# Git

Git init

Git add remote ssh:xxx

Git add remote https:xxxx

$ git pull <远程主机名> <远程分支名>:<本地分支名>

Git add .

Git commit –m”xxx”

Git push

Git fetch <https://xxxx.x.x.xx.git> master 获取xxx地址master分支下的数据

CollapsingToolbarLayout的高度应该设置为match\_parent

RecycleView onScrollStateChanged state 0停止 1触摸滑动 2惯性滑动

onScrolled(@NonNull RecyclerView recyclerView, int dx, int dy)

上滑dy>0 下滑dy<0 幅度越大Math.abs(dy)越大

判断scrollView是否可以滑动，用child和本身的高度来比较

Android对象池Pools

设计模式：

单例模式

Build模式

观测者模式

策略模式

EventBus事件处理

Butterknife插件，用注解绑定元素和事件

implementation 'com.jakewharton:butterknife:8.8.1'

annotationProcessor 'com.jakewharton:butterknife-compiler:8.8.1'

Butterknife是在编译阶段对注解进行处理，所以要用annotationProcessor添加对注解的处理

Retrofit 简化HTTP请求，可以直接将json映射为对象

addConverterFactory(GsonConverterFactory.create())，添加转化器

属性动画

ObjectAnimator ofInt（obj，attr，value）

AnimatorSet 组合动画

setInterpolator控制速率，默认的是线性变化

Rxjava

# Retrofit

添加gson converer

implementation 'com.squareup.retrofit2:converter-gson:2.4.0'

自定义Converter

new Converter.Factory() {

@Nullable

@Override

public Converter<ResponseBody, ?> responseBodyConverter(Type type, Annotation[] annotations, Retrofit retrofit) {

if(type.equals(String.class)){

return new Converter<ResponseBody, String>() {

@Override

public String convert(@NonNull ResponseBody value) throws IOException {

return value.string();

}

};

}

return super.responseBodyConverter(type, annotations, retrofit);

}

}

Retrofit retrofit = new Retrofit.Builder()

.baseUrl("http://localhost:8080/")

.addConverterFactory(GsonConverterFactory.create(gson))

.build();

Baseurl要以 ’/’ 结尾

如果后面有参数可以不用/结尾，

## Retrofit和rxjava

转换为Observable<T>

builder.addCallAdapterFactory(RxJava2CallAdapterFactory.create());

需要添加 implementation 'com.squareup.retrofit2:adapter-rxjava2:2.4.0'支持转换

Rxandroid

Kotlin写bean –> data class

要添加反射类

implementation "org.jetbrains.kotlin:kotlin-reflect:$kotlin\_version"

# MVVP设计模式

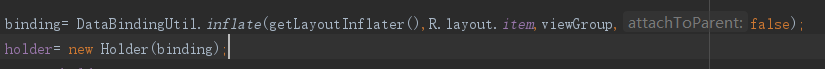
DataBinding：

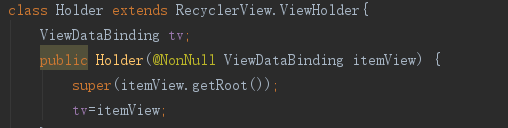
根据使用layout的布局，生成以布局名+Binding的类，如main.xml –>MainBinding

RecyclerView：

需要在onCreateViewHolder里面用ViewDatabinding绑定该布局，

ViewDatabinding的getRoot方法返回View视图





ViewModel中创建数据，Activity中给数据添加观测者，ViewModel中对数据进行更新：setValue(obj)和postValue(obj)。Activity初始化一个ViewModel：Xxx xxx=ViewModelProviders.of(this).get(Xxx.class)

MutableLiveData、MediatorLiveData（监听多个变化1对多）、ComputableLiveData

**泛型**

# AlertDialog：修改w/h参数

在调用dialog.show()之后

Window window=dialog.getWindow();

if(window!=null){

WindowManager.LayoutParams params= window.getAttributes();

params.height= WindowManager.LayoutParams.WRAP\_CONTENT;

params.width= 100;

window.setAttributes(params);

}

# 父控件监听触摸事件：

在onTouchEvent方法里面返回true

在dispatchTouchEvent里面获取MotionEvent

在代码中设置控件的透明度

View.getBackground().setAlpha()//可能没有Drawable

在LayoutParams里面设置

控件的setLeft setTop等尽量不要调用，用setX setY代替

因为在setVisibility、bringToFront等某些情况时会置0

**Native 和H5的混合开发**

# View

## DecorView

**getWindow().getDecorView()**

顶级视图，区域包含整个屏幕

好像无法控制状态栏

## ViewGroup

ViewGroup如果没有background，draw、onDraw方法都不会被调用

ViewGroup获取子View的布局属性（LayoutParams）

没有类型认领的自定义xml属性可以全局使用

重写generateDefaultLayoutParams()

generateLayoutParams()

如果要指定类型的ViewGroup下面使用，在创建自定义属性时要添加“xxx\_Layout”,如让MyViewGroup的子View具有某些属性：

在定义时需要声明为：MyViewGroup\_Layout

## ViewStub

用来占位的，里面的布局不会立即加载，只有在调用inflat或setVisibility时才加载，之后将无法再次调用

Try(InputStream is=new FileInputStream(file)){

}catch(Exception e){}

会自动调用close

## View绘制

ClipChildren=false

可以让ViewGroup显示子控件的内部控件超出的部分。超出的部分在哪里显示就需要给该区域的ViewGroup设置clipChildren

ClipToPadding=false

允许在padding区域里面绘制

Canvas clipBounds 圈出一块区域，用作**后面**的绘制

设置字体的粗细

paint.style = Paint.Style.FILL\_AND\_STROKE

paint.strokeWidth = width

ImageView的setImageTintList可以改变图片的颜色，在api21时，通过代码调用不会触发重绘，不会立即生效。此时可以使用setFilterColor代替。

在内部如果导入了appcompat包，系统会自动将ImageView替换成AppCompatImageView，导致setImageTintList不会立即生效，可以使用ImageViewCompat. setImageTintList来设置立即生效

Api21时，当给LinearLayout设置一个drawable时，如果不是一个ColorDrawable，那么backgroundTint也不会生效

# 自定义CoordinatorLayout.Behavior<View>

继承CoordinatorLayout.Behavior,重写某些方法(需要重载构造方法)

layoutDependsOn要返回true，表示哪些view和有behavior标签的view进行绑定（可以在这里获得各个view的引用）

onLayoutChild返回true，在这里对view进行定位

onDependentViewChanged如果返回true，当dependency变化时会回调（一般在这里对child，也就是有behavior标签的view进行操作）

onNestedPreScroll 滑动前被调用、里面有consumed[0,1],如果consumed[?]=dx?dy,代表偏移量被完全消耗同时onNestedScroll 不会被调用,dependency（RecyclerView）就不会在相应的方向上滚动

onNestedScroll 当dx dy不同时被完全消耗就会被调用

onStartNestedScroll 一定要返回true，否则其他函数不会回调

使用behavior时layout\_behavior值必须要写全路径 .xxx.xx这种不行，要加上包名

使用CollapsingToolbarLayout时，采用parallax模式，配合百分比可以产生视差效果 取0时，appbar一起滚动，取1时appbar固定，不滚动，其他值按百分比滚动，如0.9:内容滚动10px，appbar滚动1px

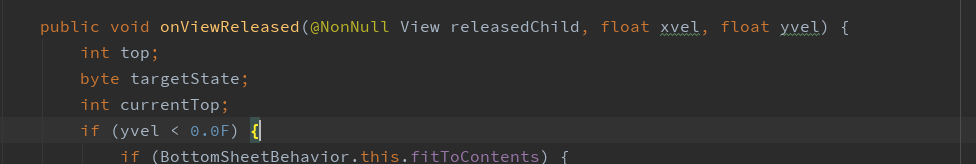
app:layout\_collapseMode="parallax"

app:layout\_collapseParallaxMultiplier="1"

# BottomSheetBehavior

bottomSheetDialog是底部弹窗，如果想要顶部弹窗，可以直接旋转coordinatorLayout 180度，然后再将内容区域旋转180度。同时需要设置窗口进入和退出的动画，否者window关闭时仍从底部消失。设置动画需要用dialog.window!!.setWindowAnimations(R.style.out) 来设置。

如果是顶部弹窗，需要将BottomSheetBehavior复制出来，同时将class NamelessClass\_1中的onViewReleased方法中的判断条件改为yvel>0.0f。用这个behavior来代替默认behavior



# 悬浮窗

使用Windowmanager.addView(view,params)添加

权限：android.permission.SYSTEM\_ALERT\_WINDOW

动态权限：

判断权限

Settings.*canDrawOverlays*(this)

申请权限

Intent intent =new Intent(Settings.*ACTION\_MANAGE\_OVERLAY\_PERMISSION*);  
startActivity(intent);

背景View响应事件：params.flags=WindowManager.LayoutParams.*FLAG\_NOT\_TOUCH\_MODAL*|WindowManager.LayoutParams.*FLAG\_NOT\_FOCUSABLE*;

Manager：remove删除、update更新

param.type:

if (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*O*) {  
 params.type = WindowManager.LayoutParams.*TYPE\_APPLICATION\_OVERLAY*;  
} else {  
 params.type = WindowManager.LayoutParams.*TYPE\_PHONE* ;  
}

# 文件选择

contentProvider

在onActivityResult中回调

URI的scheme

ContentResolver.SCHEME\_FILE

ContentResolver.SCHEME\_ANDROID\_RESOURCE

ContentResolver.SCHEME\_CONTENT

public static final String *SCHEME\_CONTENT* = "content";  
public static final String *SCHEME\_ANDROID\_RESOURCE* = "android.resource";  
public static final String *SCHEME\_FILE* = "file";

Intent中DataString的格式：

content://com.android.providers.media.documents/document/image%3A497635

图片可以直接用setImageURI（uri）来显示

可以用getContentResolver().openInputStream（uri）来获取数据流，对文件进行读操作

获取选中文件的路径

/\*String path = null;  
if(ContentResolver.SCHEME\_FILE.equals(uri.getScheme())){  
 path=uri.getPath();  
}else if(ContentResolver.SCHEME\_CONTENT.equals(uri.getScheme())){  
 if("com.android.providers.media.documents".equals(uri.getAuthority())){  
 String res[]=DocumentsContract.getDocumentId(uri).split(":");  
 if("image".equals(res[0])){  
 Cursor cursor=getContentResolver().query(MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI,  
 new String[]{"\_data"},"\_id=?",new String[]{res[1]},null);  
 if(cursor!=null&&cursor.moveToFirst()){  
 path=cursor.getString(0);  
 }  
 if (cursor != null) {  
 cursor.close();  
 }  
 }  
 }else {  
 Cursor cursor=getContentResolver().query(uri,new String[]{"\_data"},null,null,null);  
 if(cursor==null)return;  
 if (cursor.moveToFirst()){  
 do {  
 path=cursor.getString(0);  
 }while (cursor.moveToNext());  
  
 }  
 cursor.close();  
 }  
}  
if(path==null){  
 replaceIcon(uri);  
}else  
 replaceIcon(path);\*/

Tag：在android 7以上，写入图片、歌曲等文件需要发送扫描广播

Intent intent = new Intent(Intent.*ACTION\_MEDIA\_SCANNER\_SCAN\_FILE*);  
Uri uri = Uri.*fromFile*(file);  
intent.setData(uri);  
sendBroadcast(intent);

否则很有可能获取不到该文件

**调用系统图片裁剪：**

val intent = Intent("com.android.camera.action.CROP")  
intent.setDataAndType(uri, "image/\*")  
intent.putExtra("crop", "true")  
intent.putExtra("aspectX", ViewUtil.screenWidth())  
intent.putExtra("aspectY", ViewUtil.screenHeight())  
intent.putExtra("outputX", ViewUtil.screenWidth())  
intent.putExtra("outputY",ViewUtil.screenHeight())  
intent.putExtra("scale", false)  
intent.putExtra(MediaStore.*EXTRA\_OUTPUT*,Uri.fromFile(File(Environment.getExternalStorageDirectory().*absolutePath*+"/1.png")))  
intent.putExtra("return-data", false)  
intent.putExtra("outputFormat", Bitmap.CompressFormat.PNG.toString())  
intent.putExtra("noFaceDetection", true)  
intent.addFlags(Intent.*FLAG\_GRANT\_READ\_URI\_PERMISSION*)  
//intent.addFlags(Intent.FLAG\_GRANT\_WRITE\_URI\_PERMISSION)//报错、无权限  
startActivityForResult(intent,3)

# For foreach

For 用下标访问 普通数组适合 可增删

Foreach用迭代访问 链表结构适合 不可增删

# 图片选择器选择一个大图片，如何缩略显示到屏幕？

Glide设置了w h会缩略显示

# 图文混排

Spannable中插入ImageSpan，TextView中设置Spannable文字

Spannable content = new SpannableStringBuilder("占位 " + comment);

CenterImageSpan span = new CenterImageSpan(context,R.drawable.pic,ImageSpan.ALIGN\_BASELINE);

// 用ImageSpan替换文本

content.setSpan(span, 0, 2, Spannable.SPAN\_INCLUSIVE\_EXCLUSIVE);

TextView.setText(content);

每一处的ImageSpan必须新new的一个，如果多处的地方用了同一个ImageSpan，只会显示最后一个，前面的都不会显示。

## 网页式混合显示

android.text.style包下面有很多控制文字显示的类

AbsoluteSizeSpan：控制字体绝对大小的

RelativeSizeSpan：字体的相对大小 float

BulletSpan：放在段落开始才有效，显示一个圆点

UnderlineSpan：下划线

TypefaceSpan：设置字体

TextAppearanceSpan：字体样式、颜色、下划线颜色等

…………………

# Glide

GlideApp类不出来：

需要 添加编译时注解：

annotationProcessor 'com.github.bumptech.glide:compiler:4.8.0'

继承AppGlideModule 然后 make project 就会自动生成

如果使用的是kotlin，那么需要添加kotlin注解解析

Apply plugin: ‘kotlin-kapt’

kapt 'com.github.bumptech.glide:compiler:4.8.0'

kapt代替annotationProcessor

Github：

https://github.com/bumptech/glide/tree/master/samples/svg/src/main/java/com/bumptech/glide/samples/svg

对svg支持

还需要添加AndroidSVG：

implementation 'com.caverock:androidsvg-aar:1.3'

# MPAndroidChart

Android图表

LineChart 曲线表、BarChart 柱状表、CombinedChart 结合表

单一数据显示

LineDataSet set=new LineDataSet(List<?> list,String label)

Linechart.setLineData(new LineData(set))

多数据显示

LineDataSet set1,set2,set3….

List<ILineDataSet> data=new ArrayList<>();

Data.add(set1);

Data.add(set2);

Data.add(set3);

lineChar.setData(new LineData(data));

设置绘制模式：折线（默认）、递进、贝塞尔、水平贝塞尔

LineDataSet.setMode();

# 布局

getRealMetrics()和getMetrics()获取到的屏幕信息差别只在于widthPixels或heightPixels的值是否去除虚拟键所占用的像素，和是否全屏和沉浸模式无关

# Paging Library

PagedListAdapter adapter.submit(..)、PagedList- LivePagedListBulder()得到LiveData<PageList<T>>、DataSource、Diff

PagedList需要用LiveData包裹才可以

# MediaSession

**受控端**：

继承MediaBrowserService

不要重写onBind 如果重写onBind也要return supper.onBind(arg)

在onCreate里面实例化MediaSession

给MediaSession设置MediaSession.CallBack

设置FLAGS

调用setSessionToken(mMediaSession.getSessionToken())

状态变更：MediaSession.setPlaybackState(state)

歌曲切换：mediasession.setMetadata(data)

**控制端：**

实例化MediaBrowser并添加 ConnectionCallback

在onConnected()方法里面实现MediaControl，需要传入MediaBrowser的sessionToken

mediaControl.registerCallback(callback)实现回调、响应server的状态变化

MediaBrowser.subscribe(pid,callback) 监听server音乐列表的变化

# AIDL进程通信

1. aidl接口文件的包名必需相同
2. 在manifest文件里面设置explored=true
3. 服务端在OnBind里面返回IBind
4. bind=MyAIDLInterface.Stub(){//\*\*实现方法}
5. 客户端访问：Intent必需指明包名，intent.setPackegeName(“..”)
6. Intent Action 正确
7. ServiceConnection里面获取对象：MyAIDLInterface.Stub.asInterface(service)

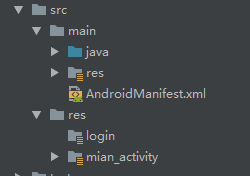
# JobScheduler、

# Files、Path API26

# Gradle里面设置多个资源路径

## Res路径





res.getSrcDirs()[0] 是指向src/main/res

根据其地址重新添加相对地址，添加方式：res.srcDirs “路径|File”、java.srcDirs “路径|File”

设置根目录

androidSourceSet.setRoot()

# Android studio 添加 C/C++支持

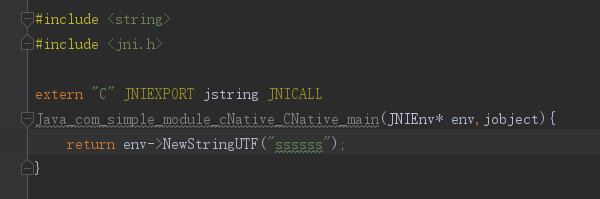
如果是新建工程：在新建时添加C支持

其他项目临时添加：

1.在SDK Manager中下载NDK、CMake、LLDB

2.在main文件夹下面新建文件夹 cpp、cpp文件夹里面写.cpp文件

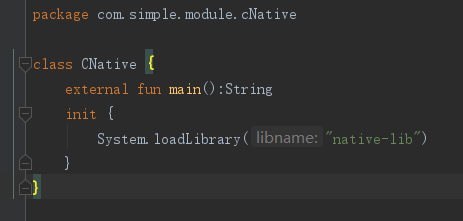
2.1 cpp格式：



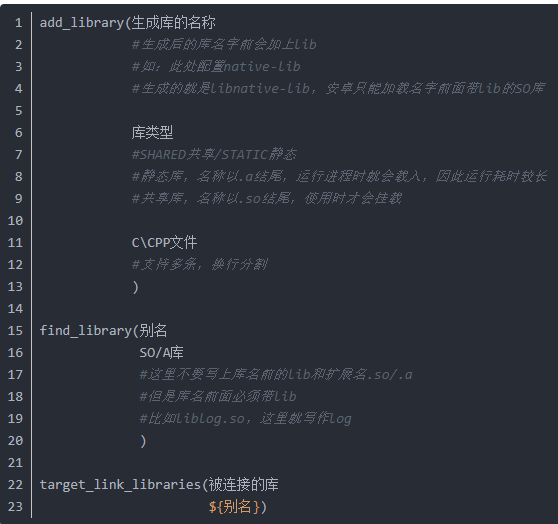
extern 那一行是返回类型

函数名称格式为Java\_包名\_类名称\_方法名称

具体kotlin代码：



3.在app文件夹下新建CMakeLists.txt（哪里都可以创建，只要最后指明路径都行）

3.1语法格式：

cmake\_minimum\_required(VERSION 3.4.1) #新版本中不能少

add\_library(

native-lib

SHARED

src/main/cpp/xxx.cpp

)

find\_library(

log-lib

log

)

target\_link\_libraries(

native-lib

${log-lib}

)

4.在build.gradle android{}内添加：

externalNativeBuild {

cmake {

path "CMakeLists.txt" //路径

}

}

5.在build.gradle defaultConfig{}内添加：

externalNativeBuild {//\*\*\*不添加就会编译全部版本

cmake {

cppFlags "x86-64,arm64-v8a,armeabi-v7a,x86"

}

}

6.重新编译就可以了

NestedScrollView +RecyclerView +notifyItemRemoved + 布局文件里面有EditText

当notifyItemRemoved(0)时会报错

NestedScrollView +RecyclerView+layerType=software

当RecyclerView的item过多时（Music里面为15）会提示toolarge然后就不显示

.9图放在不同的drawable里面显示效果不同

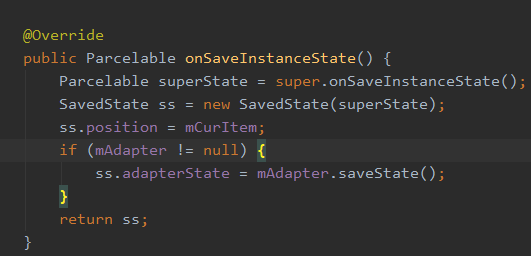
http://www.wanandroid.com/ulink

# OkHttp拦截器 Interceptor

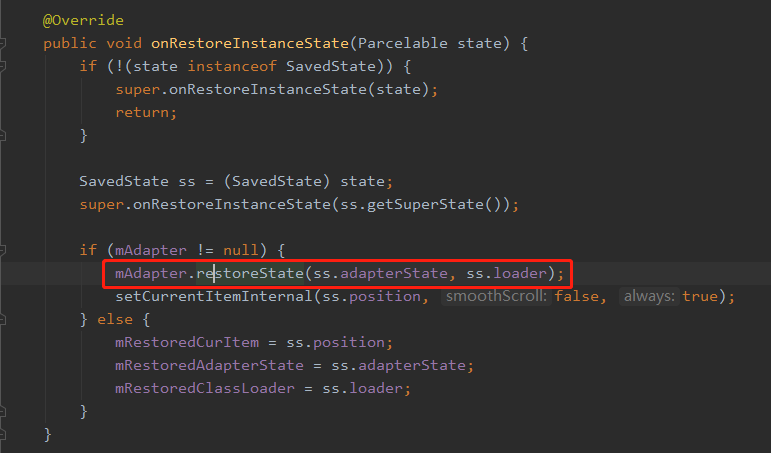
private val client=OkHttpClient.Builder().addInterceptor **{** chain **->** val request=chain.request()  
 .newBuilder()  
 .removeHeader("User-Agent")  
 .addHeader("User-Agent","xxx")  
 val response=chain.proceed(request.build())  
 return@addInterceptor response.newBuilder().body(object :ResponseBody(){  
 val body=response.body()  
 override fun contentLength(): Long { return body?.contentLength()?:0}  
  
 override fun contentType(): MediaType? { return body?.contentType() }  
 override fun source(): BufferedSource {  
 if(buffer==null){  
 buffer=Okio.buffer(object :ForwardingSource(body!!.source()){  
 override fun read(sink: Buffer, byteCount: Long): Long {  
 val len=super.read(sink, byteCount)  
 Log.i("log","\_\_\_->$len")  
 return len  
 }  
 })  
 }  
 return buffer!!  
 }  
 }).build()  
**}**.build()

RecyclerView + PagerSnapHelper 变ViewPager

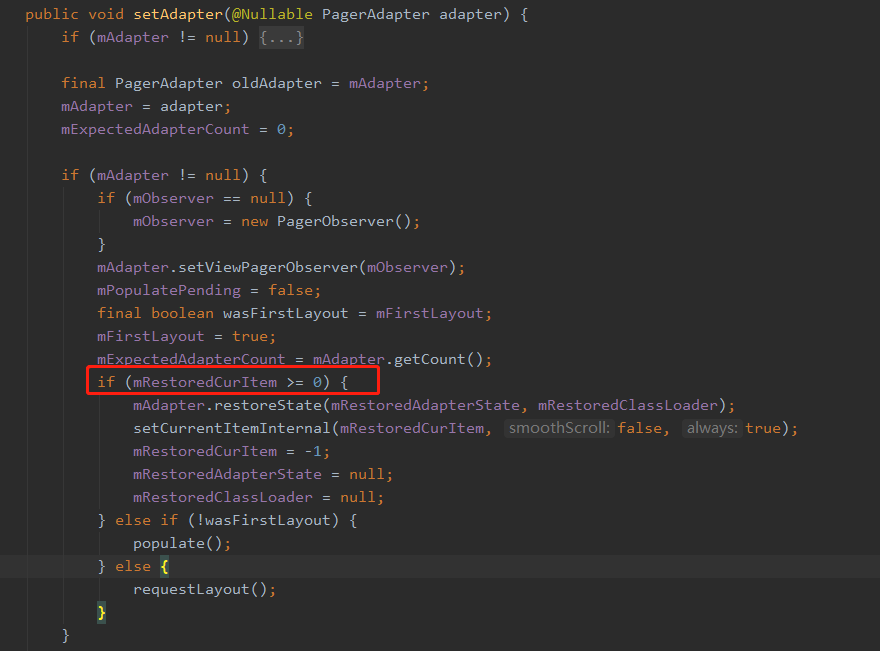
# ViewPager的状态恢复

当viewPager非正常销毁时，会调用adapter的saveState方法

恢复时会调用adapter的restoreState方法



所以ViewPager的Adapter应该重写这两个方法，当然如果只是普通使用ViewPager，可以不重写。如果要在ViewPager的setAdapter之后立即获取Fragment则必须重写。

ViewPager当销毁时adapter费控，那么重建时走的就是红色部分，在该条件下不会立即创建Fragment（需要手动创建），（populate会立即重建Fragment）

# Android Mapbox

Mapbox官方网站：<https://docs.mapbox.com/android/maps/overview/>

踩过的坑：

Source是用来提供数据的

Layer用来显示数据

所有的图片都要在Style里面注册，用时提供注册id来获取

Feature里面要保存位置信息（通过Geometry）和其他信息（通过Properties）

Layer可以设置Properties来对显示进行设置

Expression可以获取Feature里面信息

# Navigation 注意事项：

navigation xml需要app:startDestination作为默认启动项

和activity对比，Fragment无法得到onActivityResult里面的数据，可以在activity中监听回调并进行分发

某些情况下,Navigation.findNavController(v).navigate(R.id.action\_gotToSearchResult)是无法成功调用的，需要用View.post进行包裹调用（activity手动分发onActivityResult事件）。

如果是沉浸式，需要在View.post调用View.requestApplyInsets()

# Android Q文件兼容

Android Q文件系统沙盒化、无法用File类通过路径直接操作文件。

如果要创建文件，可以使用MediaStore来创建文件

创建File：

fun createFileUri(fileName: String):Uri?{  
 val cv=ContentValues()  
 cv.put(MediaStore.Files.FileColumns.*DISPLAY\_NAME*,fileName)  
 cv.put(MediaStore.Files.FileColumns.*MIME\_TYPE*,"text/plain")  
 cv.put(MediaStore.Files.FileColumns.*DATA*,Environment.getExternalStorageDirectory().*absolutePath*+fileName)  
 return MyApplication.ctx.*contentResolver*.insert(MediaStore.Files.getContentUri("external"),cv)  
}

根据返回的URI来对文件进行写入操作

如果文件存在，不会覆盖，而是新建一个文件名称为 “xxx(1)”

Tips：在compileSdkVersion API<29，如果是创建的Audio文件，DATA字段必须存在，API==29，所有的DATA字段都是Deprecate状态。

# AnnotationProcessor 代码生成

通过自定义注解，继承AbstractProcessor对自定义注解的解释可以在编译阶段生成新的代码和文件。

配置方式：

1：新建或者已经有一个项目

2：新建一个module 必须是java module (名称假若为compiler)

3：在app中的build.gradle中添加依赖

implementation project(":compiler")

annotationProcessor project(":compiler")

如果自定义的注解很简单,可以将注解的定义放在compiler中。

如果注解很多而且涉及到了android中的一些类型，就需要新建一个module（android module），同时为了在compiler中能够访问注解，需要在compiler的build.gradle文件中添加注解module的依赖。

4：compiler需要添加依赖：com.google.auto.service:auto-service:1.0-rc6

5：代码生成位置：

如果项目的classpath 版本<=3.2.0(3.2.0只是一个大概值，没有试过3.2.0-3.5.2之间的版本)，生成的java文件在build/generated/source/apt下面。

如果版本是更新的版本，还需要在compiler里面添加annotationProcessor 'com.google.auto.service:auto-service:1.0-rc6'，最终代码生成的位置在build/generated/ap\_generated\_source 下。

6：解释注解：

通过在compiler里面继承AbstractProcessor，重写和实现一些方法来生成代码。

7：build就可以生成代码了。

以上步骤是纯java代码的项目配置，如果是混用kotlin代码，kotlin里面的注解是无法识别的。要让注解器解析kotlin里面的注解，需要在app的build.gradle里面配置:

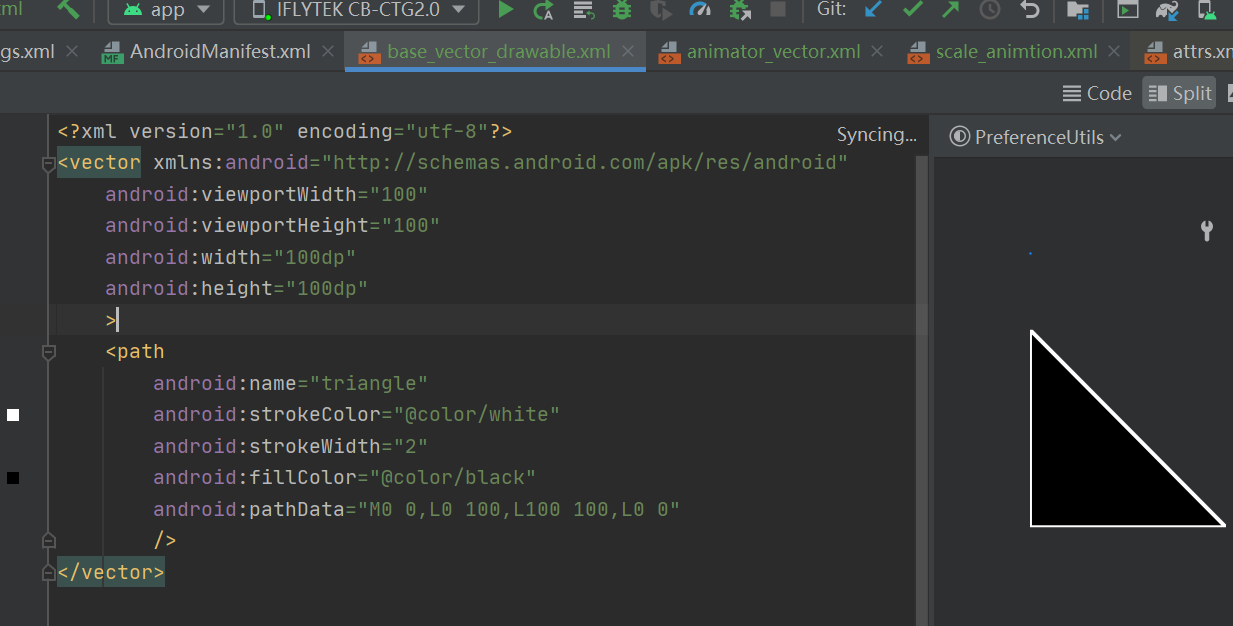
apply “apply kotlin-kapt”

将依赖annotationProcessor project(":compiler")改为kapt project(":compiler")

在defaultConfig里面配置javaCompileOptions.annotationProcessorOptions.includeCompileClasspath = true

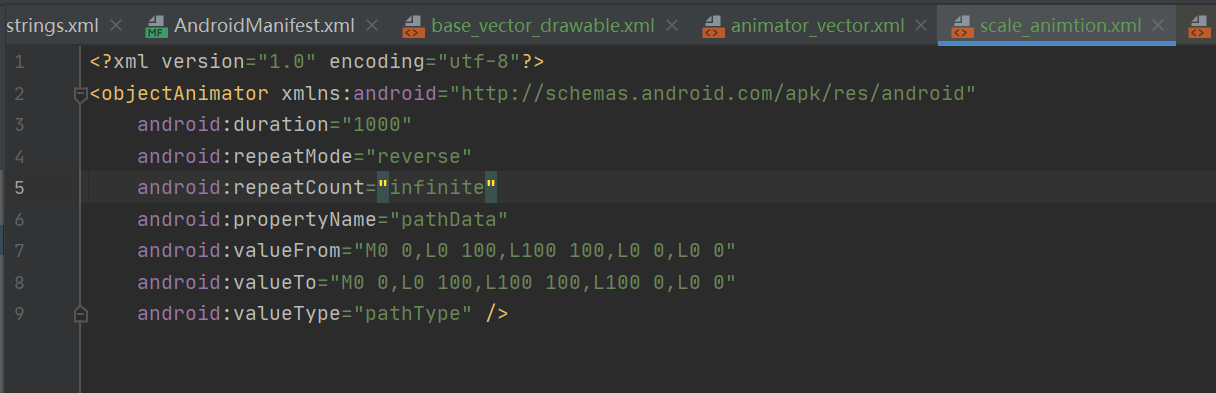
# Animated-vector 矢量图动画

1、先创建一个vector

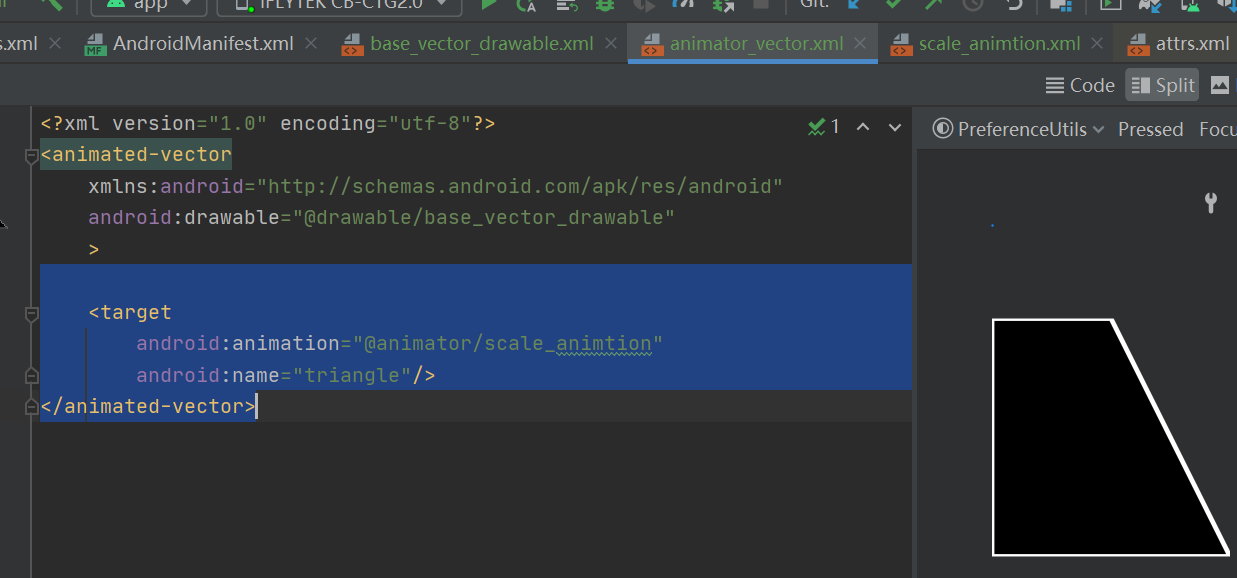


2、创建动画（位于animator文件夹内）

Valuefrom和valueTo需要点一一对应，（原理应该是不停改变点的位置来产生的动画）



3、创建animator-vector



然后就可以正常使用了

# Gradle 发布本地依赖

添加maven-publish插件

根节点添加task和上传代码

task androidSourcesJar(type: Jar) **{** from android.sourceSets.main.java.srcDirs  
**}**afterEvaluate **{** publishing**{** publications**{** mavenJava(MavenPublication)**{** groupId = 'cn.hengyumo'  
 artifactId = 'my-app'  
 version = '0.0.1'  
 artifact androidSourcesJar  
 from components.debug  
 //artifact bundleReleaseAar  
 **}  
 }  
 }  
}**

# Hook

## Activity启动hook

要监听所有activity的启动的时机，需要通过反射来实现。通过跟踪Activity.startActivity的执行流程，内部最终调用了Instrumentation里面的方法，所以需要替换activity持有的Instrumentation对象。Activity里面这个对象来源于ActivityThread，所以最终需要替换这里的Instrumentation。

1. 声明HookInstrumentation继承Instrumentation，重写execStartActivity方法，该方法是hide类型，直接写就行了。
2. execStartActivity方法内需要通过反射调用父对象的execStartActivity方法来保证流程的继续
3. Instrumentation里面还可以重写一些其它方法，比如callActivityOnNewIntent、newActivity等。
4. 通过反射将HookInstrumentation对象赋值到ActivityThread里面的mInstrumentation，ActivityThread对象可以通过sCurrentActivityThread反射获取。

针对从桌面启动的情况，由于execStartActivity需要Activity参数，所以不会走该方法

## 关于插件化的思考

插件化需要加载apk、jar中的类，需要DexClassLoader来进行加载。然后通过classLoader的loadClass加载外部的类。

针对加载apk的情况，如果需要加载apk中的drawable、color、string等资源，需要构建外部Context或者外部Resource对象。通过context或者resource来加载。

对于想实现app免安装运行，最大的难点是资源的问题。及宿主app和插件app中资源冲突的问题。（未完全解决，比如插件app自定义View中的自定义属性无法生效的问题，自定义View可以通过LayoutInflator.Factory来加载但属性不知如何解决）