# 说明

本节主要对 Android 平台主要框架、开源库源码分析进行总结，方便笔试、面试前进行系统复习，并给出了完整源码分析的链接。

# 目录

* EventBus 源码分析
* Handler 源码分析
* AsyncTask 源码分析
* View 的工作机制源码分析
* Android 触摸事件分发机制源码分析
* Android 按键事件分发机制源码分析
* OkHttp 源码分析
* Retrofit 源码分析
* ButterKnife 源码分析
* Dagger2 源码分析

# EventBus 源码分析

1、register 方法注册：首先会根据观察者的类型找出它声明要订阅的所有事件（订阅方法），然后一一订阅

2、订阅过程：首先根据事件类型获取到订阅该事件类型的订阅关系集合，（为观察者和订阅方法生成订阅关系）并把这个订阅关系对象存入到该集合中；然后根据观察者获取到该观察者订阅的事件集合，并把当前订阅的事件放入到事件集合中。

EventBus 有两个 Map 类型的成员变量，分别为：

* Map1 用于根据事件类型通过反射调用观察者的方法 -- Key：事件类型，Value：订阅了该事件的订阅关系（观察者、订阅方法）集合
* Map2 用于取消订阅 -- Key：观察者，Value：该观察者的订阅事件集合

3、观察目标 post 事件：首先获得当前线程的待发送事件队列，并把当前事件对象添加进去；接着依次发送当前队列中的事件对象。

4、事件对象的发送：首先从刚才的 Map 中，根据事件类型取得订阅关系集合；然后遍历订阅关系集合，先进行线程判断，再分别通过反射调用观察者的订阅方法

5、unregister 解除注册：首先从 Map2 中根据观察者取得该观察者订阅事件集合，然后一一解除该观察者和每个事件的订阅关系，最后再把该观察者从 Map2 中删除

6、解除观察者和每个事件的订阅关系：从 Map1 中根据事件类型获得订阅了该事件的订阅关系集合，将该观察者相关的订阅关系进行删除

补充：

* 订阅方法是对订阅事件做的一层封装
* 订阅关系是对观察者和订阅方法做的一层封装

[EventBus 源码分析完整版地址](http://xuchongyang.com/2017/07/16/EventBus-3-0-%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90%EF%BC%88%E4%B8%80%EF%BC%89%E6%95%B4%E4%BD%93%E5%88%86%E6%9E%90/)

# Handler 源码分析

一句话总结：Looper 不断从 MessageQueue 中取出一个 Message，然后交给其对应的 Handler 处理

1、Handler 的构造方法会将 Handler 对象和当前线程的 Looper 对象绑定到一起

2、Handler 的一系列 sendMessage 和 post 方法最终会调用 enqueueMessage 方法将消息放入消息队列

3、Handler 在 dispatchMessage 方法中拿到消息后，首先检查 Message 的 callback 是否为 null，不为 null 直接调用其 run 方法；否则检查 Handler.mCallback 是否为 null，不为 null 直接调用其 handleMessage 方法；否则调用 Handler 的 handleMessage 方法

4、ThreadLocal 是一个线程内部的数据存储类，通过它可以在指定的线程中存储数据，ThreadLocal 以线程为作用域。

Looper 的作用域是线程，且不同的线程具有不同的 Looper，因此可以通过 ThreadLocal 来存储线程的 Looper 对象。

[Handler 源码分析完整版地址](http://xuchongyang.com/2017/03/10/Android-%E6%B6%88%E6%81%AF%E6%9C%BA%E5%88%B6%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90/)

# AsyncTask 源码分析

1、首先调用 AsyncTask 的构造方法，构造时对 Handler、WorkerRunnable（Callable） 和 FutureTask 进行初始化

2、然后调用 AsyncTask 的 execute 方法（可以手动设置 Executor，不设置则使用系统默认的 SerialExecutor）

3、首先判断当前 AsyncTask 状态，正在运行或者已经运行过就退出

4、调用 onPreExecute 执行准备工作

5、由 Executor 调用 FutureTask 的 run 方法，在 WorkerRunnable 中执行了 doInBackground

6、依旧是在 WorkerRunnable 中，调用 postResult，将执行结果通过 Handler 发送给主线程；调用 publishProgress 时，也是通过 Handler 将消息发送到主线程的消息队列中

[AsyncTask 源码分析完整版地址](http://xuchongyang.com/2017/05/07/AsyncTask-%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90/)

# View 的工作机制源码分析

1、ViewGroup 首先根据自己的 MeasureSpec 和子 View 自己设定的宽高，来为每个子 View 生成 MeasureSpec

2、ViewGroup 循环调用每个子 View 的 measure 方法，同时把子 View 的 MeasureSpec 传递过去

3、子 View 根据父视图给它的 MeasureSpec 确定最终的测量大小

可以看出，决定子 View 测量宽高的因素有：父视图的 MeasureSpec、子 View 自身指定的大小

[View 的工作机制源码分析完整版地址](http://xuchongyang.com/2017/08/20/View%20%E7%9A%84%E4%B8%89%E5%A4%A7%E6%B5%81%E7%A8%8B%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90/)

# Android 触摸事件分发机制源码分析

[Android 触摸事件分发机制源码分析完整版地址](http://xuchongyang.com/2017/07/30/Android%20%E4%BA%8B%E4%BB%B6%E5%88%86%E5%8F%91%E6%9C%BA%E5%88%B6%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90%EF%BC%88%E4%B8%80%EF%BC%89%E8%A7%A6%E6%91%B8%E4%BA%8B%E4%BB%B6/)

触摸事件分发流程：

1、ViewGroup 的事件分发

* 首先判断自身是否需要，需要则进行拦截，并调用自身的 onTouchEvent 方法进行使用并决定是否消费，子 View 对事件无感。
* 自己不需要则不进行拦截，事件分发给子 View（事件会分发给手指触摸区域的子 View，有重叠时传递给最上层 View）
* 子 View 不消费时，事件会回传，调用自身的 onTouchEvent 方法进行使用并决定是否消费；子 View 消费时事件不回传

伪代码：

public boolean dispatchTouchEvent(MotionEvent event) {
 // 默认状态为未消费过
 boolean result = false;
 // 如果没有拦截
 if (!onInterceptTouchEvent(event)) {
 // 则交给子 view
 result = child.dispatchTouchEvent(event);
 }
 // 如果事件没有被子 View 消费
 if (!result) {
 // 则询问自身 onTouchEvent()
 result = onTouchEvent(event);
 }
 // 返回事件消费状态
 return result;
}

2、View 的事件分发

dispatchTouchEvent 方法对 View 的监听器和方法进行分发，顺序如下：

onTouchListener -- > onTouchEvent -- > onLongClickListener -- > onClickListener

# Android 按键事件分发机制源码分析

[Android 按键事件分发机制源码分析完整版地址](http://xuchongyang.com/2017/08/03/Android%20%E4%BA%8B%E4%BB%B6%E5%88%86%E5%8F%91%E6%9C%BA%E5%88%B6%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90%EF%BC%88%E4%BA%8C%EF%BC%89%E6%8C%89%E9%94%AE%E4%BA%8B%E4%BB%B6/)

分发流程：

1、Activity 的分发

* Activity 接收到事件，为 Menu 键直接消费掉。否则分发给 PhoneWindow，PhoneWindow 直接传递给它的 DecorView
* DecorView 对 Back 键进行判断是否消费，不消费则分发给 ViewGroup
* ViewGroup 不消费时，事件分发给 KeyEvent

2、ViewGroup 的分发

* ViewGroup 有焦点且大小确定时，首先自己处理
* ViewGroup 的子 View 有焦点且大小确定时，分发给子 View
* 层层向下分发，直到最底层 View

3、View 的分发

View 首先将事件传递给 onKeyListner 监听器，不消费时传递给 KeyEvent

4、KeyEvent 的分发

KeyEvent 会根据类型来分别决定回调 Activity 或 View 的 onKeyDown、onKeyUp、onKeyLongPress 方法

5、事件回传

当 Activity、View 的 onKeyDown、onKeyUp 也没有消费事件时，事件会进行回传

# OkHttp 源码分析

示例：

mOkHttpClient.newCall(request).enqueue(callback);

## 概述版

* OkHttp 将请求信息封装在 Request 对象中，并创建一个 Call 对象
* 调用 Call 对象的方法发起网络请求，采用责任链模式，将请求信息依次传递给每个拦截器，最终发起网络请求并返回结果

## 分析版

1、OkHttpClient 的构建采用了 Builder 模式，Builder 模式易写易读且线程安全，适用于类的构造器有多个参数的情况

2、OkHttpClient 实现了 Call.Factory 接口（newCall 方法），通过 newCall 方法可以得到一个 RealCall 对象，可以操作 RealCall 对象执行请求、取消等一系列操作

3、RealCall 执行 execute 方法时，首先会在 client 的同步执行任务队列中记录当前 Call，然后在当前线程通过拦截器链条获取执行结果，最后对列中移除 Call

4、拦截器链条采用责任链模式，会依次执行每一个拦截器的 proceed 方法，直到最终的 CallServerInterceptor 完成网络请求

5、RealCall 的 enqueue 方法会直接将 Call 添加到 dispatcher 的异步执行队列，执行网络请求时也是通过拦截器链条完成

6、请求相关的所有信息封装在 Request 对象中，对于 Post 请求体 RequestBody 可以由 Byte 数组、ByteString、File、String 生成

[OkHttp 源码分析完整版地址](http://wiki.xuchongyang.com/1237005)

# Retrofit 源码分析

示例：

mRetrofit.create(RequestService.class).doGet("params").enqueue(callback);

## 概述版

* Retrofit 接口层实际是对 OkHttp 中 Request 的封装，采用注解的形式来描述网络请求参数
* 采用动态代理模式创建请求接口对象，请求执行调用接口方法时，Retrofit 会根据注解创建相应的Call对象，接下来使用OkHttp发起请求

## 分析版

1、首先需要使用注解的形式定义请求接口

2、然后通过建造者模式构造出 Retrofit 对象，Retrofit 对象的 create 方法采用动态代理模式，创建出请求接口的实例

3、发起请求时调用请求接口实例的相应方法，被调用处理器（InvocationHandler）拦截，最终返回一个 OkHttpCall 对象，OkHttpCall 有实现 Call 接口

4、接下来就是 OkHttp 的工作，调用 OkHttpCall 的 enqueue 或 execute 方法发起网络请求

[Retrofit 源码分析完整版地址](http://wiki.xuchongyang.com/1253933)

# ButterKnife 源码分析

## 概述版

* 写代码时对目标元素进行注解，编译时注解处理器会扫描注解，并通过 JavaPoet 自动生成 Java 文件
* 调用 ButterKnife.bind()方法时，通过反射拿到编译时生成的类，调用其构造方法完成目标类和绑定类的绑定

## 分析版

1、注解处理器 ButterKnifeProcessor 在编译时会扫描和处理所有注解，通过 JavaPoet 自动生成 .java 文件

2、调用 ButterKnife 的 bind 方法，通过反射拿到目标类类名，后面拼接\_ViewBinding找到刚刚自动生成的 Java 类，并根据参数找到相匹配的构造器

3、调用 ViewBinding 类构造器的构造方法，将 ViewBinding 类和 目标类进行绑定

4、ViewBinding 类的构造方法中会对目标类的成员变量进行初始化及点击事件配置

[ButterKnife 源码分析完整版地址](http://wiki.xuchongyang.com/1253934)

# Dagger2 源码分析