DiskLruCache并非安卓sdk源码中的一部分，那么需要自己去下载的。

一、DiskLruCache的创建

DiskLruCache的创建使用open方法来创建自身，

```
mDiskLruCache = DiskLruCache.open(diskCacheDir, 1, 1,DISK_CACHE_SIZE);
```

参数1：缓存的位置，如果需要应用卸载后删除缓存文件，那么就选择sd卡上的缓存目录（/sdcard/Android/data/自己的包名/cache。），如果希望卸载之后任然保留的花那么就选择sd卡上的

其他目录。

参数2：表示应用的版本号。一般设置为1就可以了。

参数3：表示单个节点所对应的数据的个数。一般设置为1就可以了。(**表示一个节点只能有一个数据**)

参数4：表示缓存的总大小，比如设置为50MB，那么超过这个大小后，DiskLruCache会清除一些缓存从而保证大小不超过这个设定值。

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

```
File diskCacheDir = getDiskCacheDir(mContext, "bitmap");
if (!diskCacheDir.exists()) {
    diskCacheDir.mkdirs();
}
if (getUsableSpace(diskCacheDir) > DISK_CACHE_SIZE) {
    try {
        mDiskLruCache = DiskLruCache.open(diskCacheDir, 1, 1,DISK_CACHE_SIZE);
        mIsDiskLruCacheCreated = true;
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
public File getDiskCacheDir(Context context, String uniqueName) {
    boolean externalStorageAvailable = Environment
            .getExternalStorageState().equals(Environment.MEDIA_MOUNTED);
    final String cachePath;
    if (externalStorageAvailable) {
        cachePath = context.getExternalCacheDir().getPath();
    } else {
        cachePath = context.getCacheDir().getPath();
    }

    return new File(cachePath + File.separator + uniqueName);
}
```

二、DiskLruCache的缓存添加

DiskLruCache的缓存添加时通过DiskLruCache.Editor完成的。DiskLruCache.Editor表示一个缓存对象的编辑对象。

拿图片缓存举例子。**首先需要通过获取图片的url所对应的key，然后根据Key就可以通过edit()来获取DiskLruCache.Editor对象了。如果这个缓存正在被编辑，那么edit()就会返回null.*之所以把url转换成key是因为图片的url里面可能有特殊字符，这回影响到URL在Android中的使用。一般采用URL的Md5值作为key.***

1、将URL使用MD5的方式转换成key。

```
    private String hashKeyFormUrl(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[] 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();
    }
```

2、缓存编辑对象Editor的创建，写入，回滚和刷新。

```
String key = hashKeyFormUrl(url);
DiskLruCache.Editor editor = mDiskLruCache.edit(key);                    1.Editor对象的创建
if (editor != null) {
    OutputStream outputStream = editor.newOutputStream(DISK_CACHE_INDEX);    2、通过editor对象可以获取到一个文件输出流，DISK_CACHE_INDEX=0，是因为在创建DiskLruCache对象的时候第三个参数设置为一个节点只能有一个数据，那么这一个的数据的位置就是为0
    if (downloadUrlToStream(url, outputStream)) {
        editor.commit();          3、必须通过editor.commit()来提交写入操作
    } else {
        editor.abort();          4、如果没有写入成功，那么久通过editor.abort()进行回滚操作
    }
    mDiskLruCache.flush();       5、别忘记刷新了
}
```

```
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;
}
```

三、DiskLruCache的缓存查找

和缓存添加类似，缓存的查找也需要将URL转换成Key，然后通过get方法得到一个Snapshot对象。然后通过Snapshot对象得到缓存的文件输入流。有了文件输入流就可以得到Bitmap对象了。

1、使用BitmapFactory.Options的decodeStream加载缩放图片存在问题

但是为了避免加载图片的时候导致OOM，一般不建议直接加载原始图片。前面提到了使用BitmapFactory.Options的方式缩放原始图片，**但是这种方式会对FileInputStream的缩放存在问题，原因是FileInputStream是一种有序的文件流，而两次的decodeStream调用影响了文件的位置属性导致第二次的decodeStream时得到的时Null**。

为了解决这个问题，可以通过文件流来得到对应的文件描述符，然后通过BitmapFactory.decodeFileDescriptor()方法来加载一张缩放图片。

2、缓存查找对象Snapshot对象的获取

```
String key = hashKeyFormUrl(url);
DiskLruCache.Snapshot snapShot = mDiskLruCache.get(key);
```

3、通过Snapshot对象获取文件输入流，并使用文件描述符的方式缩放图片

```
Bitmap bitmap = null;
String key = hashKeyFormUrl(url);
DiskLruCache.Snapshot snapShot = mDiskLruCache.get(key);
if (snapShot != null) {
    FileInputStream fileInputStream = (FileInputStream)snapShot.getInputStream(DISK_CACHE_INDEX);
    FileDescriptor fileDescriptor = fileInputStream.getFD();
    bitmap = mImageResizer.decodeSampledBitmapFromFileDescriptor(fileDescriptor,
            reqWidth, reqHeight);
    if (bitmap != null) {
        addBitmapToMemoryCache(key, bitmap);
    }
}
```

```
public Bitmap decodeSampledBitmapFromFileDescriptor(FileDescriptor fd, int reqWidth, int reqHeight) {
    // First decode with inJustDecodeBounds=true to check dimensions
    final BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true;
    BitmapFactory.decodeFileDescriptor(fd, null, options);

    // Calculate inSampleSize
    options.inSampleSize = calculateInSampleSize(options, reqWidth,
            reqHeight);

    // Decode bitmap with inSampleSize set
    options.inJustDecodeBounds = false;
    return BitmapFactory.decodeFileDescriptor(fd, null, options);
}
```

4、将缓存对象添加到内存中

```
private void addBitmapToMemoryCache(String key, Bitmap bitmap) {
    if (getBitmapFromMemCache(key) == null) {
        mMemoryCache.put(key, bitmap);
    }
}
```

四、DiskLruCache缓存对象的删除

1、删除整个存储设备缓存

```
/**
 * Recursively delete everything in {@code dir}.
 */
// TODO: this should specify paths as Strings rather than as Files
public static void deleteContents(File dir) throws IOException {
    File[] files = dir.listFiles();
    if (files == null) {
        throw new IllegalArgumentException("not a directory: " + dir);
    }
    for (File file : files) {
        if (file.isDirectory()) {
            deleteContents(file);
        }
        if (!file.delete()) {
            throw new IOException("failed to delete file: " + file);
        }
    }
}
```

```
/**
 * Closes the cache and deletes all of its stored values. This will delete
 * all files in the cache directory including files that weren't created by
 * the cache.
 */
public void delete() throws IOException {
    close();
    deleteContents(directory);
}
```

2、删除指定Key的缓存对象

```java
/**
 * Drops the entry for {@code key} if it exists and can be removed. Entries
 * actively being edited cannot be removed.
 *
 * @return true if an entry was removed.
 */
public synchronized boolean remove(String key) throws IOException {
    checkNotClosed();
    validateKey(key);
    Entry entry = lruEntries.get(key);
    if (entry == null || entry.currentEditor != null) {
        return false;
    }

    for (int i = 0; i < valueCount; i++) {
        File file = entry.getCleanFile(i);
        if (!file.delete()) {
            throw new IOException("failed to delete " + file);
        }
        size -= entry.lengths[i];
        entry.lengths[i] = 0;
    }

    redundantOpCount++;
    journalWriter.append(REMOVE + ' ' + key + '\n');
    lruEntries.remove(key);

    if (journalRebuildRequired()) {
        executorService.submit(cleanupCallable);
    }

    return true;
}
```