原文链接：

http://docs.eoeandroid.com/training/displaying-bitmaps/process-bitmap.html

BitmapFactory 的 decode()方法，在 Load Large Bitmaps Efficiently 要点中进行讨论，不应该执行在主 UI 线程如果要读取源数据从磁盘或网络位置（或相对内存来说任何别的真实来源）.该数据需要加载的时间是不可预知的，并取决于多种因素（从磁盘或网络的读取速度，图像大小，CPU 的功率，等）.如果这些任务阻塞 UI 线程，系统标志您的应用程序无响应，用户可以选择关闭它响应（有关更多信息，请参阅 Designing for Responsiveness）.这节课将引导您通过在后台线程中使用 AsyncTask 处理位图，并告诉您如何处理并发问题.

使用一个异步任务

AsyncTask 类提供了一种简单的方式来在一个后台线程中执行许多任务，并且把结果反馈给 UI 线程.使用的方法是，创建一个继承与它的子类并且实现提供的方法.这里是一个使用 AsyncTask 和 decodeSampledBitmapFromResource()加载一个大图片到 ImageView 中的例子:

```
class BitmapWorkerTask extends AsyncTask {
    private final WeakReference imageViewReference;
    private int data = 0;

    public BitmapWorkerTask(ImageView imageView) {
        // Use a WeakReference to ensure the ImageView can be garbage collected
        imageViewReference = new WeakReference(imageView);
    }

    // Decode image in background.
    @Override
    protected Bitmap doInBackground(Integer... params) {
        data = params[0];
        return decodeSampledBitmapFromResource(getResources(), data, 100, 100));
    }

    // Once complete, see if ImageView is still around and set bitmap.
    @Override
    protected void onPostExecute(Bitmap bitmap) {
        if (imageViewReference null) {
            final ImageView imageView = imageViewReference.get();
            if (imageView != null) {
                imageView.setImageBitmap(bitmap);
            }
        }
    }
```

```
}
```

对于 ImageView 来说 WeakReference 确保那时 AsyncTask 并不会阻碍 ImageView 和任何它的引用被垃圾回收期回收. 不能保证 ImageView 在任务完成后仍然存在，所以你必须在 onPostExecute()方法中检查它的引用. ImageView 可能不再存在，如果例如，如果在任务完成之前用户退出了活动或者配置发生了变化. 为了异步地加载位图，简单地创建一个新的任务并且执行它:

```
public void loadBitmap(int resId, ImageView imageView) {
    BitmapWorkerTask task = new BitmapWorkerTask(imageView);
    task.execute(resId);
}
```

处理并发

常见的视图组件例如 ListView 和 GridView 如在上一节中当和 AsyncTask 结合使用时引出了另外一个问题. 为了优化内存，当用户滚动时这些组件回收了子视图. 如果每个子视图触发一个 AsyncTask，当它完成时没法保证，相关的视图还没有被回收时已经用在了别的子视图当中. 此外，还有异步任务开始的顺序是不能保证他们完成的顺序.

这篇文章透过 Multithreading for Performance 功能讨论处理并发，并且提供了一个当任务完成后 ImageView 将一个引用存储到后面能被检查的 AsyncTask 的解决方案. 使用类似的方法，从上一节的 AsyncTask 可以扩展到遵循类似的模式.

创建一个专用的 Drawable 的子类来存储一个引用备份到工作任务中. 在这种情况下，一个 BitmapDrawable 被使用以便任务完成后一个占位符图像可以显示在 ImageView 中:

```
static class AsyncDrawable extends BitmapDrawable {
    private final WeakReference bitmapWorkerTaskReference;

    public AsyncDrawable(Resources res, Bitmap bitmap,
            BitmapWorkerTask bitmapWorkerTask) {
        super(res, bitmap);
        bitmapWorkerTaskReference =
            new WeakReference(bitmapWorkerTask);
    }

    public BitmapWorkerTask getBitmapWorkerTask() {
        return bitmapWorkerTaskReference.get();
    }
}
```

```

```

执行 BitmapWorkerTask 前，你创建一个 AsyncDrawable，并将其绑定到目标 ImageView:

```
public void loadBitmap(int resId, ImageView imageView) {
    if (cancelPotentialWork(resId, imageView)) {
        final BitmapWorkerTask task = new BitmapWorkerTask(imageView);
        final AsyncDrawable asyncDrawable =
                new AsyncDrawable(getResources(), mPlaceHolderBitmap, task);
        imageView.setImageDrawable(asyncDrawable);
        task.execute(resId);
    }
}
```

如果别的正在运行的任务已经和这个 ImageView 关联，cancelPotentialWork 引用在上面的代码示例检查中.如果这样，它试图通过调用 cancel()取消先前的任务.在少数情况下，新的任务数据匹配现有的任务，而且并不需要做什么.下面是实现 cancelPotentialWork:

```
public static boolean cancelPotentialWork(int data, ImageView imageView) {
    final BitmapWorkerTask bitmapWorkerTask = getBitmapWorkerTask(imageView);

    if (bitmapWorkerTask != null) {
        final int bitmapData = bitmapWorkerTask.data;
        if (bitmapData != data) {
            // Cancel previous task
            bitmapWorkerTask.cancel(true);
        } else {
            // The same work is already in progress
            return false;
        }
    }
    // No task associated with the ImageView, or an existing task was cancelled
    return true;
}
```

一个帮助方法，getBitmapWorkerTask()，使用以上来检索一个和特定 ImageView 相关的任务:

```
private static BitmapWorkerTask getBitmapWorkerTask(ImageView imageView) {
   if (imageView != null) {
       final Drawable drawable = imageView.getDrawable();
       if (drawable instanceof AsyncDrawable) {
           final AsyncDrawable asyncDrawable = (AsyncDrawable) drawable;
```

```
            return asyncDrawable.getBitmapWorkerTask();
        }
    }
    return null;
}
```

这最后一步是在 BitmapWorkerTask 更新 onPostExecute()方法，以便任务取消时并且当前任务和这个 ImageView 关联时进行检查:

```
class BitmapWorkerTask extends AsyncTask {
    ...

    @Override
    protected void onPostExecute(Bitmap bitmap) {
        if (isCancelled()) {
            bitmap = null;
        }

        if (imageViewReference != null && bitmap != null ) {
            final ImageView imageView = imageViewReference.get();
            final BitmapWorkerTask bitmapWorkerTask =
                    getBitmapWorkerTask(imageView);
            if (this == bitmapWorkerTask && imageView != null) {
                imageView.setImageBitmap(bitmap);
            }
        }
    }
}
```

现在这个实现适合使用 ListView 和 GridView 控件组件以及回收其子视图的任何其他组件. 在你正常地给你的 ImageView 控件设置图片时简单地调用 loadBitmap 就行了.例如,在一个 GridView 中实现的方式是在支持的适配中的 android.view.View, android.view.ViewGroup) getView()方法中.

文章来源：http://wiki.eoe.cn/page/Processing_Bitmaps_Off_the_UI_Thread