Android 基于 Glide 的二次封装

为什么选择 Glide？

Glide 轻量级
速度快
可以根据所需加载图片的大小自动适配所需分辨率的图
支持多种格式图片（静态 webp，动态 gif，jpeg，jpg，png）
支持多种数据源图片（url，drawable，src，file，asserts，raw）
Google 主导
更多知识点可以看我的上一篇文章：
Picasso，Glide，Fresco 对比分析
http://blog.csdn.net/github_33304260/article/details/70213300

已经很方便了，为啥还要封装？

避免以后换框架的时候需要改的地方太多。如果封装了只需要改封装的方法而不会影响到所有的代码。
入口统一，所有图片加载都在这一个地方管理，一目了然，即使有什么改动我也只需要改这一个类就可以了。
虽然现在的第三方库已经非常好用，但是如果我们看到第三方库就拿来用的话，很可能在第三方库无法满足业务需求或者停止维护的时候，发现替换库，工作量可见一斑。这就是不封装在切库时面临的窘境！
外部表现一致，内部灵活处理原则
更多内容参考：如何正确使用开源项目？

初识 Glide

Glide 配置

1、 在 build.gradle 中添加依赖：

```
dependencies {

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

  compile 'com.android.support:support-v4:19.1.0'

}
```

2、混淆

```
-keep public class * implements com.bumptech.glide.module.GlideModule

-keep public enum com.bumptech.glide.load.resource.bitmap.ImageHeaderParser$** {

  **[] $VALUES;

  public *;

}

# for DexGuard only

-keepresourcexmlelements manifest/application/meta-data@value=GlideModule
```

3、权限
如果是联网获取图片或者本地存储需要添加以下权限：

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

Glide 基本使用

Glide 使用一个流接口（Fluent Interface）。用 Glide 完成一个完整的图片加载功能请求，需要向其构造器中至少传入 3 个参数，分别是：

with(Context context)- Context 是许多 Android API 需要调用的， Glide 也不例外。这里 Glide 非常方便，你可以任意传递一个 Activity 或者 Fragment 对象，它都可以自动提取出上下文。
load(String imageUrl) - 这里传入的是你要加载的图片的 URL，大多数情况下这个 String 类型的变量会链接到一个网络图片。
into(ImageView targetImageView) - 将你所希望解析的图片传递给所要显示的 ImageView。
example:

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

String internetUrl = "http://i.imgur.com/DvpvklR.png";



Glide

    .with(context)

    .load(internetUrl)

    .into(targetImageView);
```

更多 Glide 详细介绍可以看 Glide 官网以及 Glide 教程系列文章

如何进行二次封装

明白了为什么封装以及基本原理，接下来我们就要开工，大干一场。

先看一下本人封装后的基本使用样式：

```
ImageLoader.with(this)

    .url("http://img.yxbao.com/news/image/201703/13/7bda462477.gif")

    .placeHolder(R.mipmap.ic_launcher,false)

    .rectRoundCorner(30, R.color.colorPrimary)

    .blur(40)

    .into(iv_round);
```

更多属性我们后再详细讲解使用，主要先来看看具体的封装。

先看一下 uml：

使用者只需要关心 ImageLoader 就好了，就算里面封装的库更换、更新也没关系，因为对外的接口是不变的。实际操作中是由实现了 ILoader 的具体类去操作的，这里我们只封装了 GlideLoader，其实所有操作都是由 ImageLoader 下发指令，由 GlideLoader 具体去实现的。这里如果想封装别的第三方库，只需要实现 ILoader 自己去完成里面的方法。

初始化

```
    public static int CACHE_IMAGE_SIZE = 250;



    public static void init(final Context context) {

        init(context, CACHE_IMAGE_SIZE);

    }



    public static void init(final Context context, int cacheSizeInM) {

        init(context, cacheSizeInM, MemoryCategory.NORMAL);

    }



    public static void init(final Context context, int cacheSizeInM, MemoryCategory
memoryCategory) {
```

```
        init(context, cacheSizeInM, memoryCategory, true);
    }

    /**
     * @param context        上下文
     * @param cacheSizeInM   Glide 默认磁盘缓存最大容量 250MB
     * @param memoryCategory 调整内存缓存的大小 LOW(0.5f) / NORMAL(1f) / HIGH(1.5f);
     * @param isInternalCD   true 磁盘缓存到应用的内部目录 / false 磁盘缓存到外部存
     */
    public static void init(final Context context, int cacheSizeInM, MemoryCategory
memoryCategory, boolean isInternalCD) {
        ImageLoader.context = context;
        GlobalConfig.init(context, cacheSizeInM, memoryCategory, isInternalCD);
    }
```

从这里可以看出我们提供了四个构造器，这里注释详细说明了所有参数的用法及意义。

除了初始化，我们还需要在 Application 中重写以下方法：

```
    @Override
    public void onTrimMemory(int level) {
        super.onTrimMemory(level);
        // 程序在内存清理的时候执行
        ImageLoader.trimMemory(level);
    }

    @Override
    public void onLowMemory() {
        super.onLowMemory();
        // 低内存的时候执行
        ImageLoader.clearAllMemoryCaches();
```

```
    }
```

上面这两个方法会在下面 ImageLoader 中介绍到。

你所关心的类–ImageLoader

ImageLoader 是封装好所有的方法供用户使用的，让我们看看都有什么方法：

ImageLoader.init(Context context) //初始化
ImageLoader.trimMemory(int level);
ImageLoader.clearAllMemoryCaches();
ImageLoader.getActualLoader(); //获取当前的 loader
ImageLoader.with(Context context) //加载图片
ImageLoader.saveImageIntoGallery(DownLoadImageService downLoadImageService) // 保存图片到相册
ImageLoader.pauseRequests() //取消请求
ImageLoader.resumeRequests() //回复的请求（当列表在滑动的时候，调用 pauseRequests()取消请求，滑动停止时，调用 resumeRequests()恢复请求 等等）
ImageLoader.clearDiskCache()//清除磁盘缓存(必须在后台线程中调用)
ImageLoader.clearMomoryCache(View view) //清除指定 view 的缓存
ImageLoader.clearMomory() // 清除内存缓存(必须在 UI 线程中调用)
github 项目地址

图片的各种设置信息–SingleConfig

我们所设置图片的所有属性都写在这个类里面。下面我们详细的看一下：

url(String url) / / url
file(String filePath) //加载 SD 卡资源
file(File file) //加载 SD 卡资源
res(int resId) //加载 drawable 资源
content(String contentProvider) //加载 ContentProvider 资源
raw(String rawPath) //加载 raw 资源
asserts(String assertspath) //加载 asserts 资源
thumbnail(float thumbnail)//缩略图
rectRoundCorner(int rectRoundRadiusf) //形状为圆角矩形时的圆角半径
priority(int priority) //优先级
error(int errorResId) //错误占位图
asSquare() //形状为正方形
colorFilter(int color) //颜色滤镜
diskCacheStrategy(DiskCacheStrategy diskCacheStrategy) //DiskCacheStrategy.NONE :不缓存图片 / DiskCacheStrategy.SOURCE :缓存图片源文件 / DiskCacheStrategy.RESULT:缓存修改过的图片 / DiskCacheStrategy.ALL:缓存所有的图片，默认
ignoreCertificateVerify(boolean ignoreCertificateVerify) // https 是否忽略校验

asCircle()//加载圆形图片
placeHolder(int placeHolderResId) //占位图
override(int oWidth, int oHeight) //加载图片时设置分辨率 a
scale(int scaleMode) // CENTER_CROP 等比例缩放图片，直到图片的狂高都大于等于 ImageView 的宽度，然后截取中间的显示 ；FIT_CENTER 等比例缩放图片，宽或者是高等于 ImageView 的宽或者是高 默认：FIT_CENTER
animate(int animationId) 引入动画
animate(Animation animation) 引入动画
animate(ViewPropertyAnimation.Animator animato) 引入动画
asBitmap(BitmapListener bitmapListener)// 使用 bitmap 不显示到 imageview
into(View targetView) //展示到 imageview
colorFilter(int filteColor) //颜色滤镜
blur(int blurRadius) / /高斯模糊
brightnessFilter(float level) //调节图片亮度
grayscaleFilter() //黑白效果
swirlFilter() //漩涡效果
toonFilter() //油画效果
sepiaFilter() //水墨画效果
contrastFilter(float constrasrLevel) //锐化效果
invertFilter() //胶片效果
pixelationFilter(float pixelationLevel) //马赛克效果
sketchFilter() // //素描效果
vignetteFilter() //晕映效果
github 项目地址

中转站–GlobalConfig

GlobalConfig 类非常简单主要是选择 Loader 的操作，之所以用到这个类是因为方便以后扩展。今后我们如果需要使用其他的图片加载框架，只需要继承 ILoader，然后在 GlobalConfig 中配置即可。

```
public class GlobalConfig {

    public static void init(Context context, int cacheSizeInM,

        getLoader().init(context, cacheSizeInM, memoryCategory, isInternalCD);



    }



    public static ILoader getLoader() {
```

```
        if (loader == null) {
            // 这里只做了 glide 的封装
            loader = new GlideLoader();
        }
        // 可以接着做 fresco 或者 picasso

        return loader;
    }
{
```

最终的执行者–GlideLoader

GlideLoader 实现 ILoader 接口。在使用的时候我们虽然不用关心这个类，但是了解一下主要做了什么功能还是必要的。
GlideLoader 中主要做了两件事，一个是初始化的实现，一个是出口方法的实现。

初始化的实现

我们在 application 中调用

```
ImageLoader.init(getApplicationContext());
```

会最终调用到下面这个方法，最终的操作都是在这里进行的

```
public class GlideLoader implements ILoader {


    /**
     * @param context        上下文
     * @param cacheSizeInM   Glide 默认磁盘缓存最大容量 250MB
     * @param memoryCategory 调整内存缓存的大小 LOW(0.5f) / NORMAL(1f) / HIGH(1.5f);
     * @param isInternalCD   true 磁盘缓存到应用的内部目录 / false 磁盘缓存到外部存
     */
    @Override
```

```
    public void init(Context context, int cacheSizeInM, MemoryCategory memoryCategory, boolean isInternalCD) {
        Glide.get(context).setMemoryCategory(memoryCategory); //如果在应用当中想要调整内存缓存的大小，开发者可以通过如下方式：
        GlideBuilder builder = new GlideBuilder(context);
        if (isInternalCD) {
            builder.setDiskCache(new InternalCacheDiskCacheFactory(context, cacheSizeInM * 1024 * 1024));
        } else {
            builder.setDiskCache(new ExternalCacheDiskCacheFactory(context, cacheSizeInM * 1024 * 1024));
        }
    }
}
```

出口方法的实现

不管最终我们使用的是 into（） 还是 asBitmap（）最终都会调用 GlideLoader 的 request()方法。

request（）方法里面我们主要思路是拿到 DrawableTypeRequest，然后根据 SingleConfig 中的配置信息进行设置。

```
@Override
    public void request(final SingleConfig config) {
        RequestManager requestManager = Glide.with(config.getContext());
        DrawableTypeRequest request = getDrawableTypeRequest(config, requestManager);
    }
```

比如：设置图片伸缩
方法如下：

```
            / / 先拿到之前的配置信息。
           int scaleMode = config.getScaleMode();


            / / 然后根据配置信息去对 request 进行设置。
```

```
            switch (scaleMode) {
                case ScaleMode.CENTER_CROP:
                    request.centerCrop();
                    break;
                case ScaleMode.FIT_CENTER:
                    request.fitCenter();
                    break;
                default:
                    request.fitCenter();
                    break;
            }
```

其他的方法与此方法相同，就不在这做详细说明了。

总结

总结一下，其实主要思路就是在 GlobalConfig 中选择使用哪一个图片加载库，然后将使用者的所用设置信息保存在 SingleConfig 中，然后在具体的 Loader 中去实现，本文使用的 Glide，也可以增加 FrescoLoader，只需要实现 ILoader 然后实现 request()方法去通过 DrawableTypeRequest 设置就好了。

更多代码可以查询本人 GitHub：欢迎阅读，star 点起来。
Glide 二次封装库源码

看一下效果哦：

到这里我们的封装就结束了，就可以愉快的使用了，欢迎大家提出意见与建议。

Glide 二次封装库源码 欢迎点击 star

使用

在 gradle 中添加如下配置

```
compile 'com.libin.imageloader:ImageLoader:1.0.3'
```

在 Application 中:

```
    ImageLoader.init(getApplicationContext());
```

为了防止 oom,加入如下代码，清理内存：

```
@Override
public void onTrimMemory(int level) {
    super.onTrimMemory(level);

    ImageLoader.trimMemory(level);
}

@Override
public void onLowMemory() {
    super.onLowMemory();

    ImageLoader.clearAllMemoryCaches();
}
```