Guava 事件总线(EventBus) 分析报告

吴双 2016K8009937003

事件总线(Eventbus)是Guava中的进程内事件分发机制，用于管理事件的注册和分发。它允许组件间的发布-订阅式通信，而不需要进行显式注册。事件总线并不适用与进程间的通信。

事件总线主要包括以下几个组件：发布者、订阅者、事件总线、事件通道、事件监听器。其工作流程为：订阅者在事件总线中注册要监听的事件，将这些订阅方法和订阅对象存储在map中。当发布者在特定的通道上发布一个事件时,事件总线根据事件的参数类型和tag找到对应的订阅者对象,最后执行订阅者对象中的方法。这些订阅方法会执行在发布者指定的线程模型中，事件总线会通知订阅者从这些特定通道上获取事件消息。值得一提的是发布者同时也可以是订阅者。

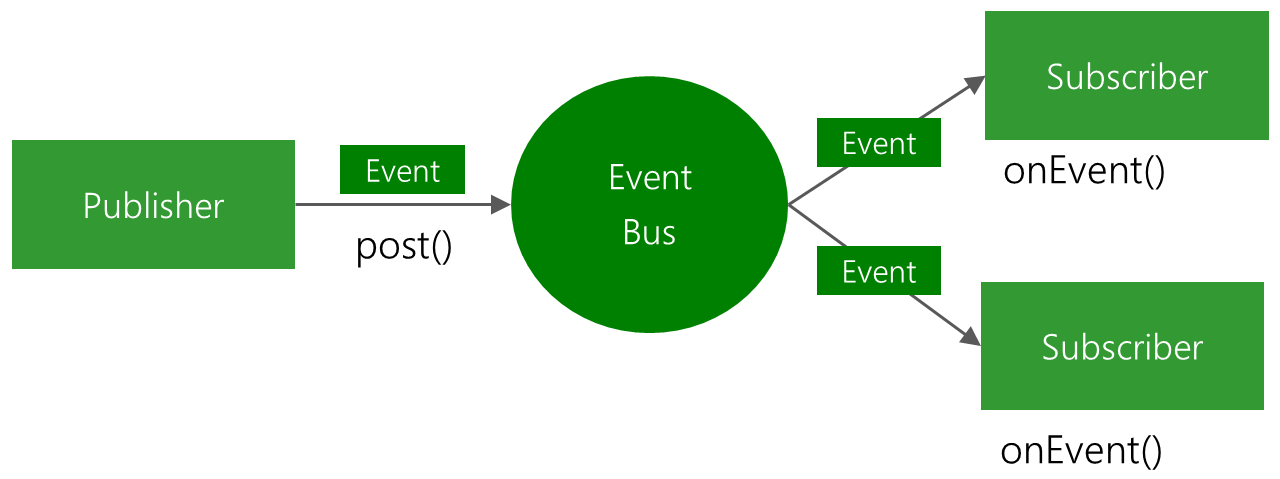


图1 Event Bus 的工作机制 （图源：<http://greenrobot.org/eventbus/>）

Event Bus主要需要实现的功能有：

1.订阅者在事件总线中注册/订阅事件；

2.发布者发布事件到事件总线中的特定通道，事件总线完成事件分发。

源码部分：

Event Bus 构造函数：

EventBus(EventBusBuilder builder) {

this.currentPostingThreadState = new ThreadLocal() {

protected EventBus.PostingThreadState initialValue() {

return new EventBus.PostingThreadState();

}

};

this.subscriptionsByEventType = new HashMap();

this.typesBySubscriber = new HashMap();

this.stickyEvents = new ConcurrentHashMap();

this.mainThreadPoster = new HandlerPoster(this, Looper.getMainLooper(), 10);

this.backgroundPoster = new BackgroundPoster(this);

this.asyncPoster = new AsyncPoster(this);

this.subscriberMethodFinder = new SubscriberMethodFinder(builder.skipMethodVerificationForClasses);

this.logSubscriberExceptions = builder.logSubscriberExceptions;

this.logNoSubscriberMessages = builder.logNoSubscriberMessages;

this.sendSubscriberExceptionEvent = builder.sendSubscriberExceptionEvent;

this.sendNoSubscriberEvent = builder.sendNoSubscriberEvent;

this.throwSubscriberException = builder.throwSubscriberException;

this.eventInheritance = builder.eventInheritance;

this.executorService = builder.executorService;

}

1.订阅者注册(register)

// Registers all subscriber methods on {@code object} to receive events.

public void register(Object object) {

// Multimap是guava自定义数据结构，类似Map<K, Collection<V>>，key就是事件类型，例如DMLExecutionEvent，value就是EventSubscriber即事件订阅者集合（说明，这个的订阅者集合是指object里符合订阅者的所有方法，例如DMLExecutionEventListener.listener()，DMLExecutionEventListener中可以有多个订阅者，注解@Subscribe即可），

Multimap<Class<?>, EventSubscriber> methodsInListener =

finder.findAllSubscribers(object);

// 获得写锁保证线程安全

subscribersByTypeLock.writeLock().lock();

try {

// 把订阅者信息放到map中缓存起来（发布事件post()时就会用到）

subscribersByType.putAll(methodsInListener);

} finally {

//释放写锁

subscribersByTypeLock.writeLock().unlock();

}

}

2. public void post(Object event) {

// 得到所有该类以及它的所有父类（因为有些注册监听器是监听其父类）

Set<Class<?>> dispatchTypes = flattenHierarchy(event.getClass());

boolean dispatched = false;

for (Class<?> eventType : dispatchTypes) {

//获得读锁

subscribersByTypeLock.readLock().lock();

try {

// 得到事件类的所有订阅者，例如DMLExecutionEvent的订阅者就是DMLExecutionEventListener（EventSubscriber有两个属性：重要的属性target和method，target就是监听器即DMLExecutionEventListener，method就是监听器方法即listener；从而知道DMLExecutionEvent这个事件由哪个类的哪个方法监听处理）

Set<EventSubscriber> wrappers = subscribersByType.get(eventType);

if (!wrappers.isEmpty()) {

// 如果有时间订阅者，那么dispatched = true，表示该事件可以分发

dispatched = true;

// 遍历所有的时间订阅者，每个订阅者的队列都增加该事件

for (EventSubscriber wrapper : wrappers) {

enqueueEvent(event, wrapper);

}

}

} finally {

subscribersByTypeLock.readLock().unlock();

}

}

if (!dispatched && !(event instanceof DeadEvent)) {

post(new DeadEvent(this, event));

}

dispatchQueuedEvents();

}

参考文献链接：

1. <https://www.cnblogs.com/moonandstar08/p/5651793.html>

2. <https://blog.csdn.net/feelwing1314/article/details/80335164>

3. <http://ifeve.com/google-guava-eventbus/>

4. <https://blog.csdn.net/u012070360/article/details/60141106#>