标签: Bitmap LruCache DiskLruCache Imageloader

转载请注明：http://blog.csdn.net/feather_wch/article/details/79457395

> 主要包括三个部分：
>
> 1. Bitmap的高效加载
> 2. Bitmap的缓存
> 3. ImageLoader的实现
> 4. ListeView等列表的优化

Bitmap的加载和Cache

版本: 2018/3/15-1(14:41)

Bitmap的高效加载

1、Bitmap在Android中是什么?

> 指一张图片，可以是 `png` 或者 `jpg` 等常见的图片格式

2、Android中加载图片的四种方法

BitmapFactory提供的四类方法	作用	备注
decodeFile()	从 文件系统 加载Bitmap对象	间接调用 `decodeStream`
decodeResource()	从 资源 加载Bitmap对象	间接调用 `decodeStream`
decodeStream()	从 输入流 加载Bitmap对象	
decodeByteArray()	从 字节数组 加载Bitmap对象	

> 四类方法最终在 `Android底层` 实现，对应 `BitmapFactory` 类的几个 `native` 方法。

3、如何高效加载Bitmap?

> 1. 利用 `BitmapFactory.Options` 能通过一定 采样率 来加载缩小后的图片，并在 `ImageView` 中显示
> 2. 这样能 降低内存占用 并从一定程度上避免 `OOM`

4、BitmapFactory.Options如何使用?

> 1. 利用其参数 `inSampleSize`-采样率 进行缩小
> 2. `inSampleSize` 的值应该是2的指数，如1、2、4、8、16
> 3. `inSampleSize` 值为1时，大小为原始大小
> 4. `inSampleSize` 值为2时，宽高缩小为1/2，因此像素数为1/4

5、如何获取采样率?

> 1. 将 `BitmapFactory.Options` 的 `inJustDecodeBounds` 参数设置为 `true` 并加载图片
> (`inJustDecodeBounds=true` 时，只会解析图片原始信息，不会真正加载图片)
> 2. 从 `BitmapFactory.Options` 中取出 图片的原始宽高 ， 他们对应于 `outWidth` 和 `outHeight` 参数
> 3. 根据采样率规则并结合目标View所需大小计算出 采样率`inSampleSize`
> 4. 将 `BitmapFactory.Options` 的 `inJustDecodeBounds` 设置为 `false` ， 重新加载图片
>
> - 四大方法都支持类似的加载方法，但是 `decodeStream()` 有些特殊

```
//加载图片
mImageView.setImageBitmap(decodeSampledBitmapFromResource(getResources(), R.id.myimage, 100, 10

//采用合适采样率进行图片加载
public static Bitmap decodeSampledBitmapFromResource(Resources res, int resId, int reqWidth, in
    //1. 获取图片原始尺寸
    final BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true;
    BitmapFactory.decodeResource(res, resId, options);
    //2. 计算出采样率
    options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);
    //3. 用采样率重新加载图片
    options.inJustDecodeBounds = false;
    return BitmapFactory.decodeResource(res, resId, options);
}

public static int calculateInSampleSize(BitmapFactory.Options options, int reqWidth, int reqHei
    final int height = options.outHeight;
    final int width = options.outWidth;
    int inSampleSize = 1;

    if(height > reqHeight || width > reqWidth){
        final int halfHeight = height / 2;
        final int halfWidth = width / 2;
        while((halfHeight / inSampleSize) >= reqHeight && (halfWidth / inSampleSize) >= reqWidt
            inSampleSize *= 2;
        }
    }

    return inSampleSize;
}
```

Android中的缓存策略

6、缓存策略的思路?

> 1. 从网络上加载图片时，先从内存中去获取
> 2. 内存中没有时，再从存储设备中获取
> 3. 存储设备也没有时，才从网络上获取

7、缓存策略

> 1. 缓存策略主要包括 缓存的添加、获取和删除
> 2. 其中特殊的 删除 ， 主要是当 存储设备 容量满时，需要去删除一些 旧缓存 并添加 新缓存

8、缓存算法

> 1. 缓存算法决定如何删除哪份缓存
> 2. 常用缓存算法是 LRU(Least Recently Used) -优先淘汰最近最少使用的 缓存对象

3. 采用LRU的缓存算法 有两种： LruCache 和 DiskLruCache
4. LruCache 用于实现 内存缓存
5. DiskLruCache 用于实现 存储设备缓存

9、强引用、软引用和弱引用的区别

引用	介绍
强引用	直接的对象引用
软引用	当一个对象只有软引用时，系统 内存不足 是该对象会被 gc回收
弱引用	当一个对象只有弱引用时，此对象随时会被 gc回收

LruCache

10、LruCache的特点和作用

1. LruCache 是 Android 3.1 提供的类，低版本中需要使用 support-v4
2. LruCache 是一个 泛型类，它的内部采用一个 LinkedHashMap 以 强引用 的方式存储外界的 缓存对象
3. LruCache 提供 get()/put() 来完成 缓存 的获取和添加操作
4. LruCache 中缓存满时就会移除较早的 缓存对象
5. LruCache 是线程安全的

11、LruCache的典型初始化过程

```
//1. 获取最大内存数(单位KB)
int maxMemory = (int)(Runtime.getRuntime().maxMemory() / 1024);

//2. 总容量：缓存大小为 1/8
int cacheSize = maxMemory / 8;
LruCache<String, Bitmap> mMemoryCache = new LruCache<String, Bitmap>(cacheSize){
    @Override
    //3. 计算出缓存对象的大小(`单位`需要与`总容量单位`一致，也应该为KB，所以除以1024)
    protected int sizeOf(String key, Bitmap value) {
        //一行总字节数 x 高度 = 总字节数
        return bitmap.getRowBytes() * bitmap.getHeight() / 1024;
    }
};

//4. 获取缓存对象
mMemoryCache.get(key);

//5. 存储缓存对象
mMemoryCache.put(key, bitmap);

//6. 可以删除指定缓存对象
mMemoryCache.remove(key);
```

DiskLruCache

12、DiskLruCache是什么

> 1. DiskLruCache 用于实现存储设备缓存，即磁盘缓存---将缓存对象写入文件系统
> 2. DiskLruCache 并不是 Android SDK 的一部分需要自己去下载，JakeWharton的Github链接

13、DiskLruCahce的创建

> 1. 不能通过构造方法进行创建，需要通过open方法

```
 private static final long DISK_CACHE_SIZE = 1024 * 1024 * 50; //50MB

File diskCacheDir = getDiskCacheDir(this, "bitmap");
if(!diskCacheDir.exists()){
    diskCacheDir.mkdirs();
}

DiskLruCache diskLruCache = DiskLruCache.open(diskCacheDir, //参数1: 缓存目录
            1,  //参数2: 应用版本号，发生改变时会清空之前的数据，一般为1
            1,  //参数3: 单个节点数据的个数，一般为1
            DISK_CACHE_SIZE); //缓存总大小


//相关权限
<!--需要操作外部内存-->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<!--从网络下载图片-->
<uses-permission android:name="android.permission.INTERNET" />
```

14、 getDiskCacheDir(context, uniqueName)获取缓存路径并转为File

```
//通过文件夹名获取缓存路径(File形式)
public static File getDiskCacheDir(Context context, String uniqueName) {
    final String cachePath = Environment.MEDIA_MOUNTED.equals(Environment.getExternalStorageSta
            ? context.getExternalCacheDir().getPath()
            : context.getCacheDir().getPath();
    return new File(cachePath + File.separator + uniqueName);
}
```

> 1. 当 SD 卡存在或者 SD 卡不可被移除的时候，调用 getExternalCacheDir() 获取路径。
> 2. 否则调用 getCacheDir() 获取缓存路径
> 3. 前者获取的路径为： /storage/emulated/0/Android/data/<application package>/cache
> 4. 后者获取的路径为： /data/data/<application package>/cache

15、外部存储的两种类型

> 1. 临时性存储：应用卸载后，存储的数据也会被删除。 `context.getExternalCacheDir().getPath()`
> 2. 永久存储：应用被卸载后，存储的数据依旧存在。 `Environment.getExternalStorageDirectory().getAbsolutePath() + "/mDiskCache"` 能获取例如 `/storage/emulated/0/mDiskCache` 的路径。

16、getDiskCacheDir()只获取缓存地址String的实现方法

```
public String getDiskCacheDir(Context context) {
    String cachePath = null;
    if (Environment.MEDIA_MOUNTED.equals(Environment.getExternalStorageState())
            || !Environment.isExternalStorageRemovable()) {
            // 路径：/storage/emulated/0/Android/data/<application package>/cache
        cachePath = context.getExternalCacheDir().getPath();
    } else {
        // 路径：/data/data/<application package>/cache
        cachePath = context.getCacheDir().getPath();
    }
    return cachePath;
}
```

17、如何将图片Url转为对应的Key值

```
/**==========================================================
 * 需要获取图片Url对应的Key
 *  1. 图片的Url中可能有特殊字符，这会影响在Android中的使用
 *==========================================================*/
private String hashKeyFromUrl(String url){
    String cacheKey;
    final MessageDigest messageDigest;
    try {
        messageDigest = MessageDigest.getInstance("MD5");
        messageDigest.update(url.getBytes());
        cacheKey = bytesToHexString(messageDigest.digest());
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        cacheKey = String.valueOf(url.hashCode());
    }
    return cacheKey;
}
private String bytesToHexString(byte[] bytes){
    StringBuilder sb = new StringBuilder();
    for(int i = 0; i < bytes.length; i++){
        String hex = Integer.toHexString(0xFF & bytes[i]);
        if(hex.length() == 1){
            sb.append('0');
        }
        sb.append(hex);
    }
    return sb.toString();
}
```

18、DiskLruCache的缓存添加

```
//DiskLruCache的缓存添加
private static final int DISK_CACHE_INDEX = 0; //open中设置了一个节点只能有一个数据，因此为0

//1. 将url转为key
String key = hashKeyFromUrl(url);
//2. DiskLruCache的缓存添加的操作是通过Editor完成的
DiskLruCache.Editor editor = diskLruCache.edit(key);
if(editor != null){
    //3. 获取到输出流
    OutputStream outputStream = editor.newOutputStream(DISK_CACHE_INDEX); //一般为0
    //4. 将图片等数据通过`outputStream`将数据写入到文件系统中
    if (downloadUrlToStream(url, outputStream)) {
        //5. 需要通过editor的commit对写入操作进行提交
        editor.commit();
    } else {
        //6. 出现了错误可以回退整个操作
        editor.abort();
    }
    //7. 进行刷新
    diskLruCache.flush();
}


//downloadUrlToStream-通过url将文件写入到outputStream中
public boolean downloadUrlToStream(String urlString, OutputStream outputStream) {
        HttpURLConnection urlConnection = null;
        BufferedOutputStream out = null;
        BufferedInputStream in = null;

        try {
            final URL url = new URL(urlString);
            urlConnection = (HttpURLConnection) url.openConnection();
            in = new BufferedInputStream(urlConnection.getInputStream(), IO_BUFFER_SIZE);
            out = new BufferedOutputStream(outputStream, IO_BUFFER_SIZE);

            int b;
            while ((b = in.read()) != -1) {
                out.write(b);
            }
            return true;
        } catch (IOException e) {
            Log.e(TAG, "downloadBitmap failed." + e);
        } finally {
            if (urlConnection != null) {
                urlConnection.disconnect();
            }
            MyUtils.close(out);
            MyUtils.close(in);
        }
        return false;
}
```

19、DiskLruCache的缓存查找

```
Bitmap bitmap = null;
//1. url转为key
String key = hashKeyFromUrl(url);
//2. 获取Snapshot对象
DiskLruCache.Snapshot snapShot = diskLruCache.get(key);
if (snapShot != null) {
    //3. 通过Snapshot对象获取缓存的文件输入流
    FileInputStream fileInputStream = (FileInputStream)snapShot.getInputStream(DISK_CACHE_INDE
    FileDescriptor fileDescriptor = fileInputStream.getFD();
    //4. 通过文件输入流获取文件描述符，再通过BitmapFactory.decodeFileDescriptor来获取缩放后的图片
    bitmap = mImageResizer.decodeSampledBitmapFromFileDescriptor(fileDescriptor,
            reqWidth, reqHeight);
    if (bitmap != null) {
        addBitmapToMemoryCache(key, bitmap);//添加到内存中
    }
}
```

20、DiskLruCache的remove和delete

> 1. 用于磁盘缓存的删除操作

ImageLoader的实现

21、ImageLoader需要具备的特点

> 1. 图片的同步加载
> 2. 图片的异步加载
> 3. 图片压缩
> 4. 内存缓存
> 5. 磁盘缓存
> 6. 网络拉取

22、ListView或者GridView的View复用缺点

> 1. View复用 既是他们的 优点 也是其 缺点
>
> 错位现象：
>
> ItemA 正从网络加载图片，其对应的 ImageView为A 。用户迅速滑动列表，如果此时 ItemB 也 复用ImageView A 。此时 Item A 的图片下载好并赋给 ImageView A ， 此时会出现 ItemB 的ImageView却显示 ItemA 的图片

23、如何进行图片压缩?

> 通过 BitmapFactory.Options 的参数 inSampleSize-采样率 进行压缩处理

24、图片压缩类ImageResizer的实现

```
public class ImageResizer {
    private static final String TAG = "ImageResizer";

    public ImageResizer() {
    }

    public Bitmap decodeSampleBitmapFromResource(Resources res, int resId, int reqWidth, int re
        final BitmapFactory.Options options = new BitmapFactory.Options();
        options.inJustDecodeBounds = true;
        BitmapFactory.decodeResource(res, resId, options);
        options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);
        options.inJustDecodeBounds = false;
        return BitmapFactory.decodeResource(res, resId, options);
    }

    public Bitmap decodeSampledBitmapFromFileDescriptior(FileDescriptor fd, int reqWidth, int r
        final BitmapFactory.Options options = new BitmapFactory.Options();
        options.inJustDecodeBounds = true;
        BitmapFactory.decodeFileDescriptor(fd, null, options);
        options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);
        options.inJustDecodeBounds = false;
        return BitmapFactory.decodeFileDescriptor(fd, null, options);
    }

    /**
     * 从内存卡中加载图片
     */
    public static Bitmap decodeSampleBitmapFromFile(FileDescriptor fd,int reqWidth,int reqHeigh
        BitmapFactory.Options options = new BitmapFactory.Options();
        options.inJustDecodeBounds = true;
        BitmapFactory.decodeFileDescriptor(fd,null,options);
        options.inSampleSize = calculateInSampleSize(options,reqWidth,reqHeight);
        options.inJustDecodeBounds = false;

        return BitmapFactory.decodeFileDescriptor(fd,null,options);
    }

    private static int calculateInSampleSize(BitmapFactory.Options options, int reqWidth, int r
        if (reqWidth == 0 || reqHeight == 0) {
            return 1;
        }
        final int height = options.outHeight;
        final int width = options.outWidth;
        int inSampleSize = 1;
        if (height > reqHeight || width > reqWidth) {
            final int halfHeight = height / 2;
            final int halfWidth = width / 2;
            while ((halfHeight / inSampleSize) >= reqHeight && (halfWidth / inSampleSize) >= re
                inSampleSize *= 2;
            }
        }
        return inSampleSize;
    }
}
```

25、ImageLoader的具体实现(同步、异步、图片压缩、缓存)

```
public class ImageLoader {
    private Context mContext;
    //内存缓存
    private LruCache<String, Bitmap> mMemoryCache;
    //磁盘缓存
    private DiskLruCache mDiskLruCache;
    //磁盘缓存大小50MB
    private static final long DISK_CACHE_SIZE = 1024 * 1024 * 50;
    //磁盘缓存是否已经创建
    private boolean mIsDiskLruCacheCreated = false;
    //磁盘缓存中一个单位只有一个数据，因此为0
    private static final int DISK_CACHE_INDEX = 0;
    //IO缓存区尺寸为8MB
    private static final int IO_BUFFER_SIZE = 8 * 1024;
    //ImageView的Tag的key值，用于查询Url，随机使用一个ID的int值
    private static final int TAG_KEY_URI = R.id.remote_imageview; //XXXXXXXX
    //Handler发送消息前，用于构造Message
    private static final int MESSAGE_POST_result = 1;

    /**==============================
     *  1、创建内存缓存和磁盘缓存
     *==============================*/
    private ImageLoader(Context context){
        mContext = context.getApplicationContext();
        int maxMemory = (int)(Runtime.getRuntime().maxMemory() / 1024);
        //1. 缓存大小为 最大内存的1/8（单位MB）
        int cacheSize = maxMemory / 8;
        //2. 创建内存缓存
        mMemoryCache = new LruCache<String, Bitmap>(cacheSize){
            @Override
            protected int sizeOf(String key, Bitmap bitmap) {
                //3. 每个Bitmap的大小，单位MB
                return bitmap.getRowBytes() * bitmap.getHeight() / 1024;
            }
        };
        //4. 获取缓存目录
        File diskCacheDir = getDiskCacheDir(mContext, "bitmap");
        if(!diskCacheDir.exists()){
            diskCacheDir.mkdirs();
        }
        //5. 创建硬盘缓存(可用硬盘控件不足就不创建)
        if(getUsableSpace(diskCacheDir) > DISK_CACHE_SIZE){
            try {
                mDiskLruCache = DiskLruCache.open(diskCacheDir, 1, 1, DISK_CACHE_SIZE);
                mIsDiskLruCacheCreated = true;
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    /**==============================
     *  2、内存缓存的添加和获取
     *==============================*/
    //通过url获取Bitmap
```

```
private Bitmap loadBitmapFromMemCache(String url) {
    String key = hashKeyFromUrl(url);
    return getBitmapFromMemCache(key);
}
//通过key获取Bitmap
private Bitmap getBitmapFromMemCache(String key){
    return mMemoryCache.get(key);
}
//添加(key, bitmap)
private void addBitmapToMemoryCache(String key, Bitmap bitmap){
    //内存缓存中不存在该Bitmap才进行添加
    if(getBitmapFromMemCache(key) == null){
        mMemoryCache.put(key, bitmap);
    }
}

/**==============================
 *  3、磁盘缓存的存储和读取
 *  -从网络中获取图片并且保存到磁盘中
 *  -DiskLruCache的Editor完成存储操作
 *  -从磁盘缓存中读取Bitmap
 *==============================*/
private Bitmap loadBitmapFromHttp(String urlString,int reqWidth,int reqHeight) throws IOExc
    Log.i(TAG,"从网络中加载数据");

    if(Looper.myLooper() == Looper.getMainLooper()){
        throw new RuntimeException("不能从主线程中访问网络图片");
    }
    if(mDiskLruCache == null){
        return null;
    }
    //将url转换成key
    String key = hashKeyFromUrl(urlString);
    DiskLruCache.Editor editor = mDiskLruCache.edit(key);
    if(editor!=null){
        OutputStream outputStream = editor.newOutputStream(DISK_CACHE_INDEX);
        if(downloadUrlToStream(urlString,outputStream)){ //存储到本地文件夹中
            editor.commit();
        }else{
            editor.abort();
        }
    }
    mDiskLruCache.flush();
    return loadBitmapFromDiskCache(urlString, reqWidth, reqHeight);
}
//磁盘缓存的存储：将url的数据(图片)保存到OutputStream流中
private boolean downloadUrlToStream(String urlString, OutputStream outputStream){
    HttpURLConnection conn = null;
    BufferedInputStream in = null;
    BufferedOutputStream out = null;
    try {
        URL url = new URL(urlString);
        conn = (HttpURLConnection) url.openConnection();
        in = new BufferedInputStream(conn.getInputStream(),IO_BUFFER_SIZE);
        out = new BufferedOutputStream(outputStream,IO_BUFFER_SIZE);
```

```
            int b;
            while((b = in.read())!=-1){
                out.write(b);
            }
            return true;
        }catch (Exception e){
            e.printStackTrace();
        }finally {
            if(conn !=null)
                conn.disconnect();
            if(in!=null) {
                try {
                    in.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if(out!=null){
                try {
                    out.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return false;
    }
    //磁盘缓存的加载：从磁盘中获取Bitmap
    private Bitmap loadBitmapFromDiskCache(String urlString, int reqWidth, int reqHeight) throv
        Log.i(TAG,"从磁盘中加载数据");
        if(Looper.myLooper() == Looper.getMainLooper()){
            throw new RuntimeException("不能从主线程中加载图片");
        }
        if(mDiskLruCache == null){
            return null;
        }
        Bitmap bitmap = null;
        String key = hashKeyFromUrl(urlString);
        //1. 通过key值获取磁盘中的输入流
        DiskLruCache.Snapshot snapshot = mDiskLruCache.get(key);
        if(snapshot !=null){
            FileInputStream fileInputStream = (FileInputStream) snapshot.getInputStream(DISK_CA
            FileDescriptor descriptor = fileInputStream.getFD();
            //2. 通过写的一个ImageResizer高效加载图片
            bitmap = ImageResizer.decodeSampleBitmapFromFile(descriptor, reqWidth, reqHeight);
            if(bitmap !=null){
                //向缓存中加入图片
                addBitmapToMemoryCache(key,bitmap);
            }
        }
        return bitmap;
    }
    /**=====================================================
     *  4、同步加载
     *  -同步加载接口需要外部在线程中调用，因为可能会比较耗时
```

```
 *  -加载图片的顺序：
 *    缓存中加载图片->磁盘中加载图片->网络中加载图片
 *=====================================================*/
public Bitmap loadBitmap(String url,int reqWidth,int reqHeight){
    //1. 从缓冲中加载图片
    Bitmap bitmap = loadBitmapFromMemCache(url);
    if(bitmap!=null){
        return bitmap;
    }
    try {
        //2. 从磁盘中加载bitmap(不能在主线程中调用，会抛出异常)
        bitmap = loadBitmapFromDiskCache(url, reqWidth, reqHeight);
        if(bitmap !=null){
            return bitmap;
        }
        //3. 从网络中加载bitmap
        bitmap = loadBitmapFromHttp(url, reqWidth, reqHeight);
    } catch (IOException e) {
        e.printStackTrace();
    }
    if(bitmap == null && !mIsDiskLruCacheCreated){
        //如果bitmap为空，并且，磁盘缓存没有创建。那么通过url路径来获取bitmap
        bitmap = downloadBitmapFromUrl(url);
    }
    return bitmap;
}
/**
 * 通过url路径加载网络图片，直接返回bitmap。
 */
private Bitmap downloadBitmapFromUrl(String urlString) {
    Bitmap bitmap = null;
    HttpURLConnection conn = null;
    BufferedInputStream bis = null;
    try {
        URL url = new URL(urlString);
        conn = (HttpURLConnection) url.openConnection();
        bis = new BufferedInputStream(conn.getInputStream(),IO_BUFFER_SIZE);
        bitmap = BitmapFactory.decodeStream(bis);
    } catch (Exception e) {
        e.printStackTrace();
    }finally {
        if(conn!=null)
            conn.disconnect();
        if(bis!=null){
            try {
                bis.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    return bitmap;
}
/**=================================================
 *  5、异步加载
```

```
     *  -先从缓存中加载bitmap，如果存在就直接返回，并设置图片。
     *  -不存在，就会在线程池中调用loadBitmap方法，来更新UI
     *  * 主线程Handler中会额外对 ListViewd等列表的错位问题进行处理
     *=======================================================*/
    public void bindBitmap(final String url, final ImageView imageView, final int reqWidth,
                           final int reqHeight){
        //0. 给ImageView设置Tag，用于异步加载后，解决列表的错位问题
        imageView.setTag(TAG_KEY_URI, url);
        //1. 内存缓存中读取
        Bitmap bitmap = loadBitmapFromMemCache(url);
        if(bitmap!=null){
            //2. 存在则直接设置
            imageView.setImageBitmap(bitmap);
            return;
        }
        //3. 不存在就在线程池中调用loadBitmap()方法进行加载
        Runnable loadBitmapTask = new Runnable() {
            @Override
            public void run() {
                Bitmap bitmap = loadBitmap(url,reqWidth,reqHeight);
                if(bitmap!=null){
                    //4. 将图片、图片地址、bitmap封装为LoaderResult
                    LoaderResult result = new LoaderResult(imageView,url,bitmap);
                    //5. 给主线程发送消息
                    mMainHandler.obtainMessage(MESSAGE_POST_result, result).sendToTarget();
                }
            }
        };
        //6. 线程池中执行
        THREAD_POOL_EXECUTOR.execute(loadBitmapTask);
    }
    //1-图片结果类
    private class LoaderResult {
        public ImageView imageView;
        public String url;
        public Bitmap bitmap;
        public LoaderResult(ImageView imageView, String url, Bitmap bitmap) {
            this.bitmap = bitmap;
            this.imageView = imageView;
            this.url = url;
        }
    }
    //cpu核心数
    private static final int CPU_COUNT = Runtime.getRuntime().availableProcessors();
    //cpu核心线程数
    private static final int CORE_POOL_SIZE = CPU_COUNT +1;
    //cpu最大线程数
    private static final int MAXIMUM_POOL_SIZE = CPU_COUNT * 2 + 1;
    //线程超时时长
    private static final long KEEP_ALIVE = 10L;
    //2-线程工厂
    private static final ThreadFactory sThreadFactory = new ThreadFactory() {
        private final AtomicInteger mCount = new AtomicInteger(1);

        @Override
```

```
    public Thread newThread(Runnable r) {
        return new Thread(r,"imageLoader#"+mCount.getAndIncrement());
    }
};
//3-线程池
public static final Executor THREAD_POOL_EXECUTOR
        = new ThreadPoolExecutor(   CORE_POOL_SIZE,
                                    MAXIMUM_POOL_SIZE,
                                    KEEP_ALIVE,
                                    TimeUnit.SECONDS,
                                    new LinkedBlockingDeque<Runnable>(),
                                    sThreadFactory);
//4-主线程Handler
private Handler mMainHandler = new Handler(Looper.getMainLooper()){
    @Override
    public void handleMessage(Message msg) {
        LoaderResult result = (LoaderResult) msg.obj;
        ImageView iv = result.imageView;
        Bitmap bitmap = result.bitmap;
        //1. 获取ImageView的tag()-bindBitmap中设置
        String getUrl = (String) iv.getTag(TAG_KEY_URI);
        //2. 比较加载结果的Url与ImageView的Url
        if(getUrl.equals(result.url)){
            //3. 相等表示没有错位
            iv.setImageBitmap(bitmap);
        }else{
            Log.i(TAG,"set image bitmap,but url has changed");
        }
    }
};

/**====================================
 * 获取缓存路径(File形式)
 *====================================*/
public static File getDiskCacheDir(Context context, String uniqueName) {
    final String cachePath = Environment.MEDIA_MOUNTED.equals(Environment.getExternalStorag
            ? context.getExternalCacheDir().getPath()
            : context.getCacheDir().getPath();
    return new File(cachePath + File.separator + uniqueName);
}

/**====================================
 * 获取磁盘中可用的空间
 *====================================*/
private long getUsableSpace(File path) {
    if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.GINGERBREAD){
        return path.getUsableSpace();
    }
    final StatFs stats = new StatFs(path.getPath());
    return stats.getBlockSize()*stats.getAvailableBlocks();
}

/**====================================
 * 将url转换为hash key
 *====================================*/
```

```
    private String hashKeyFromUrl(String url){
        String cacheKey;
        try {
            final MessageDigest mDigest = MessageDigest.getInstance("MD5");
            mDigest.update(url.getBytes());
            cacheKey = bytesToHexString(mDigest.digest());
        } catch (NoSuchAlgorithmException e) {
            cacheKey = String.valueOf(url.hashCode());
        }
        return cacheKey;
    }
    private String bytesToHexString(byte[] digest) {
        StringBuilder sb = new StringBuilder();
        for(int i=0; i<digest.length; i++){
            String hex = Integer.toHexString(0xFF & digest[i]);
            if(hex.length() == 1){
                sb.append("0");
            }
            sb.append(hex);
        }
        return sb.toString();
    }
}
```

26、ImageLoader如何解决ListView/GridView的错位问题?

> 1. 异步加载 开始前，给 ImageView 设置Tag，将 图片对应的Url 传入
> 2. 异步加载 完成后，比较加载的 图片Url 和 显示图片的ImageView 的 Url 是否相匹配。匹配就正常加载，不匹配就不加载，防止错位。

27、ImageLoader为什么不使用 AsyncTask 或者 线程 而采用 线程池 ?

> 1. 普通 线程 加载图片，随着列表滑动会产生大量线程，效率低下。
> 2. AsyncTask 封装了 线程池和handler 但是 AsyncTask 在不同版本中效果不同。
> 3. 3.0以上AsyncTask 无法实现 并发效果 ， 不满足需求。
> 4. 通过 AsyncTask 的 executeOnExecutor 或者改造 AsyncTask 来达到并发效果，过于繁琐而无太多意义。
>
>   结论：因此通过 线程池和Handler 来实现 ImageLoader 的并发能力和访问UI是最合适的方法。

ListView等列表的优化

28、ListView等列表遇到卡顿的解决办法

> 1. ListView等列表 滑动过程中 不进行图片的加载 ， 停止滑动后才进行 图片加载
> 2. 部分特殊情况下，需要 开启硬件加速，通过给 Activity 设置 android:hardwareAccelerated="true" 可以开启 硬件加速

29、ListView解决卡顿问题的实现方法

```
//1. 给ListView、GridView设置滑动监听器
mGridView.setOnScrollListener(new AbsListView.OnScrollListener() {
    @Override
    public void onScrollStateChanged(AbsListView view, int scrollState) {
       //1. 停止滑动时，告知Adpater已经停止滑动
        if(scrollState == AbsListView.OnScrollListener.SCROLL_STATE_IDLE){
            mImageAdapter.setmIsStopScroll(true);
            mImageAdapter.notifyDataSetChanged();
       //2. 滑动时不进行图片加载
        }else{
            mImageAdapter.setmIsStopScroll(false);
        }
    }
    @Override
    public void onScroll(AbsListView view, int firstVisibleItem, int visibleItemCount, int tota
    }
});
```

```
//2. Adpater的getView中进行特殊处理
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    ViewHolder holder = null;
    if(convertView == null){
        convertView = mInflater.inflate(R.layout.gridview_item, parent, false);
        holder = new ViewHolder();
        holder.imageView = (ImageView)convertView.findViewById(R.id.image);
        convertView.setTag(holder);
    }else{
        holder = (ViewHolder)convertView.getTag();
    }
    ImageView imageView = holder.imageView;
    final String tag = (String) imageView.getTag();
    final String url = (String) getItem(position);
    //1、图片已经发生了变化，使用默认图片
    if(!url.equals(tag)){
        imageView.setImageResource(R.drawable.ic_launcher);
    }
    //2、停止滚动时加载图片
    if(mIsStopScroll){
        imageView.setTag(url);
        mImageLoader.bindBitmap((String)getItem(position), holder.imageView, 200, 200);
    }
    return convertView;
}
//3. 设置是否滑动的标志
Boolean mIsStopScroll = true;
public void setmIsStopScroll(Boolean mIsStopScroll) {
    this.mIsStopScroll = mIsStopScroll;
}
```

参考资料

1. Andorid使用磁盘缓存DiskLruCache