Handler 和 Looper 来满足线程间的通信。Handler先进先出原则。Looper类用来管理特定线程内对象之间的消息交换

1、Looper: 一个线程可以产生一个Looper对象，由它来管理此线程里的MessageQueue(消息队列)。

2、Handler: 你可以构造Handler对象来与Looper沟通，通过`sendMessage()`发消息到MessageQueue里;`handleMessage(Message msg)`接收Looper从Message Queue取出)所送来的消息。

3、 Message Queue(消息队列):用来存放线程放入的消息。

4、线程：UIthread 通常就是main thread，而**Android**启动程序时会替它建立一个MessageQueue。 卖色字KU

Handler创建消息时首先查询消息池中是否有消息存在，如果有直接从消息池中取得，如果没有则重新初始化一个消息实例。使用消息池的好处是：消息不被使用时，并不作为垃圾回收，而是放入消息池，可供下次Handler创建消息时使用。消息池提高了消息对象的复用，减少系统垃圾回收的次数。

EventBus原码分析：

register原理：

1、onCreate中进行register，在onDestory中进行unregister ；

register(this)就是去当前类及所有父类，遍历所有的方法，找到onEvent开头的然后进行存储

只要把这个param发布出去，EventBus会在它内部存储的方法中，进行扫描，找到参数匹配的，就使用反射进行调用。它内部使用了Map进行存储，键就是参数的Class类型

EventBus.getDefault()其实就是个单例，使用了双重判断的方式，防止并发的问题，还能极大的提高效率。

```
01. public static EventBus getDefault() {
02.        if (defaultInstance == null) {
03.            synchronized (EventBus.class) {
04.                if (defaultInstance == null) {
05.                    defaultInstance = new EventBus();
06.                }
```

register公布给我们使用的有4个

```
01. public void register(Object subscriber) {
02.         register(subscriber, DEFAULT_METHOD_NAME, false, 0);
03.     }
04.  public void register(Object subscriber, int priority) {
05.         register(subscriber, DEFAULT_METHOD_NAME, false, priority);
06.     }
07. public void registerSticky(Object subscriber) {
08.         register(subscriber, DEFAULT_METHOD_NAME, true, 0);
09.     }
10. public void registerSticky(Object subscriber, int priority) {
11.         register(subscriber, DEFAULT_METHOD_NAME, true, priority);
12.     }
```

本质上就调用了同一个：

```
[java]

01. private synchronized void register(Object subscriber, String methodName, boolean sticky, int priority) {
02.         List<SubscriberMethod> subscriberMethods = subscriberMethodFinder.findSubscriberMethods(subscriber.getClass(),
03.                 methodName);
04.         for (SubscriberMethod subscriberMethod : subscriberMethods) {
05.             subscribe(subscriber, subscriberMethod, sticky, priority);
06.         }
07.     }
```

四个参数

subscriber 是我们扫描类的对象，也就是我们代码中常见的this;

methodName 这个是写死的：“onEvent” ，用于确定扫描什么开头的方法，可见我们的类中都是以这个开头。

sticky 这个参数，解释源码的时候解释，暂时不用管

priority 优先级，优先级越高，在调用的时候会越先调用。

肯定是去遍历该类内部所有方法，然后根据methodName去匹配，匹配成功的封装成SubscriberMethod，最后返回一个List

clazz.getMethods();去得到所有的方法：

分别判断了是否以onEvent开头，是否是public且非static和abstract方法，是否是一个参数。如果都复合，才进入封装的部分。

根据方法的后缀，来确定threadMode，threadMode是个枚举类型：就四种情况。

将method, threadMode, eventType传入构造了：new SubscriberMethod(method, threadMode, eventType)。添加到List，最终放回。

clazz = clazz.getSuperclass();可以看到，会扫描所有的父类，不仅仅是当前类。

扫描了所有的方法，把匹配的方法最终保存在subscriptionsByEventType（Map，key：eventType ； value：CopyOnWriteArrayList<Subscription> ）中；

eventType是我们方法参数的Class，Subscription中则保存着subscriber, subscriberMethod（method, threadMode, eventType），priority；包含了执行改方法所需的一切。

post它又是如何调用我们的方法的

把我们传入的event，保存到了当前线程中的一个变量PostingThreadState的eventQueue中。

判断当前是否是UI线程。

遍历队列中的所有的event，调用postSingleEvent（eventQueue.remove(0), postingState）方法。

这里大家会不会有疑问，每次post都会去调用整个队列么，那么不会造成方法多次调用么？

有个判断，就是防止该问题的，isPosting=true了，就不会往下走了。

将我们的event，即post传入的实参；以及postingState传入到postSingleEvent中。

2-3行：根据event的Class，去得到一个List<Class<?>>；其实就是得到event当前对象的Class，以及父类和接口的Class类型；主要用于匹配，比如你传入Dog extends Dog，他会把Animal也装到该List中。

6-31行：遍历所有的Class，到subscriptionsByEventType去查找subscriptions；哈哈，熟不熟悉，还记得我们register里面把方法存哪了不？

是不是就是这个Map；

12-30行：遍历每个subscription,依次去调用postToSubscription(subscription, event, postingState.isMainThread);

这个方法就是去反射执行方法了，大家还记得在register，if(sticky)时，也会去执行这个方法。

第一步根据threadMode去判断应该在哪个线程去执行该方法；

线程模式：

PostThread:直接反射调用；也就是说在当前的线程直接调用该方法；

MainThread:首先去判断当前如果是UI线程，则直接调用；否则： mainThreadPoster.enqueue(subscription, event);把当前的方法加入到队列，然后直接通过handler去发送一个消息，在handler的handleMessage中，去执行我们的方法。

BackgroundThread:如果当前非UI线程，则直接调用；如果是UI线程，则将任务加入到后台的一个队列，最终由Eventbus中的一个线程池去调用

executorService = Executors.newCachedThreadPool();

Async:将任务加入到后台的一个队列，最终由Eventbus中的一个线程池去调用；

总结一下：register会把当前类中匹配的方法，存入一个map，而post会根据实参去map查找进行反射调用。

就是在一个单例内部维持着一个map对象存储了一堆的方法；post无非就是根据

参数去查找方法，进行反射调用。

●·反射的性能：

反射需要在运行时解析字节码，同时虚拟机无法对反射的代码作相关优化，在效率上反射比正常的代码要低一些；

注解的三个角色

注解类：

应用了注解类的类：

对应用了注解类的类进行反射操作的类：

一些常用的注入框架

ButterKnife： 运行时没有使用反射的方式调用，在编译时解析注解生成新代码

AndroidAnnotation： 没有使用反射，在编译时解析注解生成新代码

Dagger2： 没有使用反射，在编译时解析注解生成新代码

RoboGuice： 使用了反射，性能上有一点损耗

如何区分：

变量可以定义为私有的并成功注入，则使用了反射；

变量定义不能定义为private的，否则无法注入，则没有使用反射，在编译时解析注解生成新代码；

Dagger2

➢·**Dagger1**是由**Square**公司受到Google的**Guice** （'juice'）(**https：//github.com/google/guice**)项目的启发，开发的依赖注入框架。

Dagger1用到了反射机制（创建图），性能上会有一定的损耗；

➢·**Dagger2**是**Dagger1**(**https：//github.com/square/dagger**)的分支，由谷歌接手开发。

Dagger2没有使用反射机制，在会在编译时生成中间代码；

●·**Dagger2**的特点：

➢·在编译时自动生成代码；

➢·没有使用反射，效率较高，谷歌表示比dagger1性能提高了**13%**

➢·容易调试；

遵循**JSR-330**标准：

三个角色及相关注解

角色1：被注入类，需要注入对象的类 [MainActivity]

角色2：**Module**类，创建依赖对象的工厂 [MainActivityModule]

角色3： **Component**桥梁类(接口&实现类)：连接被注入类与**Module**类的桥梁 [MainActivityComponent 组件]

1.1. Volley

●·Google 2013 I/O大会上推出，适合数据请求频繁且数据量少的应用场景；

●·不支持同步数据请求操作；

●·封装了异步的网络请求，底层实现：(见源码)

2.3及以上版本用的是HttpURLConnection

2.2及以下版本用的是HttpClient

●·基于接口设计，扩展性很强，比如可以封装Okhttp，使用它来做数据请求；(见源码)

●·默认启动了四个线程处理网络请求，一个线程处理数据缓存；（见源码）

●·因为只有四个线程处理网络请求，不适合文件上传下载操作，否则会等待请求无法执行；

1.2. Okhttp

●·OKHttp：Square 公司针对 Java 和 Android 程序封装的一个高性能 HTTP请求开源库。

●·API简洁易用；

●·实现了HTTP2、spdy、websocket协议；

●·支持同步和异步数据请求，但异步请求是在子线程回调 （使用时通常再作一层封装）；

●·封装了线程池、数据转换、GZIP 压缩、HTTP协议缓存等；

●·Picasso图片框架使用了OKHttp的缓存机制实现其文件缓存；

（而UniversalImageLoader自己做文件缓存，没有遵守http缓存机制）

●·为什么性能高： 基于NIO和Okio

什么是NIO?

NIO：non-blocking IO（java非阻塞式IO)，是jdk1.4 里提供的新api，为所有的原始类型提供缓存支持。

NIO能够处理的所有场景，原普通IO基本都能做到，NIO主要因效率而生，效率包括处理速度和吞吐量。

●·需要注意：

OKHttp和HttpClient，HttpUrlConnection类似，是对http协议的封装；

而Volley和AsyncHttpClient是基于HttpClient或HttpUrlConnection作封装；

Retrofit与picasso是基于okhttp作封装；

所以OKHttp和HttpClient，HttpUrlConnection是比较底层的，如果使用OkHttp通常还会再封装多一层(OKHttpUtils)，

类似Volley或AsyncHttpClient，以方便在主线程回调获取返回来的服务器数据；

Retrofit

●·Retrofit 是Square公司出品的默认基于OkHttp封装的网络请求框架。

●·服务器端需要支持RESTful 规范

●·解耦得很彻底：

比方说通过注解来配置请求参数；

通过工厂来生成CallAdapter，Converter：

可以使用不同的调用适配器(CallAdapter)，比方说RxJava，Java8， Guava

可以使用不同的数据转换器(Converter)，比方说json，xml，moshi等。

●·通常会组合RxJava(响应式编程)使用；

●·内部基于OKHttp，性能较高

Handler消息机制：