链滴

RxAndroid2.x 源码分析

作者：hiquanta

原文链接：https://ld246.com/article/1500951680234

来源网站：链滴

RxAndroid其实就是对Android 的handler， looper及Message的封装，使替变为基于观察者模式调用。理解其源码并不困难，关键在于要先弄清Andriod中handler，looper及Message的关系，才理清RxAndroid2.x的源码。这三者的关系网上的资料一大堆，我就不重复了。

由于Android UI的操作是单线程且非线程安全的，因此不可以把耗时的操作放入主线程即UI线程中去否则会引发ANR异常，Android提供了AsyncTask，及handler机制来进行线程切换，RxAndroid对handler，looper，Message进行了封装提供了另外一种思路

在分析源码之前还是先看下如何使用RxAndroid2.x，2.x和1.x试用方式基本上没啥变化，直接拿官方例：

```
public class MainActivity extends Activity {
   private static final String TAG = "RxAndroidSamples";
 //一个存放事件源或被观察者的容器
 private final CompositeDisposable disposables = new CompositeDisposable();

 @Override
 protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
 setContentView(R.layout.main_activity);
 findViewById(R.id.button_run_scheduler).setOnClickListener(new View.OnClickListener() {
         @Override
 public void onClick(View v) {
             onRunSchedulerExampleButtonClicked();
 }
     });
 }

  @Override
 protected void onDestroy() {
     super.onDestroy();
 disposables.clear();
 }

  void onRunSchedulerExampleButtonClicked() {
     disposables.add(sampleObservable()
             // Run on a background thread
             //耗时的操作放在io线程中
 .subscribeOn(Schedulers.io())
             // Be notified on the main thread
//回调操作放在主线程  .observeOn(AndroidSchedulers.mainThread())
             .subscribeWith(new DisposableObserver() {
                 @Override
 public void onComplete() {
                     Log.d(TAG, "onComplete()");
 }

                 @Override
 public void onError(Throwable e) {
                     Log.e(TAG, "onError()", e);
 }

                 @Override
 public void onNext(String string) {
                     Log.d(TAG, "onNext(" + string + ")");
```

```
  }
            }));
  }

    //事件源，即被观察者
  static Observable sampleObservable() {
       return Observable.defer(new Callableextends String>>() {
           @Override
  public ObservableSourceextends String> call() throws Exception {
              // Do some long running operation
  SystemClock.sleep(5000);
 return Observable.just("one", "two", "three", "four", "five");
  }
       });
  }
}
```

界面布局我就不放了，就一个布局文件，而且官方实例当中都有。

核心代码就两行

```
// Run on a background thread
//耗时的操作放在io线程中
.subscribeOn(Schedulers.io())
// Be notified on the main thread
//回调操作放在主线程
.observeOn(AndroidSchedulers.mainThread())
```

怎么样，和原来的代码书写方式比起来是不是简单了很多，只用两行代码！

直接看RxAndroid2.x的源码目录结构：

可以看到类不多只有四个类

RxAndroidPlugins，AndroidSchedulers，HandlerScheduler，MainThreadDisposable

这里根据流程来跟踪下源代码，

先看AndroidSchedulers中的mainThread()

```
  public static Scheduler mainThread() {
     return RxAndroidPlugins.onMainThreadScheduler(MAIN_THREAD);
  }
```

可以看到其返回一个Scheduler，其中MAIN_THREAD为主线程的调度器，

```
static final Scheduler DEFAULT = new HandlerScheduler(new Handler(Looper.getMainLooper(
));
```

内部调用了RxAndroidPlugins的onMainThreadScheduler方法，其代码如下

```
    public static Scheduler onMainThreadScheduler(Scheduler scheduler) {
      if (scheduler == null) {
          throw new NullPointerException("scheduler == null");
      }
      Function<Scheduler, Scheduler> f = onMainThreadHandler;
      if (f == null) {
          return scheduler;
```

```
    }
    return apply(f, scheduler);
  }
```

可以看到onMainThreadHandler是一个 Fuction类型，并对齐应用了主线程的调度器,返回给RxJava度

```
static <T, R> R apply(Function<T, R> f, T t) {
  try {
    return f.apply(t);
} catch (Throwable ex) {
    throw Exceptions.propagate(ex);
}
}
```

这样就完成的线程的切换，是不是很简单！

HandlerScheduler提供了在其它线程中刷新ui的方法，具体调用看AndroidSchedulers的from(Looper looper)方法，具体操作和mainThread方法类似

MainThreadDisposable提供了资源释放的方法及对主线程的检验方法，也比较简单

总结：其使用核心代码

```
//耗时的操作放在io线程中
.subscribeOn(Schedulers.io())
// Be notified on the main thread
//回调操作放在主线程
.observeOn(AndroidSchedulers.mainThread())
```

仅需两行代码即可完成线程的切换