目录

[**介绍自己的项目，为什么使用这些框架**](#介绍自己的项目) **1**

[**项目遇到的困难**](#项目遇到的困难) **4**

[**View的绘制**](#View的绘制)

[1/自定义了一个ExpandableTextView](#自定义了一个ExpandableTextView)

[**对其的TextView**](#对其的TextView)

[**Dagger单例注入**](#Dagger单例注入)

1. **[介绍自己的项目，为什么使用这些框架](#介绍自己的项目)**

**使用dagger**：a.增加开发效率，将新建对象的活都交给dagger来做,也不用写单例模式

b.更好的管理实例类，全局性对象交给application管理。各个activity对应一个component，里面的对象与activity生命周期相同。

c.解耦，若某对象的构造函数发生变化，要修改很多代码，利用dagger解耦可以只需在module中进行修改

**使用RxJava**：正常写一个异步线程更新ui的操作很繁琐，利用RXJava使得代码可读性更强，同时切换线程的代码也非常简单，利用map还可以轻松的转换获得数据的类型，通过响应式编程可解决背压问题

**使用Retrofit:**网络请求的方式非常简单利用接口即可实现，同时适配多种callAdapter，**适配RXjava，让网络请求更加的简洁。**

**MVP：**解耦activity，让activity单纯的为显示界面的功能。交数据交互工作交给presenter来使用。

1. [**项目遇到的困难**](#项目遇到的困难)

**有关自定义View**

**[View的绘制](#View的绘制)**

Perform**Measure**——measure——onMeasure——measureChildren——children的measure

MeasureSpec

1. unspecified 要多少给多少
2. at\_most 顶天给多少
3. exactly 确切的值

对于Wrap——content需要重写onMeasure

dectorView通过屏幕的大小和自身的layoutParam决定

其他的View根据父类的MeasureSpec以及自身的LayoutParam决定

Perform**Layout**——layout（确定自身的布局）——setFrame——onLayout(ViewGroup必须重写，children调用layout来完成自身的布局。)

ViewGroup一定要实现onLayout

Perform**Draw**——drawBackground——onDraw（空方法 绘制自己）——distachDraw（将绘制事件分发给子事件）——onDrawScrollBars（绘制装饰）

**Visible属性** visible可见 invisible不可见，保留控件位置，gone不可见，不保留控件位置，控件会被完全移除

**困难**。

[1/自定义了一个ExpandableTextView](#自定义了一个ExpandableTextView)

**package** com.ali.zhihu.widget;  
  
**import** android.content.Context;  
**import** android.content.res.TypedArray;  
**import** android.graphics.drawable.Drawable;  
**import** android.util.AttributeSet;  
**import** android.view.View;  
**import** android.view.ViewGroup;  
**import** android.widget.ImageButton;  
**import** android.widget.LinearLayout;  
**import** android.widget.TextView;  
  
**import** com.ali.zhihu.R;  
  
**import** org.w3c.dom.Text;  
  
*/\*\*  
 \* Created by Administrator on 2018/2/9.  
 \*/***public class** ExpandableTextView **extends** LinearLayout{  
 **private int maxCollapseLines**;  
 **private** TextView **expandableText**;  
 **private** ImageButton expandCollapseButton;  
 **private boolean collapsed**;  
 **private** Drawable **expendDrawable**;  
 **private** Drawable **collapsedDrawable**;  
 **private boolean isChange**;  
 **private** OnExpandStateChangeListener **listener**;  
  
 **public** ExpandableTextView(Context context) {  
 **super**(context,**null**);  
 }  
  
 **public** ExpandableTextView(Context context, AttributeSet attrs){  
 **super**(context,attrs);  
 init(context,attrs);  
 }  
  
 **private void** init(Context context, AttributeSet attrs){  
 setOrientation(***VERTICAL***);  
 **isChange** = **false**;  
 **collapsed** = **true**;*//表示折叠* TypedArray array = context.obtainStyledAttributes(attrs, R.styleable.***ExpandableTextView***);  
 **maxCollapseLines** = array.getInteger(R.styleable.***ExpandableTextView\_maxCollapseLines***,2);  
 **expendDrawable** = array.getDrawable(R.styleable.***ExpandableTextView\_expandDrawable***);  
 **collapsedDrawable** = array.getDrawable(R.styleable.***ExpandableTextView\_collapseDrawable***);  
 array.recycle();  
 }  
  
 @Override  
 **protected void** onFinishInflate() {  
 **super**.onFinishInflate();  
  
 **expandableText** = (TextView)findViewById(R.id.***expandable\_text***);  
 expandCollapseButton = (ImageButton)findViewById(R.id.***expand\_collapse***);  
 expandCollapseButton.setImageDrawable(**collapsed**?**collapsedDrawable**:**expendDrawable**);  
 expandCollapseButton.setOnClickListener(**new** OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **isChange** = **true**;  
 **collapsed** = !**collapsed**;  
 **if**(**listener** != **null**){  
 **listener**.onExpandStateChanged(**collapsed**);  
 }  
 expandCollapseButton.setImageDrawable(**collapsed**?**collapsedDrawable**:**expendDrawable**);  
 requestLayout();  
 }  
 });  
 }  
  
 @Override  
 **protected void** onMeasure(**int** widthMeasureSpec, **int** heightMeasureSpec) {  
 **if**(!**isChange** || expandCollapseButton.getVisibility() == ***GONE***){  
 **super**.onMeasure(widthMeasureSpec, heightMeasureSpec);  
 **return**;  
 }  
 **isChange** = **false**;  
  
 expandCollapseButton.setVisibility(***GONE***);  
 **expandableText**.setMaxLines(Integer.***MAX\_VALUE***);  
 **super**.onMeasure(widthMeasureSpec,heightMeasureSpec);  
  
 **if**(**expandableText**.getLineCount() <= **maxCollapseLines**){  
 **return**;  
 }  
 **if**(**collapsed**){  
 **expandableText**.setMaxLines(**maxCollapseLines**);  
 }  
 expandCollapseButton.setVisibility(***VISIBLE***);  
 **super**.onMeasure(widthMeasureSpec,heightMeasureSpec);  
 }  
  
 **public void** setText(String text){  
 **isChange** = **true**;  
 **expandableText**.setText(text);  
 }  
  
 **public void** setText(String text,**boolean** collapse){  
 **this**.**collapsed** = collapse;  
 expandCollapseButton.setImageDrawable(**collapsed**?**collapsedDrawable**:**expendDrawable**);  
 setText(text);  
 getLayoutParams().**height** = ViewGroup.LayoutParams.***WRAP\_CONTENT***;  
 requestLayout();  
 }  
  
 **public interface** OnExpandStateChangeListener{  
 **void** onExpandStateChanged(**boolean** collapse);  
 }  
  
 **public void** setExpandStateChangeListener(OnExpandStateChangeListener listener){  
 **this**.**listener** = listener;  
 }  
}

**[对其的TextView](#对其的TextView)**

**package** com.ali.zhihu.widget;  
  
**import** android.content.Context;  
**import** android.graphics.Canvas;  
**import** android.graphics.Paint;  
**import** android.text.Layout;  
**import** android.text.StaticLayout;  
**import** android.text.TextPaint;  
**import** android.util.AttributeSet;  
**import** android.widget.TextView;  
  
*/\*\*  
 \* Created by Administrator on 2018/2/9.  
 \*/***public class** JustifyTextView **extends** TextView {  
 **private int mLineY**;  
 **private int mViewWidth**;  
  
 **public** JustifyTextView(Context context, AttributeSet attrs){  
 **super**(context,attrs);  
 }  
  
 @Override  
 **protected void** onLayout(**boolean** changed,**int** left,**int** top,**int** right,**int** boottom){  
 **super**.onLayout(changed,left,top,right,boottom);  
 }  
  
 @Override  
 **protected void** onDraw(Canvas canvas){  
 TextPaint paint = getPaint();  
 paint.setColor(getCurrentTextColor());  
 paint.**drawableState** = getDrawableState();  
 **mViewWidth** = getMeasuredWidth() - getPaddingRight();  
 String text = getText().toString();  
 **mLineY** = getPaddingTop();  
 **mLineY** += getTextSize();  
 Layout layout = getLayout();  
  
 **if**(layout == **null**){  
 **return**;  
 }  
  
 Paint.FontMetrics fm = paint.getFontMetrics();  
 **int** textHeight = (**int**)(Math.*ceil*((fm.**descent** - fm.**ascent**)));  
 textHeight = (**int**)(textHeight \* layout.getSpacingMultiplier() + layout.getSpacingAdd());  
 **int** textLine = layout.getLineCount();  
  
 **for**(**int** i = 0; i < textLine;i++){  
 **int** lineStart = layout.getLineStart(i);  
 **int** lineEnd = layout.getLineEnd(i);  
 String lineText = text.substring(lineStart,lineEnd);  
 **float** lineWidth = StaticLayout.*getDesiredWidth*(text, lineStart, lineEnd, paint);  
  
 **if**(i == textLine - 1){  
 *//最后一行则直接画* canvas.drawText(lineText,0,**mLineY**,paint);  
 }**else**{  
 **if**(needScale(lineText)){  
 drawScaleText(canvas,lineText,paint,lineWidth);  
 }**else**{  
 canvas.drawText(lineText,0,**mLineY**,paint);  
 }  
 }  
 **mLineY** += textHeight;  
 }  
  
 }  
 **private void** drawScaleText(Canvas canvas, String lineText, TextPaint paint,**float** lineWidth){  
 **int** x = 0;  
 **int** gapCount = lineText.length() - 1;  
 */\*if(lineText.length() > 2 && lineText.charAt(0) == ' ' && lineText.charAt(1) == ' '){  
 float blankWidth = StaticLayout.getDesiredWidth(" ",paint);  
 x += blankWidth;  
 canvas.drawText(" ",0,mLineY,paint);  
 i = 2;  
 }\*/* **float** gap = (**mViewWidth** - lineWidth)/gapCount;  
 **for**(**int** i = 0;i < lineText.length();i++){  
 canvas.drawText(String.*valueOf*(lineText.charAt(i)),x,**mLineY**,paint);  
 x += gap + StaticLayout.*getDesiredWidth*(String.*valueOf*(lineText.charAt(i)),paint);  
 }  
 }  
  
 **private boolean** needScale(String lineText){  
 **if**(lineText == **null** || lineText.length() == 0){  
 **return false**;  
 }  
 *//如果没有遇到原先设置的换行，则自动调整字间的间距* **return** lineText.charAt(lineText.length() - 1) != **'\n'**;  
 }  
}

1. **[Dagger单例注入](#Dagger单例注入)**

@Scope

@Retention(RetentionPolicy.Running)

Public @interface perActivity{

}

Public class person{

Public person(){

}

}

@Module

Public class mainModule{

@Provide

@PerActivity

Person providePerson(){

}

}

@perActivity

@component(dependencies = AppComponent,module = mainModule.class)

Interface mainComponent{

Void inject(Activity actibity);

FragmentComponent fragmentComponent();

}

Public class activity eatends Activity{

@inject

Person person;

}

则在activity生命周期中，只会创建一个person。

由于被Scope修饰的对象工程会被存入DoubleCheck中

DoubleCheck用类似双重if的方式创建单例模式，保证只会创建一个对象

若没用Scope修饰，则在DaggerMainComponent中通过工厂创建

若在此时用Singleton修饰，不会创造全局单例，Singleton其实为Scope的一种，若想用它创建，只能创建在Application中，由于Application是全局性的，所以被Singleton创建出来的单例就是全局性的。

/\*\*

\* 自定义一个限定符

\*/

@Qualifier//限定符

@Documented

@Retention(RetentionPolicy.RUNTIME)

public @interface Type {

String value() default "";//默认值为""

}

@Module

public class SaladModule {

.........

@Type("normal")//使用限定符来区别使用哪个构造函数返回对象

@Provides

public Apple provideNormalApple() {

return new Apple();

}

@Type("color")//使用限定符来区别使用哪个构造函数返回对象

@Provides

public Apple provideColorApple(String color) {

return new Apple(color);

}

// 由于我们的Apple构造函数里使用了String,所以这里要管理这个String(★否则报错)

// int等基本数据类型是不需要这样做的

@Provides

public String provideString(){

return new String();

}

@Component(modules = {SaladModule.class})//指明要在那些Module里寻找依赖

public interface SaladComponent {

.........

@Type("normal")//关键靠这个限定符来区分调用哪个构造函数

Apple provideNormalApple();//这个方法名并不重要，只要是provide开头就行，但是建议和module里一致

@Type("color")

Apple provideColorApple();//若想暴露给dependenci，则需要提供该函数，若想暴露给subComponent 则不用。

String provideString();

//注意：下面的这个方法，表示要将以上的三个依赖注入到某个类中

//这里我们把上面的三个依赖注入到Salad中

//因为我们要做沙拉

void inject(Salad salad);

}

}

public class Salad {

.........

@Inject

@Type("normal")

Apple normalApple;

@Inject

@Type("color")

Apple colorApple;

public Salad() {

// DaggerSaladComponent编译时才会产生这个类，

// 所以编译前这里报错不要着急(或者现在你先build一下)

SaladComponent saladComponent = DaggerSaladComponent.create();

saladComponent.inject(this);//将saladComponent所连接的SaladModule中管理的依赖注入本类中

makeSalad(pear, banana, saladSauce);

}

private void makeSalad(Pear pear, Banana banana, SaladSauce saladSauce) {

Log.e("TAG", "我在搅拌制作水果沙拉");

}

}

1. **Activity生命周期，给你具体场景，生命周期应该是怎样的**

OnCreate——onStart——onRestoreInstanceState——onresume——onPause——onsaveOnstanceState——onStop——onDestroy

Single

singleTop（防止点击多次通知栏）

singleTask(保证只有一个聊天界面)

singleInstance(微信共享地图界面)

1. **Service的启动方式，如何关闭**

startService start之后，activity与Service无任何关系，activity不可以控制Service 用stopService关闭

bindService bind之后，activity与Service绑定，activity可以控制Service做一些事情

要unBinde

所有的客户端共享一个IBinder，要关闭BindService需要所有的客户端解除绑定

1. **广播两种注册方式，有什么区别，是不是所有的接收者都能够接收广播，广播如何截断**

静态注册：在androidManifest.xml文件中静态注册，该广播接收器android不可以自动销毁。在activity销毁后，任然可以接受信息。在<application>下添加<receiver>,在<receiver>中添加多个<intent-filter>

动态注册：继承BroadcastReceiver，重写onReceive方法。在registerReceiver中注册广播接收器，传参为BroadCastReceiver与IntentFilter，可以用unregisterReceiver解除注册，解除注册后无法接收到广播

<Intent-filter

<action> intent行为

<category> intent类型 默认有个default 注册activity时一定要添加

<date>数据类型，intent要传递的数据

/>

Public class myBroadcastReceiver extends BroadCastReceiver{

@Override

Public void onReceive(Context context,Intent intent){

}

}

Public class myActivity extends Activity{

myBroadcastReceiver;

IntentFilter intentfilter = new IntentFilter();

intentfilter.addAction(“aaaa”);

registerReceiver(myReceiver,intentfilter);

Intent = new Intent();

Intent.setAction(“ss”);

sendBroadcast(intent);

}

广播分为普通广播，有序广播，本地广播。

有序广播按接收器的优先级接受广播，可用abortBroadcast()对广播进行拦截。BroadcastReceiver可以使用setResult系列函数来结果传给下一个BroadcastReceiver，通过getResult系列函数来取得上个BroadcastReceiver返回的结果，并可以abort系列函数来让系统丢弃该广播让，使用该广播不再传送到别的BroadcastReceiver。短信接受为有序广播，可以拦截短信，电话。

本地广播广播只在本地传播。防止所有应用程序能接收到带来安全隐患

滞留（粘性）广播，若无接受者，一直滞留到有接受者了才接受，保障动态注册广播接收器时能迅速收到广播，监听电池电量

广播事件：来电，电量过低，电量充满，屏幕关闭，屏幕唤醒，飞行模式，

1. 事件分发，如何拦截事件
2. View绘制，specMeasure
3. Java集合 如何实现生产者消费者
4. Java如何构造线程
5. new Thread 重写run函数
6. 用Runnable参数去创造Thread，
7. Java锁有哪些，具体到如何使用
8. Sleep与wait差别，如何唤醒
9. 了解哪些设计模式，单利模式，dlc原理