**2019.7.25使用原生MediaPlayer做音乐播放器时，唱片不随着暂停按钮暂停。**

现象：其中在切换歌曲后再暂停出现唱片不暂停

原因：切换歌曲时，之前生成的动画没有从内存释放，暂停的是后面新生成的。

解决方案1：在切换歌曲时，判断动画是否为空，可以**节约内存**，同时也避免了新旧动画冲突。缺陷是：这只能适应多首歌曲公用一个动画，并且切换歌曲时不会重置唱片位置。

解决方案2：加入分支判断，动画不为空时**调用animator.cancel()**;取消动画，再重新申请动画。

**2019.7.25 软件权限动态获取（用户产品弹出允许框进行获取**！**）**Android 6.0之后都需要。

String[]permisson={Manifest.permission.*READ\_EXTERNAL\_STORAGE*,Manifest.permission.*WRITE\_EXTERNAL\_STORAGE*,Manifest.permission.*INTERNET*};  
ActivityCompat.*requestPermissions*(this,permisson, 0);

这里动态获取了三个权限，对应判断是否获得权限函数是：ActivityCompat.checkSelfPermission(this,Manifest.permission.READ\_EXTERNAL\_STORAGE) == PackageManager.PERMISSION\_GRANTED；已获取则为true；

**程序一旦通过动态权限获取，则不需要再次获取（第二次虽然会有弹出框，但是已经获取权限），卸载重装例外。**

**2019.7.26 歌单listview中每次进行（点击列表选择歌单播放）循环时，出现list增加四条数据（自编写公共类common的构造方法执行）。**

原因：因为common.musiclist为static属性所以导致，list为公共属性，不会随着common对象而改变。

解决方案：去掉静态属性。

**扩展：static方法不需要实例化对象就可执行，static属性被所有该类的对象公用，所以不会随着对象的注销而销毁**

**2019.7.26 在实现listview歌单时，在Item.xml文件中设置linearLayout属性background为图片时，无法显示全部的list数据。**

原因：因为每一个list的元素就对应于一个Item文件，如果背景太大会导致撑开，其他数据会挤出屏幕外。

解决方案，采用color赋予背景颜色，或者选择适中的背景图片

2019.7.30

**1.在手动写入签名时，build.gradle(app)报错找不到方法\*\*\*。**

原因：signingConfigs 需要写在defaultConfig 和buildTypes 前面，不然会编译报错。

且部分需要写在signingConfigs中的内容写入了buildtype中。

**2.在成功修改了签名配置之后，报错找不到密钥**

原因：在local文件中的string 对象多了一个空格。。。

**AS对空格识别灵敏度较高，String类型中不得多加空格，避免因小失大。**

**3.在安装了程序的机器上跑含有同标签名的程序，会报错。此时需卸载之前安装的软件**

**4.**使用git 从公司项目clone项目：

git clone [git@172.18.1.16:yecon/at8317.git -b branchname](mailto:git@172.18.1.16:yecon/at8317.git%20-b%20branchname)

此处-b后为所需要clone的分支名 前为服务器名

**5.clone后需要build 后才能获取刷机的文件夹image\_release;**

Build操作流程：1 进入获取的分支文件 2. 点+空格+“selfbuild clean” 3. 点+空格+“selfbuild“

(后续将此文件夹所有内容拷贝到格式化的u盘，并且加上空文件yc8317.img即可插入硬件进行烧机，俗称换系统)

删除服务器上的文件命令：rm -rf 文件名

6.**在使用as打开两个或者多个项目时，配置文件正常，但是build报错，后找到缘故，是因为切换项目时，没用clear project，导致环境不匹配。**

**解决方案：每切换工程时，选择clear**

**7.根据分支id同步提交分支代码（在C211分支上获取B211的分辨率兼容驱动）：**

**进入项目目录git** **cherry-pick branchId（在获取前尽量先pull代码）**

**20190809**

**1.在github修改远程库文件，本地修改对应文件，制造冲突后出现（master|merging），导致不能$git commit <filename> -m ‘massage’。**

**原因：$git commit <filename> -m ‘massage’和$git commit -m ‘massage’ ，前者会提交工作区的文件（不太清楚），后者才是提交缓存区的文件**

**20190814**

**关于两类滚动条的添加**

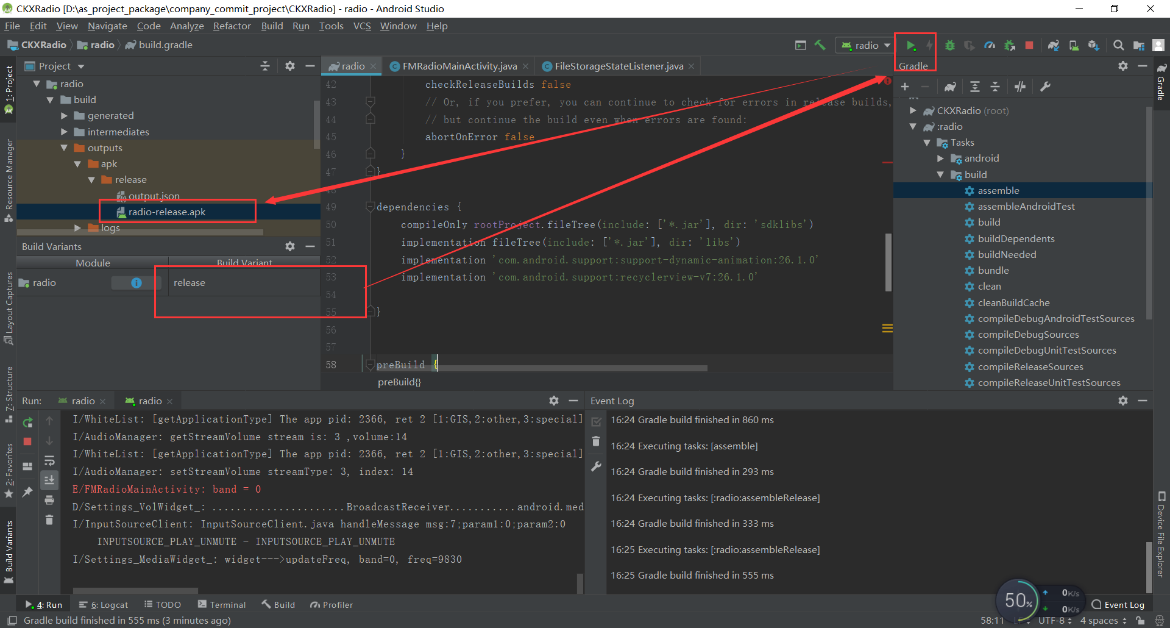
**1：ScrollView滚动条只适用于纵向滚动，HorizontalScrollView滚动条只适用于横向滚动；**

**2：两类滚动条内都只能保留一个同级别的layout，其他布局作为此layout的子控件**

**20190819**

**项目实操中项目代码取下来，没有及时刷机升级到指定版本，导致AS上编译通过，但是到了车机上出现找不到资源报错，原因是车机版本太低，环境不满足，无法支持最新版本的包。**

**20190822**

**在项目发布上线的时候不能发布dubug模式的apk，我们需要发布带了签名的release包，可以从右侧的gradle——：app——tasks——build——assemble（在tasks目录下还有各种打包模式），有特殊的情况是debug模式可以跑动并且，可以打包debug模式apk，但是不能使用assembel，这种特例，操作如下图：**

**20190827**

**Gravity和layout\_gravity 的区别：**

**Gravity是指子控件在使用gravity的控件中的布局；**

**Layout\_gravity则是指使用gravity的控件在其父控件中的布局；**

**20190828**

**'adb remount' 将 '/system' 部分置于可写入的模式，默认情况下 '/system' 部分是只读模式的。这个命令只适用于已被 root 的设备。**

**在将文件 push 到 '/system' 文件夹之前，必须先输入命令 'adb remount'。**

**Push apk 到车机流程：**

**在有权限的前提下，执行 adb+push+apk文件路径（可拖拽，避免中文字符路径）+push目标路径。如果存在同名apk会覆盖掉。**

**20190904**

**在as上修改对应CKXSytemSettings的java，然后跑到车机上没用；**

**原因：因为是系统的应用，车机上的对应apk没用remove所以导致加载的还是未修改的apk（system/app/下），通过as跑进去的apk是放在data目录下的。**

**解决方案：对于系统apk我们应该选择在system/app下先卸载移除掉，再通过as进行演示，否则极大可能运行无效（跑起的是没有卸载的apk，无法达到预期的预览效果）。**

**20190904**

**父控件ScrollView中存在一个自定义的触摸滑动view，两者对触摸都有事件响应，所以出现子控件无法正常使用，**

**解决方案：在子控件中重写以下方法，使得其祖宗控件不拦截子控件的触摸事件。**

**@Override  
public boolean** **dispatchTouchEvent(MotionEvent ev) { //重写（阻止父控件对此控件的拦截）--sgx  
  
 getParent().requestDisallowInterceptTouchEvent(true);  
 return super.dispatchTouchEvent(ev);  
}**

**20190910：**

**移位运算符中（<<;>>）左移低位都是补0，右移高位补1；**

**如果要将byte类型（八位）转换为int类型（三十二位）：则需要使用与（&）运算，**

**Int a = byte[0] & 0xFF;**

**如果不使用上面方法进行byte转换int，而是使用强转的话，补位的高24位为1，所以byte[0] & 0xFF的作用就是将这24位转为0，这样就不会出现错误的数据**

**计算机中的正负，以及原码，反码，和补码：**

**计算机中正负区别是以第一位的0和1确定的，0开头为正，1为负数；**

**正数的原码，反码，补码都一样和原码相同**

**负数的反码除去****最高位，其他都是取反，补码则为负数的反码加一；**

**例如byte A=-2，**

**A的原码为1000 0010。**

**A的反码为除去最高位,取反：1111 1101。**

**A的补码则为反码加一：1111 1110**

**在负数正数的取反运算有一条规律。  
1.十六进制正数的取反，为二进制的最高位取反，数据位（有效位）加一。**

**例如+9取反，其二进制位为01001，其取反为11010，为-10**

**20191021：**

**1：在as配置文件中android{}函数内加入以下方法，可以避免每次生成release包时修改名称。**

android.applicationVariants.all{

variant->

variant.outputs.all{

outputFileName="YeconCan.apk"

}

}

**2：**如果拉取代码的时候，出现需要merge的解决操作（可能没有冲突，而是在git pull和push之间的间断中有一个时间差，恰好别人push了），只要没有冲突系统会自动合并。（未实践）  
第一步：Esc退出  
第二步：冒号+wq进入命令输入状态

3：AS中logcat内，查看日志筛选log在右侧有选择，可选单个apk的log，也可选全部log

**20191028：**

**加入handler 防止多按钮的快速点击响应。**

**final int LEFT\_WHAT = 0;  
final int RIGHT\_WHAT = 1;  
long lastTime = 0;  
Handler clickHandler = new Handler(){  
 @Override  
 public void handleMessage(Message msg) {  
 long currentTime = System.currentTimeMillis();  
 Log.e("sgx","====currentTime=="+currentTime+"====="+lastTime+"======="+(currentTime - lastTime));  
 if(currentTime - lastTime <= 1000){  
 Log.e("sgx","11111111111111");  
 return;  
 }  
 lastTime = currentTime;  
 switch (msg.what){  
 case LEFT\_WHAT:  
 Log.e("sgx","LEFT\_WHAT");  
 onSeekupClick();  
 break;  
 case RIGHT\_WHAT:  
 Log.e("sgx","RIGHT\_WHAT");  
 onSeekdownClick();  
 break;  
 }  
 }  
};**

**20191031：**

**放弃本地所有修改、新增、删除文件和目录:**

**git checkout . && git clean -df**

20191101：

在使用git 服务器上单编时注意：

1：单编前必须全编生成必要的环境文件（sdklibs等）。并且使用source ./selfenv准备环境

2：全编后如果要单编，但是服务器上没有作任何修改，单编不会启动！

20191105：

我司无法复现bug，只能提供客户log查找问题。

核对协议时，需要注意协议对应的byte意义，根据文档来读取mcu所发指令的含义。

20191112：  
在使用notepad++查看日志时，可以使用正则表达式快速查找大法：

例如A.+?B：一行种a开头 B结尾的正则表达式。

20191202：在8227车机中，了解到焦点的数据存储结构为栈存储。退出焦点则会舍弃栈顶，并且从里取一个作为新的栈顶。

20191203：  
java几个常用的Math方法

Double Math.ceil(Double a) 向上取整

Double Math.floor(Double a) 向下取整

Int Math.abs(Int a) 取绝对值

Double Math.sqrt(Double a) 取算数平方根

Double Math.cbrt(Double a) 去立方根

Double Math.random() 获取一个[0.0,1.0)的double（这里可以通过扩展，得到对应范围的随机数）

Long Math.round(Double a)

Int Math.round(Float a) 四舍五入（实则调用了a+0.5 的floor操作）

20191205

车可讯桑德车机系统，对数据库的修改只需要修改接口即可，无须手动修改数据库，且每次通过接口修改后，记得修改database\_version ，否则无法运行成功。

20191206

当一个manifest含有多个activity时，需要在设置默认的activity里头的<intent-filter>

标签里头写入

<action android:name="android.intent.action.MAIN" />  
<category android:name="android.intent.category.LAUNCHER" />

这样才能设置成功！

20191209

Android 调用系统输入法，可能产生屏幕控件及其元素抖动现象，此时在AndroidManifest文件中的默认activity内，添加如下标签信息，即可解决.

android:windowSoftInputMode="stateAlwaysHidden|adjustNothing"

201909

Glide框架绘制图片设置色彩模式：

Glide.*with*(App.*getContext*()).load(getFilePath(position)).asBitmap().format(DecodeFormat.*PREFER\_ARGB\_8888*);

初始化Bitmap设置色彩模式：

newOpts.inPreferredConfig = Bitmap.Config.*ARGB\_8888*;  
 bitmap = BitmapFactory.*decodeFile*(srcPath, newOpts);

20191217： RadioGroup设置监听事件中需注意：

RadioGroup.setOnCheckedChangeListener(onCheckedChangeListener);

Java代码中如果设置子控件点击用RadioGroup.check(true)方法则会调用两次监听事件。

而使用Radiobutton.setChecked();则只会调用一次监听

20191220   
1.systemproperties设置的系统属性在恢复出厂设置时会被销毁清除

2.串口所抓log，中文不会显示

3.串口通讯需要加载完成才能，正常通信，恢复系统设置

20191224   
1.alt+insert按键是AS中快速新增方法的快捷键，阔以快速新建构造方法，Getting，Setting方法或者是原生方法等等。

2.handler中使用锁，是为了避免多线程读取同一数据时,出现数据误差。

3.自定义view中调用handler时，因为handler可能被强引用，如果不清空，回导致无法销毁activity，产生内存溢出，需要重写以下方法，并且在其中加入清空handler的处理。这样就可以保障activty’销毁正常。

@Override  
protected void onDetachedFromWindow() {  
super.onDetachedFromWindow();  
}

20200102：

1.安卓启动模式了解：

1. standard

默认启动模式，每次激活Activity时都会创建Activity，并放入任务栈中。

2. singleTop

如果在任务的栈顶正好存在该Activity的实例， 就重用该实例(会调用实例的onNewIntent())，否者就会创建新的实例并放入栈顶(即使栈中已经存在该Activity实例，只要不在栈顶，都会创建实例)。

3. singleTask

如果在栈中已经有该Activity的实例，就重用该实例(会调用实例的onNewIntent())。重用时，会让该实例回到栈顶，因此在它上面的实例将会被移除栈。如果栈中不存在该实例，将会创建新的实例放入栈中。

4. singleInstance

在一个新栈中创建该Activity实例，并让多个应用共享改栈中的该Activity实例。一旦该模式的Activity的实例存在于某个栈中，任何应用再激活改Activity时都会重用该栈中的实例（也回调用onNewIntent（）），其效果相当于多个应用程序共享一个应用，不管谁激活该Activity都会进入同一个应用中。

启动模式设置：  
<activity android:name=".MainActivity" android: ="standard" />

20200103：

在java中@Override用于标注重写父类的方法，或者是接口的方法。

20200110：

在实现ViewPager.OnPageChangeListener接口时有三个方法控制页面的滑动事件：  
1：onPageScrolled(int arg0, float arg1, int arg2)

arg0 :当前页面，及你点击滑动的页面

arg1:当前页面偏移的百分比

arg2:当前页面偏移的像素位置

@Override  
public void onPageScrolled(int arg0, float arg1, int arg2) {  
}

2：onPageSelected(int i)此方法是页面跳转完后得到调用，参数i为当前页面的position值。

一般我们需要在这里更新当前的position值到应用层。

@Override  
public void onPageSelected(int i) {

}

3：onPageScrollStateChanged(int i)，这里的参数有三个种类（1.2.0）：   
“1”：表示的是正在滑动的事件

“2”：表示的是滑动完成的事件

“0”：则表示的啥都没有的事件

这里我们在每个完整的滑动事件中所调的参数应为（1>2>0），

@Override  
public void onPageScrollStateChanged(int i) {

}

20200116：

全编时使用 ./selfbuild 2>&1 | tee build.log命令编译可以获取全面的log日志

（强烈建议使用该方法编译，因为全编所耗费的时间太长，一次失败再使用次命令不值得）。

若只用./selfbuild 那么将只能捕获到少量的日志，一般很难分析。如果使用前者所消耗的时间差异也不多。

20200117：

在使用git时，查询对应资源内容。

grep -rn "wifi\_tether\_configure\_ssid\_default" （此处可以再加上子目录路径，不加则在当前文件夹搜索）

或者是 grep “需要查询的字段“ -rn /\*\*\*/\*\*\*（对应需要查询的子路径）

20200118：

在android布局xml文件中，常用visibility属性包扩三种（gone，visible，invisible）：

Gone为不可见，且控件不占布局内容。（彻底隐藏）

Invisible为不可见，但是控件大小在父布局容器中是占大小的，只是不可见。

Visible 为可见，且占容器大小。

20200226：

Android activity中moveTaskToBack(true);重写onbackpress，和home按键的处理差不多。可以避免activity走ondestroy（）销毁；

20200302：在通过SystemProperties.*set*来设置系统属性时。需要注意的几点：

1. 默认命名长度不可超32，且前缀有一定要求（persist.sys.）
2. 默认String value长度不可超过92

（安卓系统platform/system/core/debuggerd/test/sys/system\_properties.h里可以修改默认属性）3

20200304：使用cmd 命令行连接adb 查找logcat日志，并且copy。

1.cmd依次输入：  
adb root

adb remount

adb logcat -v time >D:\log.txt (这里为任意路径的新建文件，此时为空文件)

2.执行相关操作

3.ctrl +c（这是将2操作至3的logcat日志copy出来）

ctrl+c中断输出log记录

需要注意的是，logcat日志不支持utf-8编码，所以建议不要输出中文日志

20200307：Android 9.0 wey项目中adb root 后adb remount（提示无权限）处理方式：

Adb root🡪adb disable-verity🡪重启车机🡪adb remount

如此即可解决该问题，原因是有可能使用了dm-ve；

20200316: 在wey项目中更新Navi下文件时需要根据配置文件已写好的方法去编译一个jar包，并且在提交代码的时候一并上传（记得本地编译的时候一定要先拉取更新Navi代码）

20200318：wey项目实战中，radio处理标准按键和radio按键有区分，基础按键在mcu处理转成了标准按键；

举例 ：radio按键包括，seek up/down，bandchange等··。

标准按键包括： 上下曲，播放暂停等。····

20200320 ：在安卓view里头每个控件都可以捕获到触摸事件。

View.setOnTouchListener(new View.OnTouchListener(){

@Override

Public Boolean onTouch(View view ,MotionEvent motionevent){

//处理逻辑

Return false;

}

});

20200320：git 服务器本地无+法修改，权限未给可以在父级目录输入 chmod -R 777 ./\*

给与最高权限（读写执行）（\*最好选定指定文件夹，节约时间）

20200325：8257wey项目./buildapk.sh CKXMedia 用于单编apk

20200326：8257wey项目中，如果chmod -R 777 ./\*提示部分文件权限不够，需要使用sudo chown gitacount 文件路径 -R来更改改文f件的信息. LL可以查看合法文件的信息。

20200328:button中setselected，setenabled以及 setchecked的区别。

Setenabel设定后，一旦buttonsetenable为false，啥都不响应，包括xml的图片，包括button的click事件。一般用于置灰效果

Setselected设定后：按下为true；松开为false；

Setchecked设定后：已经选中为true，未选中为false。

20200407：int类型默认值问题，当该对象作为类的内部属性时，无需初始化，在类被new出一个实例时，其默认为0；但是当int类型作为临时变量时，是没有默认值的，如果没有初始化则会报空指针错误。

20200409:

配置属性值预置:**persist.bean.ime.size=1248,420,1248,420,83**

**这种类似的配置值，可以通过adb直接设置，后期可以直接通过写入系统属性值来固定写入。也可以写在底层的配置文件中。**

**Abd 设置方法：**adb shell setprop persist.bean.ime.size **1248,420,1248,420,83即可**

**Adb shell getprop XXXX 可以获取对于systemproperties的属性值**

**如何对第三方apk加入系统签名：**

**将原apk，签名配置文件（两个），以及signapk.jar 放入Androidp目录下，执行**

**java -jar signapk.jar platform.x509.pem platform.pk8 需要签名的apk.apk 签名完的名字.apk 报如下错误：**

**Exception in thread "main" java.lang.UnsatisfiedLinkError: org.conscrypt.NativeCrypto.get\_cipher\_names(Ljava/lang/String;)[Ljava/lang/String;**

**at org.conscrypt.NativeCrypto.get\_cipher\_names(Native Method)**

**at org.conscrypt.NativeCrypto.<clinit>(NativeCrypto.java:764)**

**at org.conscrypt.OpenSSLProvider.<init>(OpenSSLProvider.java:56)**

**at org.conscrypt.OpenSSLProvider.<init>(OpenSSLProvider.java:49)**

**at com.android.signapk.SignApk.main(SignApk.java:942)**

**再执行：**

**java -Djava.library.path=out/host/linux-x86/lib64 -jar signapk.jar platform.x509.pem platform.pk8 需要签名的apk.apk 签名完的名字.apk**

**正常情况下稍等几秒就可以在androidp目录下生成新的已经增加签名的apk。**

**20200410：使用github提交备份日志时，生成ssh密钥时，需要注意另存的密钥文件不可以为已存在的文件。否则会生成失败。也就是说你需要输入的是目录/新增密钥名称文件。**

ssh-keygen -t rsa -C "邮箱"用于生成对应邮箱的密钥

在配置完签名文件后，需要在android{ buildTypes{release{}}}内增加如下  
**signingConfig signingConfigs.release用于指定证书打包。**

**Apk更换签名后，如果启用AndroidManifest.xml文件中的如下标签：**

**<dist:module dist:instant="true" />**

**则不会重新安装apk，而是在之前成功安装的基础上，局部修改apk，这样签名文件修改后，新的apk将无法成功被安装。**

**Git reset之三种模式：**

* **mixed(默认模式)****—>get reset <commit-id> 修改记录仍然在工作区间。**
* **soft —> get reset --soft <commit-id> 修改记录在工作区和暂存区都有，依然可以通过commit提交。**
* **hard —> get reset --hard <commit-id> 重置最彻底，所有修改记录清除，包括工作区和暂存区**

**工作区：本地电脑的文件夹就是一个工作区**

**暂存区：为git add 之后保存在版本库的东西，但是未commit。Commit后暂存区清空**

**版本库：就是工作区里有个.git，就是本地git的版本库。**

**20200413：**

**在git 中输入jobs可以查看对应的后台运行的动作**

**20200414：**

**27Git服务器中**

**ps | grep allmake 用于查看allmake执行的后台任务，显示pid**

**kill -s 9 pid 用于shutdown指定的pid进程**

**20200417：**

**System.*currentTimeMillis*()用于获取系统当前时间**

**20200418:**

**使用monkey调试测试应用**

**20200420：**

**8257项目中当需要设置系统属性时：**

**ac8257-project\ac8257-carocean\overlay\device\autochips\ac8257\_demo**

**下通过修改system.prop文件或者是device.mk文件可以达到效果。（这种是设置驱动的系统属性）**

**在ac8257-carocean /overlay/build/make/tools/buildinfo.sh下也可以配置应用层的系统属性**

**git config --global push.default simple 设置push模式为simple，保证每次push的只有指定的commit文件，而不包含某些特殊的配置文件。**

**20200427 在本地连接git服务器，下载代码时，需要注意两点：**

**第一是需要生成新的密钥，并且将公钥给上传到git，对应的ssh密钥中。**

**第二是需要使用生成的私钥，加入到ssh——bash中，这样才能保证与git远程服务器的交互数据的连接安全。具体操作如下(进入新密钥目录)：**

**1.进入git bash 输入命令行 ssh-agent bash**

**2.输入命令行** **ssh-add key**

**同一计算机，使用不同账户时，需要在ssh下配置config文件，处理密钥混淆问题。详细见20200501随笔记录**

**20200501:**

**Adb操作小技巧：**

**使用****tasklist|findstr “PID” 可以查看该pid对应的进程名称**

**使用netstat -ano | findstr "端口ID" 可以查看该占用该端口的进程pid**

**使用****taskkill /F /PID PID 可以直接杀死该pid对应的进程**

**使用****adb** **nodaemon server 可以查询adb所在端口的id**

**关于8257车机android9.0的系统在win10上出现logcat中断问题：**

**可以修改win10的adb端口号，新增系统环境变量** ANDROID\_ADB\_SERVER\_PORT值为10000-60000（避免被其他程序使用）之间的端口（需要重新启动电脑），可以解决；

**使用本地git clone后，由于本地存在多个ssh密钥分别适用于不同账号（gitlab 和github），所以需要通过~/.ssh下的config文件来配置对应的账号使用对应的密钥。否则会默认使用id\_rsa。**

**注意：如下格式**

**第一hostname 可以是域名或者是ip，这里gitlab使用的ip地址。Github则使用的域名“github.com “。**

**第二 host要和域名保持一致，其作为标识，用于标识，选择对应的密钥。IdentityFile可以填写为绝对路径。为对应密钥的路径**

**当需要配置多个的时候，主要不能留空格，否则会对第二至多个配置有影响。**

**例子：**

**#gitlab(company\_gitlab)**

**Host 172.18.1.17**

**HostName 172.18.1.17**

**PreferredAuthentications publickey**

**IdentityFile ~/.ssh/key**

**User suguixing**

**#github(personal\_github)**

**Host git-github**

**HostName github.com**

**PreferredAuthentications publickey**

**IdentityFile ~/.ssh/id\_rsa**

**User xingcheng11111**

**注：**

**第一点:使用该配置的时候，检查URI是否为ssh的路径（如果是http的方式不会调用密钥，且每次都会首选输入账号密码）**

**第二点：可以使用ssh -T git@hostname测试链接；**

**20200502：**

**Apk在install时报出install\_falled\_test\_only，导致install失败。**

**原因：**

**AS通过绿色按钮run到机器中生成的应用APK,自动添加了**testOnly="true" 属性到androidManifest.xml中。

解决方案：

1我们可以选择building去生成apk，这样就可以避免自动添加属性。

2.或者是在gradle.properties中添加android.injected.testOnly=false属性避免。

3.也可以通过**adb install -t app-debug.apk去安装指定apk**

**android textview中建议使用sp做单位，因为使用sp的时候可以根据系统字体的大小变化而变化，而使用dp的时候则不会产生变化。**

**20200505：**

**Int 转 String方法：**

第一种方式：String的valueOf方法，  
int i=1;  
String s=String.valueOf(i);  
第二种方式，直接在int后面加一个空的字符串，因为在java里面，默认任务int类型和字符串类型相加，为字符串类型。  
int i=2；  
String s=i+"";  
第三种方式：使用int的封装类Integer，在Integer里面用他的toString方法。  
int i=3；  
String s=Integer.toString(i);

String 转 Int方法：

方法1，使用Integer类中的parseInt（）方法。

方法2，使用Integer类中的valueOf()和intValue()方法。

例如，int i = integer.valueOf(String s).intValue();

20200506:

**Thread.*currentThread*().getName()**

**该方法可以输出当前代码段执行时所处的线程name。**

**线程池的四种状态：**

**Running：线程池被创建时初始状态。**

**Shutdown：调用了shutdown（）后，线程池不接受新任务，等待池内任务执行完毕。**

**Stop：调用shutdownow（）后，线程池不接受新任务，并且尝试终止池内正在执行的任务。**

**Terminated：线程池在处于Stop状态或者sh****utdown状态，并且线程池内，所有工作线程已经销毁。任务缓存列队已经清空或者执行结束时，此时线程池被设置terminated状态。**

**20200507:**

**1.如何在listview中的每个item button点击事件中绑定当前item。**

**在listview中每个item存在button按钮需要去操作对于item的数据时，可以在adapter的getview()中，绑定点击事件并且在此方法数据赋值中给与对应的button.setTag(i)，这样相当于绑定了对应的item数据，在adapter中实现view.onclicklistener()接口，通过view.getTag可以获取绑定的item index。从而获取数据。**

**2.Android提示框：AlertDialog。**

**详细见：**[**https://www.cnblogs.com/duoshou/articles/8622551.html**](https://www.cnblogs.com/duoshou/articles/8622551.html)

**以下为简单的提示框模型：**

**new AlertDialog.Builder(this)  
 .setTitle("提示")  
 .setMessage("删除失败！！")  
 .setPositiveButton("确定",null)  
 .show();**

**20200509：**

Git中查看当前文件夹大小：du -sh

android中EidtText中展示提示文字：android:hint=”\*\*\*”

**20200512：**

**关于java内存泄漏的那些事情：**

**1.什么是内存泄漏：**

**一个对象被new后，占用了堆内存，在申请后已经不再使用了，但是一直没有被GC回收，在android种大多数情况是因为 生命周期较长的对象持有生命周期较短的对象的引用。**

**2.android 中常见的内存泄漏例子：**

**1.单例造成的内存泄漏：**

**单例对象中构造方法中成员变量直接引用外部activity从而导致activity无法销毁回收，导致内存泄漏。我们可以修改引用为，activity.****getApplicationContext();(**Application 的 Context 的生命周期和单例的生命周期是一致的，所有再整个 App 运行过程中都不会造成内存泄漏。Activity（是一个界面的环境）的context一般用于UI相关，而application的生命周期是整个应用。)

**2.外部类中持有非静态内部类的静态对象（如下图）**，

这个其实和单例的原理是一样的，由于静态对象 test 的生命周期和整个应用的生命周期一致，而非静态内部类 Test 持有外部类 MainActivity 的引用，导致 MainActivity 退出的时候不能被回收，从而造成内存泄漏，解决的方法也很简单，把 test 改成非静态，这样 test 的生命周期和 MainActivity 是一样的了，就避免了内存泄漏。



3. **Handler 或 Runnable 作为非静态内部类**

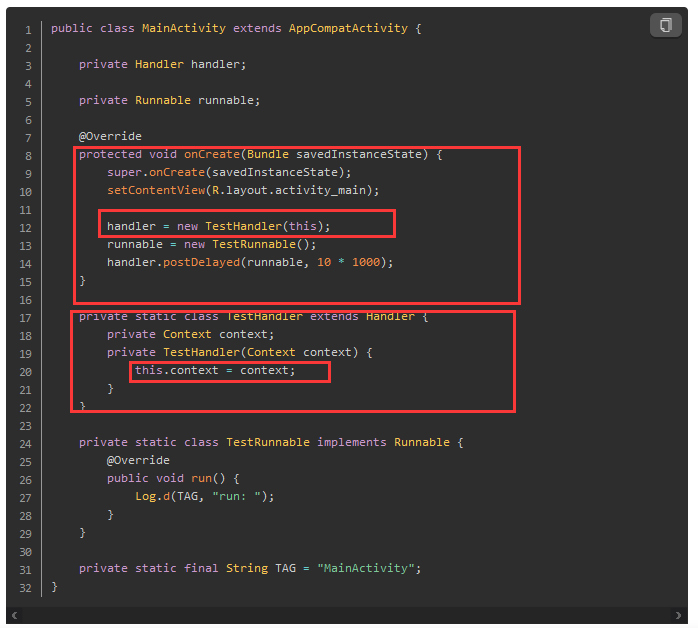
handler 和 runnable 都有定时器的功能，当它们作为非静态内部类的时候，同样会持有外部类的引用，如果它们的内部有延迟操作，在延迟操作还没有发生的时候，销毁了外部类，那么外部类对象无法回收，从而造成内存泄漏（如下图）。

所以一般我们把Handler和Runable定义为静态内部类，使得它们不再持有 MainActivity 的引用，并且同时在activity的ondestroy中需要调用removeCallbacks来移除massage，这样可以在退出时候取消了定时器。



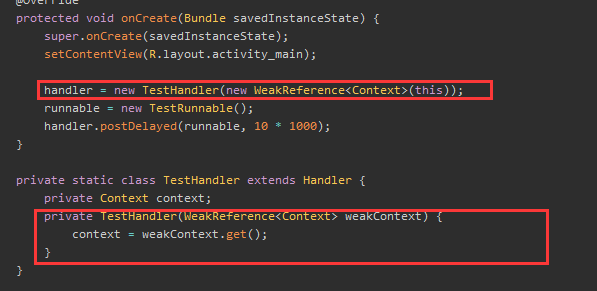
**4.在使用静态内部类的Handler和Runable时，如果构造方法调用了当前activity的对象，则会形成长生命周期handler引用短生命周期的activity，从而activity无法及时注销。**

如下图



这种情况可以使用弱引用的方式来引用 Context 来避免内存泄漏。

代码如下：



**5.其他常见的几种内存泄漏原因：**

比如 BraodcastReceiver 未取消注册，InputStream 未关闭等，这类内存泄漏非常简单，只要在平时写代码时多多注意即可避免。

关于内部类的那些事：

* 成员内部类
* 局部内部类
* 匿名内部类
* 静态内部类

**1。成员内部类：**

定义约束：实际上，成员内部类的类定义没有任何约束（不涉及static，因为这属于静态内部类的范畴），不仅可以将内部类声明为public、private的，甚至可以将其声明为一个abstract抽象类和interface接口，以及继承外部类也是允许的，其定义是十分宽松的

内部方法和属性约束：

* 不能使用static来修饰任何成员内部类中的属性和方法，但可以使用private、final等其余任意修饰符
* 可以用static final来修饰成员
* 可以允许与外部类字段和方法重名

**2.局部内部类**

* 其类上不允许有任何修饰符，但是可以使用abstract将类声明为抽象类
* 不允许将局部内部类声明为接口
* 不允许使用static来声明成员变量和方法
* 可以将局部内部类声明在静态方法中
* 任意两个方法中的局部内部类可以重名

局部内部类，就只能在声明类的方法处来使用

**3.匿名内部类**

* 不能使用static来修饰方法和属性，但是可以有static final的属性
* 可以使用this指针来访问本身定义的变量和继承得到的变量，也可以使用 外部类名.this指针来访问外部类中的所有属性
* 无法在类上进行任何修饰，因为没有class定义符和类名
* 其中定义的私有字段对外是完全可见的

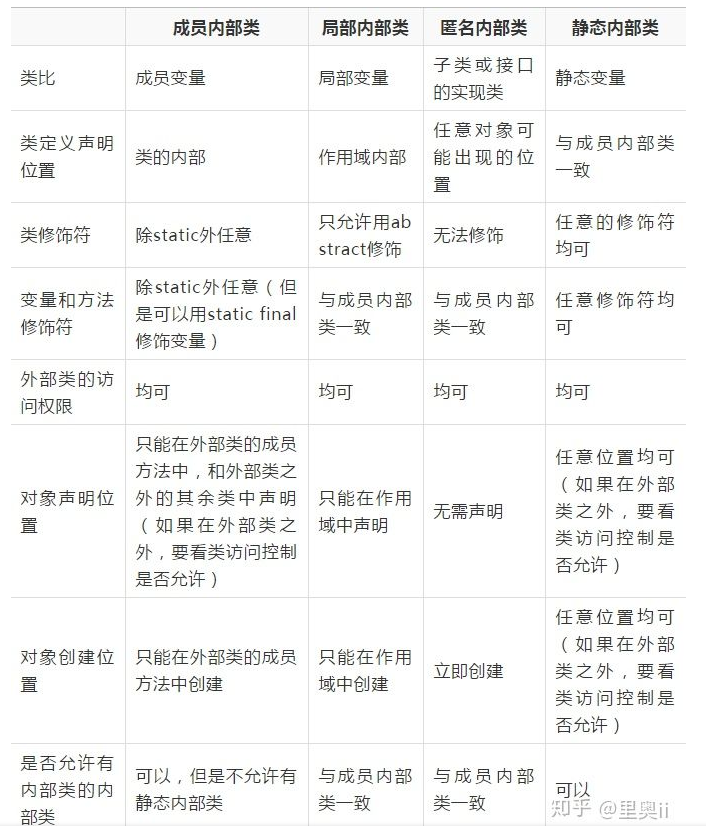
匿名内部类没有类的声明，会隐式地继承一个类或实现一个接口

**4.静态内部类：**

* 可以使用任意的修饰符来修饰类、方法和属性
* 可以访问外部类的静态方法和属性

静态内部类相当于static修饰的成员内部类，可以当作静态变量来理解

**具体各权限和使用方法如下表约束：**



**Activity之movetasktoback（boolean flag）：**

该方法参数为true时，表示将当前activity后台运行，相当于最小号，后续再次点击进入resume(),而不是oncreate 。只要进程不杀死不注销，则多次进入都是走onresume（）；

**20200514：handler在子线程中处理运行，避免主线程卡顿问题**

主线程通过新建一个子线程，并且在子线程中将looper直接做为handler的构造参数传入可以实现，handler在子线程中运行，这里发消息依然是主线程来发送，但是消息动作在子线程中处理，可以避免handler的耗时操作导致主线程的卡顿问题。注意页面更新的UI操作不可放置于子线程中。

**Handler的post和sendmasssage区别；**

前者post是将消息防止于消息队列中，只有前一个消息执行结束后，才能执行。

后者sendmassage则是同步处理。

**Handler的机制：**

**弹窗导致的内存泄漏问题：**  
1.在子线程中更新UI是禁止的，包括弹窗类，比如toast，解决方法，通过调用主线程的handler.post(Runable r)将所需要的动作通过消息发送给主线程。

2.在内部类中的弹窗问题，因为直接在内部类中直接调用弹窗，弹窗是依赖于外部类的。而如果通过内部类调用外部类handler发送消息 post（Runabel）的方式。则不属于内部类的引用，直接相当于和外部类关联，外部类销毁，弹出即销毁。

**内存泄漏检测工具：leakcanary**

**详见文：https://www.jianshu.com/p/70b8c87ea877**

20200518：

Recycleview的使用时，没有直接调用当前position的接口，如果子item监听事件需要可以在内部类中设置监听事件，并且在recycleview的onBindViewHolder（）方法对每个子item设置tag为当前postion，如何在内部类ViewHolder中获取position即可。

20200525：

Android学习笔记：

1.drawable文件分类使用。不同密度使用不同drawable

一个有六种drawable文件夹

（drawable,drawalbe-ldpi,drawable-mdpi,drawable-hdpi,drawable-xhdpi,drawable-xxhdpi）

后五个文件夹分别对应一下density。（第一个除外，其放置图片样式相关的xml）

density：120dip（low），160dip（medium）,240dip（high），320dip（xhigh），480dip（xxhdpi）

2.api获取当前设备密度以及dpi的方法。

public void getDensity(){  
 DisplayMetrics displayMetrics = getResources().getDisplayMetrics();  
Log.*e*(*TAG*," Density is"+displayMetrics.density+"densityDpi is"+displayMetrics.densityDpi +"height:"+displayMetrics.heightPixels+"width: "+displayMetrics.widthPixels );

}

3.字号设置使用Autosize框架可以避免多个手机屏幕宽度时，需要建立多个dimen文件适配。

两部曲：

1.配置：

<!--私有依赖-->

implementation 'me.jessyan:autosize:1.1.2'

<!--共有依赖-->

api 'me.jessyan:autosize:1.1.2'

2.manifest文件适配

<manifest>

<application>

<meta-data

android:name="design\_width\_in\_dp"

android:value="540"/>

<meta-data

android:name="design\_height\_in\_dp"

android:value="960"/>

</application>

</manifest>

这里填对应机器的长宽即可。

20200526：

1.Adb 输入adb shell dumpsys activity activities：

来获取activity栈数据。可以看到栈顶及其附近的activity

2.java获取前一个activity的接口：

public static String getTopAppPackageName(Context context) {

String packageName = "";

try {

if(Build.VERSION.SDK\_INT < Build.VERSION\_CODES.LOLLIPOP){

ActivityManager mActivityManager = (ActivityManager) context.getSystemService(Context.ACTIVITY\_SERVICE);

List<ActivityManager.RunningTaskInfo> rti = mActivityManager.getRunningTasks(1);

packageName = rti.get(0).topActivity.getPackageName();

}

}catch (Exception ignored){

}

return packageName;

}

该方法可以在onpause中调用（放置于super.onpause前）。

动画笔记：

Android动画一共分为三种：补间动画，帧动画和属性动画。

**补间动画：** 适用于对单一静态图片做旋转,平移以及缩放和透明度设置等等操作。

在补间动画中可以单一的对一个图片进行旋转，平移，透明以及缩放，也可以对此图片实现组合动画。  
**帧动画：** 适用于构造一个类似gif动图场景，须由多张不同形态的照片来实现。有一点需要注意的是，**帧动画在停止之后不能继续从停止帧继续向后执行，而是从头执行。（即没有所谓的暂停）。这也是帧动画的缺点。**

**属性动画：很全面，前面两种动画的效果他都可以实现，且可以更加全面的去实现。详细见**

[Android属性动画完全解析(上)，初识属性动画的基本用法](https://blog.csdn.net/guolin_blog/article/details/43536355)  
[Android属性动画完全解析(中)，ValueAnimator和ObjectAnimator的高级用法](https://blog.csdn.net/guolin_blog/article/details/43816093)  
[Android属性动画完全解析(下)，Interpolator和ViewPropertyAnimator的用法](https://blog.csdn.net/guolin_blog/article/details/44171115)

**曲线的绘制完全可以根据自定义的point和自定义的TypeEvaluator，来组合实现，算法置于typeEvaluator中，具体可参照项目animation\_test模拟动画学习项目。**

20200527 android属性动画学习下：

ValueAnimator对象中可以通过设置Interpolator控制运动速率（当前设置为加速的控制类AccelerateInterpolator），不设置的话，默认为先加速再减速（AccelerateDecelerate-Interpolator对应此实现类），也可以通过自定义的类实现TimeInterpolator接口来调用。

anim.setInterpolator(new AccelerateInterpolator(2f));  
anim.setInterpolator(new MyInterprolator());

**使用属性动画组合时候（Animatorset来组合），如果需要重复播放，可以在每个子动画调用父类ValueAnimator的setRepeatCount(int value)的接口，可以设置重复次数，当value为-1时为永久重复播放。**

自定义类如下：  
public class MyInterprolator implements TimeInterpolator {  
 @Override  
 public float getInterpolation(float v) {  
 float result = 0;  
 result = (float)((0.5 - v)\*(0.5 - v)\*4);  
 return result;  
 }  
}

这里头传入的v决定了传入TypeEvaluato接口的fraction值。

**Android 3.1后为了避免view需要显性地调用属性动画的操作才能有动画，ViewPropertyAnimator类诞生了。**

所有的view都可以通过animate（）方法来获取一个ViewPropertyAnimator实例。并且直接操作该view进行动画。

例如：

Textview.animate.alpha(0f);——————设置view为透明。

Textview.animate.x(500),y(500);——————设置view到（500.500）坐标

Textview.animate.setDuration(5000)——————设置动画时间（5000毫秒）

等等，同样的它也可以用来组合实现多个属性。

Textview.animate.x(500).y(500).setDuration(5000)————从当前移动到（500.500）坐标耗时五秒。

也可以自定义运动速率interpolator。

Textview.animate.x(500).y(500).setDuration(5000).setInterpolator(new MyInterpolator());

————从当前移动到（500.500）坐标耗时五秒,运动速率随MyInterpolator对象变化。

对于自定义控件我们可以在values文件夹下，配置一个对应的attris.xml文件来设置xml中自定义view的私有属性。同时需要注意在调用自定义view的时候需要在最外层布局写入：

xmlns:myAnimView="http://schemas.android.com/apk/res-auto"

这里myAnimView为你attris.xml中自定义的名称（首字母不可大写）。

attris.xml如下：

<?xml version="1.0" encoding="utf-8"?>  
<resources>  
 <declare-styleable name="myAnimView">  
 <!--颜色-->  
 <attr name="color" format="color"/>  
 </declare-styleable>  
</resources>

20200529：

Java字符串匹配：

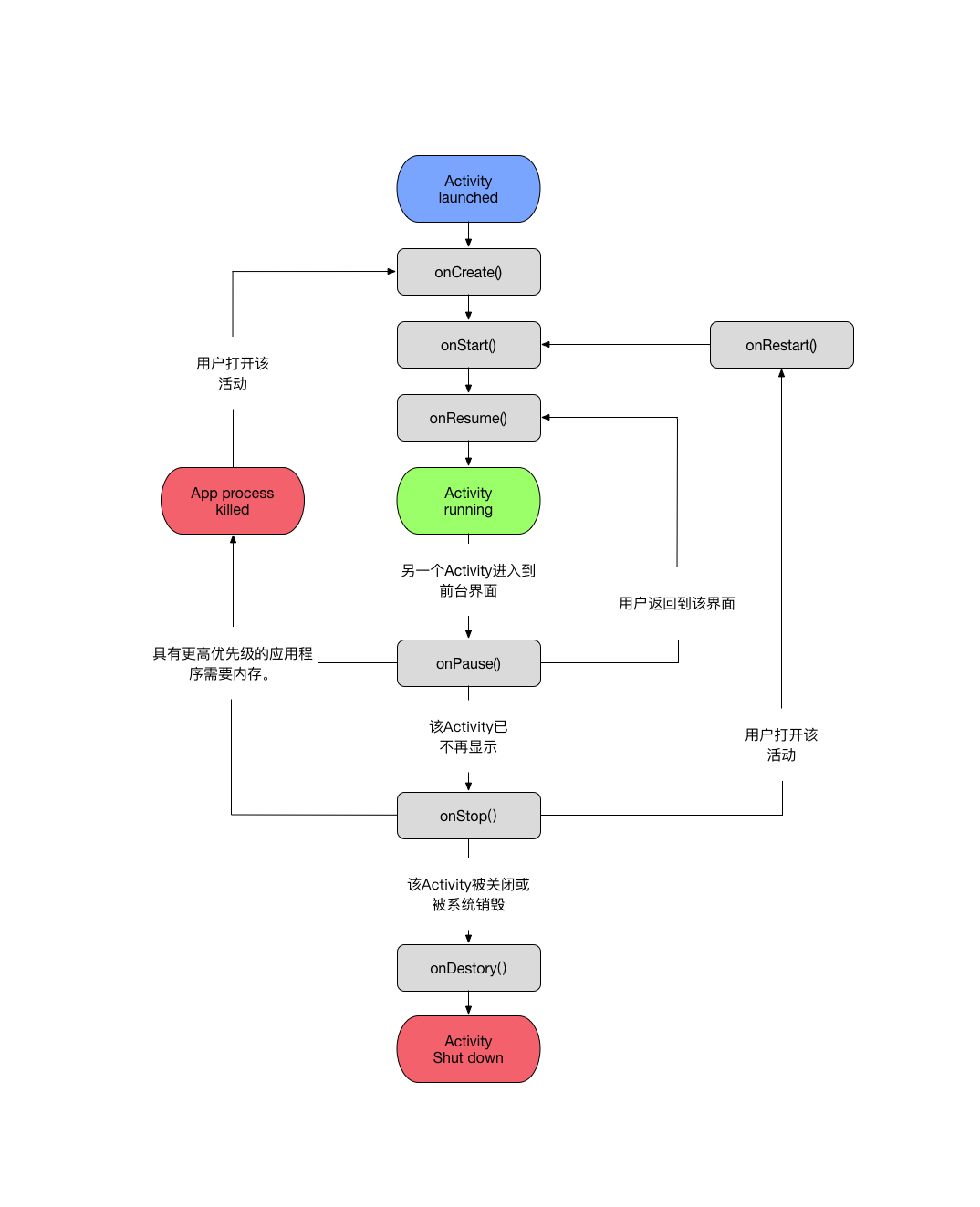
Android提供了原生的接口供我们使用Pattern以及Matcher：

Pattern pattern = Pattern.*compile*("[sgx]");  
Matcher matcher = pattern.matcher(string);

两者的构造方法都是私有，可以通过以上两种方法获取对象。

Matcher.find():如果string字符串符合添加的匹配规则则为true，否则为false；

Activity生命周期存图：



20200601 ：**java读写文件深入学习**

**1。权限问题：**

在android6.0以下相应的权限可以直接写入在manifest.xml中

而后在android6.0及以上的版本中为了安全，权限还需要在java代码中动态申请了。

**2.方法：文件读写一般使用文件输入和输出流：**

FileOutputStream： 文件输出流用于写文件（write）；

FileInputStream ：文件输入流用于读文件（read）;

20200602 ：**反射使用实践应用**

**1.在使用Object o = aClass.newInstance();时，所调用的类必须有无参构造方法，否则会报错。**因为我们用反射实例化对象，就是调用无参构造方法。

2.获取class对象时：

**1）对于获取类属性：getFields()只能获取public的字段，包括父类的,而getDeclaredFields()只能获取当前类自己声明的各种字段，包括public，protected，private。**

**其次如果需要知道单个属性的类型可以使用以下方式获取对应类型的类：**

**Class<?> typeClazz = field.getType();**

**2）获取类方法时：getMethods（）只能获取public的方法，包括父类，而getDeclaredMethods()同上属性的获取。**

**3）在调用类方法时，需要配置正确的类型数据，例如：**

原类student中的方法：

private float getWeight() {  
 return weight;  
}

private void setWeight(float weight) {  
 this.weight = weight;  
}

**Method method\_setWeight = mClass.getDeclaredMethod("setWeight",float.class);  
Method method\_getWeight = mClass.getDeclaredMethod("getWeight");**

**如果是有参函数，需要再后面加入对应的参数类型。**

**无参函数无需加入。**

**反射调用注意事项：**

**1.注意在获取私有方法时候需要使用到getDeclaredMethod接口。**

**2.其次使用该私有方法的时候需要解除私有，method.setAccessible(true);**

**3.再者在调用有返回结果的方法时候，可以用object接收，例如：Object result = method\_getWeight.invoke(object)；**