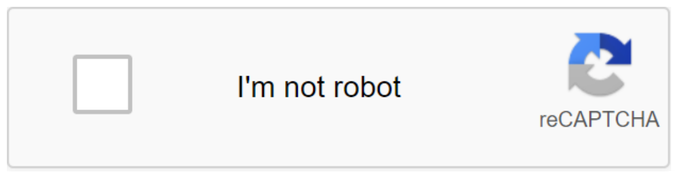
Compositedisposable rxjava 2 android



Continue

Lately, many people have been excited about the new way of writing applications, which is called jet programming. It's all gone to Android. Let's try to understand this topic. The overall page on jet programming is . Class documentation is here. The GitHub home for RxJava is . Keep in mind that Google does not officially support this direction. But some programmers from the company use reactive programming in their own projects. One of them even wrote his own version of the library, similar to RxJava, which was used in one of Google's applications. Switching to RxJava 2 Since some time there has been a split of the version into two branches: 1.x and 2.x. Branch 1.x was frozen on June 1, 2017 (bug fix only). The branch will be closed on March 31, 2018. I started studying the topic based on 1.x branches, so don't be surprised if you come across old examples for the first version. I will try to clearly warn about such cases, as the differences are significant. We connect RxJava. For the 2.x branch, the package name has changed so there are no conflicts. The difference between the two branches is described on the documentation website. Common fundamental concepts remain the same. Some types of Action and Function classes have been renamed or removed. Action Action Action1 - BiConsumer ActionN - Consumer's Action3 - Action3 - Action3 - Action3 - --gt; removed Func0 - 'gt; Callable Func1 - Function Func2 - As well as CompositeSubscription in CompositeDisposable. Classes In RxJava have a huge number of scary words to learn. Observable Observer Subject, </Object[]>also PublishSubject, AsyncSubject, BehaviorSubject, ReplaySubject Processor - subspecies Subject with BackPressure support. AsyncProcessor, BehaviorProcessor, PublishProcessor, ReplayProcessor, UnicastProcessor. Future Single is the lazy equivalent of Future. Maybe Completable Consumer Disposable - Former Subscription from RxJava 1.x Scheduler Flowable Observable and Observer There are two entities: manufacturers and consumers (Observables and Observers). The general idea is to implement three tasks - to create a stream of data, to bring the data in the right form, to get the corrected data. First acquaintance. Observable and Observer Hot and Cold Observable. ConnectableObservable Single Maybe Completable TestObserver. Testing (Kotlin) Operators In RxJava have special operators with which you can create new Observable or change existing ones. In Java, operators are implemented in the form of methods. Typically, the name of the operator coincides with the name of the method or slightly differs from the presence of additional prefixes or suffixes. The full operator page is presented in alphabetical order in the documentation. It is advisable to walk on your own on all operators to know their capabilities. Suddenly you need a specific operator for your tasks. We will consider only a part of popular operators. There is also an interactive RxMarbles scheme for all operators, allowing you to move the data to see the operator's work in action. The scheme is also called pebble diagrams. Pebble diagrams illustrate the work of operators. Typically, the diagram contains two horizontal time axes, directed from left to right. Figures on diagrams (stones) serve to visualize events. There are three types of pebbles - a circle, a pentagon and a triangle. If the thread is processed then on the right is a vertical devil. In the case of an error, a cross is drawn. If the thread never ends, nothing is drawn on the timeline. Between the a downs is an operator that changes the sequence of events coming from the original Observable and transmitted to the resultant. In most cases, the operators run in sync. You create a chain of operators and they are consistently executed. This is their power when we begin to combine them. Clutching multiple operators, branching the flow into multiple sub-streams and merging them - you need to learn some of the popular operators to use them correctly. Operators create Observable Transformation And Filtration Operators Combining Conditional and Bulev Operators Other Operators Other classes Scheduler Backpressure Flowable Subject PublishSubject Disposable Library RxLifecycle - trello/RxLifecycle: Lifecycle handling APIs for Android apps using RxJava. The author of the library decided to abandon his library, but so far supports it. RxBinding - JakeWharton/RxBinding: RxJava binding APIs for Android's UI widgets. Rx Preferences - f2prateek/rx-preferences: Reactive SharedPreferences for Android AutoDispose - uber/AutoDispose: Automatic binding+disposal of RxJava 2 streams. RxKotlin - Kotlin also got his library. Advertising java.lang.Object io.reactivex.disposables.CompositeDisposable All Implemented Interfaces: Disposable, io.reactivex.internal.disposables.DisposableContainer public final class CompositeDisposables extends Object implements Disposable, io.reactivex.internal.disposables.Container A disposable container that can hold a large number of different clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait public CompositeDisposable() Creates an empty CompositeDisposable. public CompositeDisposable(@NonNull Disposable... disposables) Creates a CompositeDisposables with the given array of initial elements. Disposable - an array of disposables to start with Throws: NullPointerException - if disposable or any of its array elements is a zero-public CompositeDisposable (@NonNull Iterable? expands disposable) creates CompositeDisposables with a given Iterable sequence of initial elements. Options: Disposable - Iterable sequence disposable to start with Throws: NullPointerException - if a disposable or any of its items is an invalid public void dispose () Description copied from the interface: Disposable recycle resource, operation should be idempotent. Indicated: Dispose of the One-Time Public boolean isDisposed interface () Description copied from the interface: Disposable returns are true if this resource has been removed. Indicated: Removed in the interface Disposable Returns: true if this resource has been removed public boolean add (@NonNull disposable disposable) adds disposable to this container or recycles it if the container has been disposed of. Indicated: add in the interface io.reactivex.internal.disposables.DisposableContainer Options: disposable - disposable - disposable to add, not zero returns: true, if successful, false if this container has been removed Throws: NPointullerException - if disposable is a zero public boolean addAll (@NonNull Disposable ... Atomicly adds this array disposable to the container or disposes of them all if the container has been disposed of. Options: Disposable - array disposable returns: true if the operation was successful, false if the container has been disposed of by Throws: NullPointerException - if disposable or any of its array items is zero public boolean remove (@NonNull Disposable disposable) removes and disposes of this disposable if it is part of that container. Indicated: remove in the interface io.reactivex.internal.disposableContainer Options: disposable - disposable for removal and recycling, not zero Returns: true, if the operation was a successful public boolean remove (@NonNull Disposable disposable) Removes (but do not dispose) of this disposable if it is part of this container. Indicated: remove in the interface io.reactivex.internal.disposables.DisposableContainer Options: disposable - disposable for removal, not zero Returns: true if the operation was successful Throws: NullPointerException - if disposable is invalid public void clearly () cleans the atomic container and then disposes of all previously contained disposable. Public size int () Returns the number of currently held disposable. Returns: The number of disposable disposables currently held