Admin MindOrks

21st August 2019

Understanding LiveData in Android

Begin an Android Developer, you must have used ViewModel in your Android application to communicate between various views of your app. But these ViewModels require multiple calls each time when the data has to alter the view. So, it is a very tedious and costly operation. Due to this, Google introduced the concept of LiveData in Android. LiveData makes the smart use of ViewModel. In this blog, we will learn about LiveData. So, let's get started.

> *LiveData is an observable data holder class. Unlike a regular observable, LiveData is lifecycle-aware, meaning it respects the lifecycle of other app components, such as activities, fragments, or services. This awareness ensures LiveData only updates app component observers that are in an active lifecycle state.*

Phew! It's a mouth full of explanation.

In simpler words, we can say that LiveData could be modified by the ViewModels. So, once the LiveData is updated or modified it will then notify all it's observers i.e. activities, fragments, services, etc. But the best part about LiveData is that it will not notify every observer. It will first check if the state of the observer if the state is live then that observer will be notified otherwise not. So, you need not worry about unsubscribing any observer. Also, if the observer is resumed then the latest data change in the LiveData will be notified to the observer.

Advantages of using LiveData

1. **A good update in UI:** With the help of LiveData, your application's UI will only be changed if there is a change in the data.
2. **No crash due to stopped activities:** Only that observer will be notified which is in a live state. So, no app crash will be there due to stopped or paused activities.
3. **Up to date data:** The observer will have the latest data ev the observer is in background or pause state. Whenever the

4. **No memory leaks:** Since all the observers are bound to Lifecycle objects and are cleaned up when their associated lifecycle is destroyed. So, there are no memory leaks.

Three basic steps to work with LiveData

In order to use LiveData in your project, all you need to do is follow the below three steps:

1. In your ViewModel class, create a LiveData instance to hold a certain type of data.
2. The next thing that you need to do is to handle the event when there is a change in the data of LiveData. So, you need to create an Observer object that will define the **onChanged()** method and you will define the operation that will be triggered when there is a change in data. Normally, you create an object of the Observer class in your observer i.e. Activity, fragment or Service.
3. The last thing that you need to do is to attach the Observer object to the LiveData object with the help of the **observer()** method.

After following the above three steps, whenever there is a change in the data stored in LiveData then all the observers associated with the LiveData will be notified if they are live otherwise they will be notified when they come into the resume state.

So, let's see how we can code for the above three steps:

```
class YourViewModel : ViewModel() {

    // Create a LiveData with a String type
    val currentName: MutableLiveData<String> by lazy {
        MutableLiveData<String>()
    }

    // Rest of the ViewModel...
}
```

After creating the object of the LiveData, we need to create the object of the Observer class so that if there is a change in data of the LiveData then that change will be observed. Generally, you perform this activity in your observer i.e. in your Activity, Fragment or Service.

```
class NameActivity : AppCompatActivity() {

    private lateinit var viewModel: YourViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Other code to setup the activity...

        // Get the ViewModel.
        viewModel = ViewModelProviders.of(this).get(YourView


        // Create the observer which updates the UI.
        val nameObserver = Observer<String> { newName ->
            // Update the UI, in this case, a TextView.
            nameTextView.text = newName
```

```
        // observe the LiveData, passing in this activity as
        viewModel.currentName.observe(this, nameObserver)
    }
}
```

The above code illustrates how to start observing the a LiveData object.

When the **observer()** method is called with nameObserver as a parameter and as discussed earlier, if there is a change in LiveData then the **onChanged()** method will be called and will provide the latest value stored in the mCurrentName.

Updating LiveData object

By default, you don't have any public methods to update the stored data in LiveData. So, you can use the **setValue(T)** and **postValue(T)** methods of the **MutableLiveData** class. You have to use the **MutableLiveData** class in the **ViewModel** only.

Following is an example, that triggers all observers when we press a button i.e. the **onChanged()** method will be called when we press the button value that the **onChanged()** method will contain is "John Doe".

```
button.setOnClickListener {
    val anotherName = "John Doe"
    viewModel.currentName.setValue(anotherName)
}
```

LiveData in observer

So, you can extend the LiveData class in your observer class and override the **onActive()** and **onInactive()** method. This **onActive()** method will be used to check if there is a change in LiveData or not and the **onInactive()** method is used to remove the updates.

```
class StockLiveData(symbol: String) : LiveData<BigDecimal>()
    private val stockManager = StockManager(symbol)

    private val listener = { price: BigDecimal ->
        value = price
    }

    override fun onActive() {
        stockManager.requestPriceUpdates(listener)
    }

    override fun onInactive() {
        stockManager.removeUpdates(listener)
    }
}
```

Here, in the above code, the **onActive()** method is used to start observing for the stock price updates that are stored in the LiveData object. In the **onInactive()** method, we stop looking for any change in the LiveData because the observer is no longer interested in getting the LiveData.

Conclusion

So, in this blog, we learned about LiveData and how to use it. We saw that if there is a change in data then that data will be

ViewModel is that LiveData only notifies the changes to the observers that are live or in the active state and not to that observer that are in the inactive state.

Hope you liked this blog. To learn more about some of the cool topics of Android, you can visit our blogging website.

Keep Learning :)

Team MindOrks!