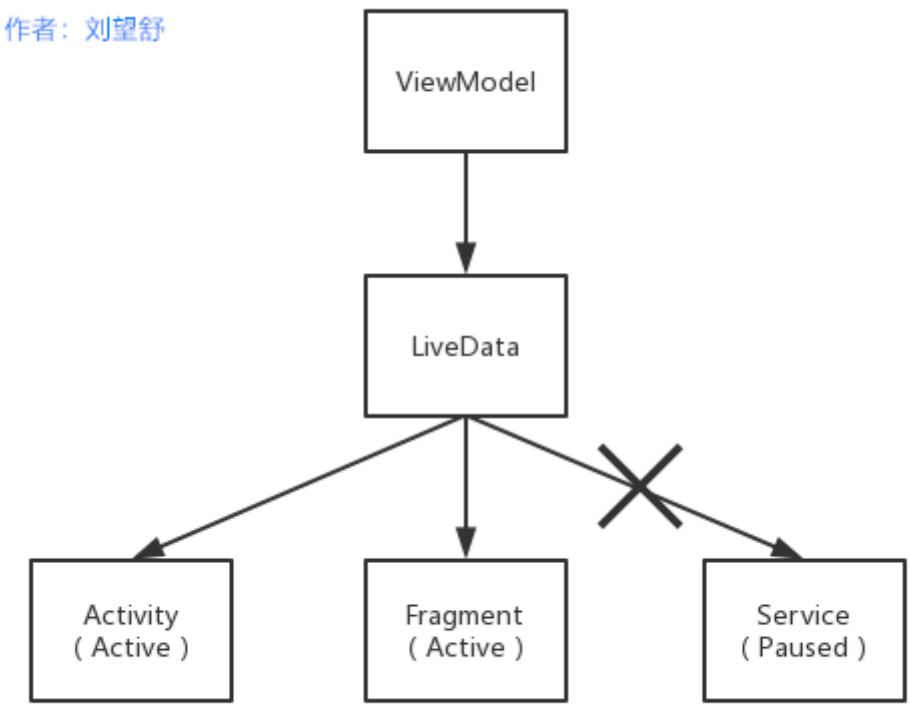
> 本文首发于微信公众号「后厂技术官」

前言

在2017年前后，RxJava一直很火，我在Android进阶三部曲第一部《Android进阶之光》中就介绍了RxJava的使用和原理。谷歌推出的LiveData和RxJava类似，也是基于观察者，你可以认为LiveData是轻量级的RxJava。起初LiveData并不被看好，随着谷歌的大力推广，LiveData也慢慢的进入了大家的视野。一般来说，LiveData很少单独使用，它更多的和Android Jetpack的其他组件搭配使用，比如和ViewModel。这篇文章就来介绍LiveData的使用。

1.什么是LiveData

LiveData如同它的名字一样，是一个可观察的数据持有者，和常规的observable不同，LiveData是具有生命周期感知的，这意味着它能够在Activity、Fragment、Service中正确的处理生命周期。



LiveData的数据源一般是ViewModel，也可以是其它可以更新LiveData的组件。当数据更新后，LiveData 就会通知它的所有观察者，比如Activiy。与RxJava的方法不同的是，LiveData并不是通知所有观察者，它 只会通知处于Active状态的观察者，如果一个观察者处于Paused或Destroyed状态，它将不会收到通知。

这对于Activiy和Service特别有用，因为它们可以安全地观察LiveData对象而不用担心内存泄漏的问

题。开发者也不需要在onPause或onDestroy方法中解除对LiveData的订阅。还有一点需要注意的是一旦观察者重新恢复Resumed状态，它将会重新收到LiveData的最新数据。

1.LiveData的基本用法

LiveData是一个抽象类，它的最简单的实现类为MutableLiveData，这里举个最简单的例子。

```java
public class MainActivity extends AppCompatActivity {
    private static final String TAG="MainActivity";
     @Override
     protected void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);
         setContentView(R.layout.activity_main);
         MutableLiveData<String> mutableLiveData  = new MutableLiveData<
         mutableLiveData.observe(this, new Observer<String>() {//1
             @Override
             public void onChanged(@Nullable final String s) {
                 Log.d(TAG, "onChanged:"+s);
             }
         });
         mutableLiveData.postValue("Android进阶三部曲");//2
     }
}
```

注释1处的observe方法有两个参数分别是LifecycleOwner和 `Observer<T>` ，第一个参数就是MainActivity本身，第二个参数新建了一个 `Observer<String>` ，在onChanged方法中得到回调。注释处的postValue方法会在主线程中更新数据，这样就会得到打印的结果。

D/MainActivity: onChanged:Android进阶三部曲

在大多数情况下，LiveData的observe方法会放在onCreate方法中，如果放在onResume方法中，会出现多次调用的问题。除了MutableLiveData的postValue方法，还可以使用setValue方法，它们之前的区别是，setValue方法必须在主线程使用，如果是在工作线程中更新LiveData，则可以使用postValue方法。

2.更改LiveData中的数据

如果我们想要在LiveData对象分发给观察者之前对其中存储的值进行更改，可以使用Transformations.map()和Transformations.switchMap()，下面通过简单的例子来讲解它们。

2.1 Transformations.map()

如果想要在LiveData对象分发给观察者之前对其中存储的值进行更改，可以使用Transformations.map()。

JAVA

```
public class MainActivity extends AppCompatActivity {
   private static final String TAG="MainActivity";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        MutableLiveData<String> mutableLiveData  = new MutableLiveData<
        mutableLiveData.observe(this, new Observer<String>() {
            @Override
            public void onChanged(@Nullable final String s) {
                Log.d(TAG, "onChanged1:"+s);
            }
        });
        LiveData transformedLiveData =Transformations.map(mutableLiveDa
            @Override
            public Object apply(String name) {
               return name + "+Android进阶解密";
            }
        });
        transformedLiveData.observe(this, new Observer() {
            @Override
            public void onChanged(@Nullable Object o) {
                Log.d(TAG, "onChanged2:"+o.toString());
            }
        });
        mutableLiveData.postValue("Android进阶之光");
    }
}
```

通过Transformations.map()，在mutableLiveData的基础上又加上了字符串"+Android进阶解密"。

打印结果为：

D/MainActivity: onChanged1:Android进阶之光

D/MainActivity: onChanged2:Android进阶之光+Android进阶解密

2.2 Transformations.switchMap()

如果想要手动控制监听其中一个的数据变化，并能根据需要随时切换监听，这时可以使用Transformations.switchMap()，它和Transformations.map()使用方式类似，只不过switchMap()必须返回一个LiveData对象。

JAVA

```java
public class MainActivity extends AppCompatActivity {
    private static final String TAG = "MainActivity";
    MutableLiveData<String> mutableLiveData1;
    MutableLiveData<String> mutableLiveData2;
    MutableLiveData<Boolean> liveDataSwitch;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mutableLiveData1 = new MutableLiveData<>();
        mutableLiveData2 = new MutableLiveData<>();
        liveDataSwitch = new MutableLiveData<Boolean>();//1

        LiveData transformedLiveData= Transformations.switchMap(liveDat
            @Override
            public LiveData<String> apply(Boolean input) {
                if (input) {
                    return mutableLiveData1;
                } else {
                    return mutableLiveData2;
                }
            }
        });

        transformedLiveData.observe(this, new Observer<String>() {
            @Override
            public void onChanged(@Nullable final String s) {
                Log.d(TAG, "onChanged:" + s);
            }
        });
        liveDataSwitch.postValue(false);//2
        mutableLiveData1.postValue("Android进阶之光");
        mutableLiveData2.postValue("Android进阶解密");
    }
}
```

注释1处新建一个 `MutableLiveData<Boolean>()` 来控制切换并赋值给liveDataSwitch，当liveDataSwitch的值为true时返回mutableLiveData1，否则返回mutableLiveData2。注释2处将liveDataSwitch的值更新为faske，这样输出的结果为"Android进阶解密"，达到了切换监听的目的。

3 合并多个LiveData数据源

MediatorLiveData继承自mutableLiveData，它可以将多个LiveData数据源集合起来，可以达到一个组件监听多个LiveData数据变化的目的。

JAVA

```java
public class MainActivity extends AppCompatActivity {
   private static final String TAG="MainActivity";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        MutableLiveData<String> mutableLiveData1  = new MutableLiveData
        MutableLiveData<String> mutableLiveData2  = new MutableLiveData
        MediatorLiveData liveDataMerger = new MediatorLiveData<String>(
        liveDataMerger.addSource(mutableLiveData1, new Observer() {
            @Override
            public void onChanged(@Nullable Object o) {
                Log.d(TAG, "onChanged1:"+o.toString());
            }
        });

        liveDataMerger.addSource(mutableLiveData2, new Observer() {
            @Override
            public void onChanged(@Nullable Object o) {
                Log.d(TAG, "onChanged2:"+o.toString());
            }
        });
        liveDataMerger.observe(this, new Observer() {
            @Override
            public void onChanged(@Nullable Object o) {
                Log.d(TAG, "onChanged:"+o.toString());
            }
        });
        mutableLiveData1.postValue("Android进阶之光");
    }
}
```

为了更直观的举例，将LiveData和MediatorLiveData放到了同一个Activity中。通过MediatorLiveData的addSource将两个MutableLiveData合并到一起，这样当任何一个MutableLiveData数据发生变化时，MediatorLiveData都可以感知到。

打印的结果为：

D/MainActivity: onChanged1:Android进阶之光

4 拓展LiveData对象

如果观察者的生命周期处于STARTED或RESUMED状态，LiveData会将观察者视为处于Active状态。关于如何扩展LiveData，官网的例子是比较简洁的，如下所示。

JAVA

```
public class StockLiveData extends LiveData<BigDecimal> {
    private static StockLiveData sInstance;
    private StockManager stockManager;

    private SimplePriceListener listener = new SimplePriceListener() {
        @Override
        public void onPriceChanged(BigDecimal price) {
            setValue(price);
        }
    };

    @MainThread
    public static StockLiveData get(String symbol) {
        if (sInstance == null) {
            sInstance = new StockLiveData(symbol);
        }
        return sInstance;
    }

    private StockLiveData(String symbol) {
        stockManager = new StockManager(symbol);
    }

    @Override
    protected void onActive() {
        stockManager.requestPriceUpdates(listener);
    }

    @Override
    protected void onInactive() {
        stockManager.removeUpdates(listener);
    }
}
```

上面的代码是一个观察股票变动的一个例子，对LiveData进行了拓展，实现了LiveData的两个空方法onActive和onInactive。当Active状态的观察者的数量从0变为1时会调用onActive方法，通俗来讲，就是当LiveData对象具有Active状态的观察者时调用onActive方法，应该在onActive方法中开始观察股票价格的更新。当LiveData对象没有任何Active状态的观察者时调用onInactive方法，在这个方法中，断开与StockManager服务的连接。

在Fragment中使用StockLiveData，如下所示。

JAVA

```java
public class MyFragment extends Fragment {
    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        StockLiveData.get(symbol).observe(this, price -> {
            // Update the UI.
        });
    }
}
```

总结

这篇文章主要介绍了什么是LiveData，以及LiveData的使用方法，这里没有介绍LiveData和ViewModel的结合使用，以及LiveData的原理，这些会在后面的文章进行介绍。