Android多媒体基础

万里鹏 wanlipeng.rd@bytedance.com

字节跳动Android工程师

ByteDance字节跳动

目录

- 图片
- 音视频

图片

图片基础

图片格式

JPEG全称Joint Photographic Experts Group

- 手机或者相机拍照的 产物
- 压缩比相对较高，文件大小相 对较小
- 如果图像需要全彩模式效果，最好使用 JPEG
- 不支持透明图和动态图

图片格式

GIF全称Graphic Interchange Format

- 最多可使用256种颜色
- 支持透明度
- 常用做动画使用
- 文件大小一般与 动画帧数有关系
- Andriod上控件不能直接播放，第三方 组件android-gif-drawable

图片格式

PNG格式，全称Portable Network Graphics

- 高压缩比的无损压缩，文件大小比jpg高一些
- 最多支持48位的色彩
- 支持α通道数据（透明度）
- 是Android开发上常用的图片格式，比如图标

图片格式

BMP全称Bitmap

- 由微软发明，最初在Windows上使用
- 主要应用位映射存储格式
- 不使用任何压缩算法，占用空间很大
- 颜色比较保真

图片格式

WebP是谷歌提供的一种支持有损压缩和无损压缩的图片文件格式

- 比JPEG或PNG更好的压缩
- 在Android 4.0(API level 14)中支持有损的WebP图像
- 在Android 4.3(API level 18)和更高版本中支持无 损和透明的WebP图像

下图是几种图片，结构相似性(SSIM)情况下，WebP与JPEG格式占用大小的对比

	Lenna	Kodak	Tecnik	Image_crawl
WebP: Average File Size (Average SSIM)	17.4 KB (0.841)	31.0 KB (0.898)	92.4 KB (0.917)	6.5 KB (0.901)
JPEG: Average File Size (Average SSIM)	23.5 KB (0.840)	42.7 KB (0.897)	124.6 KB (0.916)	9.9 KB (0.899)
Ratio of WebP to JPEG file size	0.74	0.72	0.74	0.66

Nine Patch

图中的文字的按钮，

如何做到背景图根据文字做自适应？

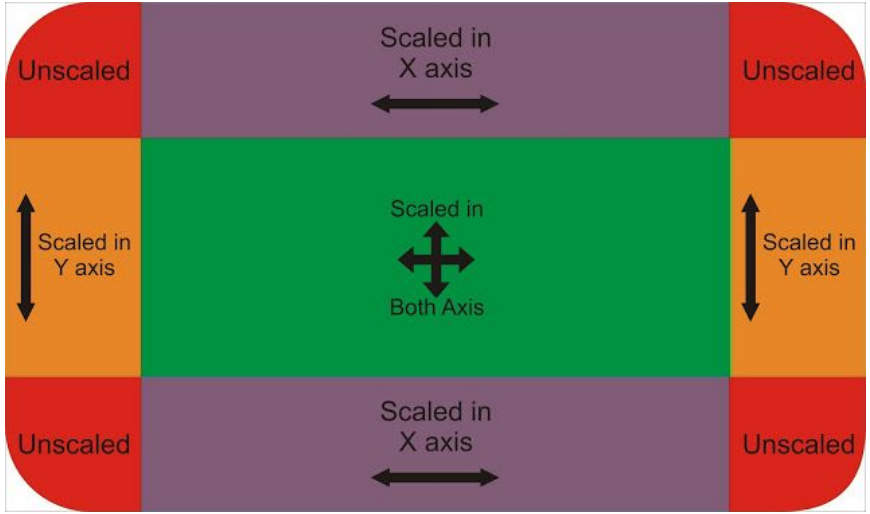
Nine Patch

普通图片会被拉伸变形,如右图

普通图片会被拉伸变形,如右图

Nine Patch 图片会分为9块区域, 按右图拉伸

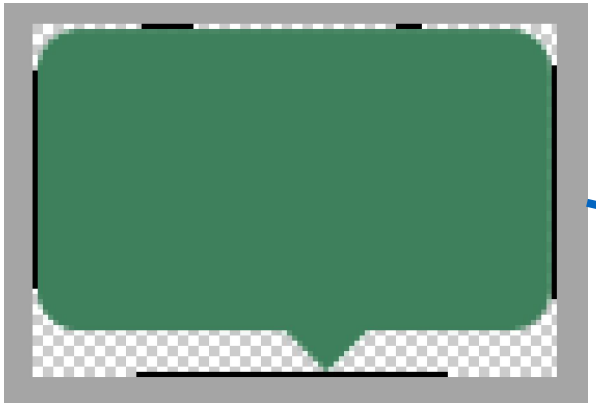
它后缀格式*.9.png, Android系统特有的图片格式



Nine Patch效果

制作气泡图

- 设置内容区(下方)
- 设置拉伸区(上方)



基础示例

多行文本示例
多行文本示例
多行文本示例

代码设置Padding

椭圆形状

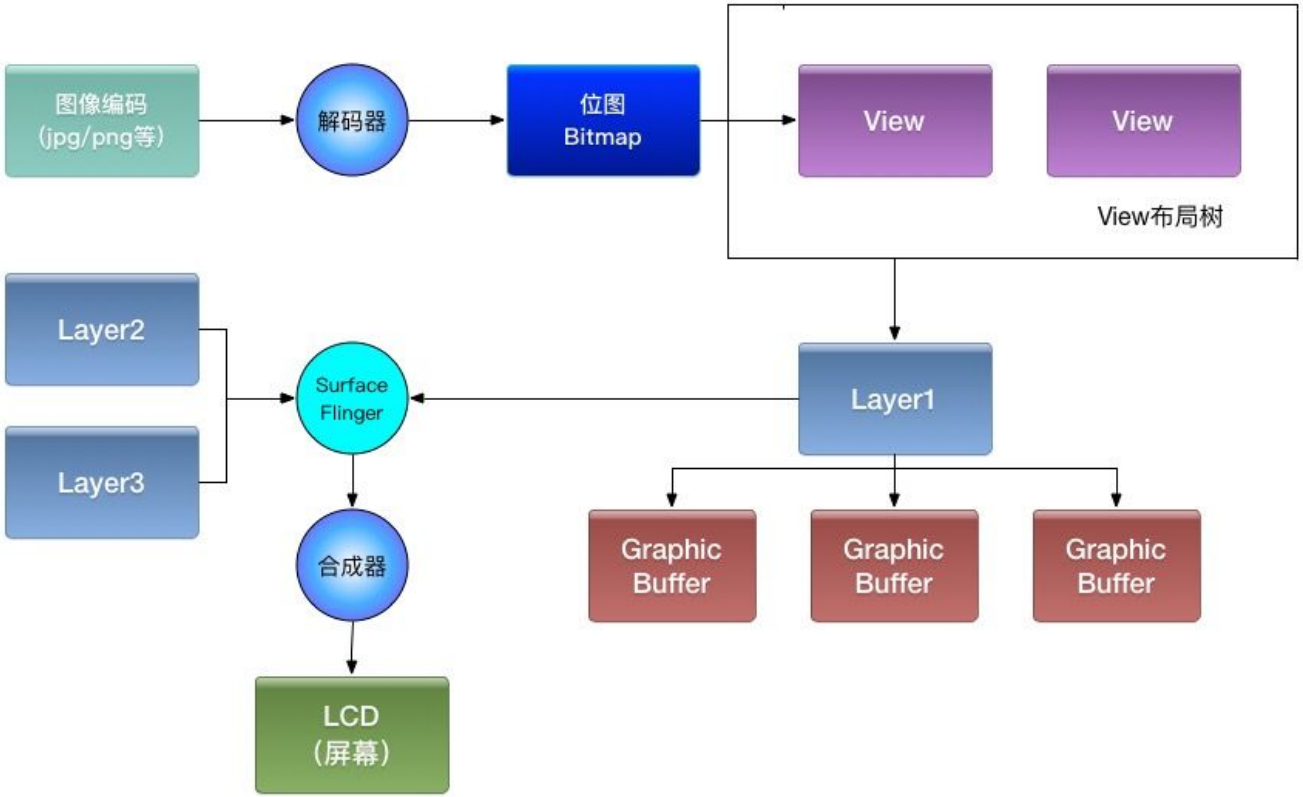
纵向不拉伸时是半圆，代码设置Gravity居中

文字过高时
会变成圆角

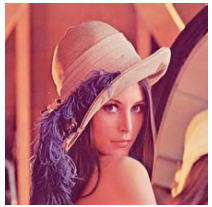
多个缩放区域会按比例均匀缩放
这里箭头左右黑色像素点数是2:1

没有定义
内容区域

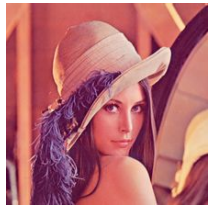
Android 图片加载流程



Android Bitmap



WebP
磁盘占用17.4K



Bitmap
内存占用？

1. 缺少了WebP的像素
2. 缺少了加载时指定的像素类型

Android Bitmap

常见像素类型(R=Red, G=Green, B=Blue, A=Alpha)

- ALPHA_8：A = 8位, 1字节
- RGB_565：R = 5位 G = 6位 B = 5位, 2字节
- ARGB_4444：A = 4位 R = 4位 G = 4位 B = 4位, 2字节
- ARGB_8888：A = 8位 R = 8位 G = 8位 B = 8位, 4字节

Android Bitmap

1. WebP的像素 ----- 512*512
2. 加载时指定的像素类型------ ARGB_8888

内存占用:

512 * 512 * 4 = 1048576B = 1MB

具体代码实现:

```
BitmapFactory.Options options = new BitmapFactory.Options();
options.inPreferredConfig = Bitmap.Config.ARGB_8888;
Bitmap bitmap = BitmapFactory.decodeResource(getResources(),
R.drawable.lenna, options);
```

加载图片

BitmapFactory提供了四类方法：

- decodeFile(String pathName, Options opts)
- decodeResource(Resources res, int id, Options opts)
- decodeStream(InputStream is, Rect outPadding, Options opts)
- decodeByteArray(byte[] data, int offset, int length)

可以从磁盘文件，资源，流以及byte数组中加载图片资源

例如从资源中加载图片

Bitmap bitmap = BitmapFactory.decodeResource(getContext().getResources(), R.drawable.sample);

加载超大图片

手机摄像头越来越好，拍出照片也越来越大
比如一张小米手机照片：

分　辨　率：3120 * 4160
文件大小：4.06MB

这个照片直接加载到内存
占用内存空间：**3120 * 4160 * 4 = 49.51MB**

问题
在内存较小的手机上，加载几张图就会出现OOM（Out of Memory）异常

加载超大图片

加载策略:

1. 图片大小
2. 图片质量

加载超大图片

预判图片大小，非真实加载图片到内存

```
BitmapFactory.Options options = new BitmapFactory.Options();
//inJustDecodeBounds为true，不返回bitmap，只返回这个bitmap的尺寸
options.inJustDecodeBounds = true;
BitmapFactory.decodeResource(getResources(), images[position], options);
```

设定缩放尺寸，像素配置

```
//利用返回的原图片的宽高，我们就可以计算出缩放比inSampleSize（只能是2的整数次幂）
options.inSampleSize = caluelateInSampleSize(options, reqWidth, reqHeight);
options.inPreferredConfig = Bitmap.Config.RGB_565;  //使用RGB_565减少图片大小

//inJustDecodeBounds为false，返回bitmap
options.inJustDecodeBounds = false;
Bitmap bitmap = BitmapFactory.decodeResource(getResources(), images[position], options);
```

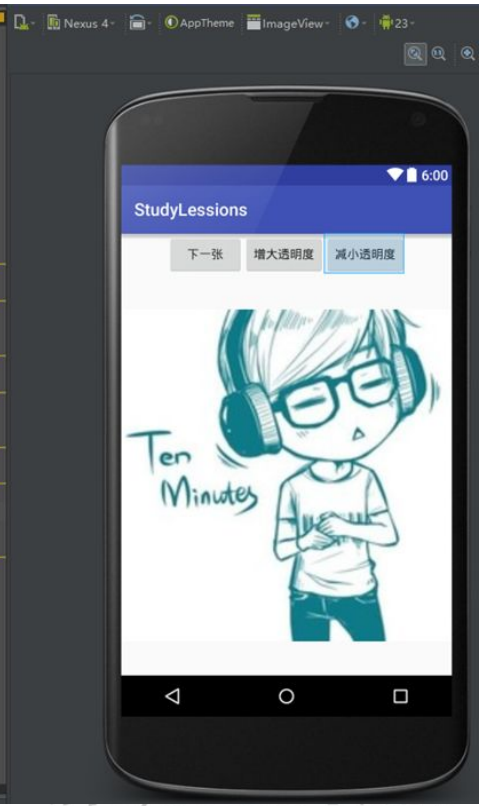
ImageView

ImageView是Android显示图片的组件

主要方法：

方法名称	说明
setImageBitmap(Bitmap bm)	使用Bitmap对象设置ImageView显示的图片
setImageDrawable(Drawable drawable)	使用Drawable对象设置ImageView显示的图片
setImageResource(int resId)	使用资源ID设置ImageView显示的图片
setImageURI(Uri uri)	使用图片的URI地址设置ImageView显示的图片

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.boyliupan.studylessions.ImageViewActivity">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:gravity="center"
        >
        <Button
        android:id="@+id/btn1"
        android:text="下一张"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
        <Button
            android:id="@+id/btn2"
            android:text="增大透明度"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <Button
            android:id="@+id/btn3"
            android:text="减小透明度"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </LinearLayout>
    <ImageView
        android:id="@+id/img1"
        android:scaleType="fitCenter"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:src="@drawable/image1"/>
</LinearLayout>
```



ImageView的主要属性

ImageView.ScaleType的主要值 设置所显示的图片如何缩放或移动以适应ImageView的大小

值	说明
ImageView.Scale.MATRIX	使用Matrix方式进行缩放
ImageView.Scale.FIT_XY	对图片横向、纵向独立缩放，使得该图完全适应于ImageView，图片的纵横比可能会改变
ImageView.Scale.FIT_START	保持纵横比缩放图片，直到该图能完全显示到ImageView，从ImageView的左上角开始显示
ImageView.Scale.FIT_CENTER	保持纵横比缩放图片，直到该图能完全显示到ImageView，从ImageView的中间显示
ImageView.Scale.FIT_END	保持纵横比缩放图片，直到该图能完全显示到ImageView，从ImageView的右下角显示
ImageView.Scale.CENTER	把图片放到ImageView中间，不进行任何缩放
ImageView.Scale.CENTER_CROP	保持纵横比缩放图片使得图片能完全覆盖ImageView，最要图片最短边能显示即可。
ImageView.Scale.CENTER_INSIDE	保持纵横比缩放图片，以使得ImageView能完全显示该图片

ImageView的主要属性

采用这两张图片做范例，左侧图片比ImageView大，右侧比ImageView小

ImageView - CENTER

按图片的原来size居中显示，当图片长/宽超过View的长/宽，则截取图片的居中部分显示

ImageView - CENTER_CROP

按比例扩大图片的size居中显示，使得图片长(宽)等于或大于View的长(宽)

ImageView - CENTER_INSIDE

将图片的内容完整居中显示，通过按比例缩小或原来的size使得图片长/宽等于或小于View的长/宽

ImageView - FIT_CENTER

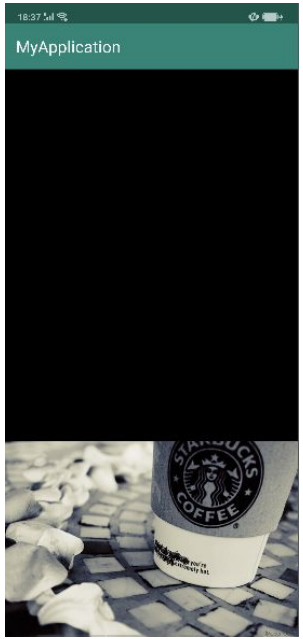
把图片按比例扩大/缩小到View的宽度，居中显示

ImageView - FIT_START

把图片按比例扩大/缩小到View的宽度，置顶显示

ImageView - FIT_END

把图片按比例扩大/缩小到View的宽度，置底显示

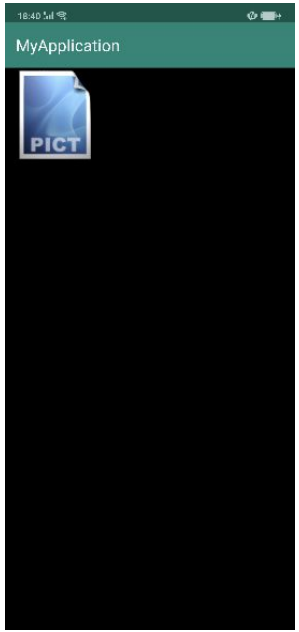


ImageView - FIT_XY

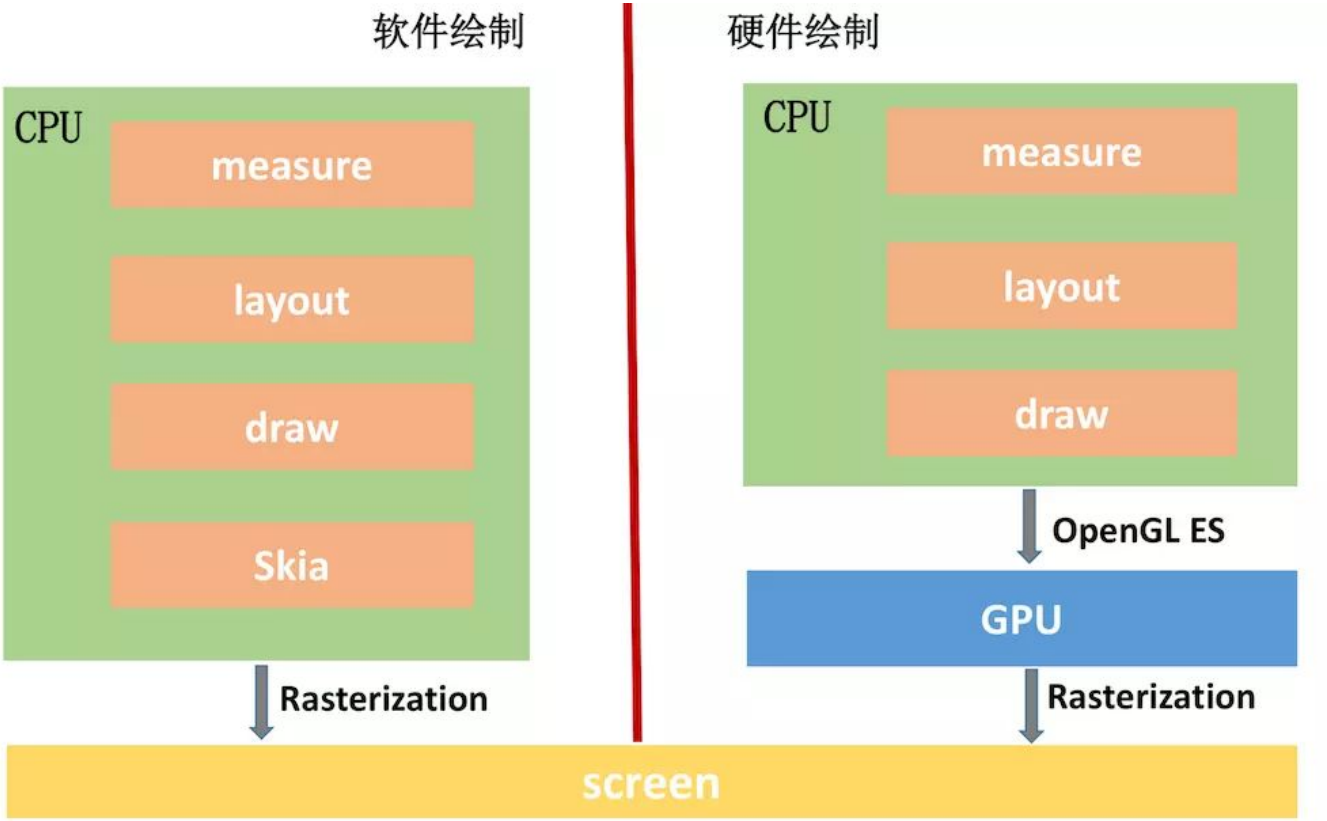
不按比例缩放图片，目标是把图片塞满整个View

ImageView - FIT_MATRIX

用矩阵来绘制(从左上角起始的矩 阵区域)



渲染



图片库

图片库介绍

常用图片库

- Picasso
- Glide
- Fresco

Picasso 轻量级

Fresco 旗舰型

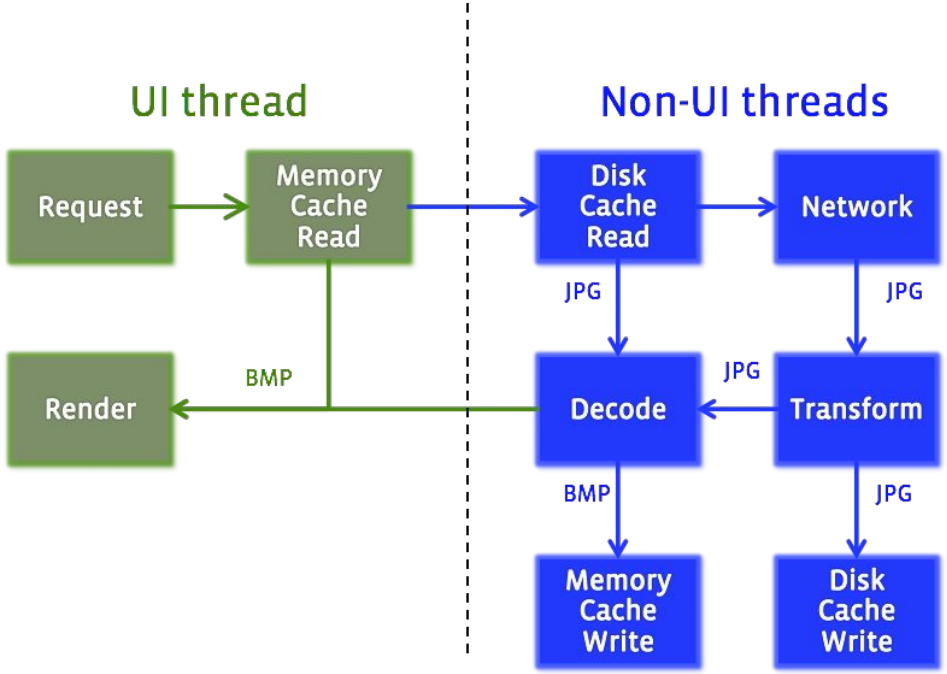
对比项	Picasso	Glide	Fresco
发布时间	2013年5月	2014年9月	2015年5月
是否支持GIF	❌	✔	✔
是否支持webp	✔	✔	✔
视频缩略图	❌	✔	✔
大小	100k	500k	2~3M
加载速度	中	高	高
图片缓存	✔	✔	✔
易用性	高	中	低
开发者	Square	Google	Facebook

图片缓存策略

为了提升用户体验，图片展示的目标

- 每次以最快的速度打开 图片
- 都缓存在内存，容易出 现OOM
- 根据内存，IO和网络的速度不同，设计三级缓存通用方案
- 包括内存缓存、本地缓存、网络缓存

很多公众的图片组件也都是采用了同样的思路，比如Picasso，Universal Image Loader，Glide，Fresco和Volley等等



Glide

Glide是一个快速高效的Android图片加载库，注重于平滑的滚动
Glide提供了易用的API，高性能、可扩展的图片解码管道（decode pipeline），以及自动的资源池技术

gradle接入

```
implementation 'com.github.bumptech.glide:glide:4.9.0'
```

代码

```
Glide.with(context)
        .load(uri2)
        .into(imageView);
```

接入代码非常简单好用，一行代码搞定，支持多种类型

Glide占位图

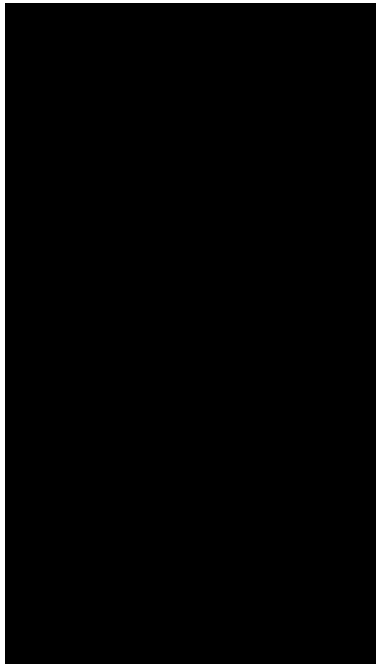
占位图就是指图片没展示出来之前，或者网 络超时、资源不存在等加载失败之后显示的图片。Glide内部直接提供了这样的接口：

```
ImageView imageView = (ImageView) findViewById(R.id.imageView);
Glide.with(context)
      .load(url)
      .placeholder(R.drawable.place_image)//图片加载出来前，显示的图片
      .error(R.drawable.error_image)//图片加载失败后，显示的图片
      .into(imageView);
```

针对一些没有返回，比如 头像，不存在时需要显示默认

```
String url = null;
GlideApp.with(context)
      .load(url)
      .fallback(R.drawable.fallback)
      .into(view);
```

传递url为null，使用fallback接口设置默认显示图片



Glide Generated API

Glide v4 使用注解的方式生成API类，该集成库可以为 Generated API 扩展自定义选项

```
dependencies {
    annotationProcessor 'com.github.bumptech.glide:compiler:4.8.0'
}
```

```
@GlideModule
public final class MyAppGlideModule extends AppGlideModule {
}
```

```
@GlideModule
public class YourAppGlideModule extends AppGlideModule {
  @Override
  public void applyOptions(Context context, GlideBuilder builder) {
    MemorySizeCalculator calculator = new MemorySizeCalculator.Builder(context)
        .setMemoryCacheScreens(2)
        .build();
    builder.setMemoryCache(new LruResourceCache(calculator.getMemoryCacheSize()));
  }
}
```

Glide Generated API

Glide v4 使用注解的方式生成API类，该集成库可以为 Generated API 扩展自定义选项

Gradle引入

```
dependencies {
  annotationProcessor 'com.github.bumptech.glide:compiler:4.9.0'
}
```

创建API类，增加注解标记

```
@GlideModule
public final class MyAppGlideModule extends AppGlideModule {}
```

编译之后生成**GlideApp**类，基本用法与**Glide**类一样

```
GlideApp.with(context)
    .load(uri1)
    .placeholder(R.drawable.place_holder)
    .error(R.drawable.icon_failure)
    .into(imageView);
```

Glide 请求选项

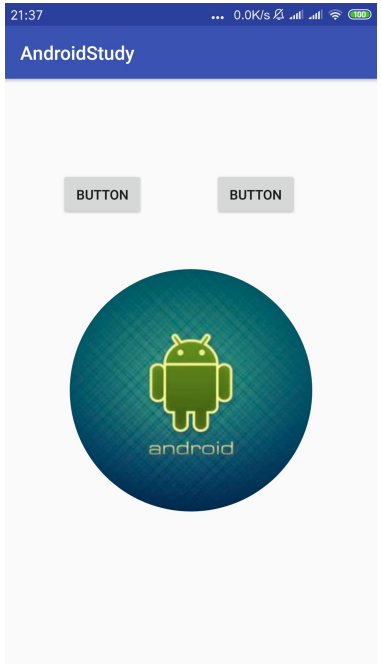
Glide中的很多设置项都可以通过 RequestOptions 类和 apply() 方法来应用到程序中

使用 Request Options 可以实现(包括但不限于):

- 占位符(Placeholders)
- 转换(Transformations)
- 缓存策略(Caching Strategies)
- 组件特有的设置项，例如编码质量，或Bitmap的解码配置等。

例如，圆形图形，效果如右图所示

```
RequestOptions cropOptions = new RequestOptions();
cropOptions.centerCrop().circleCrop();
GlideApp.with(ImageActivity.this)
    .load(uri3)
    .apply(cropOptions)
    .into(imageView);
```



Glide RequestBuilder

RequestBuilder 是Glide中请求的骨架，负责携带请求的url和你的设置项来开始一个新的加载过程。

使用 RequestBuilder 可以指定：

- 你想加载的资源类型(Bitmap, Drawable, 或其他)
- 你要加载的资源地址(url/model)
- 你想最终加载到的View
- 任何你想应用的（一个或多个）RequestOption 对象
- 任何你想应用的（一个或多个）TransitionOption 对象
- 任何你想加载的缩略图 thumbnail()

要构造一个 RequestBuilder 对象，你可以通过先调用 Glide.with()然后再调用某一个 as 方法来完成：

```
RequestBuilder<Drawable> requestBuilder = Glide.with(fragment).asDrawable();
```

或先调用 Glide.with() 然后 load()：

```
RequestBuilder<Drawable> requestBuilder = Glide.with(fragment).load(url);
```

Glide RequestBuilder的联动

RequestBuilder执行完根据情况不同，可能需要 继续请求RequestBuilder，或者同步 请求缩略图，正常图片加载比较慢的时候，缩略图速度较快可以并行加载

```
Glide.with(context)
    .load(localUri)
    .thumbnail(Glide.with(fragment)
        .load(thumbnailUri)
        .override(thumbnailSize))
    .into(view);
```

失败之后，继续发起新的请求

```
Glide.with(context)
    .load(primaryUrl)
    .error(Glide.with(fragment)
        .load(fallbackUrl))
    .into(imageView);
```

Glide缓存的支持

默认情况下，Glide 会在开始一个新的图片请求之前检查以下多级的缓存：

- 活动资源 (Active Resources) - 现在是否有另一个View 正在展示这张图片
- 内存缓存 (Memory cache) - 该图片是否最近被加载过并仍存在于内存中
- 资源类型 (Resource) - 该图片是否之前曾被解码、转换并写入过磁盘缓存
- 数据来源 (Data) - 构建这个图片的资源是否之前曾被写入过文件缓存

一些缓存的接口：

- diskCacheStrategy(DiskCacheStrategy.ALL) 磁盘缓存策略，
  - 默认为AUTOMATIC，远程资源缓存原始图片，本地资源缓存缩略图
  - ALL，缓存原始图和处理之后的资源
  - SOURCE，缓存原始图
  - NONE，关闭磁盘缓存
- onlyRetrieveFromCache(true) 仅从缓存加载图片
- skipMemoryCache(true) 跳过缓存，直接从uri获取

清除缓存

- Glide.get(context).clearMemory() 须在UI线程中调用
- Glide.get(context).clearDiskCache() 须在子线程中调用

Glide自定义网络

默认情况下，Glide使用的是基于HttpUrlConnection的网络实现，但同时也提供了与Google Volley和Square OkHttp快速集成的工具 库

OkHttp 集成库的依赖:

```
compile "com.github.bumptech.glide:okhttp3-integration:4.8.0"
```

Volley 集成库的依赖：

```
compile "com.github.bumptech.glide:volley-integration:4.8.0"
```

集成库的 Gradle 依赖将使 Glide 自动开始使用 自定义网络库来加载所有来自 http 和 https URL 的图片

权限申请

app > app.iml

AndroidManifest.xml

```
5       <uses-permission android:name="android.permission.INTERNET" />
6       <uses-permission android:name="android.permission.READ_PHONE_STATE" />
7       <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
8       <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
9       <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
10      <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
11      <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
12      <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
13      <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
14      <uses-permission android:name="android.permission.WAKE_LOCK" />
15      <uses-permission android:name="android.permission.CHANGE_CONFIGURATION" />
16      <uses-permission android:name="android.permission.REAL_GET_TASKS" />
17      <uses-permission android:name="android.permission.VIBRATE" />
18      <uses-permission android:name="android.permission.WRITE_CONTACTS" />
19      <uses-permission android:name="android.permission.READ_CONTACTS" />
20      <uses-permission android:name="android.permission.GET_ACCOUNTS" />
21      <uses-permission android:name="android.permission.RECORD_AUDIO" />
22      <uses-permission android:name="android.permission.CAMERA" />
23      <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
24      <uses-permission android:name="android.permission.CALL_PHONE" />
25      <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
26      <uses-permission android:name="android.permission.DEVICE_POWER" />
27      <uses-permission android:name="android.permission.WRITE_SETTINGS" />
28      <uses-permission android:name="android.permission.BLUETOOTH" />
29      <uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES" />
30      <uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS" />
31      <uses-permission android:name="android.permission.RECORD_AUDIO" />
32
```

动态权限

6.0之后的版本，除了AndroidManifest里面声明权限，还要动态申请

```
int[] mPermissionsArrays = new int[]{Manifest.permission.CAMERA, Manifest.permission.WRITE_EXTERNAL_STORAGE}
if (checkPermissionAllGranted(mPermissionsArrays)) {
    openSystemCamera(); // 有权限直接打开系统相机
} else {
    requestPermissions(mPermissionsArrays, REQUEST_PERMISSION);
}
// 检查授权
private Boolean checkPermissionAllGranted(String[] permissions) {
    for (permission in permissions) {
        if (checkSelfPermission(permission) != PackageManager.PERMISSION_GRANTED) {
            return false;
        }
    }
    return true;
}
// 授权结果
private void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
    if (requestCode == REQUEST_PERMISSION) {
        openSystemCamera();
    }
}
```

1

2

3

拍摄图片

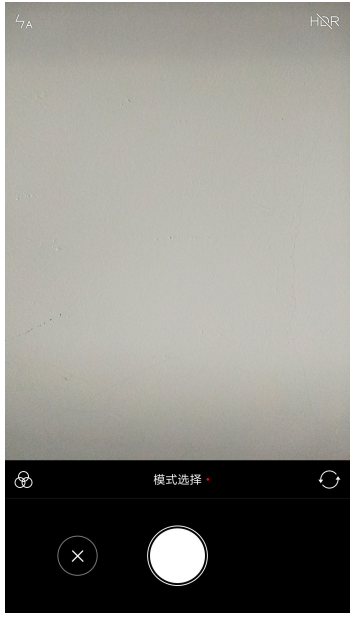
拍摄照片直接调用系统相机即可，Android内部有很多跨进程的调用，相机就是用了一样的方式

拍照之后需要知道结果，所以使用startActivityForResult方法

```
private void openSystemCamera() {
    Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    File uri = new File(Environment.getExternalStorageDirectory(), imgName);
    Uri outUri = FileProvider.getUriForFile(this, packageName + ".fileprovider", uri);
    cameraIntent.putExtra(MediaStore.EXTRA_OUTPUT, outUri);
    startActivityForResult(cameraIntent, REQ_CODE);
}
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == REQ_CODE) {
        File tempFile = new File(Environment.getExternalStorageDirectory(), imgName);
        System.out.println(tempFile.getPath());
    }
}
```

1

2



注意点：

- 6.0及以上版本，需要动态申请相机和存储的权限
- 7.0及以上版本不能直接访问SD卡路径

FileProvider

7.0之后的版本，Android框架StrictMode禁止我们的应用对外部(跨越应用分享)公开file://（编译版本低于25不会受到影响），拍照需要跨进程调用相机，属于被禁止范畴，会报FileUriExposedException异常，官方建议是使用FileProvider类

AndroidManifest.xml文件声明

```
<provider
  android:name="android.support.v4.content.FileProvider"
  android:authorities="com.domker.study.androidstudy.fileprovider"
  android:exported="false"
  android:grantUriPermissions="true">
  <meta-data
    android:name="android.support.FILE_PROVIDER_PATHS"
    android:resource="@xml/provider_paths" />
</provider>
```

XML配置

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
  <external-path name="external_files" path="/"/>
</paths>
```

生成Uri

```
Uri outUri = FileProvider.getUriForFile(context, packageName + ".fileprovider", uri)
```

图片选择

图片选择是程序中经常使用的，主要包括系统内置和自定义实现

系统内置图片选择，使用跨进程调用的方式

```
private void pickImage() {
    Intent i = new Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
    startActivityForResult(i, RESULT_LOAD_IMAGE);
}
```

在onActivityResult函数中返回选择的结果

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == RESULT_LOAD_IMAGE) {
        // 处理选择图片
    }
}
```

音视频播放

视频格式

PC和移动设备上都离不开视频播放，涉及的视频格式也比较多，不同扩展名实际上使用了不同的视频和音频编码

名称	推出机构	流媒体	支持的视频编码	支持的音频编码	目前使用领域
AVI	Microsoft Inc.	不支持	几乎所有格式	几乎所有格式	BT下载影视
MP4	MPEG	支持	MPEG-2, MPEG-4, H.264, H.263等	AAC, MPEG-1 Layers I, II, III, AC-3等	互联网视频网站
TS	MPEG	支持	MPEG-1, MPEG-2, MPEG-4, H.264	MPEG-1 Layers I, II, III, AAC,	IPTV，数字电视
FLV	Adobe Inc.	支持	Sorenson, VP6, H.264	MP3, ADPCM, Linear PCM, AAC等	互联网视频网站
MKV	CoreCodec Inc.	支持	几乎所有格式	几乎所有格式	互联网视频网站
RMVB	Real Networks Inc.	支持	RealVideo 8, 9, 10	AAC, Cook Codec, RealAudio Lossless	BT下载影视

视频编码

视频编码指通过特定的压缩技术，将某个视频内容数据转换成另一种特定格式文件的方式

名称	推出机构	推出时间	目前使用领域
HEVC(H.265)	MPEG/ITU-T	2013	研发中
H.264	MPEG/ITU-T	2003	各个领域
MPEG4	MPEG	2001	不温不火
MPEG2	MPEG	1994	数字电视
VP9	Google	2013	研发中
VP8	Google	2008	不普及
VC-1	Microsoft Inc.	2006	微软平台

网络视频平台参数对比

不同的视频编码，以及不同的压缩率直接影响播放速度，解 码速度，厂商 带宽和存储的成本

名称	协议	封装	视频编码	音频编码	播放器
CNTV	HTTP	MP4	H.264	AAC	Flash
CNTV（部分）	RTMP	FLV	H.264	AAC	Flash
华数TV	HTTP	MP4	H.264	AAC	Flash
优酷网	HTTP	FLV	H.264	AAC	Flash
土豆网	HTTP	F4V	H.264	AAC	Flash
56网	HTTP	FLV	H.264	AAC	Flash
音悦台	HTTP	MP4	H.264	AAC	Flash
乐视网	HTTP	FLV	H.264	AAC	Flash
新浪视频	HTTP	FLV	H.264	AAC	Flash

视频播放

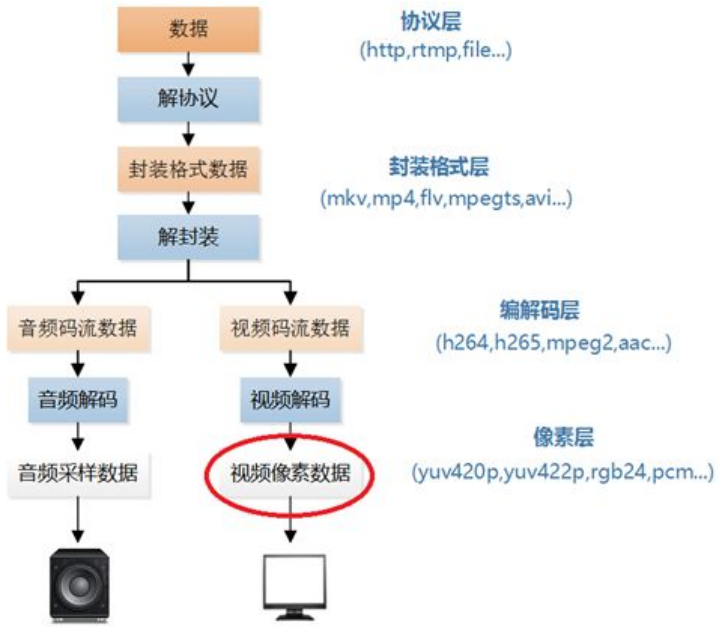
视频文件都会有特定的封装格式、比特率、时长等信息
视频解封装之后，就划分为video_stream和audio_stream，分别对应视频流和音频流。

解复用之后的音视频有自己独立的参数，包括

- 编码方式、采样率、画面大小等
- 音频参数包括采样率、编码方式和声道数等

解复用之后的音频和视频Packet进行解码之后，就变成原始的音频（PWM）和视频（YUV/RGB）数据，才可以在进行显示和播放。

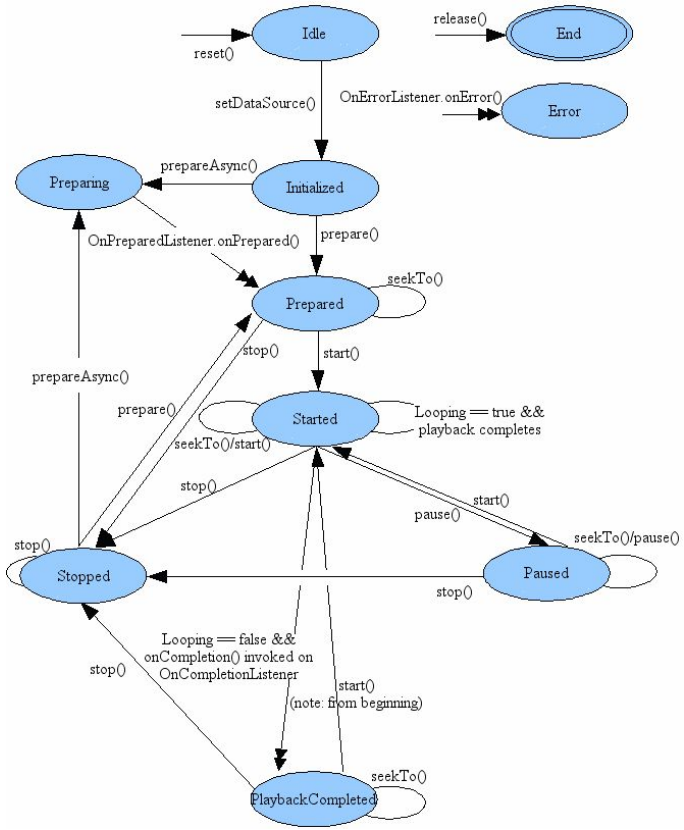
视频解码播放的大部分流程，整个视频播放的流程如右图所示



MediaPlayer

Android音视频播放相关类

MediaPlayer，应用层控制状态机



VideoView播放视频

Android系统提供了播放视频的控件VideoView

XML里配置

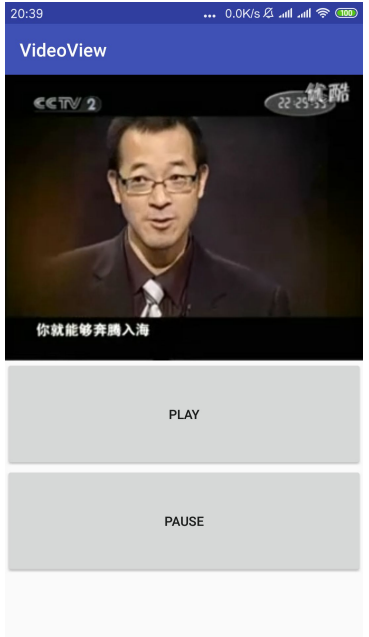
```
<VideoView
  android:id="@+id/videoView"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:layout_weight="1" />
```

代码控制

```
videoView = findViewById(R.id.videoView);
videoView.setVideoPath(getVideoPath(R.raw.yuminhong));
buttonPause = findViewById(R.id.buttonPause);
buttonPause.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    videoView.pause();
  }
});
buttonPlay = findViewById(R.id.buttonPlay);
buttonPlay.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    videoView.start();
  }
});
```

VideoView主要接口

- int getCurrentPosition()：获取当前播放的位置
- int getDuration()：获取当前播放 视频的总长度
- isPlaying()：当前VideoView是否在播放 视频
- void pause()：暂停
- void seekTo(int msec)：从第几毫秒开始播放
- void resume()：重新播放
- void setVideoPath(String path)：以文件路径的方式 设置VideoView播放的视频源
- void setVideoURI(Uri uri)：以Uri的方式设置VideoView播放的视频源，可以是网 络Uri或本地Uri
- void start()：开始播放
- void stopPlayback()：停止播放
- setMediaController(MediaController controller)：设置MediaController控制器
- setOnCompletionListener(MediaPlayer.onCompletionListener l)：监听播放完成的事件
- setOnErrorListener(MediaPlayer.OnErrorListener l)：监听播放发生错误时候的事件
- setOnPreparedListener(MediaPlayer.OnPreparedListener l) 监听视频装载完成的事件



MediaPlayer播放

MediaPlayer需要配合SurfaceView播放视频

初始化代码

```
surfaceView = findViewById(R.id.surfaceView);
player = new MediaPlayer();
try {
    player.setDataSource(getResources().openRawResourceFd(R.raw.yuminhong));
    holder = surfaceView.getHolder();
    holder.addCallback(new PlayerCallBack());
    player.prepare();
    player.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
        @Override
        public void onPrepared(MediaPlayer mp) {
            player.start(); // 初始化准 备好，立刻播放
            player.setLooping(true); // 循环播放
        }
    });
} catch (IOException e) {
    e.printStackTrace();
}
```

SurfaceView和MeidaPlayer关联

```
private class PlayerCallBack implements SurfaceHolder.Callback {

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        player.setDisplay(holder);
    }
    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {

    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {

    }
}
```

第三方播放器

第三方播放器

- Vitamio是Android平台视音频播放组件，支持播放几乎格式的视频以及主流网络视频流（http/rtsp/mms等）
- VLC 底层基于ffmpeg ios平台支持比较好，android 平台支持比较简陋，给出了android一个集成的例子
- ijkplayer是b站开源的超级好用的视频播放器，ijkplayer Android和ios都可用，还支持多种视频的硬解码。底层编解码基于FFmpeg 的ffplay实现，国内各大直播平台很多基于它进行定制开发
- JiaoZiVideoPlayer 属于个人开发者项目，中文名节操视频播放器，基于Android videoview实现的视频播放框架，可切换播放器引擎Ijkplayer Exoplayer Vitamio等
- 基于ijkPlayer，实现了多功能的视频播放器,开源项目有参考JiaoZiVideoPlayer部分代码，但后来做的扩展功能更丰富和强大

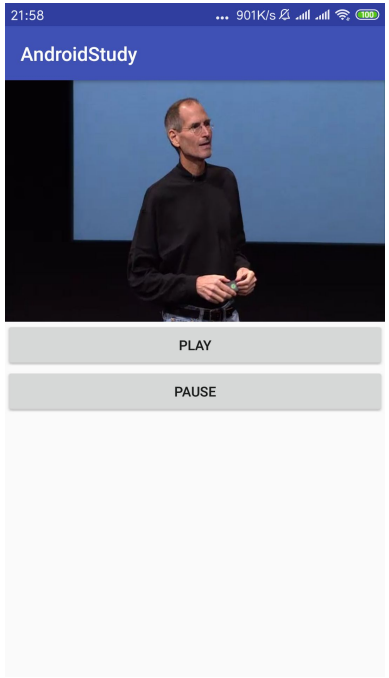
ijkPlayer播放视频

gradle引入

```
dependencies {
//     required, enough for most devices.
    compile 'tv.danmaku.ijk.media:ijkplayer-java:0.8.8'
    compile 'tv.danmaku.ijk.media:ijkplayer-armv7a:0.8.8'

//    # Other ABIs: optional
    compile 'tv.danmaku.ijk.media:ijkplayer-armv5:0.8.8'
    compile 'tv.danmaku.ijk.media:ijkplayer-arm64:0.8.8'
    compile 'tv.danmaku.ijk.media:ijkplayer-x86:0.8.8'
    compile 'tv.danmaku.ijk.media:ijkplayer-x86_64:0.8.8'

//    # ExoPlayer as IMediaPlayer: optional, experimental
    compile 'tv.danmaku.ijk.media:ijkplayer-exo:0.8.8'
}
```



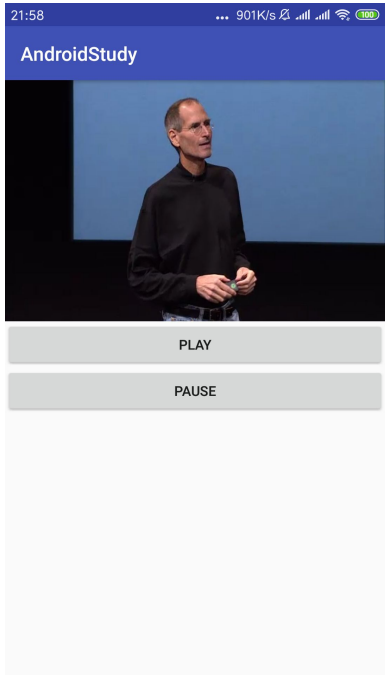
IjkMediaPlayer没有提供播放的View，需要创建SurfaceView关联在一起，思路和MediaPlayer类似

github地址 ijkplayer

ijkPlayer播放视频

初始化代码

```
IjkMediaPlayer ijkMediaPlayer = new IjkMediaPlayer();
ijkMediaPlayer.native_setLogLevel(IjkMediaPlayer.IJK_LOG_DEBUG);
// 开启硬解码
ijkMediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER, "mediacodec", 1);
// 设置路径，或者在线流媒体地址
ijkMediaPlayer.setDataSource(mPath);
// 给mediaPlayer设置视图
ijkMediaPlayer.setDisplay(surfaceView.getHolder());
ijkMediaPlayer.prepareAsync();
// 播放
ijkMediaPlayer.start();
```



作业

视频播放器

- 播放、暂停功能
- 进度条展示进度，点击进度条可以直接跳过去
- 横竖屏切换（手动，传感器自动）
- 注册Action为ACTION_VIEW，Data为Uri，Type为其MIME类型，关联播放器
- 手机相册中，录制视频列表展示，点击调起播放器

Q&A

ByteDance 字节跳动

Thanks