一、获取应用本地版本号及其它信息

通过创建包管理器对象

```
PackageManager  pm =  context.getPackageManager();
```

通过包管理对象获取包信息

参一：应用包名 可以通过context.getPackageName()获取

参二：默认为0 获取应用的所有信息

```
PackageInfo  info=pm.getPackageInfo(context.getPackageName(),  0);
```

通过应用包对象获取应用相关配置信息

```
info.versionCode;
```

包信息说明：

锁定屏幕是横屏还是竖屏

Android开发中只需在AndroidManifest.xml里面配置一下就可以了，加入这一行

android:screenOrientation="landscape" 例如(landscape是横向，portrait是纵向)。

```
<activity android:name=".Activity.WelcomeActivity" android:screenOrientation="portrait">
```

二、获取服务器上应用版本号及其它信息

xUtils 库的主要功能：联网，视图，图片，数据库

*通过第三方开源工具xUtils 进行网络连接，获取HttpUtils对象

```
HttpUtils  utils  =  new  HttpUtils(3000);
```

构造参数：网络连接超时时间

*通过HttpUtils的方法请求网络连接

参一：请求网络访问方式 GET、POST.....

参二：请求网络访问路径

参三：方法请求的回调 new RequestCallBack<String>

```
utils.send(HttpMethod.GET,  Cons.UPDATE_URL,new  RequestCallBack<String>()
```

*重写请求回调方法

```
onSuccess(ResponseInfo<String>  responseInfo)
```

表示网络访问连接成功

```
String  result  =  responseInfo.result;
```

----->获取服务器返回的数据

操作处理数据 result

```
onFailure(HttpException  error,  String  msg)
```

表示网络访问连接失败

三、去除标题栏信息的三种方式

*方案一：在代码中设置 在onCreate方法中请求没有标题 且只能在只能在setContentView之前

requestWindowFeature(Window.FEATURE_NO_TITLE); 这个方法只能作用于当前的Activity

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // 请求没有标题:只能在setContentView之前

    // requestWindowFeature(Window.FEATURE_NO_TITLE);
    // 设置布局文件
```

*方案二：在清单文件中修改主题 在AndroidManifest.xml文件的 <application 下的<activity

增加一条 android:theme="@android:style/Theme.Black.NoTitleBar"

这个方法会改变应用主题的背景颜色

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme"
    android:debuggable="true">
    <activity
        android:name="org.ljr.mobilesafe.SplashActivity"
        android:label="@string/app_name"

        android:theme="@android:style/Theme.Black.NoTitleBar"

        >
```

*方案三：最好的一种方法 在styles.xml文件中的主题下<!-- Application theme. -->写

```
<!-- notitle -->
  <item name="android:windowNoTitle">true</item>
```

styles.xml

```
<!-- Application theme. -->
<style name="AppTheme" parent="AppBaseTheme">

    <!-- All customizations that are NOT specific to a particular API-level can go
    <!-- notitle -->
    <item name="android:windowNoTitle">true</item>
</style>
```

四、延迟几秒进入新界面

1、先创建一个跳转界面的方法

2、创建一个延时的加载的方法，这个方法可以通过

Message message = Message.obtain(); 创建消息

mHandler.sendMessageDelayed(message,3000); 延迟3秒发送消息

3、创建Handler mHandler=new Handler() 重写handleMessage(Message msg)方法

在handleMessage（）方法中调用跳转界面方法

4、需要进入界面时调用延时加载的方法，即第二步的方法。

五、下载文件或应用到手机上（安装）如果是下载到SD卡要先判断SD卡是否挂载

判断方法：

```
if(Environment.getExternalStorageState().equals(Environment.MEDIA_UNMOUNTED))
```

没有挂载

*通过第三方开源工具xUtils 进行网络连接，获取HttpUtils对象

```
HttpUtils utils = new HttpUtils(3000);
```

构造参数：网络连接超时时间

*设定下载存储路径

```
Environment.getExternalStorageDirectory().getAbsolutePath() + "/" + System.currentTimeMillis() + ".xxx";
```

*开始下载

参一：访问服务器路径

参二：下载存储路径

参三：下载完成回调方法

```
utils.download(Cons.DOWNLOAD_URL,target,new RequestCallBack<File>()
```

*重写回调方法

```
onSuccess(ResponseInfo<File> responseInfo)
```

下载成功

安装应用

```
onFailure(HttpException error,String msg)
```

下载失败

```
onLoading(long total,long current,boolean isUploading)
```

下载进度条任务设定

如果有用下载进度条重写此方法 onLoading方法

参数一：文件的下载长度 参数二：当前下载进度 参数三：是否上传文件

进度条对象.setMax((int)total); 进度条对象.setProgress((int)current);

关于应用的安装：采用隐式意图调用系统完成

responseInfo是下载成功的文件资源

```
Intent  intent  =  new  Intent();
intent.setAction("android.intent.action.VIEW");
intent.addCategory("android.intent.category.DEFAULT");
intent.setDataAndType(Uri.fromFile(responseInfo.result),  "application/vnd.android.package-archive");
startActivityForResult(intent,  REQ_INSTALL);
```

六、更新提示对话框、登录注册对话框、下载进度条对话框　　AlertDialog.Builder类实现

下载进度条对话框

```
ProgressDialog  dialog  =  new  ProgressDialog(this);       用于下载的进度条对象
dialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);    设置进度条样式为水平
dialog.setMax((int)  total);        设置进度条的任务量或者说是最大值
dialog.setProgress((int)  current);     设置进度条的当前下载位置
dialog.show();       最示进度条
dialog.dismiss();     隐藏进度条
```

自定义大小的对话框方案

```
//获取layout布局
  public static View inflate(int layoutRes, ViewGroup parent) {
      return LayoutInflater.from(getContext()).inflate(layoutRes, parent,false);
}
//对话框
public void showDialog(float scale) {
            // 创建对话框对象
            Dialog dialog = new Dialog(this);
            // 设置标题
            dialog.setTitle("提示信息");
            // 给对话框填充布局
           dialog.setCancelable(false);
            //设置主体对话框内容
           View view = Utils.inflate(R.layout.abnormaldispose_view, null);
           //查找控件设置相关事件
           //TODO


            dialog.setContentView(view);
            // 获得当前activity所在的window对象
            Window window = dialog.getWindow();
            // 获得代表当前window属性的对象
            WindowManager.LayoutParams params = window.getAttributes();
            params.width = 300; // 宽度    px值
            params.height = 300; // 高度
            // 方式一：设置属性
            window.setAttributes(params);
            // 方式二：当window属性改变的时候也会调用此方法，同样可以实现
            // dialog.onWindowAttributesChanged(params);
            dialog.show();
      }
```

更新提示对话框

```
AlertDialog.Builder dialog = new AlertDialog.Builder(this);     用于更新提示对话框对象
dialog.setCancelable(false);          设置对话框点击外面，不会消失取消
dialog.setTitle("是否更新？");       设置对话框标题
dialog.setMessage("版本升级提示");     设置对话框提示内容
dialog.setIcon(android.R.drawable.ic_dialog_info)    设置标题图片    可选项
dialog.setView(new EditText(this));      设置对话框的输入框      可选项
设置单选对话框（不可与setMessage、setView共存）可选项
dialog.setSingleChoiceItems(new String[] {"选项1","选项2","选项3","选项4"}, 0,
    new DialogInterface.OnClickListener() {
  public void onClick(DialogInterface dialog, int which) {
            dialog.dismiss();
        }
   });
设置多选对话框     可选项
dialog.setMultiChoiceItems(new String[] {"选项1","选项2","选项3","选项4"}, null, null);
设置按钮升级或取消 的点击事件
如果有多个对话框时要注意区分
android.content.DialogInterface.OnClickListener()   包名以避免导包错误
dialog.setPositiveButton("升级",new android.content.DialogInterface.OnClickListener(){
    @Override
  public void onClick(DialogInterface dialog, int which) {
   设置 相关点击后的事件
  Toast.makeText(getApplicationContext(), "升级成功", 0).show();
            dialog.dismiss();
     }
  });
设置取消按钮
dialog.setNegativeButton("取消", new android.content.DialogInterface.OnClickListener(){
    @Override
     public void onClick(DialogInterface dialog, int which) {
设置 相关点击后的事件
   Toast.makeText(getApplicationContext(), "升级取消", 0).show();
隐藏对话框
dialog.dismiss();
  }
});
显示对话框
dialog.show();
```

登录注册对话框   需要一个XML文件设置到AlertDialog

```
AlertDialog.Builder dialog = new AlertDialog.Builder(this);      对话框对象
final AlertDialog create = dialog.create();                  真正可以设置XML对话框的对象
View view = View.inflate(this, R.layout.log_in, null);       采用打气筒获得XML文件
Button ko1 = (Button)view. findViewById(R.id.btu1);          获取xml文件中的按钮
Button ko2 = (Button)view. findViewById(R.id.btu2);
ko1.setOnClickListener(new OnClickListener() {       设置按钮的点击事件
   @Override
  public void onClick(View v) {      //登录
Toast.makeText(getApplicationContext(), "恭喜你登录成功", 0).show();
```

```
        create.dismiss();       隐藏对话框
    }
});
ko2.setOnClickListener(new  OnClickListener()  {    //  取消
@Override
  public  void  onClick(View  v)  {
        create.dismiss();
  Toast.makeText(getApplicationContext(),  "恭喜你登录取消",  0).show();
  }
});
create.setView(view);        将XML设置到对话框中去
create.show();                显示对话框
```

关于XML文件

中定义好相关的内容：对话框标题、提示或输入框、按钮等元素。

七、跑马灯及自定义UI组件

1、跑马灯（普通TextView组件）在布局文件中设置

*设定跑马灯属性：

```
android:singleLine="true"
android:ellipsize="marquee"
```

*设定循环次数：

```
android:marqueeRepeatLimit="marquee_forever"
```

*设定UI组件获取焦点：

```
android:focusable="true"
android:focusableInTouchMode="true"
```

2、自定义TextView组件的跑马灯

*创建一个类继承TextView重写六个方法

三个构造方法分别是：一个参数 二个参数 三个参数的构造方法

一参构造：一般是代码创建创建UI组件是调用

二参构造：在布局文件中创建控件调用

三参构造：一般不用理会

三个功能方法：

```
isFocused()      欺骗系统该组件一直有焦点时
onFocusChanged(boolean  focused,int  direction,Rect  previouslyFocusedRect)    当焦点发生改变时
onWindowFocusChanged(boolean  hasWindowFocus)当窗体发生改变时
```

方法说明：

*在 二参构造 方法中定义跑马灯属性

```
设置TextView只能显示单行
setSingleLine(true);
设置跑马灯
setEllipsize(TextUtils.TruncateAt.MARQUEE);
设置获得焦点
setFocusable(true);
setFocusableInTouchMode(true);
跑马灯无限循环跑
setMarqueeRepeatLimit(-1);
```

*在布局中创建自定义UI组件

```
 <org.ljr.mobilesafe.view.MarqueeTextView     自定义组件类全名     核心点
            android:id="@+id/tv_main_des"      核心点
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="大家好或者不好"
```

八、属性动画

target：需要做动画的View的id
propertyName：属性名字
位移 translationY或translationX　　透明度：alpha
缩放：scaleX或scaleY　　旋转：rotationY或rotationX
values：做动画值
ObjectAnimator animator = ObjectAnimator.ofFloat（参一，二，三）；
设置时间：setDuration（ 时间值毫秒 ）；
开始动画：start()；
无限循环：
setRepeatCount(ObjectAnimator.INFINITE);
setRepeatMode(ObjectAnimator.REVERSE);

九、selector选择器和shape标签

xml文件创建方法：

（shape是图片，可以用来绘制自定义的图片，selector选择器是用来做点击时的背景图或背景颜色的改变）
selector选择器创建方法：在res------>新建目录 drawable --------->新建 Android xml 文件 ---------->选择selector
文件内容：
<item android:drawable="@drawable/图片名" android:state_selected="true"/>　　点击时图片状态的改变
<item android:drawable="@drawable/图片名" android:state_selected="false"></item>
state_pressed="true" 与 state_selected="true" 区别
pressed 是点击背景颜色发生变化　　selected 是点击背景图片改变 两种状态不同
用法：
UI组件 android:background="@drawable/选择器名" 做为点击改变背景颜色的用法
UI组件 android:drawableRight="@drawable/btn_lock_selector" 做为点击改变组件图片的用法
代码中可以通过判断不同的情况来用UI组件ID来设置显示的图片 setSelected(false/true);

shape的形状，默认为矩形，可以设置为矩形（rectangle）、椭圆形(oval)、线性形状(line)、环形(ring)
shape创建方法：在res------>新建目录 drawable --------->新建 Android xml 文件 ---------->选择shape
文件内容： 通过以下方法制作自己相要的图片做为背景使用

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval" >
    <!-- 填充颜色 -->
    <solid android:color="@color/White" />
    <!-- 指定半径 -->
    <corners android:radius="16dp" />
    <!--描边 -->
    <stroke android:color="#a0f0" android:width="2dp"/>
    <!--颜色渐变 -->
    <gradient
        android:startColor="#f00"
        android:centerColor="#0f0"
        android:endColor="#00f"
        android:angle="0"
         />
    <!-- 间距 -->
    <padding
        android:bottom="8dp"
        android:left="8dp"
        android:right="8dp"
        android:top="8dp" />
</shape>
```

用法：将做好的图片（xml文件）放入选择器中做为一种点击或不点击时的背景颜色样式的状态。 也可以直接设置为背景使用android:background="@drawable/intercep_setting"

代码创建方法：

通过代码shape绘制图片Drawable　调用该方法 传入颜色值，圆角值

```
public static Drawable getShape(int rgb, int radius) {
    //渐变的Drawable
    GradientDrawable gradientDrawable=new GradientDrawable();
    //设置形状 矩形
    gradientDrawable.setShape(GradientDrawable.RECTANGLE);
    //设置颜色
    gradientDrawable.setColor(rgb);
    //设置矩形圆角半径
    gradientDrawable.setCornerRadius(radius);
    return  gradientDrawable;
  }
```

通过代码selector绘制图片Drawable的选择器　返回一个状态图片

normalDrawable ：默认的颜色　绘制一张默认的图片的状态

pressedDrawable ：按下去的颜色　绘制一张按下去的图片状态

```
public static Drawable getStateListDrawable(Drawable normalDrawable, Drawable
pressedDrawable) {
    //状态列表的Drawable ： 图片选择器
    StateListDrawable stateListDrawable=new StateListDrawable();
    stateListDrawable.addState(new int[]
{android.R.attr.state_pressed,android.R.attr.state_enabled},pressedDrawable);
    stateListDrawable.addState(new int[]{android.R.attr.state_checked},pressedDrawable);
    stateListDrawable.addState(new int[]{},normalDrawable);
    return  stateListDrawable;
  }
```

十、自定义组合控件

1 、写一个布局文件xml

xml布局文件可以有任意结构的UI组件

2 、写个类 extends LinearLayout（addView）继承任意布局

3、在构造参数为二个的构造方法中找到XML文件　xml 转换 View

通过打气筒填充获取　mView = (ImageView)View.inflate(context, R.layout.xml文件, null);

通过findViewById找到XML文件中的UI组件

4、把转换好的View添加到自定义的控件中去　addView(mView);

十一、自定义组合控件属性设定

1 先定义组件规则，在values目录新建一个attrs.xml

这个规则的写法可以参考系统的attrs.xml文件：Sdk\platforms\android-19\data\res\values\attrs.xml

文件内容：

```
<declare-styleable name="SettingItemView">      自定义控件类名
        <!-- 文本属性名，可以给什么样的值-->
        <attr name="siv_text" format="string" />     规则中的属性名称
        <!-- 开关:是否是开，或者关 -->
        <attr name="siv_switch" format="boolean" />
```

```
        <!-- 是否可见 -->
        <attr name="siv_visible" format="boolean" />
        <!-- 背景 -->
        <attr name="siv_bg">
            <enum name="first" value="0" />
            <enum name="middle" value="1" />
            <enum name="last" value="2" />
        </attr>
    </declare-styleable>
```

2、 声明完之后：系统在R文件中：Android系统做　这个不用管看一下就可以了。

R.styleable.SettingItemView(数组)

R.styleable.SettingItemView_siv_text

3 、在自定义的组合控件的构造器里面获取属性的值

获取TypeArray

TypedArray　ta=context.obtainStyledAttributes(attrs,R.styleable.SettingItemView) 自定义控件类名

这个名称要与规则中定义的名称保持一致

4、获取属性值

String text =ta.getString(R.styleable.SettingItemView_siv_text); 规则中的属性名称

ta.getBoolean(R.styleable.SettingItemView_siv_switch,　false);　默认值

ta.getInt(R.styleable.SettingItemView_siv_bg,　-1);　默认值

5、设置属性值

```
mTvText.setText(sivText);
设置有没有选择
setCheck();
设置是否可见
setVisible();
设置背景
setBackGround();
设置可点击
setClickable(true);
```

6、 回收TypeArray： ta.recycle();

核心属性： 这些属性最核心的地方主要依据就是规则怎么定的，xml怎么写的

View.*INVISIBLE*与View.*GONE*的区别

GONE 会隐藏并删除相应***UI***的位置

INVISIBLE 只会隐藏不会删除***UI***位置

```
setVisibility(View.VISIBLE);
setVisibility(View.INVISIBLE);
```

设置ImageView是否可见　visibility可见或不可见

判断View是GONE还是VISIBLE的方法：

```
view.getVisibility()== View.VISIBLE
view.getVisibility()== View.GONE
```

为true则是显示的

```
setBackgroundResource(R.drawable.first_bg_selector);
setImageResource(R.drawable.on);
```

设置点击后的背景和颜色

设置点击后的图片状态

完成后的使用： 命名空间

在布局中创建自定义UI组件在布局中加了XMLNS后才可以使用自定义的属性

```
 xmlns:自定义名称="http://schemas.android.com/apk/res/应用包名"
 xmlns:liujinrong="http://schemas.android.com/apk/res/org.ljr.mobilesafe"
 <org.ljr.mobilesafe.view.类名
liujinrong:siv_bg="last"
liujinrong:siv_text="开关拦截骚扰"
```

自定义组件类全名　核心点

attrs.xml　文件中定义的属性名

十二、在布局中画一条线 listview分割线

在layout目录里新建一个XML文件选择View

```
<View xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"      线的长度
    android:layout_height="2dp"    线的宽度
    android:background="#ff0000" >    线的颜色
</View>
```

用法：

在需要用的布局中通过：<include layout="@layout/xml文件名"/>" 来使用

将一个布局也可以通过在另一个布局中直接引用 <include layout="@layout/xml文件名"/>"

listview分割线在listview组件中直接写

android:divider="#FF0000" 线的颜色

android:dividerHeight="3dp" 线的宽度

android:scrollbars="none" android:fadeScrollbars="true" 设置隐藏滚动条

十三、ProgressBar与图片组合动画

1、在drawable目录下创建一个XML文件选择rotate类型

2、文件内容为运画

```
<!-- 进度旋转条动画 -->
<rotate xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/loading"      加载一张图片
    android:pivotX="50%"    以图片的x轴的50
    android:pivotY="50%"     以图片的Y轴的50
    android:fromDegrees="0"    从多少度开始
    android:toDegrees="2880">      到多少度结束
</rotate>
```

3、在布局中定义进度条　　布局必须要有ID号以便快速使用

```
<ProgressBar    默认为园形
   android:layout_width="150dp"
   android:layout_height="150dp"
   android:indeterminateDrawable="@drawable/pb_londing" />  将动画的XML设置给进度条
```

4、在另外一个布局中引用<include layout="@layout/loading" />

5、关于使用

引用的进度条需要在代码中找到通过布局ID来找

```
View  mPbLong  =  findViewById(R.id.pd_lond);   此ID是布局的id
```

一个布局的可见与不可见设置：setVisibility(View.VISIBLE)   setVisibility(View.GONE)不可见

在**FrameLayout**布局中**ListView**与**ImageView**共存时当**ListView**没有内容显示时就自动显示**ImageView** 在代码中这样设置

```
ListView.setAdapter(mMyAdaapter 适配器);   加载数据
```

数据加载结束后没有数据设置图片可见，有数据图片不可见

```
ListView.setEmptyView(ImageView ID);
```

十四、界面切屏动画

1、在res目录下新建一个anim目录

2、在anim目录下一个xml文件选择translate 创建Activity切换动画 这个一般需要4个动画组合

动画内容：

```
<!-- 滑动屏幕的动画 -->
```

```
<translate xmlns:android="http://schemas.android.com/apk/res/android"
    android:fromXDelta="100%"    从屏幕x坐标的100%到x坐标0    向右滑动屏幕时
    android:toXDelta="0"
    android:duration="200"      动画持续时间毫秒
    android:fillAfter="true">     是否填充满窗体
</translate>
```

3、在代码中设置滑动屏幕的动画

enterAnim：进来Activty的动画资源id：0没有动画　　滑动屏幕进来的Activity

exitAnim：出去ACtivity的动画资源id：0没有动画　　滑动屏幕从当前屏幕出去的Activity

overridePendingTransition(R.anim.next_in, R.anim.next_out);

十五、手势动作识别监听　手势监听类GestureDetector

1、创建GestureDetector对象　参一：上下文　参二：手势监听器

```
GestureDetector gd =  new  GestureDetector(this,new  SimpleOnGestureListener(){
```

按下去第一点：e1　　快速滑动结束点：e2　　velocityX：x轴方向速度　　velocityY：y轴方向速度

```
@Override
public  boolean  onFling(MotionEvent  e1,  MotionEvent  e2,  float  velocityX,  float  velocityY)  {
int  x1=(int)  e1.getX();
int  x2=(int)  e2.getX();
if(x1-x2>50){
next(null);                    调用下一步跳转界面
return  true;
    }
if(x2-x1>50){
pre(null);                              调用上一步跳转界面
return  true;
      }
return  super.onFling(e1,  e2,  velocityX,  velocityY);
    }
});
```

2、简单涉及一些：滑动屏幕的事件分发机制

重写 onTouchEvent(MotionEvent event)方法

把事件交给GestureDetector处理

```
gd.onTouchEvent(event);
```

十六、获取手机SIM卡惟一标识

*获取电话管理对象TelephonyManager，通过系统务.getSystemService(Context.TELEPHONY_SERVICE)

```
TelephonyManager  Tm  =(TelephonyManager)this.getSystemService(Context.TELEPHONY_SERVICE);
```

*通过电话管理对象获取Sim卡的标识号

```
String SimNumber  =  Tm.getSimSerialNumber();
```

十七、R文件名的修改方法 整体修改应用的包名

应用右键Android　Tools------->Rename　application package　----->输入新的包名即可

十八、来电监听实现来电拦截，获取电话的三种状态

```
获取电话管理器
TelephonyManager mTm = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
电话状态监听类
MPhoneStateListener mListener = new MPhoneStateListener();
开启电话状态监听
mTm.listen(mListener, PhoneStateListener.LISTEN_CALL_STATE);
写类继承PhoneStateistener
onCallStateChange （三种状态）
权限：READ_PHONE_STATE
class MPhoneStateListener extends PhoneStateListener {
@Override
public void onCallStateChanged(int state, String incomingNumber) {
super.onCallStateChanged(state, incomingNumber);
switch (state) {
case TelephonyManager.CALL_STATE_IDLE:// 简单看成挂断，但是初始化的时候，会先调用一次
break;
case TelephonyManager.CALL_STATE_RINGING:// 响铃的状态
通过来电号码，查询黑名单数据库，查询类型（电话，全部）
int type = mbu.getType(incomingNumber);
if (type == Blacklistdb.CALL || type == Blacklistdb.ALL) {
需要电话拦截
endCall();
删除通话记录
deleteCallLog(incomingNumber);
case TelephonyManager.CALL_STATE_OFFHOOK:// 接通状态
break;
电话拦截方法
private void endCall() {
一波三折：ITelephony----endCall方法 ITelephony iTelePhony=new ITelephony（）；
iTelePhony.endCall
aidl:一波 ServiceManger：ServiceManager：public static IBinder
getService(String name):三折
IBinder binder = null;
反射 找到class
try { 以下代码需要try{}catch{}处理一下
Class<?> clss = Class.forName("android.os.ServiceManager");
找到对象 找到方法
Method method = clss.getMethod("getService", String.class);
4 调用方法
binder = (IBinder) method.invoke(null, Context.TELEPHONY_SERVICE);
获取拦截挂断电话的方法
ITelephony iTelePhony = ITelephony.Stub.asInterface(binder);
iTelePhony.endCall();
删除通话记录
public void deleteCallLog(final String phone) {
final ContentResolver cr = getContentResolver();
cr.registerContentObserver(CallLog.Calls.CONTENT_URI, true, new ContentObserver(new Handler()) {
@Override
public void onChange(boolean selfChange) {
super.onChange(selfChange);
反注册 cr.unregisterContentObserver(this);
监听到内容发生变化，就可以删除记录
cr.delete(CallLog.Calls.CONTENT_URI, "number=?", new String[] { phone });
```

十九、删除通话记录的方法、拨打电话的广播监听件事

1、内容观察者：检测通过记录的数据库有没有添加数据，有删除

2、获取内容观察者 final ContentResolver cr = getContentResolver();

3、注册内容观察

```
cr.registerContentObserver(CallLog.Calls.CONTENT_URI, true, new ContentObserver(new Handler()) {
@Override
public void onChange(boolean selfChange) {
super.onChange(selfChange);
```

4、用完反注册

```
cr.unregisterContentObserver(this);
```

5、 监听到内容发生变化，就可以删除记录

```
cr.delete(CallLog.Calls.CONTENT_URI, "number=?", new String[] { phone });
    }
});
```

拨打电话的广播监听件事

1、注册外播电话监听广播

```
OutGoingReceiver mReceiver = new OutGoingReceiver();
IntentFilter filter=new IntentFilter();
filter.addAction(Intent.ACTION_NEW_OUTGOING_CALL);
registerReceiver(mReceiver, filter);
```

2、定义一个内部类继承OutGoingReceiver extends BroadcastReceiver

获取外播电话号码

```
String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);
```

查询号码归属

```
String cardType = AddressDao.getCardType(context, number);
```

显示查询号码归属的toast

```
mAddressToast.show(cardType);
```

二十、注解的作用，可以省略写 findViewById

写法给一个方法在onCreate方法中加载这个方法内写

```
ViewUtils.inject(this);
```

然后声明：

```
@ViewInject(R.id.et_phone)
private EditText etPhone;
```

Dagger2

```
apt 'com.google.dagger:dagger-compiler:2.0'
compile 'com.google.dagger:dagger:2.0'
```

BufferKnife

```
compile 'com.jakewharton:butterknife:6.1.0'
ButterKnife.inject(this);
@InjectView(R.id.butter_text_view_2)
TextView mTextView2;
```

BindView

```
@BindView(id = R.id.tv)
private TextView tv;
BindUtils.initBindView(this);//这里一定得放在setContentView()后
tv.setText("绑定成功...");
```

二十一、手机vibrator震动器的调用

```
//获取震动器服务
Vibrator mbrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
//震动器开始工作 运行时间 停止等待时间 运行时间 等待时间 次数
mVibrator.vibrate(new long[]{1000,2000,1000,3000}, -1);
```

二十二、判断获取服务运行状态 写一个方法来判断

```
方法参数（Context context, Class<? extends Service> clss）
//获取ActivityManager管理器
ActivityManager am =(ActivityManager)context.getSystemService(Context.ACTIVITY_SERVICE);
//通过管理器获取正在运行的服务集合
List<RunningServiceInfo> servicess = am.getRunningServices(Integer.MAX_VALUE);
//迭代集合
for (RunningServiceInfo runServiceInfo : servicess) {
//获取正在运行服务的名字
ComponentName componentName = runServiceInfo.service;
//判断正在运行的服务名与传入的服务类名是否相等
if(componentName.getClassName().equals(clss.getName())){
//返回判断结果
return true;
}
```

二十三、意图数据回传

A界面跳转到B界面，B界面操作完要返回数据给A界面

A界面操作：

```
Intent intent = new Intent(this,B界面.class);
startActivityForResult(intent,100); 跳转意图及请求码
重写onActivityResult(int requestCode, int resultCode, Intent data)方法
判断
if(requestCode==100){ 请求码
if(resultCode==120){ 返回码
String num = data.getStringExtra("number"); 获取返回数据
```

B界面操作：

```
Intent data = new Intent();
data.putExtra("number", 数据); 封装返回数据给意图
setResult(120, data); 返回码及返回数据
finish(); 关闭B界面
```

二十四、自定义Dialog样式及数据加载点击事件等

1、创建一个自定义类继承Dialog extends Dialog 并重写onCreate(Bundle savedInstanceState)方法

在方法中设置弹出样式setContentView(R.layout.样式布局文件);

```
拿到布局文件中的控件 为将来设置点击及加载数据做准备
ListView lvAddressDialog=(ListView) findViewById(R.id.lv_address_dialog);

设置这个样式的属性通过LayoutParams的属性完成
LayoutParams params = getWindow().getAttributes(); 拿到LayoutParams对象
params.width=LayoutParams.MATCH_PARENT; 设置宽高
```

```
params.height=LayoutParams.WRAP_CONTENT;
params.gravity=Gravity.BOTTOM|Gravity.CENTER_HORIZONTAL;     设置弹出的位置
```

2、完成后基本用法：

```
    继承Dialog类  addressDialog=new  继承Dialog类(SettingActiivty.this);
    addressDialog.show();
```

3、自定义样式及弹出动画等

在自定义继承Dialog类的构造方法中返回一个样式

```
public  自定义类(Context  context)  {       构造器
      super(context,R.style.样式名);
   }
```

4、写样式：样式来源于系统自带的通过复制修改放入**style**文件中即可，在构造中引用

```
<style name="AddressDialog" parent="android:Theme.Dialog">
注意一：  点击"android:Theme.Dialog"与系统 <style name="Theme.Dialog">对应
<item name="android:windowBackground">@android:color/transparent</item>
<item name="android:windowFrame">@null</item>
<item name="android:windowContentOverlay">@null</item>
<item name="android:windowAnimationStyle">@android:style/Animation.InputMethod</item>
注意二："android:windowAnimationStyle"表示弹出动画，用系统输入法的
<item name="android:backgroundDimEnabled">true</item>
<item name="android:windowIsTranslucent">true</item>
<item name="android:windowNoTitle">true</item>
注意三：样式标题,不需要可改为true
<item name="android:windowCloseOnTouchOutside">true</item>
注意四：弹出后窗口是否可点击，需要设为true
注意五：复制的代码没有android:前缀   需要为每一条加上，不然会报错
</style>
```

5、定义对外可点击及显示数据的方法，外面通过调用方法传入数据适配器对象及

```
public  void  setAdapter(BaseAdapter  adapter){      显示数据的方法
                this.adapter=adapter;
public  void  setOnItemClickLintener(OnItemClickListener  onItemClickListener){    可点击的方法
                this.onItemClickListener=onItemClickListener;
```

同时声明

```
private  BaseAdapter  adapter;     ListView加载数据的数据适配器对象
private  OnItemClickListener  onItemClickListener;    可点击的点击对象
```

6、在onCreate(Bundle savedInstanceState)方法设置控件点击事件及加载数据

```
if(adapter!=null){
   lvAddressDialog.setAdapter(adapter);
}
if(onItemClickListener!=null){
    lvAddressDialog.setOnItemClickListener(onItemClickListener);
}
```

7、外面界面可以通过某个UI控件点击后获取继承**Dialog**类对象调用继承**Dialog**类的方法弹出自定义对话框加载数据**Dialog**对话框的显示数据

```
继承Dialog类  addressDialog=new  继承Dialog类(SettingActiivty.this);
//为对话框添加数据
addressDialog.setAdapter(mAdapter);
//设置点击事件
addressDialog.setOnItemClickLintener(new  OnItemClickListener()  {
点击后的ListView更新mAdapter.notifyDataSetChanged();
addressDialog.show();    显示对话筐
```

addressDialog.dismiss(); 隐藏对话筐

二十五、应用获取或移除设备管理员权限的方法

1、要先创建一个自定义的类继承DeviceAdminReceiver

2、注册管理员广播

```
<!--  高级管理员权限注册的广播是自定义继承DeviceAdminReceiver的类两个string是权限的描述，xml是一个注册文件要重写  -->
    <receiver
    android:name="org.ljr.mobilesafe.receiver.SuperAdmin"          自定义广播类
    android:description="@string/sample_device_admin_description"      描述
    android:label="@string/sample_device_admin"          描述
    android:permission="android.permission.BIND_DEVICE_ADMIN"  >
    <meta-data
          android:name="android.app.device_admin"
          android:resource="@xml/device_admin_sample"  />
    <intent-filter>
        <action  android:name="android.app.action.DEVICE_ADMIN_ENABLED"  />
    </intent-filter>
</receiver>
```

3、写一个xml文件   在res目录下创建一个xml目录在创建一个名叫device_admin_sample的xml文件写以下内容

```
<device-admin  xmlns:android="http://schemas.android.com/apk/res/android"  >
       <uses-policies>
              <limit-password  />
              <watch-login  />
              <reset-password  />
              <force-lock  />
              <wipe-data  />
              <expire-password  />
              <encrypted-storage  />
              <disable-camera  />
       </uses-policies>
</device-admin>
```

4、创建激活管理员权限对象

```
DevicePolicyManager  Dm  =  (DevicePolicyManager)  getSystemService(Context.DEVICE_POLICY_SERVICE);
```

5、找到注册的管理员广播类

```
ComponentName  cn  =  new  ComponentName(this,  自定义广播类.class);
```

6、判断是否激活设备管理员 ，如果想移除管理员

```
Dm.isAdminActive(cn);      默认是没有激活的     Dm.removeActiveAdmin(cn);
```

7、激活管理员的方法

```
Intent  intent  =  new  Intent(DevicePolicyManager.ACTION_ADD_DEVICE_ADMIN);
  intent.putExtra(DevicePolicyManager.EXTRA_DEVICE_ADMIN,  广播对象     cn  );
  intent.putExtra(DevicePolicyManager.EXTRA_ADD_EXPLANATION,  "描述可以不要的");
  startActivityForResult(intent,  请求码     100 ); 如果没有返回处理可以不用请求码
```

如果有请求码那么一定有返回的处理

```
import android.app.admin.DeviceAdminReceiver;

public class SuperAdmin extends DeviceAdminReceiver {
```

```
<!-- 高级管理员权限注册 -->  <!-- 注册的广播是自定义继承DeviceAdminReceiver的类
两个string是权限的描述，xml是一个注册文件要重写 -->
<receiver                                                    广播类  描述说明
    android:name="org.ljr.mobilesafe.receiver.SuperAdmin"
    android:description="@string/sample_device_admin_description"
    android:label="@string/sample_device_admin"
    android:permission="android.permission.BIND_DEVICE_ADMIN" >
    <meta-data
        android:name="android.app.device_admin"              xml文件
        android:resource="@xml/device_admin_sample" />
    <intent-filter>
        <action android:name="android.app.action.DEVICE_ADMIN_ENABLED" />
    </intent-filter>

</receiver>
```

```
<device-admin xmlns:android="http://schemas.android.com/apk/res/android" >

    <uses-policies>
        <limit-password />

        <watch-login />
                                                    管理员xml文件
        <reset-password />

        <force-lock />

        <wipe-data />

        <expire-password />

        <encrypted-storage />

        <disable-camera />
    </uses-policies>

</device-admin>
```

```
DevicePolicyManager mDm = (DevicePolicyManager) getSystemService(Context.DEVICE_POLICY_SERVICE);
ComponentName mWho = new ComponentName(this, SuperAdmin.class);   //自定以管理员广播类

//判断是否激活
mBtnAdmin.setSelected(mDm.isAdminActive(mWho));
```

```
// 判断是否已经激活，假如没有激活，跳到设备管理员激活界面
Intent intent = new Intent(DevicePolicyManager.ACTION_ADD_DEVICE_ADMIN);
intent.putExtra(DevicePolicyManager.EXTRA_DEVICE_ADMIN, mWho);   数据
intent.putExtra(DevicePolicyManager.EXTRA_ADD_EXPLANATION, "大家好才是真的好，广州好迪");
startActivityForResult(intent, 100);
                                  请求码                描述
```

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    // 在onActivityResult处理，判断是否已经激活，显示不同界面          返回处理
    mBtnAdmin.setSelected(mDm.isAdminActive(mWho));
}
```

二十六、RadioGroup与RadioButton UI组件

RadioGroup mPsa = (RadioGroup)findViewById(R.id.pb_spa);

mPsa.getCheckedRadioButtonId(); 通过RadioGroup判断是否有选中RadioButton

返回-1就是没有选中

代码指定选中某一个RadioButton   mPsa.check(RadioButton的ID号);

二十七、CheckBox

UI组件 mKqcb = (CheckBox) findViewById(R.id.kaibf_cb_sp5);

代码中设置选中：mKqcb.setChecked(true);

判断是否选中：mKqcb.isChecked()

二十八、GridView九宫格

特殊的属性 用法与ListView相同另外两个：Spinner 下拉菜单 Gallery

android:numColumns="2" 分列展示

android:horizontalSpacing="3dp" 水平间距

android:verticalSpacing="3dp" 垂直间距

.setAdapter(): ArrayAdapter,SimpleAdapter,extends BaseAdapter

* getCount()
* getItem: null
* getItemId:
* getView:xml---View:View view=View.inflate(resid,null);

二十九、ExpandableListView 二级菜单的Listview用法与Listview类似

1、在布局文件中添加ExpandableListView控件写布局文件

```
<ExpandableListView
        android:id="@+id/el"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
    </ExpandableListView>
```

2、代码中获取UI并设置显示数据

```
ExpandableListView mEl = (ExpandableListView) findViewById(R.id.el);
setAdapter(new CommonNumAdapter());
```

3、写自定义类继承extends BaseExpandableListAdapter

重写 核心方法五个

getGroupCount() 获取组的数量

getChildrenCount(int groupPosition) 获取某组下子Item的数量

获取某组的View

View getGroupView(int groupPosition,boolean isExpanded,View convertView,ViewGroup parent)

获取某组下子Item的View

View getChildView(int groupPosition,int childPosition,boolean isLastChild,View convertView,ViewGroup parent

设置可以点击某子Item

isChildSelectable(int groupPosition, int childPosition)

3、写两个布局文件一个用于组的View,一个用于组下子Item的View

如：

```
View groupView=View.inflate(getApplicationContext(),R.layout.view_commonnum_group, null);
TextView tvGroup = (TextView) groupView.findViewById(R.id.tv_commonnum_group);
tvGroup.setText(CommonNumDao.getGroupText(getApplicationContext(), groupPosition));
return groupView;
```

4、设置点击事件

```
mEl.setOnChildClickListener(new OnChildClickListener() {
```

三十、ViewHolder的使用 listview滑动监听 分页加载

ViewHolder的使用

1、创建一个ViewHolder类，类中声明Item中UI组件的成员变量 如：public TextView name;

2、在getView(int position, View convertView, ViewGroup parent)方法中绑定convertView.setTag(houderView);

3、找到houderView.name = (ImageView) convertView.findViewById(R.id.name);

4、 如果已经有这个item已经创建　获取houderView = (HouderView) convertView.getTag();
5、用法　houderView.name.setText(appInfo.appName);

listview滑动监听
android:scrollbars="none"　android:fadeScrollbars="true" 设置隐藏滚动条

```
listview.setOnScrollListener(new OnScrollListener() {
public void onScrollStateChanged(AbsListView view, int scrollState) {
      firstVisibleItem　第一个可见的itme　visibleItemCount　屏幕可以显示的itme数
      totalItemCount　总的itme数
public void onScroll(AbsListView view, int firstVisibleItem, int visibleItemCount, int totalItemCount)
如果第一个可见项大于用户应用的长度
if (firstVisibleItem > mAppuser.size()) {
显示系统条目
mAppGrouping.setText("系统程序:" + mAppsys.size() + "个");
else {
显示用户条目
mAppGrouping.setText("用户程序:" + mAppuser.size() + "个");
```

分页加载
数据库查询的方式按每加载的条目查从offset位置开始每次查count条数据

```
rawQuery("select _id,phone,type from blacklist limit ?,?", new String[]
{String.valueOf(offset),String.valueOf(count)});
```

当滑动屏幕的时候判断最后一个可见项的位置
然后再重新查询加载此时的开始位置为集合的长度

三十一、GPS定位 火星坐标　GPS卫星定位坐标进行偏移计算后的坐标

1、获取位置服务
LocationManager mLm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
2、请求位置变化监听
参数　provider：定位的方式（GPS）　mimTime:最小更新位置时间（单位毫秒）
　　minDistance：最短更新距离(单位米)　LocationListener：位置的监听
mLm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 5 * 1000, 5, mListener);
3、创建一个类实现监听回调服务
class MLocationListener implements LocationListener{
4、在onLocationChanged(Location location)方法中获取经纬度
获取经度　double longitude = location.getLongitude();
获取纬度　double latitude = location.getLatitude();
5、短信发送获取的坐标
SmsManager.getDefault().sendTextMessage(电话号, null, "longitude:" + longitude + ";latitude:" + latitude, null, null);

三十二、自定义Toast　实现过程是通过WindowManager添加移除View实现

系统自带Toast的实现的过程

```
WindowManager mWM = (WindowManager)context.getSystemService(Context.WINDOW_SERVICE);
mWM.addView(mView, mParams);
mWM.removeView(mView);
updateViewLayout(View view, ViewGroup.LayoutParams params);
```

总结：Toast利用WindowManager去添加View和隐藏View

1、需要一个自定义的布局做为显示的样式
2、自定义一个自定义Toast类　AddressToast implements OnTouchListener实现触摸事件
在类中定义一个有参构造器 在构造中获取WindowManager及定义LayoutParams的相关参数

```
AddressToast(Context context) {
this.mContext = context;
```

```
获取窗口管理器
mWM = (WindowManager) context.getSystemService(Context.WINDOW_SERVICE);
params.height = WindowManager.LayoutParams.WRAP_CONTENT;
params.width = WindowManager.LayoutParams.WRAP_CONTENT;
params.format = PixelFormat.TRANSLUCENT; // 格式
// <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
// 类型 决定了显示的优先级 由于优先级的不同有可能需要添加权限
params.type = WindowManager.LayoutParams.TYPE_PRIORITY_PHONE;
params.setTitle("Toast");
// 标识 决定是否有焦点 是否可以触摸等
params.flags = WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON| WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE;
3、定义两个方法一个用于显示，一个用于隐藏
核心2、显示toast的方法
public void show(String msg) {
if (mView == null) {
mView = View.inflate(mContext, R.layout.view_toast, null);
}
int style = PreferencesUtils.getInt(mContext, Cons.ADDRESS_STYLE,R.drawable.toast_address_blue);
mView.setBackgroundResource(style);
显示的核心是必须要有一个toast的样式布局
TextView tvToast = (TextView) mView.findViewById(R.id.tv_custom_toast);
tvToast.setText(msg);
mWM.addView(mView, params);
mView.setOnTouchListener(this);
}
核心3、隐藏toast的方法
public void hide() {
if (mView != null) {
if (mView.getParent() != null) {
mWM.removeView(mView);
}
mView = null;
4、需要个简单布局用来做为Toast显示的样式
5、定义触摸事件
核心4、触摸事件
getX()是表示Widget相对于自身左上角的x坐标,而getRawX()是表示相对于屏幕左上角的x坐标值
@Override
public boolean onTouch(View v, MotionEvent event) {
switch (event.getAction()) {
case MotionEvent.ACTION_DOWN: // 按下
mDownX = event.getRawX(); mDonwY = event.getRawY();
break;
case MotionEvent.ACTION_MOVE: // 移动的
float moveX = event.getRawX(); float moveY = event.getRawY();
int dx = (int) (moveX - mDownX + 0.5f);//四舍五入：（int）4.1（int）4.8
int dy = (int) (moveY - mDonwY + 0.5f);
params.x += dx; params.y += dy;
  更新按下去点
mDownX = moveX; mDonwY = moveY; mWM.updateViewLayout(mView, params);
break;
case MotionEvent.ACTION_UP: // 放开的事件
```

```
break;
return  false;
```

三十三、归属地通过归属地数据库获取电话的归属地

打开数据库：SQLiteDatabase.openDatabase(path, null, SQLiteDatabase.OPEN_READONLY);

按号码查询数据库的对应类型返回数据

三十四、LayoutParams及 对话框样式样式的的设计

LayoutParams布局参数　　用于告诉父亲怎么样去布局

三十五、ProgressBar的自定义样式

查找----->复制----->新建----->修改----->使用

找到系统自代的样式进行修改

找：先在布局中创建<ProgressBar定义style="*android:attr/progressBarStyleHorizontal*"样式

复制*progressBarStyleHorizontal*去打开*styles.xml*找到系统的*parent*进入——>*Theme.xml*文件------>搜索*progressBarStyleHorizontal*----->找到对的*@android:style/Widget.ProgressBar.Horizontal</item>* 中定义的样式点击进入*styles.xml*文件----->找到*progressDrawable*">@android:drawable/progress_horizontal</item>---->点击进入*styles.xml*点开之后就可能复制文件内容了在*drawable*目录新建一个文件　　最终样式文件名：*progress_horizontal*

建： 在***drawable***目录新建一个　***progressbar_horizontal_1.xml***文件　粘贴所有样式内容

改：进度条的颜色

用：在<ProgressBar 组件中添加

```
android:progressDrawable="@drawable/progressbar_horizontal_1"
```

三十六:工程中数据的拷贝　***assets***目录　可以通过***get***方法拿到

从***assets***目录拷贝文件到应用的***data/data/***应用包名***/files***文件夹下的方法

```
getAssets().open("address.zip");     可以获得一个输入流对象来读文件
new  FileOutputStream(getFilesDir().getAbsolutePath()  +  "/address.db");
```

可以获得一个输出流对象来写文件

三十七、文件的压缩和解压方法

GZIPOutputStream 对象　构造需要一个输出流　　write（）写压缩文件

GZIPInputStream 对象　　构造需要一个输入流　read（）读取压缩文件

三十八、EditText输入文本框实时监听输入内容

```
EditText.addTextChangedListener(new  TextWatcher()  {
有一个输入正在改变的时候的方法
s:    真正输入的内容
start:    开始输入的
before:    结束输入的
count:    输入的个数
onTextChanged(CharSequence  s,  int  start,  int  before,  int  count)
```

可以通过输入的内容对其进行一些操作

三十九、PopupWindow 可以显示任意的view,悬浮在一个view上

如果popupwindow显示的内容是ListView那么就要new一个ListView对象设置到popupwindow中显示

1、创建一个要显示出来的布局文件

2、通过打气筒View.inflate获取布局文件

3、创建PopupWindow对象，对象的构造参数决定了PopupWindow弹出的大小

参一：打气筒返回的对象　　参二：显示出来的宽带

参三：显示的高度　　参四：是否可以有焦点

```
PopupWindow对象 = new PopupWindow(contentView, 400, view.getHeight(), true);
```

4、显示出来及显示的所有位置

参一：在谁上面显示　　参二：在谁的左边　　参三：与谁对齐

PopupWindow显示的方法有三个,showAsDropDown(anchor),showAsDropDown(anchor, xoff, yoff)和showAtLocation(parent, gravity, x, y)。

前两个showAsDropDown方法是让PopupWindow相对于某个控件显示，而showAtLocation是相对于整个窗口的。

showAsDropDown(VIEW);

showAsDropDown(VIEW, 150, 0);相对某控件 x y 的偏移量

```
showAtLocation(tv_center, Gravity.CENTER, 0, 0);
```

```
PopupWindow对象.showAsDropDown(view, 160, -view.getHeight());
```

5、设置外面是否可点击会消失　小bug　　必须要设置一个背景颜色才会有效

```
PopupWindow对象.setOutsideTouchable(true);
```

必须要设置一个背景颜色才会有效

```
ColorDrawable colorDrawable = new ColorDrawable(Color.parseColor("#ffffff"));
PopupWindow对象.setBackgroundDrawable(colorDrawable);
```

6、设置弹出动画样式

```
PopupWindow对象.setAnimationStyle(android.R.style.Animation_InputMethod);
```

7、隐藏对话框

```
PopupWindow对象.dismiss();
```

四十、隐式意图跳转到软件卸载，打开，分享，信息

卸载

```
Intent intent = new Intent();
intent.setAction("android.intent.action.DELETE");
intent.addCategory("android.intent.category.DEFAULT");
intent.setData(Uri.parse("package:" + mAppInfo.packageName));    软件包名
startActivity(intent);
```

打开

```
//  获取包名
String packageName = mAppInfo.packageName;      获取包名
//  获取packageManager
PackageManager pm = getPackageManager();
//  根据包名获得Intent
Intent intent = pm.getLaunchIntentForPackage(packageName);
//  跳转
startActivity(intent);
```

用短信分享

```
Intent intent = new Intent();
intent.setAction("android.intent.action.SENDTO");//  发送信息的动作
intent.addCategory("android.intent.category.DEFAULT");//  附加信息
```

```
intent.putExtra("sms_body",  "分享广州黑马Android12期安全卫士");
intent.setData(Uri.parse("sms:10086"));//  具体的数据，发送给10086
startActivity(intent);
```

软件信息

```
Intent  intent  =  new  Intent();
intent.setAction("android.settings.APPLICATION_DETAILS_SETTINGS");
intent.addCategory("android.intent.category.DEFAULT");
intent.setData(Uri.parse("package:"  +  mAppInfo.packageName));    软件包名
startActivity(intent);
```

四十一、*SlidingDrawer*抽屉样式

四十二、系统进程获取

四十三、五种定时方式

Thread 线程定时

```
new Thread(new Runnable() {
@Override
public void run() {
while (true) {
System.out.println("我被定时");  定时任务
SystemClock.sleep(1000);
}).start();
```

Handler 定时

```
handler.postDelayed(new Runnable() {
@Override
public void run() {
System.out.println("我又被定时");  定时任务
handler.postDelayed(this, 1000);
}
}, 1000);
```

结束定时任务：*handler.removeCallbacksAndMessages(null);*

Timer 定时

```
timer.schedule(new TimerTask() {
@Override
public void run() {
System.out.println("我又又被定时");   定时任务
}
}, 1000, 1000);
```

结束定时任务：*timer.cancel();*

线程池 定时

```
ScheduledExecutorService ThreadPool= Executors.newScheduledThreadPool(5);
ThreadPool.scheduleAtFixedRate(new Runnable() {
@Override
public void run() {
System.out.println("我又又又被定时");
}
}, 1000, 1000, TimeUnit.MILLISECONDS);
```

结束定时任务：*ThreadPool.shutdown();*

AlarmManager 定时

```
class AlarmReceiver extends BroadcastReceiver {
@Override
public void onReceive(Context context, Intent intent) {
        System.out.println("我又又又又被定时");    伤务
}
```

注册接收任务广播

```
mAlarmReceiver = new AlarmReceiver();
IntentFilter filter=new IntentFilter();
filter.addAction("org.heimar.test.timing");
registerReceiver(mAlarmReceiver, filter);
```

2、获取方式

```
AlarmManager mLm= (AlarmManager) getSystemService(Context.ALARM_SERVICE);
long triggerAtMillis = SystemClock.elapsedRealtime();
long intervalMillis = 1000;
```

自定义任务广播

```
Intent intent = new Intent();
intent.setAction("org.heimar.test.timing");
PendingIntent mOperation= PendingIntent.getBroadcast(getApplicationContext(), 100,
intent,PendingIntent.FLAG_UPDATE_CURRENT);
```

定时任务的设置

参一：定时的模式　　参二：从什么时间开始算起

参三：间隔多长时间　　参四：任务

```
mLm.setRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP, triggerAtMillis, intervalMillis,
mOperation);
```

结束定时任务：*mLm.cancel(mOperation);*

* *triggerAtMillis:*触发时间
* *intervalMillis*：间隔事件
* *PendingIntent*：发送广播
* *RTC_WAKEUP*：时间算起：*1970*，关闭的时候也会唤醒
* *RTC*：时间算起：*1970*，关闭的时候也不会唤醒
* *ELAPSED_REALTIME_WAKEUP*：开机的时候，关闭的时候也会唤醒
* *ELAPSED_REALTIME*：开机的时候关闭的时候不会唤醒

四十四、Android的版本适配

```
int  sdkInt=Build.VERSION.SDK_INT;
if(sdkInt>=21){
}
```

获取系统运行版本号

如果版本大于了某个值就做出其它相应

四十五、将应用提升为前台进程应用没有窗口时，产生通知小图标显示在手机顶端

四十六、Widget应用有小窗口展示并更新数据

四十七、AysncTask的使用

异步请求 内部采用了线程池来处理任务和Handler来发消息给UI线程 Android 3.0以后才有的类

核心线程：　CPU核心数

最大线程：　CPU核心数 * 2 +1

闲置线程存活时长：30

最大缓存任务列队：128

处理机制：

内部还有一个创建线程的工厂类：

- AsyncTask主要存在以下局限性：
  - 在Android 4.1版本之前，AsyncTask类必须在主线程中加载，这意味着对AsyncTask类的第一次访问必须发生在主线程中；在Android 4.1以及以上版本则不存在这一限制，因为ActivityThread（代表了主线程）的main方法中会自动加载AsyncTask
  - AsyncTask对象必须在主线程中创建
  - AsyncTask对象的execute方法必须在主线程中调用
  - 一个AsyncTask对象只能调用一次execute方法

使用试例：

执行线程：加载数据
执行完线程之后:更新UI
参数类型： <Void, Void, Void>
参数一：要传入的数据类型 对应在 doInBackground(String 方法中
参数二：返回任务的进度更新数据 onProgressUpdate(Integer[] values)
参数三：要返回的数据类型 onPostExecute(Boolean result)
六个方法：
execute()：执行任务 这个方法如果给传递了参数那么这个参数是在doInBackground()这个方法中接收
doInBackground()： 执行线程：加载数据:run 非UI线程里面
onPreExecute： 执行线程之前：更新UI：run UI线程
onPostExecute： 执行完线程之后:更新UI，run UI线程
publishProgress
onProgressUpdate 更新进度的方法
execute()方法：

```
new AsyncTask<Void, Void, Void>() {
  执行线程之前：更新UI
protected void onPreExecute() {
        比如隐藏控件
};
执行线程：加载数据:run 非UI线程里面
@Override
protected Void doInBackground(Void... params) {
        请求数据
```

相当于:

```
        new Thread(){
            @Override
            public void run() {
             请求数据
            }
         }.start();
}
执行完线程之后:更新UI
protected void onPostExecute(Void result) {
        返回请求结果
   };
}.execute(可以有参数);
```

注：这里传入的参数在doInBackground方法接收

四十八、接口回调解耦（UI更新与业务代码分离开，UI变更不影响代码，代码变更不影响UI）
四十九、LayoutInflater的三种使用
五十、程序锁原理

五十一、AccessibiltyService辅助功能（服务）

开发步骤：
1、一个自定义类继承extends AccessibilityService
2、注册服务

```
<service
    android:name="org.heima12.mobilesafe.service.WatchDogService2"   服务类
    android:label="广州12期NB程序锁"   需要一个名称
    android:permission="android.permission.BIND_ACCESSIBILITY_SERVICE" >
    <intent-filter>
    <action android:name="android.accessibilityservice.AccessibilityService" />
    </intent-filter>
    <meta-data
    android:name="android.accessibilityservice"
    android:resource="@xml/accessibility_service_config" />   需要一个XML文件
</service>
```

XML文件：

3、

五十二、缓存的产生和清理

五十三、线程池的详细解剖

五十四、病毒的定义

五十五、开关门动画

56、ExpandableListView 可以展开的列表控件