应用详细信息统计、应用管理三部分，其中统计图谱部分如期的制作了时间维度上的各类动态统计图，详细信息统计部分精确的统计应用的使用时间和使用次数，最后的应用管理模块如期的实现了使用限制、超时惩罚、限时挑战的功能。其中惩罚类型的关机、重启、锁屏挑战也都很好的完成了。除此之外，应用从整体的架构和 UI 的设计上我们也进行了完善，设计了很多的 UI 和动画效果，仿照市面上成熟的 APP 设计了启动页、引导页、侧滑菜单等等页面，也自行设计了 Logo，完善了消息提醒、信息提示等细节功能。我们认为，本次 APP 征战手机的设计的完成度是非常高的，较开题要求而言甚至额外设计了一些 UI 和功能部件。

项目分工：

杨明亮：整体架构、UI 及应用管理（通知，关机，锁屏，重启等操作）。

董泰佑：应用详细使用信息统计、应用图表绘制（扇形图和条形图等）

服务器配置共同实现

难点参考网站：

<https://developer.android.com/reference/android/app/usage/package-summary.html>

<https://github.com/PhilJay/MPAndroidChart>

<http://www.cnblogs.com/jico/articles/1838293.html>

<http://blog.csdn.net/superkris/article/details/7709504>

<http://blog.csdn.net/z1074971432/article/details/37940701>