# Kotlin项目实战

1. 需要继承父类的非abstract方法时，必须在父类的该方法前加上open关键字
2. 在使用reified关键时，该方法前必须加上inline关键字
3. 在封装startActivity方法时，需要封装的参数是一个泛型
4. reified T:BaseActivity 指定泛型的范围，必须为BaseActivity或者它的子类
5. reified使用，具体化一个泛型T并增加一个getName方法，返回值为当前类名
6. ?attr/actionBarSize，可以适应不同主题下的actionBar的高度
7. Error : Program type already present: android.support.design.widget.CoordinatorLayout$

原因：constrain:constrain-layout 版本与design版本依赖冲突

解决方案：加入implementation 'com.android.support:design:27.1.1'

1. Kotlin中interface中的方法是可以实现的，java中是不允许实现的
2. by lazy 线程安全，且只有在使用的时候才被初始化
3. val关键字，一旦初始化就不能被改变
4. abstract和interface可被继承方法的区别：abstract中必须加上abstract或者open关键字，interface中则不需要加任何关键字
5. app:showAsAction="always" 设置菜单栏永久显示，而不是点击打开显示
6. toolbar.setOnMenuItemClickListener {  
    when(it.itemId){  
    R.id.setting -> {  
    //跳转到设置界面  
    toolbar.context.startActivity<SettingActivity>()  
    }  
    }  
    true  
   }

kotlin和java调用的特性

如果java接口中只有一个未实现的方法，可以省略这个对象，直接覆写这个方法

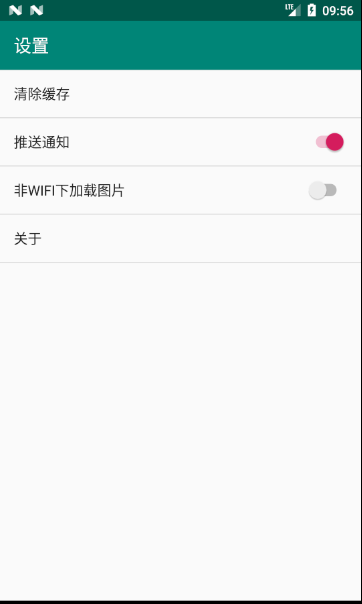
1. PreferenceFragment和PreferenceActivity可以实现实现设置界面

class SettingFragment : PreferenceFragment()

在onCreateView方法中调用addPreferencesFromResource(R.xml.setting)，setting.xml为配置标签标选项的文件，格式如下：

<?xml version="1.0" encoding="utf-8"?>  
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">  
 <Preference  
 android:key="clear\_cache"  
 android:title="清除缓存"/>  
 <SwitchPreference  
 android:key="push"  
 android:title="推送通知"/>  
 <SwitchPreference  
 android:key="no\_wifi"  
 android:title="非WIFI下加载图片"/>  
 <Preference  
 android:key="about"  
 android:title="关于"/>  
</PreferenceScreen>

实现效果为：



或者切换按钮的状态：

//获取推送通知有没有被选中  
val preferences = PreferenceManager.getDefaultSharedPreferences(this)  
val push = preferences.getBoolean("push", false)

1. class FragmentUtil private constructor() 私有化构造方法
2. companion object {  
    val fragmentUtil by lazy { FragmentUtil() }  
   }

处理静态变量和静态方法，类似于java的static关键字

1. "?"加在变量名后，系统在任何情况不会报它的空指针异常。  
   "!!"加在变量名后，如果对象为null，那么系统一定会报异常！
2. init {  
    //传入了this，不用再调addView去添加  
    View.inflate(context, R.layout.item\_home, this)  
   }

调用该类的构造方法时会被执行

1. android.support.v7.widget.CardView使用：

android:elevation="5dp" 设置井深宽度

app:cardUseCompatPadding="true" 在没有井深效果时加上用于适配

1. val s = response.body()?.string() 在使用okhttp请求时，用string()方法替换toString()方法
2. 对象获取：

javaClass :

person.javaClass

person::class.java

类获取：

kClass:

Person::class

person.javaClass.kotlin

(Person::class as Any).javaClass

Person::class.java

1. val videoList = gson.fromJson<List<HomeItemBean>>(video.toString(),object :TypeToken<List<HomeItemBean>>(){}.type)

Gson的使用，注意第二个参数的写法

1. objec关键字语法含义：将类的声明和定义该类的单例对象结合在一起（即通过object就实现了单例模式）

object RepositoryManager{

fun method(){

println("I'm in object declaration")

}

}

1. 类型强转使用as关键字：

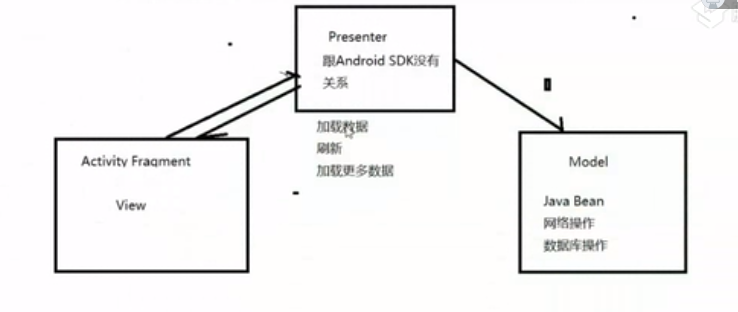
val itemView = p0.itemView as HomeItemView

1. 监听RecyclerView滑动
2. //监听列表滑动  
   recycleView.addOnScrollListener(object :RecyclerView.OnScrollListener(){  
     
    override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {  
    println("dx=$dx,dy=$dy")  
    }  
     
    override fun onScrollStateChanged(recyclerView: RecyclerView, newState: Int) {  
    when(newState){  
    //空闲状态  
    RecyclerView.SCROLL\_STATE\_IDLE -> println("idle")  
    //滑动状态  
    RecyclerView.SCROLL\_STATE\_DRAGGING -> println("dragging")  
    //滑动后自然沉降的状态  
    RecyclerView.SCROLL\_STATE\_SETTLING -> println("settling")  
    }  
    }  
   })
3. 图片加载框架Glide使用注意事项：placeholder方法必须放在load方法之前
4. Glide  
    .with(context)  
    .load("http:"+data.image)  
    .centerCrop()  
    .placeholder(R.mipmap.placeholder)  
    .into(bg);
5. MVP模型：

view：视图界面

presenter：逻辑处理

model：数据服务



1. 获取当前泛型类型的type值：

//获取泛型类型的type（可能有多个泛型，所以是一个数组）  
val type = (this.javaClass.genericSuperclass as ParameterizedType).actualTypeArguments[0]