

EventBus 3.X 的使用



changchengfeng [关注](#)

0.264 2018.05.12 04:13:37 字数 1,391 阅读 1,508

这篇文章主要记录一下EventBus的使用,详情[EventBus官方文档](#)

1. 概念

EventBus能够简化各组件间的通信，让我们的代码书写变得简单，能有效的分离事件发送方和接收方(也就是解耦的意思)，能避免复杂和容易出错的依赖性和生命周期问题。

- 事件(**Event**)：又可称为消息，本文中统一用事件表示。其实就是一个POJO对象
事件分为一般事件和 **Sticky** 事件，相对于一般事件，Sticky 事件不同之处在于，当事件发布后，再有订阅者开始订阅该类型事件，依然能收到该类型事件最近一个 Sticky 事件。
- 订阅者(**Subscriber**)：订阅某种事件类型的对象。当有发布者发布这类事件后，EventBus 会执行订阅者的 @Subscribe注解标记的方法，叫做事件响应。订阅者通过 register 接口订阅某个事件类型，unregister 接口退订。订阅者存在优先级，优先级高的订阅者可以取消事件继续向优先级低的订阅者分发(但是取消事件的线程必须和发布事件的线程一致)，默认所有订阅者优先级都为 0。



广告

上海市网站建设



changchengfeng

[关注](#)

总资产19 (约1.41元)

Android jni 学习总结

阅读 144

java 安全体系算法调用过程

阅读 649

推荐阅读

花了两个小时写的自定义Toast

阅读 136

- 发布者(Publisher)：发布某事件的对象，通过 post 接口发布事件。

2. 简单使用

2.1. 添加EventBus库

```
1 | api 'org.greenrobot:eventbus:3.1.1'
```

2.2. 新建一个POJO对象代表一个事件(Event)

```
1 public class MessageEvent {  
2  
3     public final String message;  
4  
5     public MessageEvent(String message) {  
6         this.message = message;  
7     }  
8 }
```

2.3. 在生命周期里面注册和取消注册

```
1 @Override  
2 public void onStart() {  
3     super.onStart();  
4     EventBus.getDefault().register(this);  
5 }  
6  
7 @Override
```

这15个Android开源库，只有经常逛Github的才知道！

阅读 6,090

Android Studio 导饺子播放器library时候问题

阅读 32

JetPack中的Lifecycle，你真的有了解过吗？

阅读 55

Android实现一个比较炫酷的自定义View

阅读 3,041



程序员外包

```
8 | public void onStop() {  
9 |     EventBus.getDefault().unregister(this);  
10 |     super.onStop();  
11 | }
```

2.4. 订阅者处理事件

```
1 | // 当 MessageEvent 事件被发布这个方法会被调用(在主线程中显示一个 Toast)  
2 | @Subscribe(threadMode = ThreadMode.MAIN)  
3 | public void onMessageEvent(MessageEvent event) {  
4 |     Toast.makeText(getActivity(), event.message, Toast.LENGTH_SHORT).show();  
5 | }  
6 |
```

2.5. 发布事件

发布事件的类和订阅者的类不相同时,两个类不直接引用就可以进行通信.

```
1 | EventBus.getDefault().post(new MessageEvent("Hello everyone!"));
```

3. 扩展

3.1. ThreadMode (线程模式)

订阅者响应事件的方法是通过 **@Subscribe** 注解标注的,其中有个 ThreadMode 属性可以进行控制响应事件方法的执行线程,一共有 5 种线程模式可以进行配置

- **ThreadMode: POSTING** (默认)

订阅者(**Subscriber**) 在 发布者(Publisher) post 事件相同的线程中执行响应事件的方法,若 发布者(Publisher) post 的线程是主线程,则 订阅者(**Subscriber**) 处理事件的线程也在主线程中执行,若 发布者(Publisher) post 事件的线程是子线程,则 订阅者(**Subscriber**) 处理消息事件的线程也是在子线程中执行.

```
1 // Called in the same thread (default)
2 // ThreadMode is optional here
3 @Subscribe(threadMode = ThreadMode.POSTING)
4 public void onMessage(MessageEvent event) {
5     log(event.message);
6 }
```

- **ThreadMode: MAIN**

订阅者(**Subscriber**) 在主线程中执行响应事件的方法,如果 发布者(Publisher) post 事件是主线程,则直接调用响应事件的方法,如果 post 的是子线程,则加入到主线程的消息循环队列中执行响应事件的方法,

```
1 // Called in Android UI's main thread
2 @Subscribe(threadMode = ThreadMode.MAIN)
3 public void onMessage(MessageEvent event) {
4     textField.setText(event.message);
5 }
```

- **ThreadMode: MAIN_ORDERED**

订阅者(**Subscriber**) 在主线程中执行响应事件的方法 和 ThreadMode: MAIN 区别在于,不管 发布者(Publisher) post 事件是什么线程 ,MAIN_ORDERED会把事件加入到主线程的消息循环队列中执行,而不会直接调用处理消息的方法

```
1 // Called in Android UI's main thread
2 @Subscribe(threadMode = ThreadMode.MAIN_ORDERED)
3 public void onMessage(MessageEvent event) {
4     textField.setText(event.message);
5 }
```

- **ThreadMode: BACKGROUND**

订阅者(**Subscriber**) 在子线程中执行响应事件的方法.若 发布者(Publisher) post 事件为主线程,则在后台子线程中执行.所有的 ThreadMode: BACKGROUND 事件要转化在子线程处理的都共用一个相同的后台子线程 ,若 发布者(Publisher) post 事件的线程为子线程,则就直接在 post 事件的线程中处理.

```
1 // Called in the background thread
2 @Subscribe(threadMode = ThreadMode.BACKGROUND)
3 public void onMessage(MessageEvent event){
4     saveToDisk(event.message);
5 }
```

- **ThreadMode: ASYNC**

订阅者(**Subscriber**) 在独立的子线程中执行响应事件的方法,既不是 发布者(Publisher) post 事件的线程,不是主线程,也不是 ThreadMode: BACKGROUND 的后台子线程 .发布者(Publisher) post 事件的线程不会等待订阅者(**Subscriber**) 处理事件的线程响应.适用于处理事件时间较长的情况

```
1 // Called in a separate thread
2 @Subscribe(threadMode = ThreadMode.ASYNC)
3 public void onMessage(MessageEvent event){
4     backend.send(event.message);
5 }
```

3.2. EventBus 配置

```
1 EventBus eventBus = EventBus.builder()
2     .logSubscriberExceptions(false) // 默认为 true 是否记录 调用订阅者响应事件的方法出现异常时
3     .logNoSubscriberMessages(false) // 默认为 true 是否记录 发布者(Publisher) 发布事件时没有订阅者
4     .sendNoSubscriberEvent(false) // 默认为 true 当发布者(Publisher) 发布事件时没有订阅者(Subscriber)
5     .sendSubscriberExceptionEvent(false) // 默认为 true 调用订阅者响应事件的方法出现异常时 是否
6     .throwSubscriberException(BuildConfig.DEBUG) // 默认为 false 调用订阅者响应事件的方法出现异常时
7     .eventInheritance(false) // 默认为 true 若 发布者(Publisher) 发布的事件是订阅者(Subscriber)
8     .ignoreGeneratedIndex(true) //默认为 false 是否忽略 Index
9     .strictMethodVerification(true) // 默认为 false ,是否严格认证 @Subscribe 注解标注的订阅者
10    .installDefaultEventBus();
```

3.3. Sticky 事件

Sticky 事件不同之处在于,当事件发布后,再有订阅者开始订阅该类型事件,依然能收到该类型事件最近一个 Sticky 事件。

- 订阅**Sticky**事件

```
1 // UI updates must run on MainThread
2 @Subscribe(sticky = true, threadMode = ThreadMode.MAIN)
3 public void onEvent(MessageEvent event) {
4     textField.setText(event.message);
5 }
```

- 发布**Sticky**事件

```
1 EventBus.getDefault().postSticky(new MessageEvent("Hello everyone!"));
```

- 移除**Sticky**事件

Sticky事件 在事件发布之后依然会接收的到,所以不会丢失,若不需要时必须手动移除

```
1 MessageEvent stickyEvent = EventBus.getDefault().getStickyEvent(MessageEvent.class);
2 // Better check that an event was actually posted before
3 if(stickyEvent != null) {
4     // "Consume" the sticky event
5     EventBus.getDefault().removeStickyEvent(stickyEvent);
6     // Now do something with it
7 }
```

移除 Sticky事件 还有一个重载的方法

```
1 | MessageEvent stickyEvent = EventBus.getDefault().removeStickyEvent(MessageEvent.class)
2 | // Better check that an event was actually posted before
3 | if(stickyEvent != null) {
4 |     // Now do something with it
5 | }
```

3.4. 优先级和取消事件传递

- 优先级

默认优先级是 0,高优先级的的订阅者先接收到事件.只有相同在ThreadMode 下才能比较优先级不同的 ThreadMode下的订阅者的优先级别不起作用

```
1 | @Subscribe(priority = 1);
2 | public void onEvent(MessageEvent event) {
3 |     ...
4 | }
```

- 取消事件传递

只有 和 发布者(Publisher) post 事件相同的线程的 订阅者(**Subscriber**) 才能取消事件传递,不然的话会报异常

```
1 | // Called in the same thread (default)
2 | @Subscribe
3 | public void onEvent(MessageEvent event){
4 |     // Process the event
```



```
5    ...
6    // Prevent delivery to other subscribers
7    EventBus.getDefault().cancelEventDelivery(event) ;
8 }
```

3.5. Index(索引)

Index是 EventBus 3 上添加的新特性,默认是用使用反射,而 Index 是编译时 使用 annotationProcessor 生成辅助的 SubscriberInfoIndex 类 ,里面会记录订阅者信息,就不用反射扫描类中方法的,所以 Android上推荐使用 Index ,效率更高

- 使用 **annotationProcessor** 生成 Index

```
1  android {
2      defaultConfig {
3          javaCompileOptions {
4              annotationProcessorOptions {
5                  arguments = [ eventBusIndex : 'com.example.myapp.MyEventBusIndex' ]
6              }
7          }
8      }
9  }
10
11 dependencies {
12     implementation 'org.greenrobot:eventbus:3.1.1'
13     annotationProcessor 'org.greenrobot:eventbus-annotation-processor:3.1.1'
14 }
```

- 使用 **index**

配置好 `annotationProcessor` 后使用 `index` 和之前的唯一区别是 `EventBus.getDefault()` 之前必须使用 `addIndex` 方法进行初始化

```
1 EventBus.builder().addIndex(new MyEventBusIndex()).installDefaultEventBus();
2 // Now the default instance uses the given index. Use it like this:
3 EventBus eventBus = EventBus.getDefault();
```

3.6. AsyncExecutor 辅助类

`AsyncExecutor` 是一个辅助类会创建一个线程池,在 `RunnableEx` 里面执行出现异常会被捕获不用自己处理,并被转化为一个 `ThrowableFailureEvent` 事件并 `post` 出去


```
1 AsyncExecutor.create().execute(
2     new AsyncExecutor.RunnableEx() {
3         @Override
4         public void run() throws LoginException {
5             // No need to catch any Exception (here: LoginException)
6             remote.login();
7             EventBus.getDefault().postSticky(new LoggedInEvent());
8         }
9     }
10 );
```


```
1
2 @Subscribe(threadMode = ThreadMode.MAIN)
3 public void handleLoginEvent(LoggedInEvent event) {
4     // do something
5 }
6
```

```
7 | @Subscribe(threadMode = ThreadMode.MAIN)
8 | public void handleFailureEvent(ThrowableFailureEvent event) {
9 |     // do something
10 | }
```

 2人点赞 >



 Android学习



"小礼物走一走，来简书关注我"

赞赏支持

还没有人赞赏，支持一下



changchengfeng

总资产19 (约1.41元) 共写了1.3W字 获得19个赞 共8个粉丝

关注



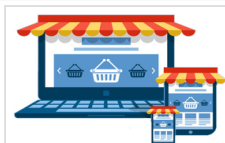
霍山铁皮石斛



脸上两边是肝斑



设计logo



多用户商城系统



喜茶加盟多少钱



服务器要多少钱

推荐阅读

更多精彩内容 >

Android框架之路——EventBus的使用

一、简介 EventBus是由greenrobot 组织贡献的一个Android事件发布/订阅轻量级框架。Even...



Mr \ sorrow 阅读 13,622 评论 0 赞 13

Android消息传递EventBus的使用

好久没更新博客了，思来想去，时隔半年又重新回来了 最近项目更新想用下greenrobot的eventbus，之前...



Luke_单车 阅读 6,720 评论 0 赞 10

Android中EventBus的使用

什么是EventBus EventBus是Android下高效的发布/订阅事件总线机制。作用是可以代替传统的Int...



时光磨棱角 阅读 593 评论 0 赞 1

4. EventBus的使用

EVentBus的使用： 简介下载地址使用步骤粘性事件例子 1.简介 EventBus是一个Android端优化的...



妖颜TMD祸众 阅读 105 评论 0 赞 0



EventBus的使用

EventBus是一个Android平台上的事件发送/订阅框架，采用观察者模式实现，可以优化组件间的信息传递过程。...



华枯荣 阅读 3,185 评论 2 赞 3

