Rxjava的思想就是观察者模式（将用户界面作为一个观察者，业务数据为被观察者，用户界面观察业务数据的变化，发现数据变化后就显示在界面上）

Rxjava关于Disposable

Disposable类

dispose():主动解除订阅

isDisposed():查询是否解除订阅 true 代表 已经解除订阅

取消订阅使用CompositeDisposable

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.

<!-- 以下这部分总结来自博客：<https://www.jianshu.com/p/2241d4238deb> -->

Retrofit与RxJava结合使用产生了Observable接口的使用



传统的用callback做网络请求：



现在用RxJava方式做网络请求：



|  |
| --- |
| 代码：  // 创建Retrofit对象  Retrofit retrofit = new Retrofit.Builder()  .baseUrl("http://fy.iciba.com/")  .addConverterFactory(GsonConverterFactory.create())  .addCallAdapterFactory(RxJava2CallAdapterFactory.create())  .build();  // 创建网络请求接口的实例  GetTranslationInterface request = retrofit.create(GetTranslationInterface.class);  // 封装请求  Observable<Translation> observable = request.getCall();  // 发送网络请求,子线程请求，主线程更新  observable.subscribeOn(Schedulers.io())  .observeOn(AndroidSchedulers.mainThread())  .subscribe(new Observer<Translation>() {  @Override  public void onSubscribe(Disposable d) {  Log.d(TAG, "开始采用subscribe连接");  }  @Override  public void onNext(Translation translation) {  translation.show();  }  @Override  public void onError(Throwable e) {  Log.d(TAG, "请求失败");  }  @Override  public void onComplete() {  Log.d(TAG, "请求成功");  }  });  } |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

下面总结学习来自博客：<https://www.jianshu.com/p/cd3557b1a474>

|  |
| --- |
| **public class** Test {  *//创建被观察者  // void 被观察者(){  //Observable中文意思：被观察者，通过create生成对象里面放入参数  //ObservableOnSubscribe可以理解为计划表。重写subscribe方法里面放入具体的计划* Observable **observable** = Observable.*create*(**new** ObservableOnSubscribe<String>() {  *//subscribe里的OnservableEmitter对象的emitter是发射器的意思  //ObservableEmitter有三种发射器分别为onNext() onError() onComplete()  //在onNext里面任何都能就收到，而onError和onComplete两者是互斥的* @Override  **public void** subscribe(ObservableEmitter<String> emitter) **throws** Exception {  emitter.onNext(**"连载1"**);  emitter.onNext(**"连载2"**);  emitter.onComplete();  }  });  *// }   //创建观察者  // void 观察者(){* Observer<String> **reader** = **new** Observer<String>() {   @Override  **public void** onSubscribe(Disposable d) {    }   @Override  **public void** onNext(String value) {   }   @Override  **public void** onError(Throwable e) {   }   @Override  **public void** onComplete() {   }  };  *// }* **void** ds(){  **observable**.subscribe(**reader**);*//将观察者和被观察者建立关系* }   */\*\*  \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*下面将上面的代码实现异步\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*/  //RxJava实现异步的方式是通过observeOn(AndroidSchedulers.mainThread())和subscribeOn(Schedulers.io())这两个方法  //observeOn是事件回调的线程，AndroidSchedulers.mainThread()一看就知道是主线程  //Schedulers.io()是子线程* **void** ddss(){   Observable.*create*(**new** ObservableOnSubscribe<String>() {  @Override  **public void** subscribe(ObservableEmitter<String> emitter) **throws** Exception {  emitter.onNext(**"连载1"**);  emitter.onNext(**"连载2"**);  emitter.onNext(**"连载3"**);  emitter.onComplete();    }  }).observeOn(AndroidSchedulers.*mainThread*())*//回调在主线程* .subscribeOn(Schedulers.*io*())*//执行在io线程* .subscribe(**new** Observer<String>() {  @Override  **public void** onSubscribe(Disposable d) {   }   @Override  **public void** onNext(String value) {   }   @Override  **public void** onError(Throwable e) {   }   @Override  **public void** onComplete() {   }  }); } } |

GreenDao的学习：

|  |
| --- |
| insert 插入一条数据  实体对象为参数  insertInTx 批量插入数据  List<>型参数  insertOrReplace 插入数据，传入的对象主键如果存在于数据库中，有则更新，否则插入  实体对象为参数  insertOrReplaceInTx 批量插入数据 List<>型参数  save 插入数据，判断对象是否有Key值，有则更新，否则插入  实体对象为参数  insertOrReplace：  会去重，保存最新的数据，也就是会进行更新 |