### 0，Okhttp相关

操作：

* 创建一个OkhttpClient对象
* 然后创建一个request对象，通过内部类Builder调用生成的Request对象，对象携带请求参数
* 创建一个Call对象，调用execute（同步，线程阻塞）/enqueue（异步）（回调方法是运行在子线程中，记住无法更新UI，得通过handler）

优点：

1，Okhttp的最大并发量为64，比volley的只有4个强很多倍

2，使用连接池技术，支持5个并发的socket连接，默认keepAlive时间为5分钟，解决TCP握手挥手的效率问题，减少握手次数

3，支持Gzip压缩，且操作对用户透明，可以通过header设置，在发起请求的时候自动加入header，Accept-Encoding:gzip,而我们的服务器返回的时候header中有Content-Encoding:gzip

4，利用响应缓存来避免重复的网络请求

5，很方便的添加拦截器，通常情况下，拦截器用来添加，移除，转换请求和响应的头部信息，比如添加公共参数等

6，请求失败自动重连，发生异常时重连，看源码调用recover方法重连了一次

7. 支持SPDY协议(SPDY是Google开发的基于TCP的应用层协议，用以最小化网络延迟，提升网络速度，优化用户的网络使用体验。SPDY并不是一种用于替代HTTP的协议，而是对HTTP协议的增强。新协议的功能包括数据流的多路复用、请求优先级以及HTTP报头压缩。谷歌表示，引入SPDY协议后，在实验室测试中页面加载速度比原先快64%)

8.使用Okio来简化数据的访问与存储，提高性能

缺点：

1.消息回来需要切到主线程，主线程要自己去写。

2.调用比较复杂，需要自己进行封装。

3.缓存失效：网络请求时一般都会获取手机的一些硬件或网络信息，比如使用的网络环境。同时为了信息传输的安全性，可能还会对请求进行加密。在这些情况下OkHttp的缓存系统就会失效了，导致用户在无网络情况下不能访问缓存

框架中用到了那些设计模式：

1.最明显的Builder设计模式，如构建对象OkHttpClient，还有单利模式

2.工厂方法模式，如源码中的接口Call

3.观察者模式如EventListener，监听请求和响应

4.策略模式

5.责任链模式，如拦截器

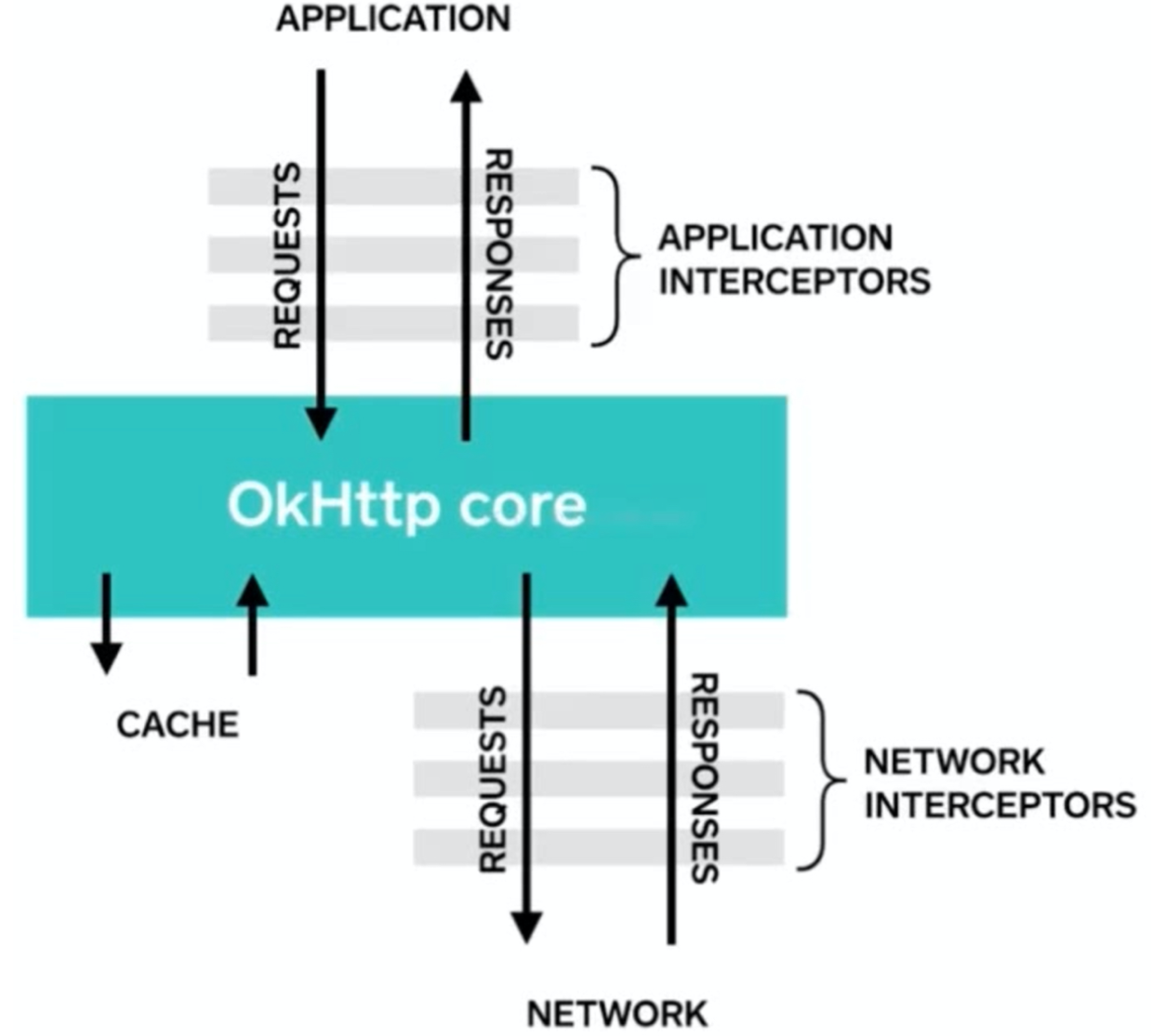
OK的网络请求缓存如何处理？

网络缓存优先考虑强制缓存，再考虑对比缓存

首先判断强制缓存中的数据是否在有效期内，如果在，就直接用缓存，如过了有效期，则进入对比缓存

在对比缓存过程中，判断ETag是否有变动，如果服务端返回没有变动，说明资源未改变，使用缓存。如果有变动，判断Last-Modified。

判断Last-Modified，如果服务端对比资源的上次修改时间没有变化，则用缓存，否则重新请求服务端的数据，并作缓存工作。

Okhttp源码解析 

Okhttp构造的时候会new 一个dipatcher对象

同步请求：dispatcher（分发器--保存同步请求，移除异步请求），execute 异步请求：dispatcher（一个等待请求队列，一个正在请求队列，一个线程池），enqueue

判断当前call，封装一个AsyncCall对象，通过client.dispatcher().enqueue()请求

通过 getResponseWithInterceptorChain()

dispatcher的作用为维护请求的状态，并维护一个线程池，用于执行请求。

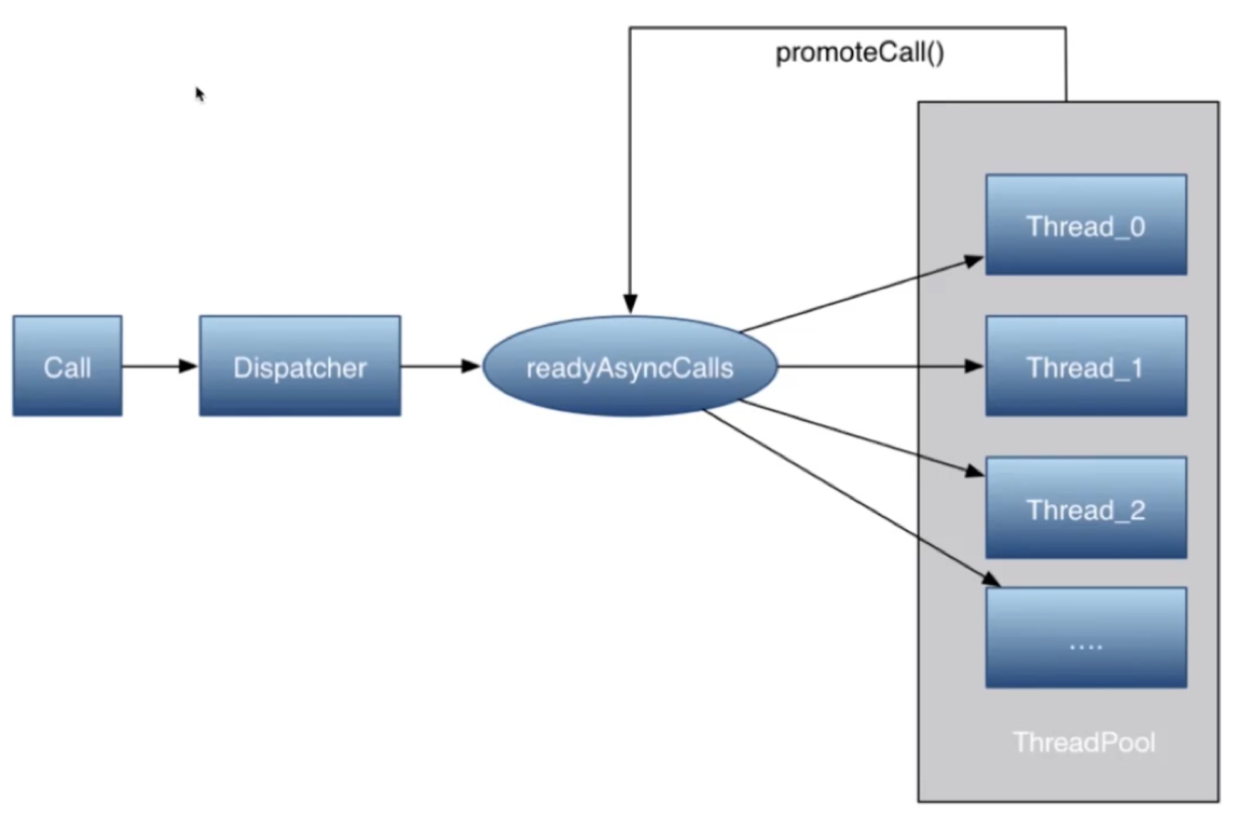
ok相对其他的来说高效的原因：内部有个异步的线程池用来维护

同步请求直接一个队列？

直接runningSyncCalls.add(call)；不再判断，直接添加，因为同步请求不需要像异步还要判断

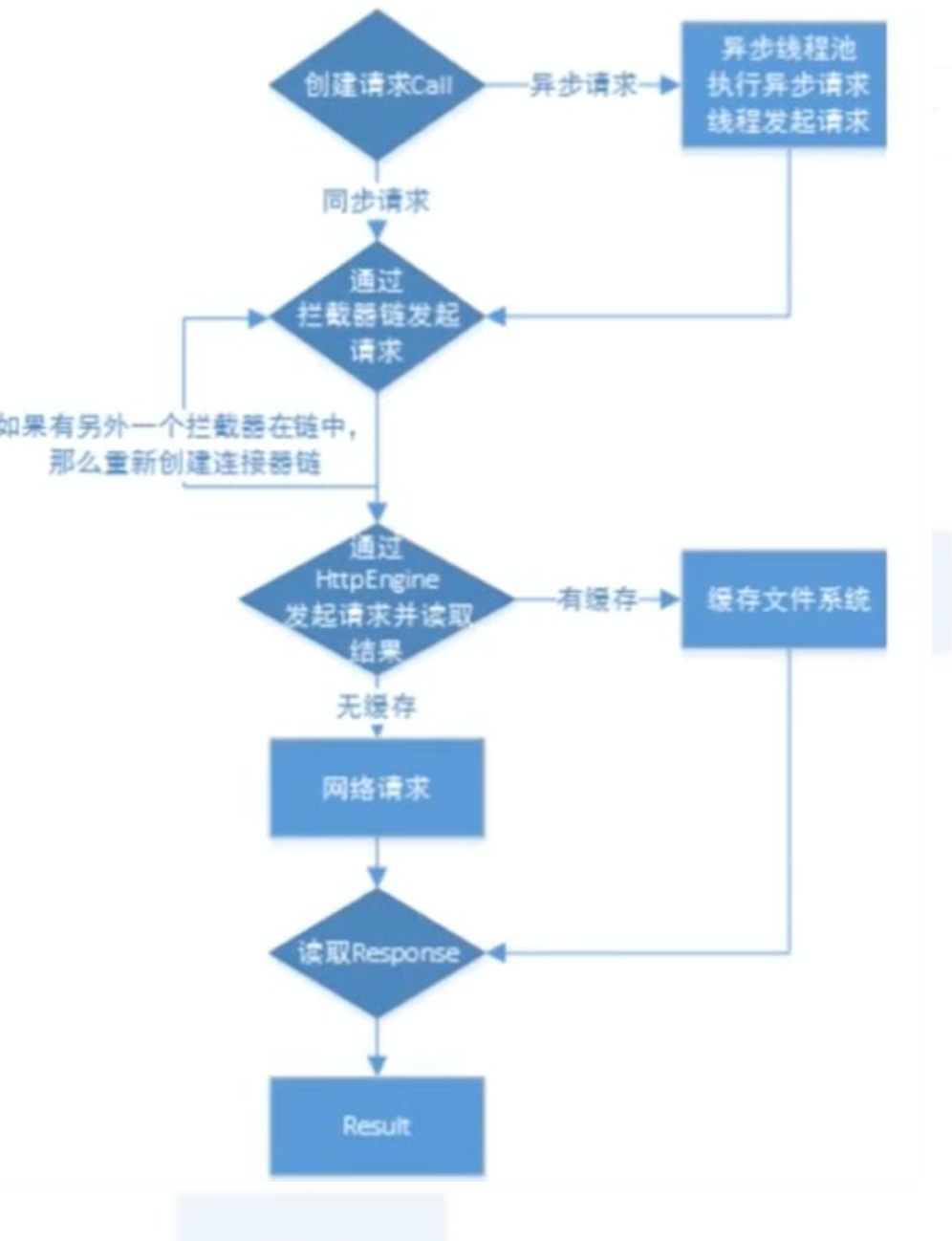
异步请求为什么需要2个队列？

类似生产者消费者模型，生产者只生产，消费者只消费，因为异步有最大运行数64限制，所以超过数量后需要有缓存仓库；

* Dispatcher 生产者，
* ExecutorService 消费者池，
* Deque缓存，
* Deque正在运行的任务 

开启一个线程池，然后

不管同步请求还是异步请求都是通过拦截器链发起请求

然后通过HttpEngine发起请求并读取结果，有缓存就进缓存文件系统，无缓存就走网络请求，然后读取Response最后获取结果 

### 1，retrofit源码解析

Retrofit底层是基于Okhttp实现的，使用运行时注解的方式

1、 原理

通过java接口以及注解来描述网络请求，并用动态代理的方式生成网络请求的request，然后通过client调用相应的网络框架（默认okhttp）去发起网络请求，并将返回的response通过converterFactorty转换成相应的数据model，最后通过call adapter转换成其他数据方式（如rxjava Observable）

2、 Retrofit流程

（1）通过解析 网络请求接口的注解 配置 网络请求参数

（2）通过 动态代理 生成 网络请求对象

（3）通过 网络请求适配器 将 网络请求对象 进行平台适配

（4）通过 网络请求执行器 发送网络请求

（5）通过 数据转换器 解析服务器返回的数据

（6）通过 回调执行器 切换线程（子线程 ->>主线程）

（7）用户在主线程处理返回结果

3、 Retrofit优点

1.可以配置不同HTTP client来实现网络请求，如okhttp、httpclient等；

2.请求的方法参数注解都可以定制；

3.支持同步、异步和RxJava；

4.超级解耦；

5.可以配置不同的反序列化工具来解析数据，如json、xml等

6.框架使用了很多设计模式

如何使用：

* 在retrofit中通过一个接口作为http请求的api接口
* 创建一个Retrofit实例
* 调用api接口

Retrofit动态代理：

* 首先，通过method把它转换成ServiceMethod
* 然后，通过serviceMethod，args获取到okHttpCall对象
* 最后，再吧okHttpCall进一步封装并返回Call对象

源码：

* 创建一个retrofit对象
* 通过Retrofit.create()方法把定义的接口转换成接口实例，并使用接口中的方法
* 最终的网络请求调用okhttp

### 2，volley源码解析

eg：<https://a.codekk.com/detail/Android/grumoon/Volley%20%E6%BA%90%E7%A0%81%E8%A7%A3%E6%9E%90>

通过new RequestQueue() 函数新建并启动一个请求队列RequestQueue后，只需往这个RequestQueue不断add Request即可。

2个请求队列 CacheDispatcher，NetworkDispatcher

先判断缓存，如果有且没过期，就从缓存拿数据

没有缓存，就从NetworkDispatcher中获取并保持在hashmap

默认情况下，volley中开启四个网络调度线程和一个缓存调度线程，首先请求会加入缓存队列，缓存调度线程从缓存队列中取出线程，如果找到该请求的缓存就

直接读取该缓存并解析，然后回调给主线程，如果没有找到缓存的响应，则将这个请求加入网络队列，然后网络调度线程会轮询取出网络队列中的请求，发起http请求，解析响应并将响应存入缓存，回调给主线程。

volley基于请求队列，volley的网络请求线程池默认大小为4，意味着只能并发进行4个请求，大于4个会排在队列中，并发量太小，所以适合请求高频率

volley下载文件会将流入内存中（是一个小于4k的缓存池），大文件会导致内存溢出，所以不能下载大文件不能上传大文件，比如上传了四个大文件，同时占用了volley的四个线程导致其他网络请求都阻塞在队列中，造成反应慢的现象。

### 3，Butterknife原理

依托Java的注解机制来实现辅助代码生成的框架。没有采用反射机制，用的是编译时再生成代码

1. 开始 会扫描Java代码中所有的ButterKnife注解
2. ButterKnifeProcessor 生成->$$ViewBinder
3. 调用bind方法加载深沉的ViewBinder类

注意：注解的方法必须是public，protected，不能是private，会影响性能，而且private是反射注入的

### 4，Glide源码解析

1）Glide.with(context)创建了一个RequestManager，同时实现加载图片与组件生命周期绑定：在Activity上创建一个透明的ReuqestManagerFragment加到FragmentManager中，通过添加的Fragment感知Activty。因为添加到Activity中的Fragment会跟随Activity的生命周期。在RequestManagerFragment中的相应生命周期方法中通过liftcycle传递给在lifecycle中注册的LifecycleListener

2）RequestManager.load(url) 创建了一个RequestBuilder对象 T可以是Drawable对象或是ResourceType等

1. RequestBuilder.into(view)

-->into(glideContext.buildImageViewTarget(view, transcodeClass))返回的是一个DrawableImageViewTarget, Target用来最终展示图片的，buildImageViewTarget-->ImageViewTargetFactory.buildTarget()根据传入class参数不同构建不同的Target对象，这个Class是根据构建Glide时是否调用了asBitmap()方法，如果调用了会构建出BitmapImageViewTarget，否则构建的是GlideDrawableImageViewTarget对象。

-->GenericRequestBuilder.into(Target),该方法进行了构建Request，并用RequestTracker.runRequest()

Request request = buildRequest(target);//构建Request对象，Request是用来发出加载图片的，它调用了buildRequestRecursive()方法以，内部调用了GenericRequest.obtain()方法

target.setRequest(request);

lifecycle.addListener(target);

requestTracker.runRequest(request);//判断Glide当前是不是处于暂停状态，若不是则调用Request.begin()方法来执行Request，否则将Request添加到待执行队列里，等暂停态解除了后再执行

-->GenericRequest.begin()

1）onSizeReady()--> Engine.load(signature, width, height, dataFetcher, loadProvider, transformation, transcoder,

            priority, isMemoryCacheable, diskCacheStrategy, this) --> a)先构建EngineKey; b) loadFromCache从缓存中获取EngineResource，如果缓存中获取到cache就调用cb.onResourceReady(cached)； c)如果缓存中不存在调用loadFromActiveResources从active中获取，如果获取到就调用cb.onResourceReady(cached)；d)如果active中也不存在，调用EngineJob.start(EngineRunnable), 从而调用decodeFromSource()/decodeFromCache()-->如果是调用decodeFromSource()-->ImageVideoFetcher.loadData()-->HttpUrlFetcher()调用HttpUrlConnection进行网络请求资源-->得于InputStream()后,调用decodeFromSourceData()-->loadProvider.getSourceDecoder().decode()方法解码-->GifBitmapWrapperResourceDecoder.decode()-->decodeStream()先从流中读取2个字节判断是GIF还是普通图，若是GIF调用decodeGifWrapper()来解码，若是普通静图则调用decodeBitmapWrapper()来解码-->bitmapDecoder.decode()

6、Glide使用什么缓存？

1. 内存缓存：LruResourceCache(memory)+弱引用activeResources

Map>> activeResources正在使用的资源，当acquired变量大于0，说明图片正在使用，放到activeResources弱引用缓存中，经过release()后，acquired=0,说明图片不再使用，会把它放进LruResourceCache中

2）磁盘缓存：DiskLruCache,这里分为Source(原始图片)和Result（转换后的图片）

第一次获取图片，肯定网络取，然后存active，再把图片显示出来，第二次读取相同的图片，并加载到相同大小的imageview中，会先从memory中取，没有再去active中获取。如果activity执行到onStop时，图片被回收，active中的资源会被保存到memory中，active中的资源被回收。当再次加载图片时，会从memory中取，再放入active中，并将memory中对应的资源回收。

之所以需要activeResources，它是一个随时可能被回收的资源，memory的强引用频繁读写可能造成内存激增频繁GC，而造成内存抖动。资源在使用过程中保存在activeResources中，而activeResources是弱引用，随时被系统回收，不会造成内存过多使用和泄漏。

7、Glide内存缓存如何控制大小？

Glide内存缓存最大空间(maxSize)=每个进程可用最大内存*0.4（低配手机是   每个进程可用最大内存*0.33）

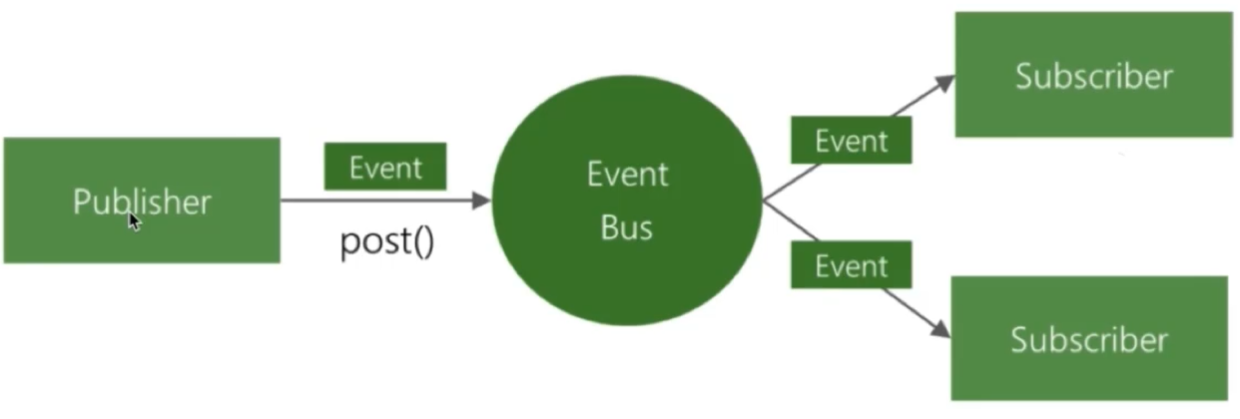
磁盘缓存大小是250MB   int DEFAULT\_DISK\_CACHE\_SIZE = 250 \* 1024 \* 1024;

### 5，leakcanary、blockcanary

### 6，eventbus

Android事件发布订阅框架

事件传递既可以用于Android四大组件间的通讯

EventBus 

1，定义事件Event

2，准备订阅者

3，订阅者同时需要在总线上注册和注销自己

4，发送事件

通过构造者模式构建EventBusBuilder，ThreadMode有四个线程，POSTING，MAIN，BACKGROUND，ASYNC，

### 7，dagger2

### 8，rxjava

Rxjava是基于响应式编程，基于事件流，实现异步操作的库。

RxJava原理是基于一种扩展的观察者模式，有四种角色：被观察者Observable 观察者Observer 订阅subscribe 事件Event。RxJava原理可总结为：被观察者Observable通过订阅(subscribe)按顺序发送事件（Emitter)给观察者(Observer)， 观察者按顺序接收事件&作出相应的响应动作。

### 9，picasso