**Glide 4.0 详解**

阅读 22593

收藏 145

2017-05-24

原文链接：[blog.csdn.net](https://link.juejin.im/?target=http%3A%2F%2Fblog.csdn.net%2Fgithub_33304260%2Farticle%2Fdetails%2F72526237" \t "_blank)

[RTC2018 实时互联网大会经典回放www.itdks.com](http://www.itdks.com/eventlist/detail/2543?channel=wx_shouciguanzhu)

**Glide 4.0.0 RC0 官方说明**

[Glide 4.0.0 RC0 官方说明](https://link.juejin.im/?target=https%3A%2F%2Fgithub.com%2Fbumptech%2Fglide%2Freleases%2Ftag%2Fv4.0.0-RC0)

**1.变化**

单独列出的更改太多，但这里有一些亮点：

* 新的文档，用户可以通过提交请求到Glide’s gh-pages分支贡献。
* 用户可以添加新类型或自定义选项集来轻松地自定义Glide流畅的API。
* 大量简化个人请求类型，确保选项始终如一，易于使用，即使您正在加载不同类型的资源。
* 各种性能改进，包括在下载采样图像时大量减少垃圾，更加智能的默认磁盘缓存策略，以及加载GIF时性能提升。
* 改进了视图大小和布局的处理，特别是在RecyclerView中。

**2.状态**

Glide 4.0由Google的各种团队内部使用，4.0被认为是内部稳定的。但外部用户可能会发现内部尚未发现的问题。因此，将此作为RC发布。如果没有发现稳定性或API中的重大问题，预计不久之后就会发布非RC版本。

**3.发布时间表**

Glide在过去采取了一种相当随意的方式发布，主要是因为在空闲的时候采取做。未来，Glide有望尝试提供定期发布：

* 每个月15日左右发行（确切的日期可能有所不同）
* 只有在前版本中没有更改的情况下才会跳过此版本。
* 只有在主要版本升级的时候才会去更改API。

**4.下载**

| **链接** | **大小** |
| --- | --- |
| [glide-full-4.0.0-RC0-javadoc.jar](https://link.juejin.im/?target=https%3A%2F%2Fgithub.com%2Fbumptech%2Fglide%2Freleases%2Fdownload%2Fv4.0.0-RC0%2Fglide-full-4.0.0-RC0-javadoc.jar) | 1.3 MB |
| [glide-full-4.0.0-RC0-sources.jar](https://link.juejin.im/?target=https%3A%2F%2Fgithub.com%2Fbumptech%2Fglide%2Freleases%2Fdownload%2Fv4.0.0-RC0%2Fglide-full-4.0.0-RC0-sources.jar) | 341 KB |
| [glide-full-4.0.0-RC0.jar](https://link.juejin.im/?target=https%3A%2F%2Fgithub.com%2Fbumptech%2Fglide%2Freleases%2Fdownload%2Fv4.0.0-RC0%2Fglide-full-4.0.0-RC0.jar) | 582 KB |
| [Source code (zip)](https://link.juejin.im/?target=https%3A%2F%2Fgithub.com%2Fbumptech%2Fglide%2Farchive%2Fv4.0.0-RC0.zip) |  |
| [Source code (tar.gz)](https://link.juejin.im/?target=https%3A%2F%2Fgithub.com%2Fbumptech%2Fglide%2Farchive%2Fv4.0.0-RC0.tar.gz) |  |

更多下载请移步官网：[Glide 4.0.0 RC0 官方说明](https://link.juejin.im/?target=https%3A%2F%2Fgithub.com%2Fbumptech%2Fglide%2Freleases%2Ftag%2Fv4.0.0-RC0)

**从V3迁移到V4**

[About Glide](https://link.juejin.im/?target=http%3A%2F%2Fsjudd.github.io%2Fglide%2F)

**1.Options（选项）**

一个在GlideV4变化较大的是库处理选项的方式（centerCrop()，placeholder()等）。在Glide v3中，选项是由一系列复杂的多类型构建器单独处理的。在Glide v4中，这些已被具有单一类型的单个构建器和可以提供给构建器的一系列选项的对象所替代。Glide 生成的API通过将选项对象和任何包含的集成库与构建器的选项合并，来创建单个流畅的API。 如下，options分类被放在不同的对象里，我们挨着看每个具体的方法：

**(1)RequestBuilder**

包括以下方法：

listener()   
thumbnail()   
load()   
into()

在Glide v4中，只有一个RequestBuilder，它使用单一的类型加载到你的项目(类型Bitmap，Drawable，GifDrawable等)。RequestBuilder提供了影响加载过程本身的选项，比如要加载的类型(url, uri etc)，任何 thumbnail()和listener()请求，RequestBuilder也提供在哪里开始加载的方法， into() or preload()。 我们看一下，下面的示例代码：

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

.load(url);

requestBuilder

.thumbnail(Glide.with(fragment)

.load(thumbnailUrl))

.listener(requestListener)

.load(url)

.into(imageView);

***实战示例：***

Glide.with(**this**)

.load(ImageConfig.URL\_WEBP)

.thumbnail(Glide.with(**this**).load(ImageConfig.URL\_JPEG))

.listener(**new** RequestListener<Drawable>() {

@Override

**public** **boolean** **onLoadFailed**(@Nullable GlideException e, Object model, Target<Drawable> target, **boolean** isFirstResource) {

Logger.e("onLoadFailed --->" +e);

**return** **false**;

}

@Override

**public** **boolean** **onResourceReady**(Drawable resource, Object model, Target<Drawable> target, DataSource dataSource, **boolean** isFirstResource) {

Logger.e("onResourceReady---> " + resource);

**return** **false**;

}

})

.into(iv\_test1);

效果图：   
我们可以看到第一次安装完成之后，在gif没有加载出来先加载缩略图，然后再加载gif图片，个人感觉加载gif图片的速度是比之前快多了，这些都是加载网络图片和网速也有一定关系。   
加载完成后，推出APP，再次进入会首先加载之前缓存在本地的图片，所以还是那么的爽。

**(2)RequestOptions**

包括以下方法：

centerCrop()   
placeholder()   
error()   
priority()   
diskCacheStrategy()

大多数的设置已经放到了RequestOptions对象中：   
***示例：***

RequestOptions options = new RequestOptions()

.centerCrop()

.placeholder(R.drawable.placeholder)

.error(R.drawable.error)

.priority(Priority.HIGH)

.diskCacheStrategy(DiskCacheStrategy.NONE);

RequestOptions一次设置之后，可以在多处使用

RequestOptions myOptions = new RequestOptions()

.fitCenter()

.override(100, 100);

Glide.with(fragment)

.load(url)

.apply(myOptions)

.into(drawableView);

Glide.with(fragment)

.asBitmap()

.apply(myOptions)

.load(url)

.into(bitmapView);

***实战实例：***

RequestOptions options = new RequestOptions()

.centerCrop()

.placeholder(R.mipmap.ic\_launcher\_round)

.error(R.mipmap.ic\_launcher)

.priority(Priority.HIGH)

.diskCacheStrategy(DiskCacheStrategy.NONE);

Glide.with(this)

.load(ImageConfig.URL\_GIF)

.apply(options)

.thumbnail(Glide.with(this)

.load(ImageConfig.URL\_JPEG))

.into(iv\_test1);

Glide.with(this)

.load(ImageConfig.URL\_WEBP)

.apply(options)

.thumbnail(Glide.with(this)

.load(ImageConfig.URL\_JPEG))

.into(iv\_test2);

效果图： 

**(3)TransitionOptions**

顾名思义，变换相关的设置在这里：

包括以下方法：

crossFade() animate()

选择一个你所需要的TransitionOptions：

* [GenericTransitionOptions](https://link.juejin.im/?target=http%3A%2F%2Fsjudd.github.io%2Fglide%2Fjavadocs%2F400%2Fcom%2Fbumptech%2Fglide%2FGenericTransitionOptions.html)
* [DrawableTransitionOptions](https://link.juejin.im/?target=http%3A%2F%2Fsjudd.github.io%2Fglide%2Fjavadocs%2F400%2Fcom%2Fbumptech%2Fglide%2Fload%2Fresource%2Fdrawable%2FDrawableTransitionOptions.html)
* [BitmapTransitionOptions](https://link.juejin.im/?target=http%3A%2F%2Fsjudd.github.io%2Fglide%2Fjavadocs%2F400%2Fcom%2Fbumptech%2Fglide%2Fload%2Fresource%2Fbitmap%2FBitmapTransitionOptions.html)

如果想去掉默认的变换效果：[TransitionOptions.dontTransition().](https://link.juejin.im/?target=http%3A%2F%2Fsjudd.github.io%2Fglide%2Fjavadocs%2F400%2Fcom%2Fbumptech%2Fglide%2FTransitionOptions.html%23dontTransition--" \t "_blank)

Transitions被要求用在 [RequestBuilder](https://link.juejin.im/?target=http%3A%2F%2Fsjudd.github.io%2Fglide%2Fjavadocs%2F400%2Fcom%2Fbumptech%2Fglide%2FRequestBuilder.html" \t "_blank)

***示例：***

Glide.with(fragment)

.load(url)

.transition(withCrossFade(R.anim.fade\_in, 300));

***实战示例：***

//使用变换效果

Glide.with(this)

.load(ImageConfig.URL\_WEBP)

.apply(options)

.transition(new DrawableTransitionOptions().crossFade(2000))

.thumbnail(Glide.with(this)

.load(ImageConfig.URL\_JPEG))

.into(iv\_test1);

//不使用变换效果

Glide.with(this)

.load(ImageConfig.URL\_WEBP)

.apply(options)

.transition(new DrawableTransitionOptions().dontTransition())

.thumbnail(Glide.with(this)

.load(ImageConfig.URL\_JPEG))

.into(iv\_test2);

***效果图：*** 

**(4)生成的API**

***关于***

GlideV4：使用一个注释处理器生成了一个API，允许应用程序访问的所有选项RequestBuilder，RequestOptions以及任何包含集成库在一个流畅的API。

生成的API有两个目的：

* 集成库可以通过自定义选项扩展Glide的API。
* 应用程序可以通过添加捆绑常用选项的方法来扩展Glide的API。

尽管这两个任务都可以通过编写RequestOptions的自定义子类来手工完成，但是这样做是具有挑战性的，并且会产生一个不那么流畅的API。

***入门***

要触发API生成，请在应用程序中包含一个AppGlideModule实现：

**package** com.example.myapp;

**import** com.bumptech.glide.annotation.GlideModule;

**import** com.bumptech.glide.module.AppGlideModule;

@GlideModule

**public** **final** **class** **MyAppGlideModule** **extends** **AppGlideModule** {}

请注意，AppGlideModule实现必须始终注释@GlideModule。如果注释不存在，则不会发现该模块，并且您将在日志中看到一条带有Glide日志标记的警告，该日志标记指示该模块无法找到。

API在与应用程序AppGlideModule提供的实现相同的包中生成，并且GlideApp默认命名。应用程序可以通过启动所有，加载GlideApp.with()而不是使用API Glide.with()：

GlideApp.with(fragment)

.placeholder(R.drawable.placeholder)

.fitCenter()

.load(myUrl)

.into(imageView);

请注意，与Glide.with()类似的选项fitCenter()，并placeholder()提供直接的建设者，并不需要传递作为一个独立的RequestOptions对象。

**实战演示**

创建自定义GlideModle： 

⚠️ 有的童鞋发现@GlideModule之后并没有生成GlideAPP对象，这时候需要注意两步：

1、依赖是否都导入   
compile 'com.github.bumptech.glide:glide:4.0.0-RC0'   
compile 'com.github.bumptech.glide:compiler:4.0.0-RC0'   
compile 'com.[**Android**](https://link.juejin.im/?target=http%3A%2F%2Flib.csdn.net%2Fbase%2Fandroid).support:support-v4:25.3.1'   
  
2、点击“Make Project”

这时候会发现自定义的AppGlideModle被注入到GeneratedAppGlideModuleImpl里面，如图： 

这时候已经可以去使用GlideApp.with(this)，接下来就会让我们回到熟悉的感觉:

GlideApp.with(this)

.load(ImageConfig.URL\_WEBP)

.sizeMultiplier(0.5f)

.centerCrop()

.diskCacheStrategy(DiskCacheStrategy.ALL)

.error(R.mipmap.ic\_launcher)

.into(iv\_test1);

是不是有种找回了记忆中的味道！！！

**2.Types and Targets （类型和目标）**

**(1)加载的资源类型**

Glide允许您指定要加载的资源类型。如果指定超类型，则Glide将尝试加载任何可用的子类型。例如，如果您要求Drawable，Glide可能会加载BitmapDrawable或GifDrawable。如果您要求GifDrawable，如果图像不是GIF，Glide将加载GifDrawable或错误（即使它恰好是完全有效的图像）。

默认情况下请求可绘制值：

Glide.with(fragment).load(url)

要求位图：

Glide.with(fragment).asBitmap()

获取文件路径（最适合本地映像）：

Glide.with(fragment).asFile()

要将远程文件下载到缓存中并获取文件路径：

Glide.with(fragment).downloadOnly()

// **or** if you have the url already:

Glide.with(fragmetn).download(url);

**(2)Drawables**

GlideDrawable在Glide v3已被删改为Android Drawable。GlideBitmapDrawable已被删改为BitmapDrawable。

如果你想知道一个Drawable是否是动画的，你可以检查它是否是一个实例[Animatable](https://link.juejin.im/?target=https%3A%2F%2Fdeveloper.android.com%2Freference%2Fandroid%2Fgraphics%2Fdrawable%2FAnimatable.html" \t "_blank)：

**boolean** isAnimated = drawable **instanceof** Animatable

**(3)Targets**

onResourceReady的回调已经发生了改变，例如Drawables：

***以前：***

**onResourceReady**(GlideDrawable drawable, GlideAnimation<? super GlideDrawable> anim)

***现在：***

onResourceReady(Drawable drawable, Transition<? **super** Drawable> transition);

同样onLoadFailed也改变了：

***以前：***

**onLoadFailed**(Exception e, Drawable errorDrawable)

***现在：***

**onLoadFailed**(Drawable errorDrawable)

如果需要有关加载失败或成功的信息，可以使用[RequestListener](https://link.juejin.im/?target=http%3A%2F%2Fsjudd.github.io%2Fglide%2Fjavadocs%2F400%2Fcom%2Fbumptech%2Fglide%2Frequest%2FRequestListener.html" \t "_blank)。

**3.Configuration（配置）**

在Glide v3中，可以配置一个活着多个GlideModules。在Glide v4中，通过类似但稍微更复杂的系统进行配置。

**(1)Applications（应用）**

应用中的[GlideModule](https://link.juejin.im/?target=http%3A%2F%2Fsjudd.github.io%2Fglide%2Fjavadocs%2F360%2Fcom%2Fbumptech%2Fglide%2Fmodule%2FGlideModule.html" \t "_blank)转换成[AppGlideModule](https://link.juejin.im/?target=http%3A%2F%2Fsjudd.github.io%2Fglide%2Fjavadocs%2F400%2Fcom%2Fbumptech%2Fglide%2Fmodule%2FAppGlideModule.html" \t "_blank)。

在Glide v3中，你可得GlideModule可能是这样的：

**public** **class** **GiphyGlideModule** **implements** **GlideModule** {

@Override

**public** **void** **applyOptions**(Context context, GlideBuilder builder) {

builder.setMemoryCache(**new** LruResourceCache(10 \* 1024 \* 1024));

}

@Override

**public** **void** **registerComponents**(Context context, Registry registry) {

registry.append(Api.GifResult.class, InputStream.class, **new** GiphyModelLoader.Factory());

}

}

在Glide v4中，您可以将其转换成AppGlideModule如下所示：

@GlideModule

**public** **class** **GiphyGlideModule** **extends** **AppGlideModule** {

@Override

**public** **void** **applyOptions**(Context context, GlideBuilder builder) {

builder.setMemoryCache(**new** LruResourceCache(10 \* 1024 \* 1024));

}

@Override

**public** **void** **registerComponents**(Context context, Registry registry) {

registry.append(Api.GifResult.class, InputStream.class, **new** GiphyModelLoader.Factory());

}

}

⚠️注意： Glide4.0 必须要使用注解： `@GlideModule

您的应用程序有多个GlideModules，将其中一个转换为AppGlideModule，其他的转换为LibraryGlideModules。除非出现一个AppGlideModule，否则不会发现LibraryGlideModules，因此您不能只使用LibraryGlideModules。`

**(2)Libraries**

有一个或多个GlideModules的库应该使用LibraryGlideModule而不是AppGlideModule。库不应该使用AppGlideModules，因为每个应用程序只能有一个，所以在一个库中包含它不仅会阻止库的用户设置他们自己的选项，而且如果多个库包含一个AppGlideModule，它也会导致冲突。

例如， the Volley GlideModule 在 V3:

**public** **class** **VolleyGlideModule** **implements** **GlideModule** {

@Override

**public** **void** **applyOptions**(Context context, GlideBuilder builder) {

// Do nothing.

}

@Override

**public** **void** **registerComponents**(Context context, Registry registry) {

registry.replace(GlideUrl.class, InputStream.class, **new** VolleyUrlLoader.Factory(context));

}

}

在V4中转换成如下：

@GlideModule

**public** **class** **VolleyLibraryGlideModule** **extends** **LibraryGlideModule** {

@Override

**public** **void** **registerComponents**(Context context, Registry registry) {

registry.replace(GlideUrl.class, InputStream.class, **new** VolleyUrlLoader.Factory(context));

}

}

**(3)Manifest parsing**

为了简化迁移，清单解析和旧GlideModule界面已被弃用，但在v4中仍然支持。AppGlideModules，LibraryGlideModules和被弃用GlideModule的都可以在应用程序中共存。

但是，为了避免检查元数据（和相关的错误）的性能开销，您可以在迁移完成后通过覆盖以下方法来禁用清单解析AppGlideModule：

@GlideModule

**public** **class** **GiphyGlideModule** **extends** **AppGlideModule** {

@Override

**public** **boolean** **isManifestParsingEnabled**() {

**return** **false**;

}

...

}

**4.** **获取 Bitmap 对象**

1. 同步加载 Bitmap 对象

有时我们需要在子线程中获取 Bitmap 对象，就需要同步获取的方式。

val bitmap = Glide.with(activity)

.asBitmap()

.load(imageUrl)

.submit().get()

借助 asBitmap() 和 submit().get() 就可以从某个源中，直接获得 Bitmap 对象。

submit() 还可以约束加载图片的尺寸，方便我们处理。

FutureTarget<TranscodeType> submit()

FutureTarget<TranscodeType> submit(int width, int height)

2. 异步加载 Bitmap 对象

Glide 也支持异步加载 Bitmap，异步加载，就涉及到线程的切换问题。

Glide.with(activity)

.asBitmap()

.load(imageUrl)

.into(object:CustomTarget<Bitmap>(){

override fun onLoadCleared(placeholder: Drawable?) {

}

override fun onResourceReady(resource: Bitmap,

transition: Transition<in Bitmap>?) {

val loadBitmap = resource

}

})

异步加载，需要用到 Target，这里直接使用 Glide 提供的 CustomTarget。

3. 加载图片的 File

我们知道用 Glide 加载的图片，在缓存容量允许的范围内，Glide 都会帮我们将图片文件缓存到本地磁盘。

那么我们如何通过 Glide 加载一个图片资源，然后获得缓存的图片文件呢？

其实只需要将上面的 asBitmap() 换成 asFile() 即可。

// sync

val bitmapFile = Glide.with(this)

.asFile()

.load(imageUrl)

.submit().get()

// async

Glide.with(this)

.asFile()

.load(imageUrl)

.into(object:CustomTarget<File>(){

override fun onLoadCleared(placeholder: Drawable?) {

}

override fun onResourceReady(resource: File, transition:

Transition<in File>?) {

val bitmapFile = resource

}

})

除了 asBitmap() 和 asFile()，还有一些其他的方法，例如 asDrawable() 等，有兴趣可以自行了解。

拓展

<https://www.jianshu.com/p/9db8f314b286>

<https://www.jianshu.com/p/cfb3d467687d>

**options** = **new** RequestOptions().format(DecodeFormat.***PREFER\_RGB\_565***);

修改图片质量