# **[Android消息传递之基于RxJava实现一个EventBus - RxBus](http://www.cnblogs.com/whoislcj/p/5816992.html)**

### **需求：**

      虽然软件工程师就害怕听见“需求”这两个字，但是一切功能来源于需求，我们要基于RxJava实现一个publish/subscribe消息总线管理模型，这里假设你已经知道如何使用RxJava了。下面看看如何实现。

### **RxBus实现过程：**

#### **1.）build.gradle中添加RxJava、RxAndroid引用**

compile 'io.reactivex:rxandroid:1.1.0'

compile 'io.reactivex:rxjava:1.1.2'

#### **2.）声明一个管理事件总线的单例**

[IMG_256](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

public class RxBus {

private ConcurrentHashMap<Object, List<Subject>> subjectMapper = new ConcurrentHashMap<>();

private static volatile RxBus instance;

private RxBus() {

}

public static RxBus getInstance() {

RxBus inst = instance;

if (inst == null) {

synchronized (RxBus.class) {

inst = instance;

if (inst == null) {

inst = new RxBus();

instance = inst;

}

}

}

return inst;

}

}

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

考虑到RxBus单例很有可能多线程并发访问，这种存储事件总线采用ConcurrentHashMap

#### **3.）订阅者实现注册/解除注册方法**

注册/解注册都以一个tag为唯一标示，这里采用的是事件的Class为tag，来实现一个Event事件对象可以对应多个订阅者。

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

/\*\*

\* 注册

\*

\* @param tag

\* @return

\*/

public <T> Observable<T> register(@NonNull Class<T> tag) {

List<Subject> subjectList = subjectMapper.get(tag);

if (null == subjectList) {

subjectList = new ArrayList<>();

subjectMapper.put(tag, subjectList);

}

Subject<T, T> subject = PublishSubject.create();

subjectList.add(subject);

return subject;

}

/\*\*

\* 解除注册

\*

\* @param tag

\*/

public <T> void unregister(@NonNull Class<T> tag, @NonNull Observable observable) {

List<Subject> subjects = subjectMapper.get(tag);

if (null != subjects) {

subjects.remove(observable);

if (subjects.isEmpty()) {

subjectMapper.remove(tag);

}

}

}

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

#### **4.）实现发送消息、清除所有注册**

发送一个事件消息是通过获取当前事件下的订阅者列表，然后通过一个循环进行事件传递。至于事件运行在哪个线程中由订阅者决定。

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

/\*\*

\* 发送消息

\*

\* @param event

\*/

public <T> void post(@NonNull Object event) {

List<Subject> subjectList = subjectMapper.get(event.getClass());

if (subjectList != null && !subjectList.isEmpty()) {

for (Subject subject : subjectList) {

subject.onNext(event);

}

}

}

/\*\*

\* 清除订阅

\*/

public void clear() {

if (subjectMapper.isEmpty()) {

return;

}

subjectMapper.clear();

}

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

#### **5.）如何使用**

事件的订阅者的注册和清除所有注册，订阅者没订阅一个类型的事件都要返回一个Observable对象，这也是个人觉得比较头疼的一件事，很难实现一个订阅者对应多个事件类型，而且解注册的时候也需要这些Observable对象进行一一解除。

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

private Observable<DataSynEvent> observable;

private void registerObservable() {

observable = RxBus.getInstance().register(DataSynEvent.class);

observable.observeOn(AndroidSchedulers.mainThread()).subscribe(new Action1<DataSynEvent>() {

@Override

public void call(DataSynEvent event) {

Log.e(TAG, "event--2-->" + event.getCount());

}

});

}

private void unRegisterObservable() {

if (observable != null) {

RxBus.getInstance().unregister(DataSynEvent.class, observable);

}

}

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

事件的发送者发送事件，发送事件的处理还算比较友好，和EventBus很类似。

RxBus.getInstance().post(new DataSynEvent());

### **RxBus、EventBus 对比：**

**1.从引入依赖包对比**

  RxBus 需要引入两个包，EventBus需要引入一个，如果项目中没有使用到RxJava编程的话，并不能减少包的依赖。

**2 .从开发难度上对比**

  上面也提到了实现RxBus是基于RxJava的，作为一个新的编程方式函数式编程，对开发者的要求多多少少提高了那么一点，而且每一个订阅都需要返回一个Observable，由订阅者写具体的代码 需要执行在哪个线程中，而EventBus 最新版本采用注解预编译的方式，订阅者注册解注册只需调用一个函数，而且通过注解的方式可以标明事件的优先级，接收事件运行在哪个线程，并且能够实现粘性事件，这些RxBus都需要进行二次开发。

**3.）从开发效率上对比**

   RxBus 需要进行大量的二次开发，否则只能实现简单的事件传递，而EventBus只需简单了解一下API就能上手。如果一个订阅者需要注册多个事件的时候，需要多个Observable全局变量，这不是疯掉了吗，而EventBus已经实现了一个订阅者订阅多个事件，和一个事件对应多个订阅者。

**4.）从功能完善上对比**

 上面三个对比也说明了，EventBus实现了事件的优先级，订阅事件运行的线程场景，以及粘性事件，这些在RxBus上面都需要进行二次实现。

### **总结：**

    基于RxJava实现简单的事件总线管理是可以的，但是个人觉得想要取代EventBus难免有点说过头了。所以如果项目中没有使用RxJava的话 还是采用EventBus比较靠谱。