科大讯飞 iFLYTEK

考试报告性能优化方案

安徽知学科技有限公司 知学宝 研发组 苏晟

邮箱：shengsu@iflytek.com

地址：天津空港经济区空港商务园东区E8楼B座

两个概念

- **响应时间:**指从用户操作开始到系统给用户以正确反馈的时间。一般包括逻辑处理时间 + 网络传输时间 + 展现时间（界面渲染时间）。
- **TPS(Transaction Per Second):**每秒处理的事务数，是系统吞吐量的指标，TPS一般与响应时间反相关。

- 响应时间是用户对性能最直接的感受。
- 通常所说的性能问题就是指响应时间过长、系统吞吐量过低。

性能调优方式

性能优化实际就是优化系统的响应时间、提高TPS

(1) 降低执行时间（逻辑优化、渲染优化）

(2) 同步改异步，利用多线程提高TPS

(3) 提前或延迟操作，错峰提高TPS

显示时才加载网络：

UserVisibleHint=true

降低执行时间

(1) 逻辑优化（Lrucache缓存、数据结构）

可进一步放宽缓存策略。

(2) 渲染优化

- 不应在Application以及Activity的生命周期回调中做任何费时操作，具体指标大概是你在onCreate，onStart，onResume等回调中所花费的总时间最好不要超过400ms；
- 布局Layout过于复杂，无法在16ms内完成渲染；（for loop 减少不必要的子控件或层级（RelativeLayout、），复用布局、复用ViewHolder）

放宽缓存策略

主动获取网络数据：

第一次请求数据时，保存一份到数据库，并记下时间戳，当下次重新初始化时，判断是否超过最大时间间隔(如5分钟)，如果没有，只加载数据库数据，不需要再做网络请求。

目前仅仅是用Lrucache缓存数据。

Hugo-How to use

注解触发的Debug日志库，设计规范的AOP框架，在Debug模式下，Hugo利用aspectj库来进行切面编程，插入日志代码。

1、直接使用@DebugLog注解给想要调试的方法

```
@DebugLog
public String getName(String first, String last) {
      SystemClock.sleep(15); // Don't ever really do this!
      return first + " " + last;
}
```

2、打印输出函数的参数，执行时间，以及返回值

```
V/Example: ⇢ getName(first="Jake", last="Wharton")
V/Example: ⇠ getName [16ms] = "Jake Wharton"
```

https://github.com/JakeWharton/hugo

http://blog.csdn.net/xxxzhi/article/details/53048476

Hugo-Getting started

1、build.gradle（Project）

```
buildscript {
  repositories {
    mavenCentral()
  }

  dependencies {
    classpath 'com.jakewharton.hugo:hugo-plugin:1.2.1'
  }
}
```

2、build.gradle（Module）

```
apply plugin: 'com.android.application'
apply plugin: 'com.jakewharton.hugo'

android {
        defaultPublishConfig "debug"
}
```

原方案渲染时间

ExamActivity:

- onCreateActivity [127ms]
  - initData [21ms]
    - initNavigateView [9ms]
    - initViewPager [2ms]

第一次ExamReportFragment（非VIP）:

- onCreate [14ms]
- onCreateView [166ms]
  - initView [157ms]
    - assembleView [156ms]
    - loadExampleData [7ms]

原方案渲染时间

第一次ExamReportFragment（VIP）：

- onCreate [6ms]
- onCreateView [741ms]
  - initView [710ms]
    - assembleView [707ms]
    - loadExampleData [28ms]
    - loadData [181ms]

非第一次ExamReportFragment（VIP）：

300ms左右

为什么使用RecyclerView

- ViewHolder
- 复用单位，避免了不必要的findViewById

- 在复写Adapter的时候，有两个方法：

- onCreateViewHolder
- onBindViewHolder

当RecyclerView需要一个新的类型的item的ViewHolder的时候调用这个方法。

为什么使用RecyclerView

- onCreateViewHolder()
- 这个方法主要生成为每个Item inflater出一个View，但是该方法返回的是一个ViewHolder。该方法把View直接封装在ViewHolder中，然后我们面向的是ViewHolder这个实例，当然这个ViewHolder需要我们自己去编写。

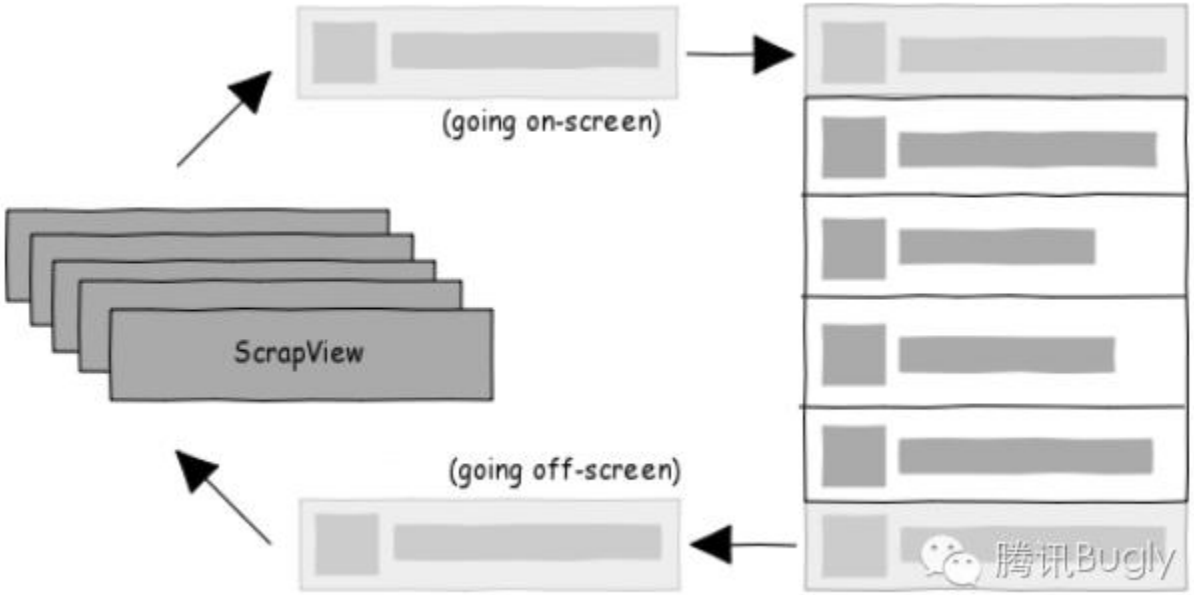
- onBindViewHolder()
- 这个方法主要用于适配渲染数据到View中。方法提供给你了一viewHolder而不是原来的convertView。

为什么使用RecyclerView

- 新的ViewHolder将会被用来通过adapter调用onBindViewHolder展示item。由于它将会被复用去展示在数据集中的不同items，所以缓存View的子view引用去避免不必要的对findViewById方法的调用。
- ViewHolder对象的数量“够用”之后就停止调用
- 调用这个createViewHolder方法的时候做了限制
- ViewHolder 屏幕Item+2

为什么使用RecyclerView

RecyclerView缓存机制原理



过程中，离屏的ItemView即被回收至缓存，入屏的ItemView则会优先从缓存中获取，只是ListView与RecyclerView的实现细节有差异。

缓存层级不同

ListView				
	是否需要回调createView	是否需要回调bindView	生命周期	备注
mActiveViews	否	否	onLayout函数周期内	用于屏幕内ItemView快速重用
mScrapViews	否	是	与mAdapter一致，当mAdapter被更换时，mScrapViews即被清空	

http://www.open-open.com/lib/view/open1477896837105.html

缓存层级不同

RecyclerView				
	是否需要回调createView	是否需要回调bindView	生命周期	备注
mAttachedScrap	否	否	onLayout函数周期内	用于屏幕内ItemView快速重用
mCacheViews	否	否	与mAdapter一致，当mAdapter被更换时，mCacheViews即被缓存至mRecyclerPool	默认上限为2，即缓存屏幕外2个ItemView
mViewCacheExtension				不直接使用，需要用户在定制，默认不实现
mRecyclerPool	否	是	与自身生命周期一致，不再被引用时即被释放	默认上限为5，技术上可以实现所有RecyclerViewPool共用同一个

http://www.open-open.com/lib/view/open1477896837105.html

缓存内容不同

1). RecyclerView缓存RecyclerView.ViewHolder，抽象可理解为：

View + ViewHolder(避免每次createView时调用findViewById) + flag(标识状态)；

2). ListView缓存View。

http://www.open-open.com/lib/view/open1477896837105.html

RecyclerView

- Support.v7包中的控件，是ListView和GridView的增强升级版。

（1）、RecyclerView.Adapter
（2）、LayoutManager
（3）、ItemAnimator

RecyclerView封装了viewholder的回收复用，也就是说RecyclerView标准化了ViewHolder，编写Adapter面向的是ViewHolder而不再是View了，复用的逻辑被封装了，写起来更加简单。

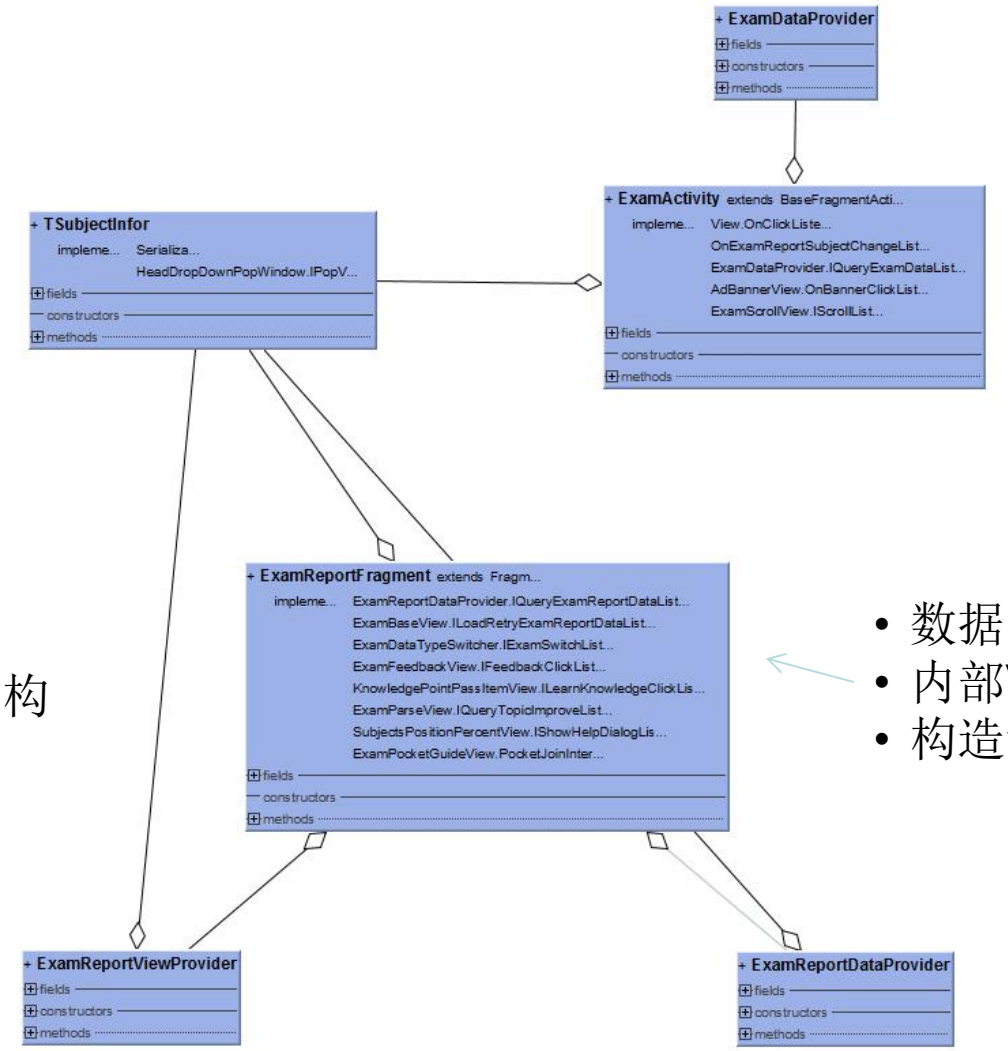
RecyclerView

- 它不关心如何将子View放在合适的位置，也不关心如何分割这些子View，更不关心每个子View各自的外观。更进一步来说就是RecyclerView它只负责回收和重用的工作，这也是它名字的由来。

- Adapter：包装数据集合并且为每个条目创建视图。

- ViewHolder：保存用于显示每个数据条目的子View。

目前項目大致结构



- 根据身份等数据构造List<View>

- 数据回调
- 内部View绑定监听
- 构造请求参数

类型池系统

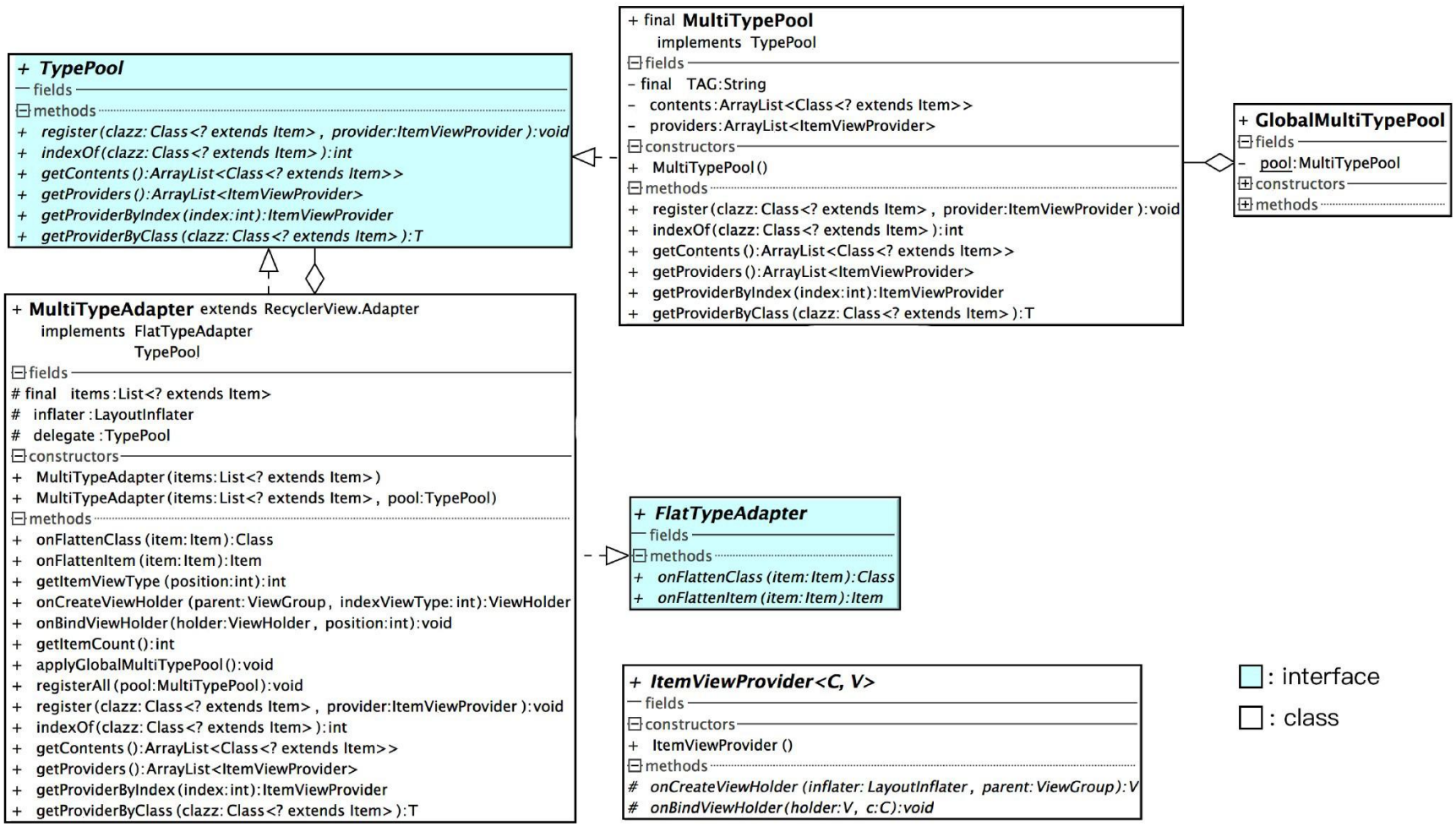
以前：

- 覆写 RecyclerView.Adapter 的 getItemViewType 方法，罗列一些 type 整型常量，并且 ViewHolder 转型、绑定数据也比较麻烦。

现在：

- 不同的 item 类型众多，而且随着业务发展，还会更多。
- 多类型列表视图的中间分发
- 类型 - View 的复用池系统

工程结构


+ TypePool
fields
methods
+ register(clazz: Class<? extends Item>, provider:ItemViewProvider):void
+ indexOf(clazz: Class<? extends Item>):int
+ getContents():ArrayList<Class<? extends Item>>
+ getProviders():ArrayList<ItemViewProvider>
+ getProviderByIndex(index:int):ItemViewProvider
+ getProviderByClass(clazz: Class<? extends Item>):T
+ final MultiTypePool
implements TypePool
fields
- final TAG:String
- contents:ArrayList<Class<? extends Item>>
- providers:ArrayList<ItemViewProvider>
constructors
+ MultiTypePool()
methods
+ register(clazz: Class<? extends Item>, provider:ItemViewProvider):void
+ indexOf(clazz: Class<? extends Item>):int
+ getContents():ArrayList<Class<? extends Item>>
+ getProviders():ArrayList<ItemViewProvider>
+ getProviderByIndex(index:int):ItemViewProvider
+ getProviderByClass(clazz: Class<? extends Item>):T
+ GlobalMultiTypePool
fields
- pool:MultiTypePool
constructors
methods
+ MultiTypeAdapter extends RecyclerView.Adapter
implements FlatTypeAdapter
TypePool
fields
final items:List<? extends Item>
inflater:LayoutInflater
delegate:TypePool
constructors
+ MultiTypeAdapter(items: List<? extends Item>)
+ MultiTypeAdapter(items: List<? extends Item>, pool:TypePool)
methods
+ onFlattenClass(item: Item):Class
+ onFlattenItem(item: Item):Item
+ getItemViewType(position:int):int
+ onCreateViewHolder(parent: ViewGroup, indexViewType: int):ViewHolder
+ onBindViewHolder(holder:ViewHolder, position:int):void
+ getItemCount():int
+ applyGlobalMultiTypePool():void
+ registerAll(pool:MultiTypePool):void
+ register(clazz: Class<? extends Item>, provider:ItemViewProvider):void
+ indexOf(clazz: Class<? extends Item>):int
+ getContents():ArrayList<Class<? extends Item>>
+ getProviders():ArrayList<ItemViewProvider>
+ getProviderByIndex(index:int):ItemViewProvider
+ getProviderByClass(clazz: Class<? extends Item>):T
+ FlatTypeAdapter
fields
methods
+ onFlattenClass(item: Item):Class
+ onFlattenItem(item: Item):Item
+ ItemViewProvider<C, V>
fields
constructors
+ ItemViewProvider()
methods
onCreateViewHolder(inflater: LayoutInflater, parent: ViewGroup):V
onBindViewHolder(holder:V, c:C):void
: interface
: class

100次3个Item放大渲染测试

优化前

ExamActivity:

- onCreateActivity [365ms]

ExamReportFragment:

- onCreate [55ms]
- onCreateView [8523ms]
- onCreateView [5021ms]
- onCreateView [5317ms]
- onCreateView [5159ms]

优化后

ExamActivity:

- [19ms]

ExamReportFragment:

- [15ms]
- [261ms]
- [211ms]
- [239ms]
- [214ms]

ExamActivity内存占用对比

优化前：

61.95　73.32　88.92　86.75 ⇠ 89.33

优化后：

58.21　58.87　55.48　57.40 ⇠ 67.44

单位MB，前四个数值为第1~4页，第五个数值为返回第三页。

Issues

1、一类型对多 ViewProvider的注册问题：很多Provider是对应一个相同的数据类型，这样当给出这个数据类型时候，程序无法分辨是生成那个View;

2、基类封装问题：因为Vip布局的View有很多相同的控件布局以及公用的方法，原先是继承BaseView，现在我就写了个BaseViewHolder，又因为一些方法是public的，在外部到子类中调用的，每个子类都不一样，又抽象了这些方法和类；问题是原逻辑是遍历所以View然后判断是哪个子类，去做操作；现在我就要把这个BaseViewHolder抽象类放到一个抽象的父类BaseViewProvider里，但这样外部无法传一个泛型到没有实例的一个父类里。

Issues

```
public class OptionalHolder<Bean> {
    private View contentView;
    // 构造方法
    public OptionalHolder() {
        contentView = initView();
    }
    public View initView(){
        View view = UIUtils.inflate(R.layout.layout_optional);
        return view;

        // 在此 find 控件
    }
    // 设置数据
    public abstract void setData(Bean data){
        // 在此设置数据
    }
    // 获取根布局
    public View getContentView() {
        return contentView;
    }
}
```

```
Container.addView(holder.get
ContentView());
```

Issues

3、另外一些监听，需要在外层ExamReportFragment去给需要的View去setListener的，或者是调用Holder中的方法的，但现在因为ViewHolder不知道什么时候有实例，经常通过Type获取时候holder是null。

4、局部刷新问题，局部刷新需要postion，因为data是逐一set的，所以每次全部刷新很浪费资源。

一类型对多 ViewProvider

- 在不同的列表中， adapter不同，注册到的局部类型池也不同，所以在这里一对多无问题；
- 在同个列表里，解决方案：

1、空继承你的类型，然后把它视为新的类型，注册到类型池中。（这样多了很多看上去很奇怪的Java bean/model类）；

2、在实体类中创建多个对应的公有静态抽象类，然后复写adpter去根据数据去setTypeClass；

但无论怎样数据必须setType();而且对应List暂时无解。

仍可做的优化

- 仅仅将View->ViewHolder实现，在单一列表中复用ViewHolder（非VIP统一可做，优化模块结构）；

- 一次性加载3、4个报告页面，减少切换时卡顿、切后时重新布局和填充数据，减少内存抖动。

设置一次性加载全部

mViewPager.setOffscreenPageLimit(0);//无效，默认1，也就是左右都缓存1页

FreeOffscreenPageLimitViewPager

114MB　　109MB （放大优化前后内存）

正常VIP 4页：83.16MB

正常VIP 1页：59.20MB

一次加载4页可行

正常VIP（4页第一次）：

ExamActivity: ⇠ onCreateActivity [203ms]

xamReportFragment: ⇠ onCreateView [742ms]
xamReportFragment: ⇠ onCreateView [562ms]
xamReportFragment: ⇠ onCreateView [151ms]
xamReportFragment: ⇠ onCreateView [138ms]

非第一次：

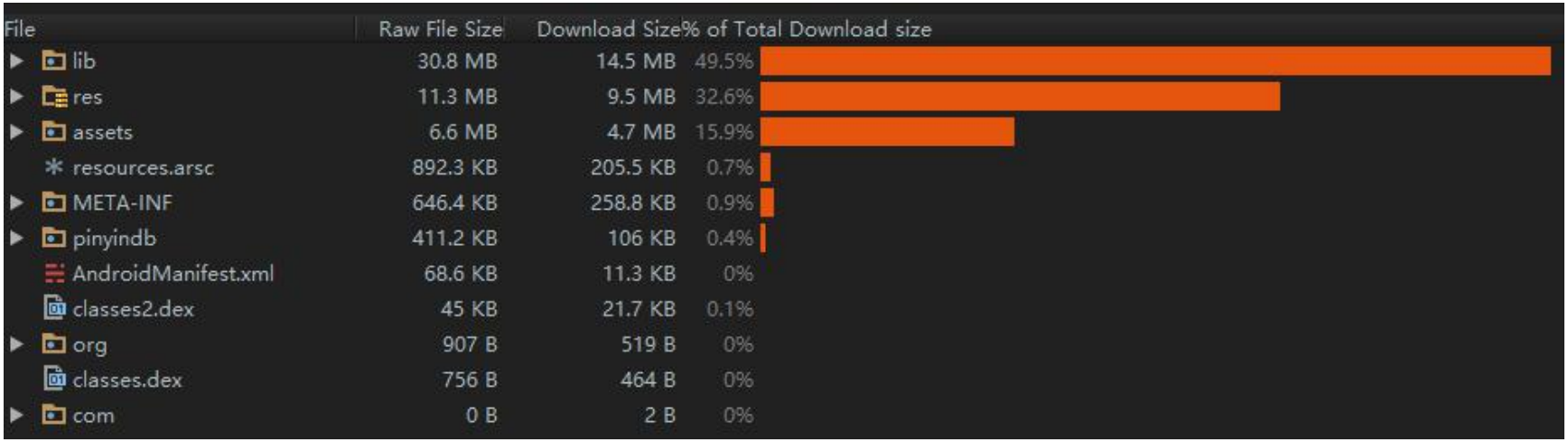
ExamActivity: ⇠ onCreateActivity [39ms]

xamReportFragment: ⇠ onCreateView [63ms]
xamReportFragment: ⇠ onCreateView [124ms]
xamReportFragment: ⇠ onCreateView [156ms]
xamReportFragment: ⇠ onCreateView [128ms]

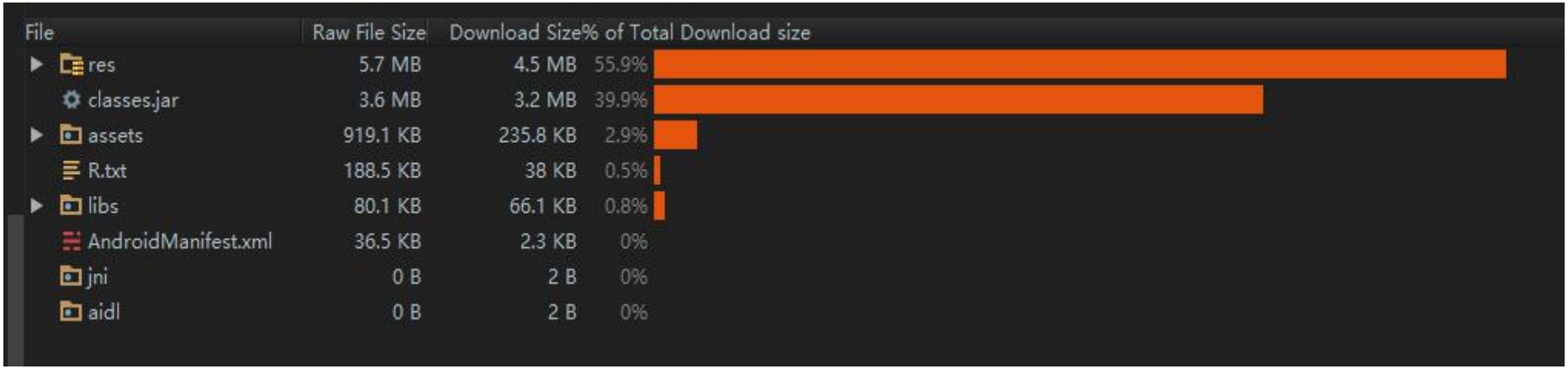
后续调研

- 调研AOP框架设计原理、参照设计规范的AOP框架，修改Hugo达到完成目前日志埋点的需求；
- 优化Vip介绍页逻辑、界面模块化、统一获取身份数据后再分发各模块处理；
- 尝试优化项目资源，瘦身APK。（debug-）；

后续调研



File	Raw File Size	Download Size	% of Total Download size
lib	30.8 MB	14.5 MB	49.5%
res	11.3 MB	9.5 MB	32.6%
assets	6.6 MB	4.7 MB	15.9%
resources.arsc	892.3 KB	205.5 KB	0.7%
META-INF	646.4 KB	258.8 KB	0.9%
pinyindb	411.2 KB	106 KB	0.4%
AndroidManifest.xml	68.6 KB	11.3 KB	0%
classes2.dex	45 KB	21.7 KB	0.1%
org	907 B	519 B	0%
classes.dex	756 B	464 B	0%
com	0 B	2 B	0%



File	Raw File Size	Download Size	% of Total Download size
res	5.7 MB	4.5 MB	55.9%
classes.jar	3.6 MB	3.2 MB	39.9%
assets	919.1 KB	235.8 KB	2.9%
R.txt	188.5 KB	38 KB	0.5%
libs	80.1 KB	66.1 KB	0.8%
AndroidManifest.xml	36.5 KB	2.3 KB	0%
jni	0 B	2 B	0%
aidl	0 B	2 B	0%

科大讯飞 iFLYTEK

谢谢，

简单尝试，有误跪求指正。