Glide是Google官方推荐的图片加载框架，使得我们可以在Android平台上以极度简单的方式加载和展示图片。

要想使用Glide，首先需要将这个库引入到我们的项目当中：在app/build.gradle文件当中添加如下依赖：

dependencies {

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

}

另外，Glide中需要用到网络功能，因此你还得在AndroidManifest.xml中声明一下网络权限才行：

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

然后我们就可以自由地使用Glide中的任意功能了。

Glide.with(this).load(url).into(imageView);

这一行代码已经可以做非常非常多的事情了，包括加载网络上的图片、加载手机本地的图片、加载应用资源中的图片等等。

Glide.with()方法用于创建一个加载图片的实例。with()方法可以接收Context、Activity或者Fragment类型的参数。也就是说我们选择的范围非常广，不管是在Activity还是Fragment中调用with()方法，都可以直接传this。那如果调用的地方既不在Activity中也不在Fragment中呢？也没关系，我们可以获取当前应用程序的ApplicationContext，传入到with()方法当中。注意with()方法中传入的实例会决定Glide加载图片的生命周期，如果传入的是Activity或者Fragment的实例，那么当这个Activity或Fragment被销毁的时候，图片加载也会停止。如果传入的是ApplicationContext，那么只有当应用程序被杀掉的时候，图片加载才会停止。

load()方法用于指定待加载的图片资源。Glide支持加载各种各样的图片资源，包括网络图片、本地图片、应用资源、二进制流、Uri对象等等。因此load()方法也有很多个方法重载

// 加载本地图片

File file = new File(getExternalCacheDir() + "/image.jpg");

Glide.with(this).load(file).into(imageView);

// 加载应用资源

int resource = R.drawable.image;

Glide.with(this).load(resource).into(imageView);

// 加载二进制流

byte[] image = getImageBytes();

Glide.with(this).load(image).into(imageView);

// 加载Uri对象

Uri imageUri = getImageUri();

Glide.with(this).load(imageUri).into(imageView);

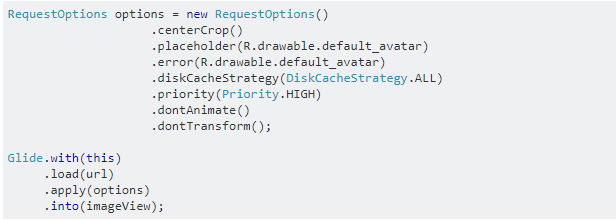
into()方法，让图片显示在哪个ImageView上，把这个ImageView的实例传进去就可以了。当然，into()方法不仅仅是只能接收ImageView类型的参数（后面会讲更多用法）。

先with()，再load()，最后into()，是Glide最核心的东西

Glide提供了占位图功能，占位图就是指在图片的加载过程中，我们先显示一张临时的图片，等图片加载出来了再替换成要加载的图片

~~在load()和into()方法之间串接任意想添加的功能，插入一个placeholder()方法就可以有占位图功能~~

新版的用法如下图所示：



Glide有非常强大的缓存机制，diskCacheStrategy(DiskCacheStrategy.NONE)可以禁用掉Glide的硬盘缓存功能。调用skipMemoryCache()方法并传入true，就表示禁用掉Glide的内存缓存功能。

异常占位图是指因为某些异常情况导致图片加载失败，比如说手机网络信号不好。串接一个error()方法就可以指定异常占位图了。

Glide是支持加载GIF图片的，而使用Glide加载GIF图并不需要编写什么额外的代码，Glide内部会自动判断图片格式。如果想指定图片的格式，串接一个一个asBitmap()方法。

既然能强制指定加载静态图片，就也能强制指定加载动态图片。如果指定了只允许加载动态图片，又传入了一张静态图片的URL地址，会加载失败。

Glide在绝大多数情况下是不需要指定图片大小的。内存浪费：比如说一张图片的尺寸是1000\*1000像素，但是我们界面上的ImageView只有200\*200像素，这个时候如果不对图片进行任何压缩就直接读取到内存中，这就属于内存浪费。而Glide会自动判断ImageView的大小，然后只将这么大的图片像素加载到内存当中，帮助我们节省内存开支。

不过如果真的必须给图片指定一个固定的大小，Glide仍然是支持这个功能的。串接一个override()方法，里面的参数是长和宽的像素。

阅读源码时的技巧，简单概括就是八个字：抽丝剥茧、点到即止。应该认准一个功能点，然后去分析这个功能点是如何实现的，但只要去追寻主体的实现逻辑即可，千万不要试图去搞懂每一行代码都是什么意思，那样很容易会陷入到思维黑洞当中，而且越陷越深。

Glide默认的图片变换，如果ImageView的scaleType是CENTER\_CROP，则会去调用applyCenterCrop()方法，如果scaleType是FIT\_CENTER、FIT\_START或FIT\_END，则会去调用applyFitCenter()方法。

Glide给我们提供了专门的API来添加和取消图片变换，调用一个dontTransform()方法

添加图片变换的用法非常简单，我们只需要调用transform()方法，并将想要执行的图片变换操作作为参数传入transform()方法即可。至于具体要进行什么样的图片变换操作，这个通常都是需要我们自己来写的。不过Glide已经内置了两种图片变换操作，我们可以直接拿来使用，一个是CenterCrop，一个是FitCenter。

FitCenter会将图片按照原始的长宽比充满全屏，CenterCrop是对原图的中心区域进行裁剪。centerCrop()方法还可以配合override()方法来实现更加丰富的效果

.override(500, 500)

.centerCrop()

但是使用dontTransform()方法存在着一个问题，就是调用这个方法之后，所有的图片变换操作就全部失效了，这种情况下我们只需要借助override()方法强制将图片尺寸指定成原始大小就可以了，代码如下所示：

Glide.with(this)

.load(url)

.override(Target.SIZE\_ORIGINAL, Target.SIZE\_ORIGINAL)

.into(imageView);

glide-transformations的项目主页地址是 https://github.com/wasabeef/glide-transformations

它实现了很多通用的图片变换效果，如裁剪变换、颜色变换、模糊变换等等，使得我们可以非常轻松地进行各种各样的图片变换。该库都是专门针对静态图片变换来进行设计的