Android中图片的三级缓存：

1. 将图片临时保存到内存中，再从内存中加载出来

Java中对象的保存机制有四种：

强引用 Object obj = new Object(); 只要对象在使用，就不会被垃圾回收器回收

非强引用：

1. 软引用：只保证该软件能够正常运行
2. 弱引用：当发现内存不足，马上就回收
3. 虚引用：只要回收器一运行，马上就回收

将bitmap对象保存到内存中：如果内存中保存过多的bitmap对象，会发生OOM异常导致程序终止，所以创建一个软引用的集合，用于保存Bitmap图片对象，步骤：

**1.创建一个软引用的集合：private** Map<String ,SoftReference<Bitmap>> map ;

map = **new** HashMap<String, SoftReference<Bitmap>>();

**2.将bitmap对象保存到集合中**：在通过网络成功加载了一个bitmap图片的对象时，将该bitmap对象保存到map集合中：

//1.将图片保存到内存的集合中

map.put(url, **new** SoftReference<Bitmap>(bitmap));

从内存中读取数据：

**public** Bitmap getPictureFromCache(String url) {

System.***out***.println("当前正在从内存缓存中获取图片");

Bitmap bitmap = **null**;

SoftReference<Bitmap> softReference = map.get(url);

// 对引用对象进行是否为空的判断

**if**(softReference != **null**){

bitmap = softReference.get() ;

}

**return** bitmap;

}

1. 将图片保存到sd卡的文件中，然后从文件中读取出来

//2.将图片保存到sd卡的文件中，getFileName是一个通过Md5算法将url加密的方法，加密后的密文作为文件名

File bitmapFile = **new** File(dir,getFileName(url));

FileOutputStream fos = **new** FileOutputStream(bitmapFile);

// 将bitmap图片压缩到sd卡文件中，压缩成JPEG格式，100%压缩

bitmap.compress(CompressFormat.***JPEG***, 100, fos);

1. 通过网络从服务器加载图片，然后本地化
2. 连接服务器
3. 获取到图片对应的的输入流InputStream
4. 通过BitmapFactory将流中的数据转码成一个bitmap对象
5. 设置图片显示到界面上

代码：

HttpClient client = **new** DefaultHttpClient();

client.getParams().setParameter(HttpConnectionParams.***CONNECTION\_TIMEOUT***, 3000);

HttpGet httpGet = **new** HttpGet(url);

HttpResponse response = client.execute(httpGet);

**if**(response.getStatusLine().getStatusCode() == 200){

InputStream is = response.getEntity().getContent();

Bitmap bitmap = BitmapFactory.*decodeStream*(is);

Android图片缓存到内存的优化一：

LruCache是Android API 12之后提供的一个静态类，用于缓存图片，该类中拥有高级机制：如果某张图片使用的少，就从内存中移除，使用频率大的图片则保留在内存中的时间就越长

**int** maxSize = (**int**) (Runtime.*getRuntime*().maxMemory()/8) ;

**private** LruCache< String, Bitmap> lruCache = **new** LruCache<String, Bitmap>(maxSize){

//重写的sizeOf方法，可以不重写

**protected** **int** sizeOf(String key, Bitmap value) {

**return** value.getRowBytes() \* value.getHeight() / 1024;

};

};

maxSize参数：为当前应用设置的缓存空间的大小

//1.将图片保存到内存的集合中

从网络加载并获取了一张图片的bitmap对象后，直接put到lruCache中即可

lruCache.put(url, bitmap);

Android图片缓存到内存的优化二：

每次通过网络去加载图片时，都会去创建一个线程，通过在该线程中去完成连接网络、得到bitmap对象的操作，当时，如果需要加载的图片比较多时，则每次去开启一个线程加载的话，就会消耗大量的系统资源，有可能造成应用异常终止，所以通过创建一个线程池,d当每次需要通过一个线程去加载图片的时候，就去线程池里面直接拿一个已经创建好的并且当前可以使用的线程对象去完成加载图片的操作

**创建一个线程池对象：**

**private** ExecutorService newFixedThreadPool;

newFixedThreadPool = Executors.*newFixedThreadPool*(5);

参数5：指定创建好的线程池中拥有的线程的数量

直接调用线程池的execute方法，该方法会获取到该线程池中的一个线程对象，并通过获取到的线程对象执行相应的操作

newFixedThreadPool.execute(**new** GetInternetPictureTask(iv, url));

Android图片缓存到内存的优化三：

通过AsyncTask异步任务加载器去加载网络图片

AsyncTask类中实现了线程池、继承并实现了了Handler消息传递的类InternalHandler，所以直接使用该类封装好的方法直接调用即可，无需自己创建线程池和InternalHandler类，使用步骤：

1).创建一个类去继承AsyncTask，并实现AsyncTask中的onPreExecute、doInBackground、onPostExecute方法,并在doInBackground方法中去获取网络图片，onPostExecute方法在doInBackground方法之后执行，参数正好是doInBackground方法的返回值（返回一个BitmapImageBounding对象）：

ImageViewUrl类是自定义的用于将ImageView和图片资源的Url进行绑定的类

BitmapImageBounding类是自定义的用于将ImageView和通过网络获取到的Bitmap进行绑定的类

**private** **class** MyAsyncTask **extends**

AsyncTask<ImageViewUrl, Integer, BitmapImageBounding> {

@Override

**protected** **void** onPreExecute() {

**super**.onPreExecute();

}

@Override

**protected** BitmapImageBounding doInBackground(ImageViewUrl... params) {

String url = params[0].url;

ImageView iv = params[0].iv;

Bitmap bitmap = **null**;

**try** {

HttpClient client = **new** DefaultHttpClient(); client.getParams().setParameter( HttpConnectionParams.***CONNECTION\_TIMEOUT***, 3000); HttpGet httpGet = **new** HttpGet(url); HttpResponse response = client.execute(httpGet); **if** (response.getStatusLine().getStatusCode() == 200) {

InputStream is = response.getEntity().getContent(); bitmap = BitmapFactory.*decodeStream*(is); // 1.将图片保存到内存的集合中

lruCache.put(url, bitmap);

// 2.将图片保存到sd卡的文件中

File bitmapFile = **new** File(dir, getFileName(url)); FileOutputStream fos = **new** FileOutputStream(bitmapFile); // 将bitmap图片压缩到sd卡文件中

bitmap.compress(CompressFormat.***JPEG***, 100, fos); }

} **catch** (Exception e) {

e.printStackTrace();

}

**return** **new** BitmapImageBounding(iv, bitmap);

}

@Override

**protected** **void** onPostExecute(BitmapImageBounding result) {

**super**.onPostExecute(result); result.iv.setImageBitmap(result.bitmap);

}

}

执行异步任务：

要每次new出来一个继承了AsyncTask类的对象去执行execute方法，

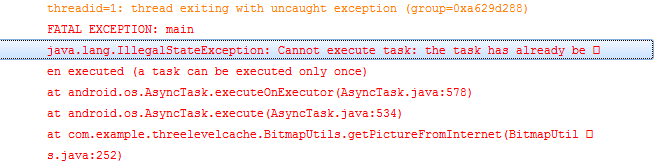
// 不能通过在类中创建一个全局变量，并每次通过该全局变量去执行execute方法，会报出如下的异常：

// Cannot execute task: the task has already been executed (a task can be executed only once)

**new** MyAsyncTask().execute(**new** ImageViewUrl(iv, url));

// 不能通过全局变量调用execute方法，否则会报出如下的异常：

// mAsyncTask.execute(new ImageViewUrl(iv, url));



将url和ImageView绑定的自定义类：

**private** **class** ImageViewUrl {

**private** ImageView iv;

**private** String url;

**public** ImageViewUrl(ImageView iv, String url) {

**super**();

**this**.iv = iv;

**this**.url = url;

}

}