# 1、简介Picasso图片加载框架

介绍：Picasso，可译为“毕加索”，是Android中一个图片加载开源库。

主要作用：实现图片加载。

Picasso实现了图片的异步加载,并解决了Android中加载图片时常见的一些问题,它有以下特点:

在Adapter中取消了不在视图范围内的ImageView的资源加载,因为可能会产生图片错位;

使用复杂的图片转换技术降低内存的使用

自带内存和硬盘的二级缓存机制

为什么要用Picasso？

　　Android系统作为图片资源加载的主角,它是通过图像的像素点来把图像加载到内存中的;现在一张500W的摄像头拍出的照片(2592x1936),加载到内存中需要大约19M的内存;如果你加入了信号强度不一的网络中进行了复杂的网络请求,并进行图片的缓存与其他处理,你会耗费大量的时间与精力来处理这些问题,但如果用了Picasso, 这些问题都一消而散。

# 2、功能介绍

## 2.1 图片的异步加载（最基础功能）

ImageView targetImageView = (ImageView) findViewById(R.id.ImageView);

String Url = "http://218.192.170.132/1.jpg";

//Picasso使用了流式接口的调用方式

//Picasso类是核心实现类。

//实现图片加载功能至少需要三个参数：

Picasso

//with(Context context)

//Context对于很多Android API的调用都是必须的，这里就不多说了

.with(context)

//load(String imageUrl)：被加载图像的Url地址。

//大多情况下，一个字符串代表一个网络图片的URL。

.load(Url)

//into(ImageView targetImageView)：图片最终要展示的地方。

.into(targetImageView);

## 2.2 图片转换 resize()

使用最少的内存完成复杂的图片转换，转换图片以适合所显示的ImageView，来减少内存消耗。（扩展：加载网络图片的屏幕适配）

Picasso.with(context)

.load(url)

//裁剪图片尺寸

.resize(50, 50)

//设置图片圆角

.centerCrop()

.into(imageView)

如果你想对这张图片进行剪裁，可以使用resize方法：

Picasso.with(this).load("http://n.sinaimg.cn/translate/20160819/9BpA-fxvcsrn8627957.jpg")

.resize(200,200)

.into(iv);

注意这里的200表示200px，如果你想在resize时指定dp，可以使用如下方法：

Picasso.with(this).load("http://n.sinaimg.cn/translate/20160819/9BpA-fxvcsrn8627957.jpg")

.resizeDimen(R.dimen.iv\_width,R.dimen.iv\_height)

.into(iv);

很多时候我还可以给Picasso下载的图片设置缩放模式，也就是ImageView的ScaleType属性（不了解的请移步这里），但是注意：

缩放模式centerCrop和centerInside要和resize一起使用，否则会抛异常，

而缩放模式fit不可以和resize一起使用，如下：

使用fit：

Picasso.with(this).load("http://n.sinaimg.cn/translate/20160819/9BpA-fxvcsrn8627957.jpg")

.fit()

.into(iv);

使用centerCrop：

Picasso.with(this).load("http://n.sinaimg.cn/translate/20160819/9BpA-fxvcsrn8627957.jpg")

.resizeDimen(R.dimen.iv\_width,R.dimen.iv\_height)

.centerCrop()

.into(iv);

## 2.3 占位图、错误图

Picasso支持加载过程中和加载错误时显示对应图片

Picasso.with(context)

.load(url)

//占位图，图片加载出来之前显示的默认图片

.placeholder(R.drawable.user\_placeholder)

//错误图，图片加载出错时显示的图片

//如果重试3次（下载源代码可以根据需要修改）还是无法成功加载图片，则用错误占位符图片显示。

.error(R.drawable.user\_placeholder\_error)

.into(imageView);

## 2.4 在Adapter中的回收不在视野的ImageView和取消已经回收的ImageView下载进程

在ListView或者RecyclerView中加载图片时，当列表处于滑动状态的时候，我们可以停止图片的加载，当列表停止滚动的时候，我们又可以继续加载图片，如下：

Object tag = new Object();

Picasso with = Picasso.with(this);

with.load("http://n.sinaimg.cn/translate/20160819/9BpA-fxvcsrn8627957.jpg")

.into(iv);

//暂停加载

with.pauseTag(tag);

//恢复加载

with.resumeTag(tag);

## 2.5 从不同资源源加载

支持多种数据源 网络、本地、资源、Assets 等

//加载资源文件

Picasso.with(context).load(R.drawable.landing\_screen).into(imageView1);

//加载本地文件

Picasso.with(context).load(new File("/images/oprah\_bees.gif")).into(imageView2);

## 2.6 自动添加磁盘和内存二级缓存功能

（Picasso是二级缓存，而其他框架很多是三级缓存。

什么是网络缓存呢？）

现在大部分的图片缓存框架都是支持三级缓存的，在Picasso中，我们也可以手动设置缓存策略，比如说当我们查看一张大图的时候，可能由于图片太大，不想将其缓存在内存中，那么可以自定义缓存策略，如下：

Picasso picasso = Picasso.with(this);

//开启指示器

picasso.setIndicatorsEnabled(true);

picasso

.load("http://n.sinaimg.cn/translate/20160819/9BpA-fxvcsrn8627957.jpg")

//第一个参数是指图片加载时放弃在内存缓存中查找

//第二个参数是指图片加载完不缓存在内存中

.memoryPolicy(MemoryPolicy.NO\_CACHE, MemoryPolicy.NO\_STORE)

.into(iv);

关于图片缓存的地址，也是可以更改。

## 2.7 支持优先级处理

每次任务调度前会选择优先级高的任务，比如 App 页面中 Banner 的优先级高于 Icon 时就很适用。

## 2.8 支持飞行模式、并发线程数根据网络类型而变

手机切换到飞行模式或网络类型变换时会自动调整线程池最大并发数，比如 wifi 最大并发为 4， 4g 为 3，3g 为 2。

## 2.9 “无”本地缓存

无”本地缓存，不是说没有本地缓存，而是 Picasso 自己没有实现，交给了 Square 的另外一个网络库 okhttp 去实现，这样的好处是可以通过请求 Response Header 中的 Cache-Control 及 Expired 控制图片的过期时间。

## 2.10 转换器Transformation

很多时候，我们可能想显示一个用户图像，但是这个用户图像是个圆形图片，这个用Picasso该怎么实现呢？首先定义一个Transformation，在transform方法中对图片进行二次处理，包括剪裁重新处理等等，那我这里想把原图变为一个圆形图，就可以按下面的写法来：

Transformation transformation = new Transformation() {

@Override

public Bitmap transform(Bitmap source) {

int width = source.getWidth();

int height = source.getHeight();

int size = Math.min(width, height);

Bitmap blankBitmap = Bitmap.createBitmap(width, height,

Bitmap.Config.ARGB\_8888);

Canvas canvas = new Canvas(blankBitmap);

Paint paint = new Paint();

paint.setAntiAlias(true);

canvas.drawCircle(size / 2, size / 2, size / 2, paint);

paint.setXfermode(new PorterDuffXfermode(PorterDuff.Mode.SRC\_IN));

canvas.drawBitmap(source, 0, 0, paint);

if (source != null && !source.isRecycled()) {

source.recycle();

}

return blankBitmap;

}

@Override

public String key() {

return "squareup";

}

};

paint的setXfermode表示最终显示的图形取所绘制图形的交集，我这里先绘制了圆形，又绘制了一个矩形的Bitmap，圆形没有Bitmap大，所以交集肯定是圆形，所以最终显示结果就为圆形，在加载图片的时候可以通过transform属性来使用自定义的这个transformation，如下：

Picasso.with(this).load("http://n.sinaimg.cn/translate/20160819/9BpA-fxvcsrn8627957.jpg")

.transform(transformation)

.into(iv);

## 2.11 开启指示器

Picasso还可以通过开启指示器，让你看到这个图片是从内存加载来的还是从SD卡加载来的还是从网络加载来的，设置方式如下：

Picasso picasso = Picasso.with(this);

//开启指示器

picasso.setIndicatorsEnabled(true);

picasso.load("http://n.sinaimg.cn/translate/20160819/9BpA-fxvcsrn8627957.jpg")

.into(iv);

开启之后，图片的加载效果，左上角会有一个带颜色的三角符号。不同的颜色表示图片的来源不同，红、蓝、绿三种颜色分别代表网络、SD卡和内存。

## 2.12 Picasso进行图片的旋转(Rotation)

简单的旋转,只需要增加一个角度即可,调用rotate(float degrees)

当我们对一张图片需要进行简单的旋转处理时,只需要调用传入旋转的角度,大于0小于360即可,上代码:

Picasso

.with(context)

.load(UsageExampleListViewAdapter.eatFoodyImages[0])

.rotate(90f) //旋转90度

.into(imageViewSimpleRotate);

根据某个点进行复杂的旋转操作

因为图片的旋转都是相对(0,0)进行操作,所以如果我们想自定义相对于某个点,也是可以的,只需要调用

.rotate(float degrees, float pivotX, float pivotY)

Picasso

.with(context)

.load(R.drawable.floorplan)

.rotate(45f, 200f, 100f)

.into(imageViewComplexRotate);

## 2.13关于缓存机制

Picasso默认的缓存分配大小特点:

LRU缓存占应用程序可用内存的15%

本地缓存占到硬盘空间的2%但不超过50M并且不小于5M(前提是这种情况只在4.0以上有效果,或者你能像OKHttp那样提供一个本地缓存库来支持全平台)

Picasso默认开启3个线程来进行本地与网络之间的访问

Picasso加载图片顺序, 内存–>本地–>网络

（1）Memory Policy

可能有的时候你不想让Picasso去内存中进行读取而跳过此步骤,这时你可以在进行网络请求时调用memoryPolicy(MemoryPolicy policy, MemoryPolicy... additional),MemoryPolicy是一个枚举,只有两个值 NO\_CACHE 和 ‘NO\_STORE`

NO\_CACHE - 让Picasso跳过从内存中读取图片这一操作

NO\_STORE - 如果你的图片只加载一次就没用了,就调用该值,这样的话Picasso就不会在内存及本地进行缓存了

代码示例:

Picasso

.with(context)

.load(UsageExampleListViewAdapter.eatFoodyImages[1])

.memoryPolicy(MemoryPolicy.NO\_CACHE)

.into(imageViewFromDisk);

当然,你也可以这样调用:

Picasso

.with(context)

.load(UsageExampleListViewAdapter.eatFoodyImages[1])

.memoryPolicy(MemoryPolicy.NO\_CACHE, MemoryPolicy.NO\_STORE)

.into(imageViewFromDisk);

注意:调用.memoryPolicy(MemoryPolicy.NO\_CACHE)虽然能避免Picasso从内存中读取资源,但是并不能避免从本地读取资源,如果你也想跳过从本地读取这一过程,请看NetworkPolicy.

（2）NetworkPolicy

就像MemoryPolicy负责管理内存缓存一样,NetworkPolicy就是负责管理本地缓存的,而且二者的用法一模一样,NetworkPolicy也是一个枚举,不过它有三个值:

NO\_CACHE - 让Picasso跳过从本地读取资源这一过程

NO\_STORE - 让Picasso不进行本地图片缓存

OFFLINE - 让Picasso加载图片的时候只从本地读取除非联网正常并且本地找不到资源的情况下

示例代码:

Picasso

.with(context)

.load(UsageExampleListViewAdapter.eatFoodyImages[2])

.networkPolicy(NetworkPolicy.NO\_CACHE)

.into(imageViewFromNetwork);

当然,你也可以配合MemoryPolicy一起使用

Picasso

.with(context)

.load(UsageExampleListViewAdapter.eatFoodyImages[2])

.memoryPolicy(MemoryPolicy.NO\_CACHE, MemoryPolicy.NO\_STORE)

.networkPolicy(NetworkPolicy.NO\_CACHE)

.into(imageViewFromNetwork);

## 2.14 调用.noFade() .noPlaceholder()

　　Picasso的默认图片加载方式有一个淡入的效果,如果调用了noFade(),加载的图片将直接显示在ImageView上

Picasso

.with(context)

.load(UsageExampleListViewAdapter.eatFoodyImages[0])

.placeholder(R.mipmap.ic\_launcher)

.error(R.mipmap.future\_studio\_launcher)

.noFade()

.into(imageViewFade);

调用.noPlaceholder()

　　有一个场景,当你从网上加载了一张图片到Imageview上,过了一段时间,想在同一个ImageView上展示另一张图片,这个时候你就会去调用Picasso,进行二次请求,这时Picasso就会把之前的图片进行清除,可能展示的是.placeholder()的图片,给用户并不是很好的体验,如果调用了noPlaceholder(),就不会出现这种情况.

Picasso

.with(context)

.load(UsageExampleListViewAdapter.eatFoodyImages[0])

.placeholder(R.mipmap.ic\_launcher)

.into(imageViewNoPlaceholder, new Callback() {

@Override

public void onSuccess() {

// 当上次加载完成后,进行二次加载

Picasso

.with(context)

.load(UsageExampleListViewAdapter.eatFoodyImages[1])

.noPlaceholder()

.into(imageViewNoPlaceholder);

}

@Override

public void onError() {

}

});

## 2.15 调用resize(x, y)来自定义图片的加载大小

如果图片很大或者想自定义图片的显示样式,可以调用该API来解决这个问题;

Picasso

.with(context)

.load(UsageExampleListViewAdapter.eatFoodyImages[0])

.resize(600, 200)

.into(imageViewResize);

调用`onlyScaleDown()来缩短图片的加载计算时间

如果我们调用了resize(x,y)方法的话,Picasso一般会重新计算以改变图片的加载质量,比如一张小图变成一张大图进行展示的时候,但是如果我们的原图是比我们从新resize的新图规格大的时候,我们就可以调用onlyScaleDown()来直接进行展示而不再重新计算.

Picasso

.with(context)

.load(UsageExampleListViewAdapter.eatFoodyImages[0])

.resize(6000, 2000)

.onlyScaleDown() // 如果图片规格大于6000\*2000,将只会被resize

.into(imageViewResizeScaleDown);

## 2.16 对拉伸图片的处理

如果图片被操作了,可能在展示的时候就会比较丑,我们是想改变这种情景的,Picasso给我们提供了两种选择进行图片展示,centerCrop() 或者centerInside().

centerCrop() - 图片会被剪切,但是图片质量看着没有什么区别

Inside()- 图片会被完整的展示,可能图片不会填充满ImageView`,也有可能会被拉伸或者挤压

Picasso

.with(context)

.load(UsageExampleListViewAdapter.eatFoodyImages[0])

.resize(600, 200)

.centerInside() 或者调用 .centerCrop()

.into(imageViewResizeCenterInside);

调用.fit()来智能展示图片

如果调用了该API, Picasso会对图片的大小及ImageView进行测量,计算出最佳的大小及最佳的图片质量来进行图片展示,减少内存,并对视图没有影响;

Picasso

.with(context)

.load(UsageExampleListViewAdapter.eatFoodyImages[0])

.fit()

.into(imageViewHero);

## 2.17 调用.priority()设置图片加载的优先级

如果一个屏幕上顶部图片较大,而底部图片较小,因为Picasso是异步加载,所以小图会先加载出来,但是对于用户来说,更希望看到的是上面的图片先加载,底部的图片后加载,Picasso支持设置优先级,分为HIGH, MEDIUM, 和 LOW,所有的加载默认优先级为MEDIUM;

Picasso

.with(context)

.load(UsageExampleListViewAdapter.eatFoodyImages[0])

.fit()

.priority(Picasso.Priority.HIGH)

.into(imageViewHero);

注意:设置优先级并不能保证图片就一定会被优先加载,只是会偏向倾斜于先加载;

## 2.18 调用tag()为请求添加标记提升用户体验

　　我们都知道,在一个ListView的子item中加载一张图片是很常见的,这些图片都来源于网络请求,如果这个listview有上千条数据,当用户快速滑动的时候,每个item会不断的被复用,当然Picasso的请求也不断地进行请求,取消请求,再次请求,再次取消的操作(对屏幕外的自动取消请求),但是如果有一个方案,可以在用户在快速滑动的时候全部停止请求,只有在滑动停止时再去请求,就非常完美了;

Picasso提供了三种设置Tag的方式

暂停标记 pauseTag()

可见标记 resumeTag()

取消标记 cancleTag()

pauseTag() 和 resumeTag()的用法

在图片请求时添加标记

Picasso

.with(context)

.load(UsageExampleListViewAdapter.eatFoodyImages[0])

.tag("Profile ListView") //参数为 Object

.into(imageViewWithTag);

然后让listview实现滑动监听

@Override

public void onScrollStateChanged(AbsListView view, int scrollState) {

final Picasso picasso = Picasso.with(context);

if (scrollState == SCROLL\_STATE\_IDLE || scrollState == SCROLL\_STATE\_TOUCH\_SCROLL) {

picasso.resumeTag("Profile ListView");

} else {

picasso.pauseTag("Profile ListView");

}

}

cancleTag()的使用场景

试想一下,当你在浏览购物车的时候,这个时候就会去展示所有被选中item的图片资源,如果这个时候用户点击了购买按钮,就会弹出一个progressdialog去请求数据以进行页面跳转,这个时候原来的请求就需要取消掉了;

public void buyButtonClick(View v) {

showDiaolg();

// 取消网络请求

Picasso

.with(context)

.cancelTag("ShoppingCart");

}

注意:如果tag状态为pause或者resume的话,Picasso会对tag持有一个引用,如果此时用户退出了当前Activity,垃圾回收机制进行回收的时候,就会出现内存泄露,所以需要在onDestory()方法中进行相应处理;

.fetch() , .get() 及 Target之间的区别

.fetch() - 该方法会在后台异步加载一张图片,但是不会展示在ImageView上,也不会返回Bitmap,这个方法只是为了将获取到的资源加载到本地和内存中,为了后期加载缩短时间;

.get() - 该方法也是一个异步线程,不过加载完成后会返回一个Bitmap,但是需要注意,该方法不能在主线程中调用,因为会造成线程阻塞;

Target - 我们之前调用.into()方法,只是将获取到的资源加载到ImageView中,但我们还可以将资源作为回调放到Target中,上代码:

private Target target = new Target() {

@Override

public void onBitmapLoaded(Bitmap bitmap, Picasso.LoadedFrom from) {

//加载成功后会得到一个bitmap,可以自定义操作

}

@Override

public void onBitmapFailed(Drawable errorDrawable) {

// 加载失败进行相应处理

}

@Override

public void onPrepareLoad(Drawable placeHolderDrawable) {

}

};

Picasso

.with(context)

.load(UsageExampleListViewAdapter.eatFoodyImages[0])

.into(target);

注意:你可以使用.get()或者Target获取图片的Bitmap,但是当你使用Target时,不能使用匿名内部类的方式,因为垃圾回收机制在你获取不到Bitmap的时候会把对象回收;

# 3、Demo实例

步骤1：在gradle添加依赖

compile 'com.squareup.picasso:picasso:2.5.2'

步骤2：添加网络权限

<uses-permission android:name="android.permission.INTERNET"/>

步骤3：在MainActivity中

ImageView targetImageView = (ImageView) findViewById(R.id.ImageView);

String Url = "http://218.192.170.132/1.jpg";

Picasso

.with(this)

.load(Url)

.into(targetImageView);

# 4. 特点

## 4.1 优点

使用简单、方便（一行代码的事情）

由于同样是出品自Square的，Square 公司的其他开源库如 Retrofit 或者 OkHttp和Picasso搭配使用兼容性会更好些，占用体积也会少些。

所以，如果项目已经使用了 Square 公司的其他开源库（如 Retrofit 或者 OkHttp），在满足需求的前提下建议使用Picasso。

## 4.2 缺点

功能较为简单-图片加载；

性能（加载速度等等）较其他图片加载库（Glide、Fresco）较差。

自身无实现“本地缓存”。