## Imageloader

### 常用的图片处理框架

Fresco 正在实现了三级缓存，2级内存缓存，一级本地缓存、Glide、Picasso、UIL-ImageLoader

### 图片占用内存

大小 = 每个像素占用的字节数 \* 总像素

Bitmap.Config

* RGB\_565
* ARGB\_8888

内存溢出

### 图片乱序错位问题

imageview.setTag(url)

弱引用双向关联

使用volley的NetImageView

### 四种引用类型

* 强引用
* 弱引用
* 软引用
* 虚引用

### 三级缓存

1、DiskLrucache 本地/磁盘缓存

2、Lrucache 内存缓存

原理

LinkedHashMap 双向循环链表

* 参数1：初始大小
* 参数2：装载因子
* 参数3：true 按访问顺序排序，false 按插入顺序排序

内存大小

int memory = (int) (Runtime.getRuntime().maxMemory()/8);  
LruCache<String,Bitmap> cache = new LruCache<String,Bitmap>(memory){  
 @Override  
 protected int sizeOf(String key, Bitmap value) {  
 return value.getByteCount();  
 }  
};

Lrucache .get(key) --> LinkedHashMap.get(key) : 命中，移动到双向循环链表的的尾部

Lrucache .put(key) --> LinkedHashMap.put(key) : 如果size > maxSize，删除链表header节点直到size < maxSize

3、 网络缓存

4、缓存路径

* getExternalCacheDir()
* getCacheDir()

### 图片压缩

BitmapFactory.Options

* inJustDecodeBounds
* inSampleSize
* outWidth
* outHeight
* inBitmap

/\*\*  
 \* Decode and sample down a bitmap from resources to the requested width and height.  
 \*  
 \* @param res The resources object containing the image data  
 \* @param resId The resource id of the image data  
 \* @param reqWidth The requested width of the resulting bitmap  
 \* @param reqHeight The requested height of the resulting bitmap  
 \* @param cache The ImageCache used to find candidate bitmaps for use with inBitmap  
 \* @return A bitmap sampled down from the original with the same aspect ratio and dimensions  
 \* that are equal to or greater than the requested width and height  
 \*/  
 public static Bitmap decodeSampledBitmapFromResource(Resources res, int resId,  
 int reqWidth, int reqHeight, ImageCache cache) {  
  
 // BEGIN\_INCLUDE (read\_bitmap\_dimensions)  
 // First decode with inJustDecodeBounds=true to check dimensions  
 final BitmapFactory.Options options = new BitmapFactory.Options();  
 options.inJustDecodeBounds = true;  
 BitmapFactory.decodeResource(res, resId, options);  
  
 // Calculate inSampleSize  
 options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);  
 // END\_INCLUDE (read\_bitmap\_dimensions)  
  
 // If we're running on Honeycomb or newer, try to use inBitmap  
 if (Utils.hasHoneycomb()) {  
 addInBitmapOptions(options, cache);  
 }  
  
 // Decode bitmap with inSampleSize set  
 options.inJustDecodeBounds = false;  
 return BitmapFactory.decodeResource(res, resId, options);  
 }  
  
 /\*\*  
 \* Decode and sample down a bitmap from a file to the requested width and height.  
 \*  
 \* @param filename The full path of the file to decode  
 \* @param reqWidth The requested width of the resulting bitmap  
 \* @param reqHeight The requested height of the resulting bitmap  
 \* @param cache The ImageCache used to find candidate bitmaps for use with inBitmap  
 \* @return A bitmap sampled down from the original with the same aspect ratio and dimensions  
 \* that are equal to or greater than the requested width and height  
 \*/  
 public static Bitmap decodeSampledBitmapFromFile(String filename,  
 int reqWidth, int reqHeight, ImageCache cache) {  
  
 // First decode with inJustDecodeBounds=true to check dimensions  
 final BitmapFactory.Options options = new BitmapFactory.Options();  
 options.inJustDecodeBounds = true;  
 BitmapFactory.decodeFile(filename, options);  
  
 // Calculate inSampleSize  
 options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);  
  
 // If we're running on Honeycomb or newer, try to use inBitmap  
 if (Utils.hasHoneycomb()) {  
 addInBitmapOptions(options, cache);  
 }  
  
 // Decode bitmap with inSampleSize set  
 options.inJustDecodeBounds = false;  
 return BitmapFactory.decodeFile(filename, options);  
 }  
  
 /\*\*  
 \* Decode and sample down a bitmap from a file input stream to the requested width and height.  
 \*  
 \* @param fileDescriptor The file descriptor to read from  
 \* @param reqWidth The requested width of the resulting bitmap  
 \* @param reqHeight The requested height of the resulting bitmap  
 \* @param cache The ImageCache used to find candidate bitmaps for use with inBitmap  
 \* @return A bitmap sampled down from the original with the same aspect ratio and dimensions  
 \* that are equal to or greater than the requested width and height  
 \*/  
 public static Bitmap decodeSampledBitmapFromDescriptor(  
 FileDescriptor fileDescriptor, int reqWidth, int reqHeight, ImageCache cache) {  
  
 // First decode with inJustDecodeBounds=true to check dimensions  
 final BitmapFactory.Options options = new BitmapFactory.Options();  
 options.inJustDecodeBounds = true;  
 BitmapFactory.decodeFileDescriptor(fileDescriptor, null, options);  
  
 // Calculate inSampleSize  
 options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);  
  
 // Decode bitmap with inSampleSize set  
 options.inJustDecodeBounds = false;  
  
 // If we're running on Honeycomb or newer, try to use inBitmap  
 if (Utils.hasHoneycomb()) {  
 addInBitmapOptions(options, cache);  
 }  
  
 return BitmapFactory.decodeFileDescriptor(fileDescriptor, null, options);  
 }  
 /\*\*  
 \* Calculate an inSampleSize for use in a {@link android.graphics.BitmapFactory.Options} object when decoding  
 \* bitmaps using the decode\* methods from {@link android.graphics.BitmapFactory}. This implementation calculates  
 \* the closest inSampleSize that is a power of 2 and will result in the final decoded bitmap  
 \* having a width and height equal to or larger than the requested width and height.  
 \*  
 \* @param options An options object with out\* params already populated (run through a decode\*  
 \* method with inJustDecodeBounds==true  
 \* @param reqWidth The requested width of the resulting bitmap  
 \* @param reqHeight The requested height of the resulting bitmap  
 \* @return The value to be used for inSampleSize  
 \*/  
 public static int calculateInSampleSize(BitmapFactory.Options options,  
 int reqWidth, int reqHeight) {  
 // BEGIN\_INCLUDE (calculate\_sample\_size)  
 // Raw height and width of image  
 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;  
  
 // Calculate the largest inSampleSize value that is a power of 2 and keeps both  
 // height and width larger than the requested height and width.  
 while ((halfHeight / inSampleSize) > reqHeight  
 && (halfWidth / inSampleSize) > reqWidth) {  
 inSampleSize \*= 2;  
 }  
  
 // This offers some additional logic in case the image has a strange  
 // aspect ratio. For example, a panorama may have a much larger  
 // width than height. In these cases the total pixels might still  
 // end up being too large to fit comfortably in memory, so we should  
 // be more aggressive with sample down the image (=larger inSampleSize).  
  
 long totalPixels = width \* height / inSampleSize;  
  
 // Anything more than 2x the requested pixels we'll sample down further  
 final long totalReqPixelsCap = reqWidth \* reqHeight \* 2;  
  
 while (totalPixels > totalReqPixelsCap) {  
 inSampleSize \*= 2;  
 totalPixels /= 2;  
 }  
 }  
 return inSampleSize;  
 // END\_INCLUDE (calculate\_sample\_size)  
 }

## EventBus

### 1. 事件总线

组件（activity，fragment，service）间的交互，通信，主线程子线程间的通信

观察者模式，发布订阅，注解，Map

事件类型EventType，事件响应函数

### 2. 发布接收事件

* 发布事件post()

EventBus.getDefault().post(new User("mr.simple"), "my\_tag");

反射调用，回调

* 注册事件register()

扫描订阅对象的所有方法，包括分类的方法，匹配方法，保存到Map集合中

* 接收事件，事件响应函数

@Subscriber(tag = "my\_tag", mode=ThreadMode.POST)

### 3. EventBus

订阅队列

Map

判断是否是主线程：Looper.getMainLooper() == Looper.myLooper();

Handler mHandler = new Handler(Looper.getMainLooper());

ThreadLocal<PostingThreadState>

存储了⼀个事件队列以及事件的状态

class PostingThread  
{  
List<Object> mEventQueue = new ArrayList<Object>();  
boolean isMainThread;  
boolean isPosting;  
}

Map<Class, CopyOnWriteArrayList<SubscribeMethod>>

通过订阅对象的字节码拿到订阅对象匹配的方法以及方法的参数，其中threadmode是通过截取字符串获得，EventBus3.0是通过注解的方式拿到threadmode，方法的参数就是EventType

## Android EventBus

### EventType

方法的参数和tag组成EventType

## 依赖注入IOC

## ImageLoader

面向接口编程，策略模式，BitmapRequest，链式调用 return this Builder模式，单例模式，生产者消费者模式

模板方法模式（算法骨架）

### BitmapRequest

图片请求

### ImageLoaderConfig

设置一些基本的东西，比如加载中的图片、加载失败的图片、缓存策略

### RequestQueue

内部维持着一个PriorityBlockingQueue

BlockingQueue<BitmapRequest> mRequestQueue = new PriorityBlockingQueue<BitmapRequest>();

### RequestDispatcher

## HttpUtils

### 常用的网络请求框架

* Retrofit
* OkHttp
* NoHttp
* Volley
* HttpURLConnection（省电省流量）
* HttpClient（6.0后删除）
* AsyncHttpclient

请求头Map mHeaders，请求参数Map mBodyParams，URLEncoder.encode()

* NetworkExecutor 网络请求线程
* HttpStack Http执行器
* ResponseDelivery Response分发

### Http协议

请求Request

* 请求首行
* 请求头
* 空行
* 请求体

响应Response

* 响应首行
* 响应头
* 空行
* 响应体

请求头

响应头

通用头

ContentType

* get 查
* post 改
* put 增 提交表单
* delete 删
* head 只返回首部
* trace 诊断，跟踪http请求是否被修改
* options 请求服务器告知其支持的各种功能

### 网络模型

TCP/IP参考模型

* 应用层 http，https，ftp
* 传输层 tcp udp
* 网络层 ip地址
* 物理层
* 数据链路层

### 三次、四次握手

### 三要素

ip地址，端口号，协议

### 文件上传

表单数据，Content-Type:multipart/form-data

## ORM框架

注解（运行时注解或编译时注解）反射

注解标记表和表中的列，javabean上添加注解，指定一个类对应的表名，一个字段在表中对应的列名

在清单文件中配置数据库名，版本号，还可以配置model

model类可以从清单文件中获取，也可以通过扫描整个应用获取，扫描当前App的dex文件

创建表：通过注解标记表名和字段名，通过反射拿到model的信息来拼接sql语句

### 数据库操作类

* Selection
* Delete
* Update
* Insert