# Android glide url to bitmap

Starting uploading images with Glide is easy and in many cases requires only one line: Glide.with .load (myUrl) .into (imageView); You don't need to cancel loads anymore, it's also simple: Glide.with (fragment).clear (imageView); While it's good practice to clear the load you no longer need, you don't have to do it. In fact, Glide will automatically clear the load and recycle any resources used load when the activity or piece you pass in Glide.with () is destroyed. Set up queries. Glide offers a variety of options that can be applied to individual queries, including conversions, conversions, caching options, etc. Options can be shared across all queries with RequestOptions: RequestOptions sharedOptions - new Queries () .placeholder .fitCenter (); Glide.with .load .apply .apply .in (imageView1); Glide.with .load .apply .apply .in (imageView2); The Glide API can be expanded to include custom options that use the Glide API created for extended use cases. ListView and RecyclerView download images in ListView or RecyclerView use the same load line as they do when downloaded into a single form. Processes Glide Reuse View and automatically requests cancellation: @Override public void on BendViewHolder (ViewHolder holder, int position) - String URL and urls.get (position); Glide.with .load .url .into (holder.imageView); You don't need to cancel the URL check either, Glide will either clear the view or install any Drawable or Reverse Drawable you pointed out if the URL is zero. The only requirement for Glide is that any re-use of the view or purpose that you may have started to load in on the previous position either has a new loaded start into it or is clearly cleared through a clear () API. @Override public void on BendViewHolder (ViewHolder holder, int position) - if (isImagePosition (position)) - String URL and urls.get (position); Glide.with .load .url .into (holder.imageView); - Glide.with (snippet).clear (holder.imageView); holder.imageView.setImageDrawable By calling Clear () or (View) in The View, you cancel the load and ensure that Glide will not change the content of the view once the call is complete. If you forget to call clear () and don't start a new load, the load you started in the same view for the previous position may end after you install a special Drawable and change the content to the old image. While the examples we've shown here apply to RecyclerView, the same principles apply to ListView. Goals not viewed in addition to downloading Bitmaps and Drawables in views, you can start asynchronous loads in your own custom goals: Glide.with (context .load .url) .in (new CustomTarget)lt;Drawable))@Override public void onResourceReady (Drawable resource, Transition) -/ Do something with Drawable here. - @Override public void onLoadCleared (@Nullable Drawable placeholder) // Remove The Drawable provided in onResourceReady, from any views and make sure, / No references to it remains. So be sure to check out the Goals Documents page for more information. background streams Download images on background streams are also straight forward via send (int, int): FutureTarget'lt;futureTarget - Glide.with (context) .asBitmap () .load (url) .submit (width, height); Bitmap bitmap - futureTarget.get(); Do something with Bitmap and then when you're done with it: Glide.with (context).clear (futureTarget); Вы также можете начать асинхронные нагрузки на фоновые потоки так же, как вы бы на переднем плане потока, если вам не нужно Bitmap или Drawable на фоновом потоке себя: Glide.with (контекст) .asBitmap() .load (url) .into (новая цель<Bitmap>() . . . . . Примечание: Есть несколько библиотек, которые следуют лучшим практикам для загрузки изображений. Эти библиотеки можно использовать в приложении для загрузки изображений наиболее оптимизированным способом. Мы рекомендуем библиотеку Glide, которая загружает и отображает изображения как можно быстрее и плавно. Другие популярные библиотеки загрузки изображений включают Пикассо с площади, катушки из Instacart, и Фреско из Facebook. Эти библиотеки упрощают большинство сложных задач, связанных с бит-картуми и другими типами изображений на Android. Изображения бывают всех форм и размеров. Во многих случаях они больше, чем требуется для типичного пользовательского интерфейса приложения (UI). Например System Gallery displays photos taken using android camera devices that typically have a much higher resolution than the device's screen density. Given that you work with limited memory, ideally you want to download only a lower-resolution version in memory. The lower-resolution version should be the size of the user interface component that displays it. The higher-resolution image does not provide any visible benefit, but still takes up precious memory and carries additional overhead performance due to the extra on Scale. This lesson allows you to decode large bit cards without exceeding the app's memory limit, downloading a smaller version to your memory. Read Bitmap Dimensions and Type BitmapFactory Class provides several decoding methods (decodeByteArray),, decodeFile (), decodeResource () etc.) to create Bitmap from a variety of sources. Choose the most appropriate method of decoding based on the source of the image data. These methods try to highlight the memory for the built bit-card and, therefore, can easily lead to a zlt;/Bitmap/bitmap'gt; Exception. Each type of decoding method has additional signatures that allow you to specify decoding options through the BitmapFactory.Options class. Installing the inJustDecodeBounds property to the true, when deciphering avoids memory allocation, returning zero for the bitmap object, but setting out Videt, outHeight and outMimeType. This method allows you to read the size and type of image data before building (and distributing memory) of a bit card. Val Variants - BitmapFactory.Options ().Apply inJustDecodeBounds - truth - BitmapFactory.decodeResource (resources, R.id.myimage, options) val imageHeight: Int - options.outHeight val imageWidth: Int - options.outWidth val imageType: String options.inJustDecodeBound BitmapFactory.decodeResource (getResources), R.id.myimage, options); int imageHeight - options.outHeight; int imageWidth - options.outWidth; String imageType and options.outMimeType; To avoid java.lang.OutOfMemory exceptions, check the bit card sizes before deciphering if you don't trust the source to provide you with predictable-size image data that fits comfortably into your available memory. Download the smaller version in memory Now that the dimensions of the image are known, they can be used to decide whether to upload the full image to memory or instead a sub-company version should be downloaded. Here are some factors to consider: Estimated memory use to upload a full image to memory. The amount of memory you are willing to commit to downloading this image, given any other memory requirements of your app. The size of the target component of ImageView or the user interface to which the image should be uploaded. The size of the screen and the density of the current device. For example, you shouldn't upload a 1024x768 pixel image to memory if it ends up appearing in a 128x96 pixel miniature in ImageView. To tell the decoder to upload the image by uploading a smaller version to your memory, install inSampleSize to make it true in your BitmapFactory.Options object. For example, an image with a resolution of 2048x1536, which is deciphered with inSampleSize 4 produces a bit of a map of approximately 512x384. Loading this into memory uses 0.75MB, not 12MB for a full image (provided the configuration ARGB_8888). Here is a method for calculating the sample size value, which is two-figure based on target width and height: fun to calculateInSampleSize (options: BitmapFactory.Options, reqWidth: Int, reqHeight: Int: Int // Crude height and image width val (height: Int, width: Int) - options.run - outHeight to outWidth - var inSampleSize No 1 if (height of qgt; reqHeight) width of the qgt; reqWidth val halfHeight: Int - height / 2 half valdth inSampleSize, which is 2 and retains both a/ height and width greater than Height and width. (halfHeight/inSampleSize reqHeight - halfWidth / inSampleSize qgt; reqWidth) - inSampleSize No 2 - return inSampleSize - public static int calculateInSampleSize (bitmapFactory.Options, int reqWidth, int reqHeight) // Raw height and width of the image final inte. Final width int - options.outWidth; int inSampleSize No 1; If (height of the reqHeight - width of the reqWidth) final int halfWidth - width / 2; Calculate the greatest value of inSampleSize, which has power 2 and saves as / height and width more than the requested height and width. while (halfHeight /inSampleS zgt;) reqHeight (halfWidth/inSampleSize) qgt; reqWidth) - inSampleSize No 2; Note: Power in two values is calculated because the decoder uses the final value, rounding up to the nearest power of two, according to the documentation inSampleSize. To use this method, first decipher with inJustDecodeBounds set actually, pass the parameters to the end, and then decipher again using the new value inSampleSize and inJustDecodeBounds set false: fun decodeSampledBitmapFromResource (res: Resources, resId: Int, reqWidth: Int, reqHeight: Int : Bitmap / First decipher with inJustBoundDecode launch inJustDecodeBounds - true BitmapFactory.decodeResource (res, resId, this) / Calculate inSampleSize reqHeight) / Decode a bit card with inSampleSize set inJustcodeDecodeBounds - false BitmapFactory.deCodeResource (res, resId, it) - public static Bitmap decodeSampledBitFromResource (Resources res, in resId, int reqWidth, int reqHeight) / First decode with inJustDecodeBounds'true to check the size of the final BitmapFactory.Options options - new BitmapFactory.Options (); options.inJustDecodeBounds - the truth; BitmapFactory.decodeResource (res, resId, options); Calculate inSampleSize options.inSampleSize - calculateInSampleSize (options, reqWidth, reqHeight); Decode a bit card with inSampleSize options set.inJustDecodeBounds - false; return BitmapFactory.decodeResource (res, resId, options); This method makes it easy to download a bit of an arbitrarily large map into ImageView, which displays a miniature of 100x100 pixels, as shown in the following code example: imageView.setImageBitmap (decodeSampledBitmapFromResource (resources, R.id.myimage, 100, 100) ) imageView.setImageBitmap (decodeSampledBitFromResources(getResources),, R.id.myimage, 100, 100);; You can follow a similar process to decode bit cards from other sources, replacing the relevant BitmapFactory.decode method as needed. Need.