# 1. 简介

介绍：Glide，是Android中一个图片加载开源库，Google的开源项目。

主要作用：实现图片加载。

# 2. 功能特点

从上面可以看出，Glide不仅实现了图片异步加载的功能，还解决了Android中加载图片时需要解决的一些常见问题。

## 2.1图片的异步加载（基础功能）

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

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

//Glide使用了流式接口的调用方式

//Glide类是核心实现类。

**Glide.with(context).load(Url).into(targetImageView);**

//实现图片加载功能至少需要三个参数：

//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

//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。

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

## 2.2设置加载尺寸

Glide.with(this).load(imageUrl).override(800, 800).into(imageView);

## 2.3占位图、错误图

api里面对占位图placeholder()、错误图error()函数中有多态实现，用的时候可以具体的熟悉一下

Glide

.with(this)

.load(imageUrl)

.placeholder(R.mipmap.ic\_launcher)

.error(R.mipmap.ic\_launcher).into(imageView);

## 2.4设置加载动画

Glide.with(this).load(imageUrl).animate(R.anim.item\_alpha\_in).into(imageView);

api也提供了几个常用的动画：比如crossFade()

R.anim.item\_alpha\_in

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

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

<alpha

android:duration="500"

android:fromAlpha="0.0"

android:toAlpha="1.0"/>

</set>

## 2.5设置要加载的内容

项目中有很多需要先下载图片然后再做一些合成的功能，比如项目中出现的图文混排，该如何实现目标下

Glide.with(this).load(imageUrl).centerCrop().into(new SimpleTarget<GlideDrawable>() {

@Override

public void onResourceReady(GlideDrawable resource, GlideAnimation<? super GlideDrawable> glideAnimation) {

imageView.setImageDrawable(resource);

}

});

## 2.6多样式的媒体加载

Glide

.with(context)

.load(imageUrl)；

.thumbnail(0.1f)；//设置缩略图支持：先加载缩略图 然后在加载全图

//传了一个 0.1f 作为参数，Glide 将会显示原始图像的10%的大小。

//如果原始图像有 1000x1000 像素，那么缩略图将会有 100x100 像素。

.asBitmap()//显示gif静态图片

.asGif();//显示gif动态图片

.into(imageView)；

## 2.7关于缓存

设置磁盘缓存策略

Glide.with(this).load(imageUrl).diskCacheStrategy(DiskCacheStrategy.ALL).into(imageView);

// 缓存参数说明

// DiskCacheStrategy.NONE：不缓存任何图片，即禁用磁盘缓存

// DiskCacheStrategy.ALL ：缓存原始图片 & 转换后的图片（默认）

// DiskCacheStrategy.SOURCE：只缓存原始图片（原来的全分辨率的图像，即不缓存转换后的图片）

// DiskCacheStrategy.RESULT：只缓存转换后的图片（即最终的图像：降低分辨率后 / 或者转换后 ，不缓存原始图片

设置跳过内存缓存

Glide

.with(this)

.load(imageUrl)

.skipMemoryCache(true)

.into(imageView);

//设置跳过内存缓存

//这意味着 Glide 将不会把这张图片放到内存缓存中去

//这里需要明白的是，这只是会影响内存缓存！Glide 将会仍然利用磁盘缓存来避免重复的网络请求。

清理缓存

Glide.get(this).clearDiskCache();//清理磁盘缓存 需要在子线程中执行

Glide.get(this).clearMemory();//清理内存缓存 可以在UI主线程中进行

## 2.8其他设置

（1）生命周期集成

通过设置绑定生命周期，我们可以更加高效的使用Glide提供的方式进行绑定，这样可以更好的让加载图片的请求的生命周期动态管理起来

.with(Context context)// 绑定Context

.with(Activity activity);// 绑定Activity

.with(FragmentActivity activity);// 绑定FragmentActivity

.with(Fragment fragment);// 绑定Fragment

注意：

传入的context类型影响到Glide加载图片的优化程度

Glide可以监视Activity的生命周期，在Activity销毁的时候自动取消等待中的请求。但是如果你使用Application context，你就失去了这种优化效果。

（2）设置动态转换

Glide.with(this).load(imageUrl).centerCrop().into(imageView);

（3）设置下载优先级

Glide.with(this).load(imageUrl).priority(Priority.NORMAL).into(imageView);

# 3. Demo实例

没有Demo的代码讲解不是好文章，让我们来一步步学会使用Glide。

步骤1：在gradle添加依赖

compile 'com.github.bumptech.glide:glide:3.7.0'

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";

Glide

.with(this)

.load(Url)

.into(targetImageView);

# 4. 特点

## 4.1 优点

多样化媒体加载

Glide 不仅是一个图片缓存，它支持 Gif、WebP、缩略图。甚至是 Video。

生命周期集成

通过设置绑定生命周期，我们可以更加高效的使用Glide提供的方式进行绑定，这样可以更好的让加载图片的请求的生命周期动态管理起来

高效的缓存策略

A. 支持Memory和Disk图片缓存

B. Picasso 只会缓存原始尺寸的图片，而 Glide 缓存的是多种规格，也就意味着 Glide 会根据你 ImageView 的大小来缓存相应大小的图片尺寸

比如你 ImageView 大小是200200，原图是 400400 ，而使用 Glide 就会缓存 200200 规格的图，而 Picasso 只会缓存 400400 规格的。这个改进就会导致 Glide 比 Picasso 加载的速度要快，毕竟少了每次裁剪重新渲染的过程，非常灵活 & 加载速度快

C. 内存开销小

默认的 Bitmap 格式是 RGB\_565 格式，而 Picasso 默认的是 ARGB\_8888 格式，这个内存开销要小一半。

Android关于图片内存计算，共有四种，分别是：

ALPHA\_8：每个像素占用1byte内存

ARGB\_4444:每个像素占用2byte内存

ARGB\_8888:每个像素占用4byte内存（默认，色彩最细腻=显示质量最高=占用的内存也最大）

RGB\_565:每个像素占用2byte内存（8bit = 1byte）

举例说明：一个32位的PNG=ARGB\_8888=1204x1024,那么占用空间是:1024x1024x(32/8) = 4,194,304kb=4M左右

在解析图片的时候，为了避免oom和节省内存，最好使用ARGB\_4444模式（节省一半的内存空间）

## 4.2 缺点

使用方法复杂

由于Glide其功能强大，所以使用的方法非常多，其源码也相对的复杂较大。

# 5. 相比其他图片加载库（Picasso & Fresco）

对比Picasso

Glide 是在Picasso 基础之上进行的二次开发做了不少改进，不过这也导致包比 Picasso 大不少，不过也就不到 500k（Picasso 是100多k），用法较为复杂，不过毕竟级别还是蛮小的，影响不是很大

对比Fresco

使用较Fresco简单，但性能（加载速度 & 缓存）却比不上Fresco

# 6. 应用场景

根据Glide的特点和与其他图片加载库的对比，可以得出其使用场景：

需要更多的内容表现形式(如Gif)；

更高的性能要求（缓存 & 加载速度）。

# 7.Glide使用SampleTarget 出现OOM

## （1）bitmap内存大小计算

density 的数值是 1dp=density px；densityDpi 是屏幕每英寸对应多少个点（不是像素点），在 DisplayMetrics 当中，这两个的关系是线性的。

1 density = 160densityDpi 。

一张522\*686的PNG 图片，我把它放到 drawable-xxhdpi 目录下，在三星s6上加载，占用内存2547360B，其中 density 对应 xxhdpi 为480，targetDensity 对应三星s6的密度为640：

522/480 \* 640 \* 686/480 \*640 \* 4 = 2546432B

scaledWidth = int( 522 \* 640 / 480f + 0.5) = int(696.5) = 696

scaledHeight = int( 686 \* 640 / 480f + 0.5) = int(915.16666…) = 915

下面就是见证奇迹的时刻：

915 \* 696 \* 4 = 2547360

Bitmap 在内存当中占用的大小其实取决于：

•色彩格式，前面我们已经提到，如果是 ARGB8888 那么就是一个像素4个字节，如果是 RGB565 那就是2个字节。

•原始文件存放的资源目录（是 hdpi 还是 xxhdpi 应该分清楚，如果你放在低分辨率的目录很可能就要放大，导致内存要大）

•目标屏幕的密度（所以同等条件下，红米在资源方面消耗的内存肯定是要小于三星S6的，你的设备的分辨率越低需求就越小）

## （2）使用 inSampleSize

既然图片最终是要被模糊的，也看不太情况，还不如直接用一张采样后的图片，如果采样率为 2，那么读出来的图片只有原始图片的 1/4 大小，真是何乐而不为呢？

## （3）Bitmap的像素格式

其实前面我们已经多次提到这个问题。ARGB8888格式的图片，每像素占用 4 Byte，而 RGB565则是 2 Byte。我们先看下有多少种格式可选：

格式 描述

ALPHA\_8 只有一个alpha通道

ARGB\_4444 这个从API 13开始不建议使用，因为质量太差

ARGB\_8888 ARGB四个通道，每个通道8bit

RGB\_565 每个像素占2Byte，其中红色占5bit，绿色占6bit，蓝色占5bit

这几个当中，

ALPHA8 没必要用，因为我们随便用个颜色就可以搞定的。

ARGB4444 虽然占用内存只有 ARGB8888 的一半，不过已经被官方嫌弃，失宠了。。『又要占省内存，又要看着爽，臣妾做不到啊T T』。

ARGB8888 是最常用的，大家应该最熟悉了。

RGB565 看到这个，我就看到了资源优化配置无处不在，这个绿色。。（不行了，突然好邪恶XD），其实如果不需要 alpha 通道，特别是资源本身为 jpg 格式的情况下，用这个格式比较理想。

## （4）Glide 加载大尺寸图片 OOM

•如果 Target 是 ImageView

•xml 中布局宽高自适应，且没有配置 override 参数，加载内存增加也就 3M 左右。

•如果 xml 中指定了更小的宽高或配置了 override 参数，那么内存会更小。

•如果 Target 不是 ImageView，比如 SimpleTarget

•SimpleTarget 中设置了参数或者设置了 override，根据尺寸的不同，内存增大不一，但基本在可控范围，10M 以内。

•未在构造时传入指定尺寸或者 override

Glide.with(getApplicationContext())

.load(url)

.asBitmap()

.into(new SimpleTarget<Bitmap>() {

@Override

public void onResourceReady(Bitmap bitmap, GlideAnimation<? super Bitmap> glideAnimation) {

// imageView.setImageBitmap(bitmap);

}

});

首先 into 方法将图片加载到内存中，然后回调 onResourceReady 这个方法，可见 Java 层内存飙升了 96M 左右，主要解码图片的操作。

可见有三种情况：

1.SimpleTarget 未设置宽高，加载原图尺寸

2.设置的宽高比原图尺寸还要大，加载原图尺寸

3.设置的宽高比原图尺寸小，用原图尺寸除以设置宽高，取最小值取整再向下取 2 的指数。因此最终获得的图片尺寸可能会比设置尺寸稍大。

•所以

在使用 SimpleTarget 的时候一定要先通过 override 设置尺寸，或者构造时传入尺寸。

•虽然实际图片尺寸可能比设置尺寸更大，但这样终究会有一个限制，限制在一定范围内。

•假设要显示的控件尺寸 20x20，图片尺寸 80x80，没有设置尺寸虽然不太可能导致 OOM，但终究也是对内存不必要的浪费。