# **[Android消息传递之EventBus 3.0使用详解](http://www.cnblogs.com/whoislcj/p/5595714.html)**

#### **EventBus产生需求背景：**

    在做项目的时候往往需要应用程序内各组件间、组件与后台线程间的通信。比如耗时操作，等耗时操作完成后通过Handler或Broadcast将结果通知给UI，N个Activity之间需要通过Listener通信，之前的实现方式我们在[Android消息传递之组件间传递消息（二）](http://www.cnblogs.com/whoislcj/p/5593056.html)中已经介绍过了，其实这些都可以通过EventBus轻松实现，EventBus通过发布/订阅(publish/subscribe)方式来管理事件总线。其实EventBus的实现方式更加接近上篇文章的方式二，不同的是EventBus通过注解和反射机制 将订阅者连同订阅函数保存起来，然后在发送订阅的时候 遍历订阅函数数组进行调用，其实从这方面就可以EventBus执行效率多少会受到一点影响。

#### **EventBus介绍:**

     EventBus出自greenrobot，和之前大名鼎鼎的GreenDao出自同一家。之前一直使用的是2.4版本，今天我们将学习分析最新的Event 3.0，EventBus 3.0 最新的特性就是加入了注解，通过注解的方式 告知订阅函数运行在哪个线程中。

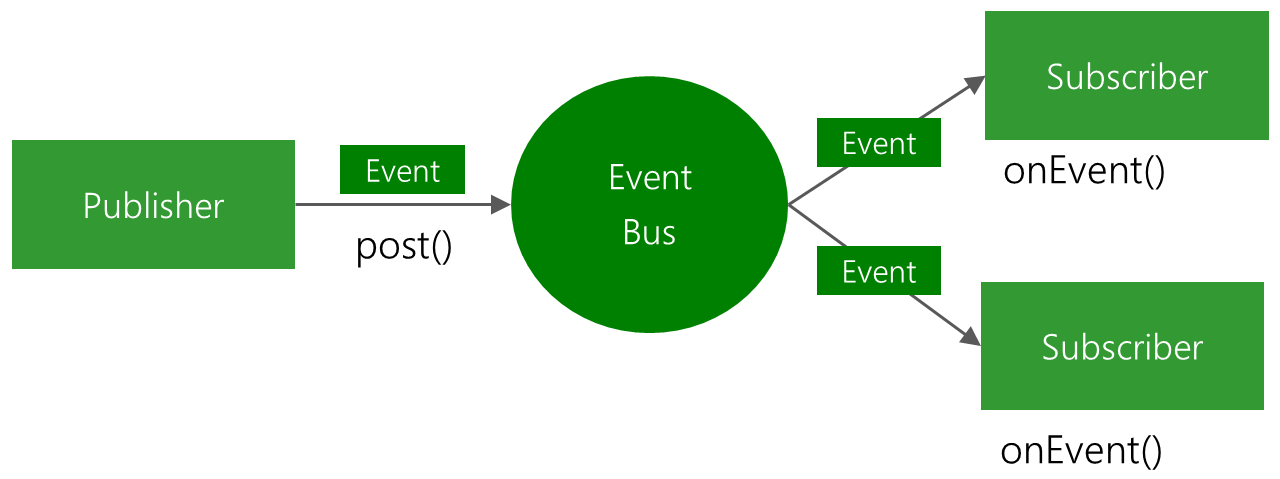
     github地址：https://github.com/greenrobot/EventBus

     官方文档：http://greenrobot.org/eventbus/documentation

#### **EventBus主要角色:**

* **Event 传递的事件对象**
* **Subscriber  事件的订阅者**
* **Publisher  事件的发布者**
* **ThreadMode 定义函数在何种线程中执行**

  官网给出的各种角色的协作图



#### **EventBus配置:**

  EventBus框架也是采用建造者模式设计的，可以通过EventBusBuilder来设置一些配置信息，例如设置debug模式下要抛出异常

EventBus eventBus=EventBus.builder().throwSubscriberException(BuildConfig.DEBUG).build();

#### **EventBus示例:**

 之前做图片社交App的时候，需要处理一个点赞数据的同步，比如在作品的详情页点赞 需要同时更新列表页该作品的点赞数量，这里还是以此为例。

##### **1.）build.gradle添加引用**

compile 'org.greenrobot:eventbus:3.0.0'

##### **2.）定义一个事件类型**

[IMG_257](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

public class DataSynEvent {

private int count;

public int getCount() {

return count;

}

public void setCount(int count) {

this.count = count;

}

}

[IMG_258](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

##### **3.）订阅/解除订阅**

 订阅

EventBus.getDefault().register(this);//订阅

 解除订阅

EventBus.getDefault().unregister(this);//解除订阅

##### **4.）发布事件**

EventBus.getDefault().post(new DataSynEvent());

##### **5.）订阅事件处理**

@Subscribe(threadMode = ThreadMode.MAIN) //在ui线程执行

public void onDataSynEvent(DataSynEvent event) {

Log.e(TAG, "event---->" + event.getCount());

}

ThreadMode总共四个：

* NAIN UI主线程
* BACKGROUND 后台线程
* POSTING 和发布者处在同一个线程
* ASYNC 异步线程

##### **6.）订阅事件的优先级**

   事件的优先级类似广播的优先级，优先级越高优先获得消息

@Subscribe(threadMode = ThreadMode.MAIN,priority = 100) //在ui线程执行 优先级100

public void onDataSynEvent(DataSynEvent event) {

Log.e(TAG, "event---->" + event.getCount());

}

##### **7.）终止事件往下传递**

 发送有序广播可以终止广播的继续往下传递，EventBus也实现了此功能

EventBus.getDefault().cancelEventDelivery(event) ;//优先级高的订阅者可以终止事件往下传递

##### **8.）处理代码混淆**

[IMG_259](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

-keepattributes \*Annotation\*

-keepclassmembers class \*\* {

@org.greenrobot.eventbus.Subscribe <methods>;

}

-keep enum org.greenrobot.eventbus.ThreadMode { \*; }

# Only required if you use AsyncExecutor

-keepclassmembers class \* extends org.greenrobot.eventbus.util.ThrowableFailureEvent {

<init>(java.lang.Throwable);

}

[IMG_260](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

#### **EventBus黏性事件**

   EventBus除了普通事件也支持粘性事件，这个有点类似广播分类中的粘性广播。本身粘性广播用的就比较少，为了方便理解成订阅在发布事件之后，但同样可以收到事件。订阅/解除订阅和普通事件一样，但是处理订阅函数有所不同，需要注解中添加sticky = true

@Subscribe(threadMode = ThreadMode.MAIN,sticky = true) //在ui线程执行

public void onDataSynEvent(DataSynEvent event) {

Log.e(TAG, "event---->" + event.getCount());

}

发送粘性事件

EventBus.getDefault().postSticky(new DataSynEvent());

对于粘性广播我们都比较清楚属于常驻广播，对于EventBus粘性事件也类似，我们如果不再需要该粘性事件我们可以移除

EventBus.getDefault().removeStickyEvent(new DataSynEvent());

或者调用移除所有粘性事件

EventBus.getDefault().removeAllStickyEvents();

#### **EventBus processor使用：**

   EventBus提供了一个EventBusAnnotationProcessor注解处理器来在编译期通过读取@Subscribe()注解并解析,  
处理其中所包含的信息,然后生成java类来保存所有订阅者关于订阅的信息,这样就比在运行时使用反射来获得这些订阅者的  
信息速度要快.

##### **1.)具体使用：在build.gradle中添加如下配置**

[IMG_261](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

buildscript {

dependencies {

classpath 'com.neenbedankt.gradle.plugins:android-apt:1.8'

}

}

apply plugin: 'com.neenbedankt.android-apt'

dependencies {

compile 'org.greenrobot:eventbus:3.0.0'

apt 'org.greenrobot:eventbus-annotation-processor:3.0.1'

}

apt {

arguments {

eventBusIndex "com.whoislcj.eventbus.MyEventBusIndex"

}

}

[IMG_262](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

##### **2.)使用索引**

此时编译一次，自动生成生成索引类。在\build\generated\source\apt\PakageName\下看到通过注解分析生成的索引类，这样我们便可以在初始化EventBus时应用我们生成的索引了。

自动生成的代码

[IMG_263](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

/\*\* This class is generated by EventBus, do not edit. \*/public class MyEventBusIndex implements SubscriberInfoIndex {

private static final Map<Class<?>, SubscriberInfo> SUBSCRIBER\_INDEX;

static {

SUBSCRIBER\_INDEX = new HashMap<Class<?>, SubscriberInfo>();

putIndex(new SimpleSubscriberInfo(com.whoislcj.testhttp.MainActivity.class, true, new SubscriberMethodInfo[] {

new SubscriberMethodInfo("onDataSynEvent", com.whoislcj.testhttp.eventBus.DataSynEvent.class,

ThreadMode.MAIN, 100, false),

new SubscriberMethodInfo("onDataSynEvent1", com.whoislcj.testhttp.eventBus.TestEvent.class, ThreadMode.MAIN,

0, true),

}));

}

private static void putIndex(SubscriberInfo info) {

SUBSCRIBER\_INDEX.put(info.getSubscriberClass(), info);

}

@Override

public SubscriberInfo getSubscriberInfo(Class<?> subscriberClass) {

SubscriberInfo info = SUBSCRIBER\_INDEX.get(subscriberClass);

if (info != null) {

return info;

} else {

return null;

}

}

}

[IMG_264](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

添加索引到EventBus默认的单例中

EventBus.builder().addIndex(new MyEventBusIndex()).installDefaultEventBus();

##### **3.）对比添加前后注册效率对比**

分别EventBus.getDefault().register(this);

  添加之前：前后用了**9**毫秒

IMG_265

  添加之后：前后用了**2**毫秒

IMG_266

#### **EventBus优缺点：**

   优点：简化组件之间的通信方式，实现解耦让业务代码更加简洁，可以动态设置事件处理线程以及优先级

   缺点：目前发现唯一的缺点就是类似之前策略模式一样的诟病，每个事件都必须自定义一个事件类，造成事件类太多，无形中加大了维护成本

#### **EventBus 3.0 与2.x的区别**

##### **1.）代码更加简洁**

   EventBus 2.x 必须定义以onEvent开头的几个方法，代码中语境比较突兀，且有可能会导致拼写错误，例如数据同步事件

[IMG_267](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

public void onEvent(DataSynEvent event) {

//事件在哪个线程发布出来的，onEvent就会在这个线程中运行， 同 @Subscribe(threadMode = ThreadMode.POSTING） }

public void onEventMainThread(DataSynEvent event) {

// 不论事件是在哪个线程中发布出来的，onEventMainThread都会在UI线程中执行，接收事件就会在UI线程中运行，同 @Subscribe(threadMode = ThreadMode.MAIN） }

public void onEventBackgroundThread(DataSynEvent event) {

//那么如果事件是在UI线程中发布出来的，那么onEventBackground就会在子线程中运行，如果事件本来就是子线程中发布出来的，那么onEventBackground函数直接在该子线程中执行,同 @Subscribe(threadMode = ThreadMode.BACKGROUND） }

public void onEventAsync(DataSynEvent event) {

//使用这个函数作为订阅函数，那么无论事件在哪个线程发布，都会创建新的子线程在执行onEventAsync，同 @Subscribe(threadMode = ThreadMode.ASYNC）

}

[IMG_268](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

EventBus  3.0 函数名字不再受到权限，而且可以在一个函数中体现出在哪个线程执行，并且可指定接收事件的优先级

[IMG_269](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

/\*\*

\* 普通事件

\* @param event

\*/

@Subscribe(threadMode = ThreadMode.MAIN, priority = 100)

public void onDataSynEvent(DataSynEvent event) {

}

/\*\*

\* 粘性事件

\* @param event

\*/

@Subscribe(threadMode = ThreadMode.MAIN, priority = 100, sticky = true)

public void onDataSynEvent(DataSynEvent event) {

}

[IMG_270](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

EventBus 2.x 注册方式也比较繁琐

[IMG_271](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

　 public void register(Object subscriber) {

　　register(subscriber, false, 0);

　　}

　　public void register(Object subscriber, int priority) {

　　register(subscriber, false, priority);

　　}

　　public void registerSticky(Object subscriber) {

　　register(subscriber, true, 0);

　　}

　　public void registerSticky(Object subscriber, int priority) {

　　register(subscriber, true, priority);

　　}

　　private void subscribe(Object subscriber, SubscriberMethod subscriberMethod, boolean sticky, int priority) {

　　...

　　}

[IMG_272](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

EventBus  3.0 注册方式只有一个

[IMG_273](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

public void register(Object subscriber) {

Class<?> subscriberClass = subscriber.getClass();

List<SubscriberMethod> subscriberMethods = subscriberMethodFinder.findSubscriberMethods(subscriberClass);

synchronized (this) {

for (SubscriberMethod subscriberMethod : subscriberMethods) {

subscribe(subscriber, subscriberMethod);

}

}

}

[IMG_274](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

以上还是在一个订阅者仅仅订阅一个事件的情况下，如果订阅多个事件，可想而知EventBus 2.x势必导致订阅者要写大量的多态函数，如果订阅多种类型事件，比如普通事件和粘性事件并存，估计要同时调用register，registerSticky两个函数。

##### **2.）性能更优**

  EventBus 2.x 是采用反射的方式对整个注册的类的所有方法进行扫描来完成注册，当然会有性能上的影响。EventBus  3.0中EventBus提供了EventBusAnnotationProcessor注解处理器来在编译期通过读取@Subscribe()注解并解析、处理其中所包含的信息，然后生成java类来保存所有订阅者关于订阅的信息，这样就比在运行时使用反射来获得这些订阅者的信息速度要快

#### **小结：**

     EventBus 3.0的使用基本上总结完了，之前一直担心EventBus通过注解或者反射会影响太多性能，随着3.0的发布这部分影响已经很小了。