Android JetPack架构之LifeCycles+LiveData+ViewModel 使用与源码解析

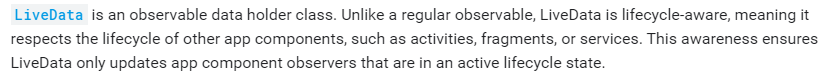
之二：LiveData

上面一篇文章我们介绍了JetPack之LifeCycle组件，这篇文章就是主要讲解如何编写生命周期敏感的数据，并且分析LiveData源码与实现方式

问题：

LiveData是什么？

下面是官网介绍截图：

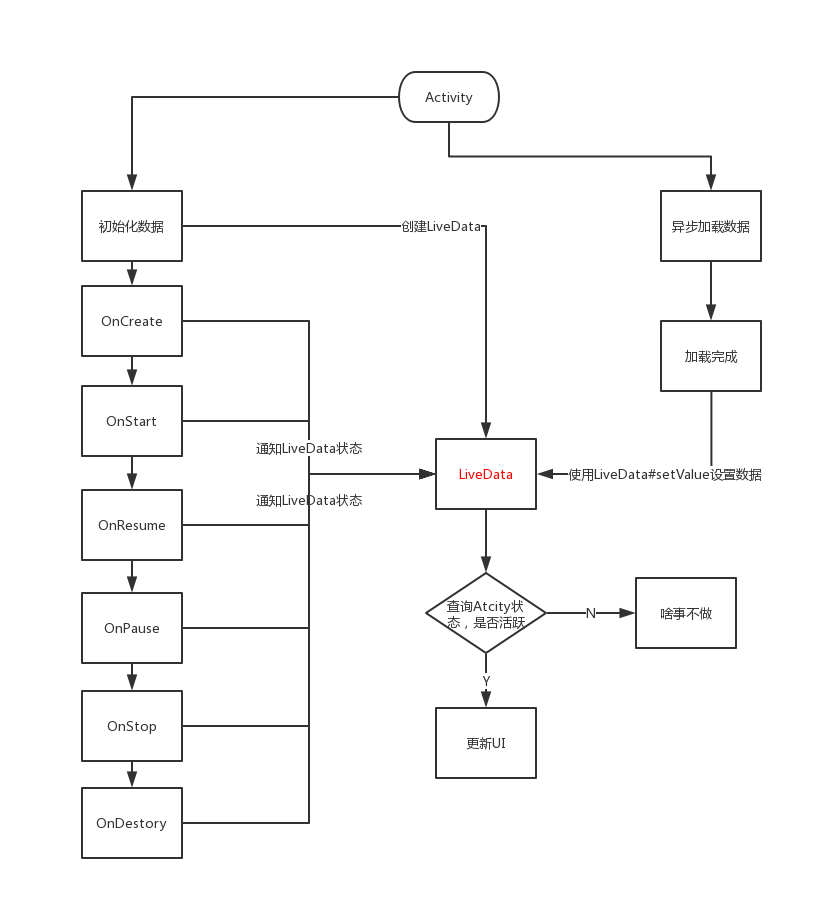


翻译的意思如下：

LiveData是观察者的持有类。与常规的可观察对象不同，LiveData具有生命周期感知功能，这意味着它检视其他应用程序组件（例如Activity，Fragment或Service）的生命周期。这种机制确保LiveData仅在被观察者生命周期处于活跃状态下触发更新。

可见LiveData是对生命周期进行观察，并且持有观察者，并且在设置数据的时候查询生命周期状态并判断是否需要通知观察者，使用方法就是与LifeCycle组件结合使用，

下面是LiveData的流程图



图片主要流程简单就是，Activity状态改变通知LiveData，LiveData记录状态标志（isActive是否活跃），在异步操作setValue设置数据的时候判断isActive标志表示是否需要通知观察者进行回调更新UI

下面我们将模拟LiveData的实现方式：

1.定义LiveData类，添加外部observe接口，引用LifeCycleOwner用于主动查询状态

public class LiveData<T> {  
 private LifecycleBoundObserver observer;  
  
 public void observe(@NonNull LifecycleOwner owner, @NonNull Observer<T> observer) {  
 this.observer = new LifecycleBoundObserver(owner, observer);  
 }  
  
 // 包装类  
 class LifecycleBoundObserver implements LifecycleObserver {  
 private LifecycleOwner owner;  
  
 private Observer<T> observer;  
  
 public LifecycleBoundObserver(LifecycleOwner owner, Observer<T> observer) {  
 this.owner = owner;  
 this.observer = observer;  
 }  
  
 @OnLifecycleEvent(Lifecycle.Event.*ON\_ANY*)  
 public void onStateChanged(LifecycleOwner source, Lifecycle.Event event) {  
  
 }  
 }  
}

我们定义了一个LiveData类，使用泛型用于包装原始数据，定义内部类LifecycleBoundObserver用于引用LifeCycleOwner（文章一中默认SupportActivity实现），以及提供观察的方法，使用LifecycleBoundObserver包装类来包装真实的回调，并且在包装类里面处理生命周期变化的方法

2.LiveData提供setValue<T> 方法，用于外部主动调用设置数据

private T value;  
public void setValue(T value) {  
 this.value = value;  
 //开始判断是否需要通知观察者数据变化  
 dispatchingValue();  
}

3.使用dispatchingValue判断是否需要通知观察者数据

private void dispatchingValue() {  
 // 1.查询状态  
 if (observer == null) {  
 return;  
 }  
 if (observer.isActive) {  
 considerNotify(observer);  
 }  
}  
  
private void considerNotify(LifecycleBoundObserver observer) {  
 if (!observer.isActive) {  
 return;  
 }  
 //通知状态改变  
 observer.observer.onChanged(value);  
}

4.setValue触发了回调方法通知数据改变，使用了observer.isActive标记位来判断是否需要通知改变，那么observer.isActive如何变更状态的呢，我们需要实现LifecycleBoundObserver#onStateChanged的方法：

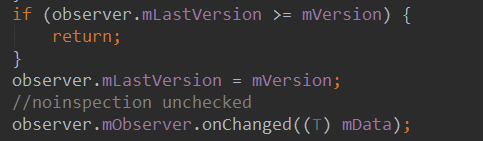
// 包装类  
class LifecycleBoundObserver implements LifecycleObserver {  
 private LifecycleOwner owner;  
  
 private Observer<T> observer;  
  
 // 是否activity处于活跃状态  
 private boolean isActive;  
  
 public LifecycleBoundObserver(LifecycleOwner owner, Observer<T> observer) {  
 this.owner = owner;  
 this.observer = observer;  
 }  
  
 //每次activity状态改变的时候通知触发该方法，这里判断是否activity处于活跃状态  
 @OnLifecycleEvent(Lifecycle.Event.*ON\_ANY*)  
 public void onStateChanged(LifecycleOwner mOwner, Lifecycle.Event event) {  
 if (mOwner.getLifecycle().getCurrentState() == *DESTROYED*) {  
 observer = null;  
 return;  
 }  
 isActive = mOwner.getLifecycle().getCurrentState().isAtLeast(*STARTED*);  
 }  
}

主要实现在onStateChanged方法里面，每次activity生命周期变化都会调用LifecycleBoundObserver包装类观察者的onStateChanged方法，这里方法里面记录一个isActive变量用于实时记录Activity变化的状态，这样在setValue的时候可以查询是否activity处于活跃状态，当然了，如果处于DESTROYED状态，表示activity被销毁了，那么就会移除observer引用（一般observer都是间接引用activity，如果不移除异步模式下将会产生内存泄漏），setvalue的时候就不会产生activity引用，就不会导致内存泄漏了（这个就是LiveData避免内存泄漏的实现方法）

疑惑解答：

笔者在使用LiveData的时候，发现如果调用void observe(@NonNull LifecycleOwner owner, @NonNull Observer<T> observer)方法的时候，如果LiveData之前被调用过setValue，那么参数二（observer）将会主动触发一次onChanged方法，为什么呢？

查看源码我们可以发现，LiveData里面有一个变量：private int mVersion = START\_VERSION;标志当前setValue被调用次数，而包装类LifecycleBoundObserver里面也有一个变量（int mLastVersion = START\_VERSION;），LiveData在每次通知完包装类LifecycleBoundObserver的时候都会将两者同步，这样就表示LifecycleBoundObserver被通知了，如果两个版本不一致，那么将会触发通知一次，这里observe方法参数二（observer）将会主动触发一次onChanged方法，因为参数二是使用new方法创建，其mVersion还是默认START\_VERSION，与LiveData不一致就会触发通知改变



上面就是jetpack组件架构LiveData的具体的实现原理，我们上面自编写代码，简化其逻辑

参考：android.arch.lifecycle.LiveData