Rxjava汇总

创建Observable的方法有：from just

|  |
| --- |
| Observable<String> o = Observable.from("a", "b", "c");  def list = [5, 6, 7, 8]Observable<Integer> o = Observable.from(list);  Observable<String> o = Observable.just("one object"); |

对于Observable要发出的每个项，这些转换后的Observable将同步调用订阅它们的任何订阅者的onNext（）方法，然后调用订阅者的onCompleted（）方法。

just可用来创建单挑数据，数字，字符串，数组，对象集合

Consumer可以当作是观察者Observer功能的单一化产物-消费者

比如：

|  |
| --- |
| public void demo4() {  Observable.just("Hello World")  .subscribe(new Consumer<String>() {  @Override  public void accept(@NonNull String s) throws Exception {  System.out.println(s);  }  });  }  使用Observer时我们需要实现三个方法onNext onError onComplete  如我我们想通过Consumer也接收到完整的信息可以通过如下方法：  public void demo4() {  Observable.just("Hello World")  .subscribe(new Consumer<String>() {  @Override  public void accept(@NonNull String s) throws Exception {  System.out.println(s);  //接收的消息  }  }, new Consumer<Throwable>() {  @Override  public void accept(@NonNull Throwable throwable) throws Exception {  throwable.printStackTrace();  //接收的异常消息  }  }, new Action() {  @Override  public void run() throws Exception {  System.out.println("接受完成");  }  });  } |

通过create方法创建Observable

下面我将完整的写一个关于订阅与订阅者的代码：

|  |
| --- |
| //被观察者者通过create方式创建  //在subscribe(ObservableEmitter e)这个方法里，我们可以通过e向观察者发射任何想发射的数据  //ObservableEmitter 是一个接口类 同时 也继承类Emitter这个接口类，Emitter里面有三个抽象方法分别为：//onNext onError onComplete  E05F22E7-D3BE-4A38-818A-369D83B0716C  //e通过调用onComplete()方法告诉观察者消息发送完毕,然后与此同时观察者可以通过Disponsable中的//disponse取消连接  //创建观察者  Observable observable = Observable.*create*(**new** ObservableOnSubscribe() {  @Override  **public void** subscribe(ObservableEmitter e) **throws** Exception {  *//ObservableEmitter是一个发射器* print\_content(**"发送线程"** + Thread.*currentThread*().getName());  print\_content(**"发送：hello"**);  e.onNext(**"hello"**);*//想观察者发送消息，观察者接收到消息后就将这条信息打印出来* print\_content(**"发送：world"**);  e.onNext(**"world"**);   print\_content(**"发送hello world"**);  e.onNext(**"Hello world"**);   print\_content(**"发送onComplete"**);  e.onComplete();  } });  //创建接收者：  Observer<String> observer = **new** Observer<String>() {  @Override  **public void** onSubscribe(Disposable d) {  //通过这个d我们可以取消连接 d.disponse();  //过Disposable来控制Observer与Observable之间的订阅。  print\_content(**"onSubscribe"**);  }   @Override  **public void** onNext(String s) {  print\_content(**"接收到线程："** + Thread.*currentThread*().getName());  print\_content(**"接收："** + s);  }   @Override  **public void** onError(Throwable e) {  print\_content(**"onerror"**);  }   @Override  **public void** onComplete() {  print\_content(**"onComplete"**);  } };  //建立被观察者和观察者间的联系  observable.subscribe(observer);  测试结果：  952660E4-9F34-4A38-8EC9-282A3EF0C4B3  如上代码可以简写如下：  **void** logogram() {  Observable.*create*(**new** ObservableOnSubscribe<String>() {  @Override  **public void** subscribe(ObservableEmitter<String> e) **throws** Exception {  *//ObservableEmitter是一个发射器* print\_content(**"发送线程"** + Thread.*currentThread*().getName());   print\_content(**"发送：hello"**);  e.onNext(**"hello"**);*//想观察者发送消息，观察者接收到消息后就将这条信息打印出来* print\_content(**"发送：world"**);  e.onNext(**"world"**);   print\_content(**"发送hello world"**);  e.onNext(**"Hello world"**);   print\_content(**"发送onComplete"**);  e.onComplete();  }  }).subscribe(**new** Observer<String>() {  @Override  **public void** onSubscribe(Disposable d) {   print\_content(**"onSubscribe"**);   }  @Override  **public void** onNext(String s) {  print\_content(**"接收到线程："** + Thread.*currentThread*().getName());  print\_content(**"接收："** + s);  }   @Override  **public void** onError(Throwable e) {  print\_content(**"onerror"**);  }   @Override  **public void** onComplete() {  print\_content(**"onComplete"**);   }  }); } |

上面我们看到被观察者和观察者是通过subscribe这个方法建立联系的

但是在有的时候，我们会需要通过subscribeWith这个方法来建立联系，

他两者的差别:

subscribe的返回类型是void类型。

subscribeWith的返回类型是observer类型。

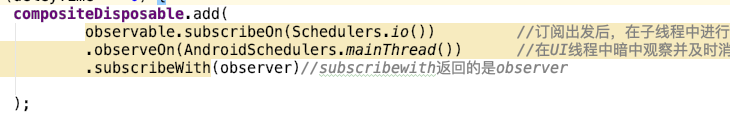
我们在后面的开发中经常回使用CompositeDisposable来管理我们的Observable和Observer。

CompositeDisposable是一个disposable的容器，可以容纳多个disposable，并添加和去除。通常都用CompositeDisposable来管理订阅事件disposable。在activity销毁的时候调用compositeDisposable.dispose()来切断所有订阅事件

CompositeDisposable通过add方法添加observer所以这时我们需要通过使用subscribewith方法。

#### CompositeDisposable类

1. 可以快速解除所有添加的Disposable类.  
   2、每当我们得到一个Disposable时就调用CompositeDisposable.add()将它添加到容器中, 在退出的时候, 调用CompositeDisposable.clear() 即可快速解除.



subscribeOn通过接收一个Scheduler参数，来指定对数据的处理运行在特定的线程调度器Schedular上。

observeOn同样接收一个Scheduler参数，用来指定下游操作运行在特定的线程调度器Scheduler上