Bilibili 反编译学习记录

Bilibili的客户端还是非常漂亮而高效的，至少比支付宝、微信的客户端看起来要快很多。

由于比较喜欢bilibili的底部dock栏，因此反编译安装包看其实现，同时也学习一下大公司的产品开发手段 嘿嘿~

分包策略：

Bilibili安装包内有3个classdex文件，分别为

Classes2：放的是bilibili自己实现的一些方案，首层只有一个bl包，里面所有代码都经过了混淆，所以应该是纯自家代码。

Classes3：这个包里面的东西就比较复杂了，几乎所有开源包和api都被打包到这里面了，所以这里面应该主要放的是相对更新较慢或者不怎么需要更新的代码和开源库、api。

Classes4：这里主要放的是okhttp3，retrofit2，kotlin的包，应该也算是更新较少的内容。

Bilibili是支持增量更新的，而源码的这种分布也给增量更新带来了好处，更新比较频繁的代码在一个单独的源码包里面，省去了另外两个不经常更新部分代码的重复下载，app只有小变动的话，增量更新包可能只有1Mb不到，这个设计带来的用户体验是非常良好的，用户不需要为大量的更新流量发愁，也不需要那么长的更新等待时间，同时还减少了服务器压力，一局多得呀。

首先安利一个英语单词，后面会大量出现哦。

Tint n，色彩，浅色

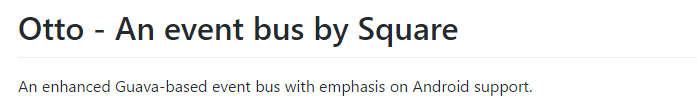
v，染色

bili客户端就是通过这种重写Tint\*View的机制，来实现其粉色少女系界面的，值得学习。

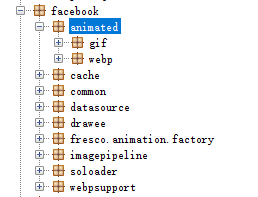
1. 使用的开源库

这里可以看到很多熟悉的名字，看来安卓开发中，一些库的使用是必不可少的。

* 1. AOSP：开发者还真是诚实得可爱，用来作为参考代码，嗯 学到了。
  2. Apache Commons IO/Lang： 这个库应该是用来控制流的
  3. Citrus Project：同样没有听说过
  4. Otto：事件管理器 square出品



* 1. Okhttp：http的一个封装，支持高并发连接
  2. Okio：io事件的封装库
  3. 阿里爸爸的FastJSON库
  4. Square Picasso：Square的图像库，实现了图像的在线获取、批量展示。
  5. Google Zxing ： 谷歌的二维码生成、识别库
  6. Google Archieve Patcher：bilibili增量升级的实现就是依靠这个库来实现的，好东西，要学着。
  7. Facebook animates ： 脸书的开源动图库，支持gif和webp两种格式



* 1. Android-job： 来自印象笔记的Evernore-job 开源库，用于定期执行任务Scheduler。
  2. **Retorfit2**：网络加载框架 常用来给网站制作客户端，需要配合GSON okhttp使用。
  3. Lottie ： 终于有一个我知道的东西了，airbnb开发的动画库
  4. Butterknife：大名鼎鼎的黄油刀

**第三方SDK**

1. Tencent … Map?
2. Tencent Bugly：一个应用崩溃信息反馈组件
3. Tencent Connect：应该是接入QQ和微信的api，用于登录、分享
4. Sina.Weibo：新浪微博的接入与分享
5. Huawei HMS：华为账号服务
6. ColorOSMcSdk：oppo的系统推送sdk，真的是做手机的都是大爷，谁都要照顾着。
7. umeng：看样子是广告咯
8. Alipay 付款api

其他东西

1. Httpdns：使用自己的dns ?

Bilibili自家应用框架

1. Live
2. Pay
3. Login
4. Player 使用的好像是nativeGL
5. Accounts
6. 首页布局

由外而内分别是

CoordinatorLayout

FrameLayout 【Content】

Tabhost【底部NavigationBar，自定义实现】

TintAppBarLayout【似乎是顶部导航，自动收缩特性】

Toolbar 【使用了V7库的Toolbar实现】

LinearLayout [horizontal]

Imageview

FrameLayout

VerifyAvatarFrameLayout

ImageView

ImageView

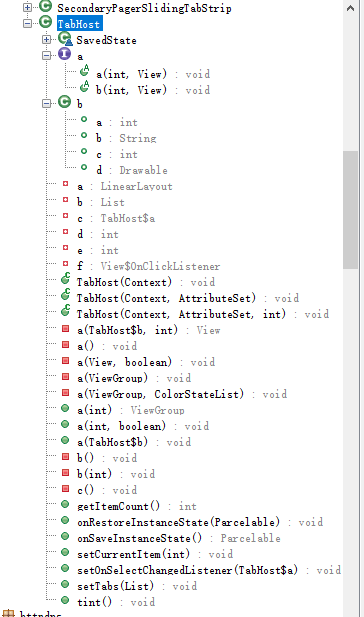
TintTextView

ToolbarCenterTextView【似乎是标题】

1. Bilibili底部导航栏实现

首先我们打开bili的Tabhost实现，位于com.bilibili.lib.widgets.Tabhost

一个很关键的point：虽然和系统的Tabhost同名，但是其实并没有继承原生Tabhost类，而是完全自己的实现（似乎是因为AOSP的Tabhost不支持底部导航）。

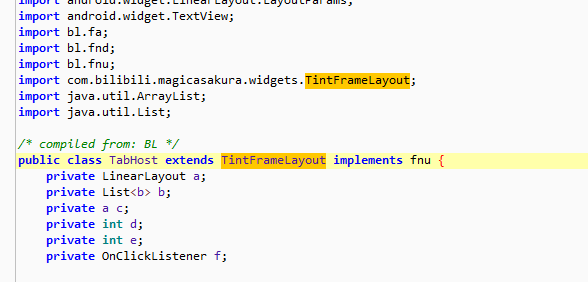


这里先给出几个结论，否则可能看起来一头雾水。

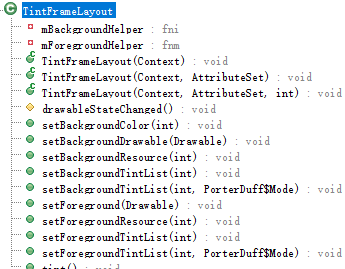
* 所有的view都有方法setSelected（）
* a(view,bool) 对view调用setSelected(bool），如果view是个viewgroup，递归地对其子view调用setSelected(bool)
* a(int,bool) int是Tab在容器中的下标，切换当前选中view为Tabs容器中的第int个，调用a(<之前选中的view>，false)取消其选中状态，然后调用a(<int对应的view>,true)选中当前view，如果bool为真，调用int对应view的选中回调方法。

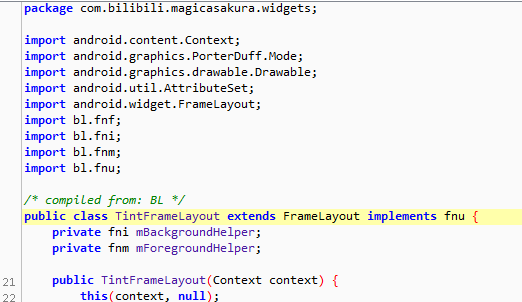
分析一个View的主要步骤是看其点击事件和绘制流程，由于是最终继承与FrameLayout，所以这里不能直接看到其绘制view的过程。

然后发现其继承自magic sakura（魔法樱花？）包下面的TintFrameLayout



跳过去看一看！

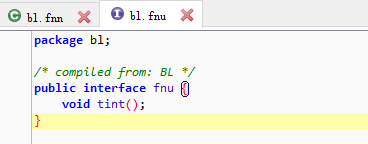




其中定义了一些简单的函数，然后比较关键的地方应该在于tint()这个函数

TintFrameLayout 继承自FrameLayout，实现了fnu接口。

Fnu接口出乎意料的简洁：



单一个tint()函数。

通过对这个TintFrameLayout的分析发现，这个类主要是实现了背景的接口化管理。

包括SetBackground\*(),SetForeground\*()这一类函数，然后重写了当状态实效时的回调函数drawableStateChanged()函数，然后tint()这个函数暂时没有搞懂。

1. SavedState恢复机制

static class SavedState extends BaseSavedState {

    public static final Creator<SavedState> CREATOR = new Creator<SavedState>() {

        public /\* synthetic \*/ Object createFromParcel(Parcel parcel) {return a(parcel);}

        public /\* synthetic \*/ Object[] newArray(int i) {return a(i);}

        public SavedState a(Parcel parcel) {return new SavedState(parcel, null);}

        public SavedState[] a(int i) {return new SavedState[i];}

    };

    int a;

    /\* synthetic \*/ SavedState(Parcel parcel, AnonymousClass1 anonymousClass1) {

        this(parcel);

    }

    public SavedState(Parcelable parcelable) {

        super(parcelable);

    }

    private SavedState(Parcel parcel) {

        super(parcel);

        this.a = parcel.readInt();

    }

    public void writeToParcel(Parcel parcel, int i) {

        super.writeToParcel(parcel, i);

        parcel.writeInt(this.a);

    }

}

首先实现了一个SavedState的Creator。

public static class b {

    public int a;

    public String b;

    @DrawableRes

    public int c;

    public Drawable d;

}

这一部分应该是显示的底部图表+文本组合，使用了android.support.annoation.DrawableRes注释，能力有限，不知道干什么用的。

下面是主要的构造方法。

public TabHost(Context context, AttributeSet attributeSet, int i) {

    super(context, attributeSet, i);

    this.b = new ArrayList();

    this.d = 0;

    this.e = 0;

    this.f = new OnClickListener(this) {

        final /\* synthetic \*/ TabHost a;

        this.a = r1;

        public void onClick(View view) {

            int indexOfChild = this.a.a.indexOfChild(view); //获取被点击的view下标

            if (this.a.e != indexOfChild) { //如果不是当前选中的view

                this.a.a(this.a.a.getChildAt(this.a.e), false); //似乎是取消选中状态，取消高亮显示图标和文字（选中状态）

                this.a.a(this.a.a.getChildAt(indexOfChild), true);

                this.a.e = indexOfChild; //记录当前选中的view下标

                this.a.d = view.getId(); //当前选中的viewid 应该是个linearLayout吧

                if (this.a.c != null) { // selectChangedListener回调方法

                    this.a.c.a(indexOfChild, view);

                    return;

                }

                return;

            }

            this.a.e = indexOfChild;

            this.a.d = view.getId();

            if (this.a.c != null) {

                this.a.c.b(indexOfChild, view);

            }

        }

    };

    a();

}

private void a() {

    LayoutInflater.from(getContext()).inflate(2131428158, this, true);

    this.a = (LinearLayout) findViewById(2131297096);

}

构造方法中，给f对象设置了点击事件，点击事件里面主要是对a对象进行操作。然后我们回过头来看看a和f是什么东东。

private LinearLayout a; //Tabs容器，也就是可选的icon+文字组合 的容器

private List<b> b;

private a c; // Tab标签被选状态改变的回调方法。

private int d; //当前选中的viewid ，也就是LinearLayout的id

private int e; //当前选中的标签下标

private OnClickListener f;

a其实是一个LinearLayout, 然后OnClickListener中的this.a.a 实际上就是指向的this.a，这样就看的明白了。

然后就是最后一行调用了a()这个函数，这个是对界面进行了初始化，根据id fb131428158，对应16进制的0x7f0b033e，然后检索到Layout名



打开后，得到布局为

merge【双fill】 <? merge是个什么东东 ?>

magicsakura.TintView【height：0.5dip】

LinearLayout【双fill，水平，id=container】

this.a = (LinearLayout) findViewById(2131297096);

这里就不用多说了吧，a就是上面那个布局里面的linearlayout，id 都不用去校对了。

就这样结束了吗？好像只有个空壳，没去添加内容诶…?

别急嘛，再去看看还有没有别的初始化代码。。

在另一个方法里面找到了这个layout



<?xml version="1.0" encoding="utf-8"?>

<com.bilibili.magicasakura.widgets.TintFrameLayout android:layout\_width="fill\_parent" android:layout\_height="fill\_parent" android:foreground="?android:selectableItemBackground"

xmlns:android="http://schemas.android.com/apk/res/android">

<LinearLayout android:gravity="center" android:layout\_gravity="center" android:orientation="vertical" android:layout\_width="wrap\_content" android:layout\_height="wrap\_content">

<ImageView android:id="@id/tab\_icon" android:layout\_width="20.0dip" android:layout\_height="20.0dip" />

<com.bilibili.magicasakura.widgets.TintTextView android:textSize="@dimen/text\_size\_supplementary" android:textColor="@color/selector\_tab\_item" android:ellipsize="end" android:gravity="center" android:id="@id/tab\_text" android:layout\_width="wrap\_content" android:layout\_height="wrap\_content" android:layout\_marginTop="3.0dip" android:lines="1" />

</LinearLayout>

</com.bilibili.magicasakura.widgets.TintFrameLayout>

结构如下：

TintFrameLayout【双fill，forground=?android:selectableBackgound】

LinearLayout【vertical竖直，重力center，双wrap】

ImageView【固定值20dip x 20dip】

TintTextView【居中，单行，省略号居右，顶部3dpmargin】

#### onRestoreInstanceState方法

public void onRestoreInstanceState(Parcelable parcelable) {

     SavedState savedState = (SavedState) parcelable;

     this.d = savedState.a;

     super.onRestoreInstanceState(savedState.getSuperState());

     b(this.d);

}

这里主要是从 存储状态信息 中获取了当前选中的Tab的viewid，然后调用了b(int)，方法，b的参数就是这个view的id，

private void b(int i) {

    int itemCount = getItemCount();

    for (int i2 = 0; i2 < itemCount; i2++) {

        View childAt = this.a.getChildAt(i2);

        if (i == childAt.getId()) {

            this.d = i;

            this.e = i2;

            a(childAt, true);

        } else {

            a(childAt, false);

        }

    }

}

逻辑很清晰，在a这个Tabs容器中搜索这个id，搜到之后记录其在父layout中的下标到变量e中，另外依次调用了每个Tab的状态改变方法，也就是

a(int childindex,bool selected)这个方法。