转载请注明链接: https://blog.csdn.net/feather_wch/article/details/88648559

LiveData基本教程

版本号:2019-03-22(11:30)

简介

1、LiveData的简介

> 1. `LiveData` 是一种类，持有 可被观察的数据 。
> 2. `LiveData` 是一种 可感知生命周期的组件 ， 意味着该组件重视其他 app组件的生命周期 ，如 `Activity、Fragment、Service`
>   - 该组件能确保，仅仅在 `Activity\Fragment\Service`等组件 都处于活跃的生命周期状态的时候，才去更新app组件。

2、LiveData只有当观察者的生命周期处于 活跃状态 时才会去通知观察者。

> 1. 实现了 `Observer`类 的观察者，可以注册监听 `LiveData`
> 2. 活跃状态就是指处于 `STARTED`或者`RESUMED` 状态
> 3. 处于非活跃的观察者，LiveData不会去通知这些观察者

3、可以注册一种观察者, 该观察者与 `LifecycleOwner` 对象(如：Activity、Fragment)相关联。

> 1. 在对应的 Lifecycle Object 处于 DESTORYED 状态时，会自动解除 LiveData和该观察者的注册关系

4、在Activity、Fragment中这种自动解除注册的特性非常有用

> 1. Activity、Fragment不用担心会出现 内存泄露
> 2. 在 Activity、Fragment 销毁时， LiveData会自动解除其注册关系

优势

5、LiveData能确保UI和数据状态相符

> 1. 因为是观察者模式，LiveData会在生命周期状态改变时，通知观察者
> 2. 可以在 观察者对象 中进行UI的更新操作

6、LiveData没有内存泄露

> 1. 观察者和 Lifecycle对象 绑定，能在销毁时自动解除注册

7、LiveData不会给已经停止的Activity发送事件

> 1. 如果观察者处于非活跃状态， LiveData 不会再发送任何事件给这些 Observer对象

8、LiveData能确保不再需要手工对生命周期进行处理

> 1. UI组件仅仅需要对相关数据进行观察
> 2. LiveData自动处理生命周期状态改变后，需要处理的代码。

9、LiveData能保证数据最新

> 1. 一个 非活跃的组件 进入到 活跃状态 后，会立即获取到最新的数据
> 2. 不用担心数据问题

10、LiveData在横竖屏切换等Configuration改变时，也能保证获取到最新数据

> 1. 例如 Acitivty、Fragment 因为 屏幕选装 导致 重建 ，能立即 接收到最新的数据

11、LiveData能资源共享

> 1. 如果将 LiveData对象 扩充，用 单例模式 将 系统服务进行包裹 。这些服务就可以在app中共享。
> 2. 只需要 LiveData和系统服务 connect，其他 观察者 只需要 监视LiveData 就能获取到这些资源

使用LiveData

1、LiveData的使用遵循下面三个步骤

> 1. 创建 LiveData的实例 ， 并持有具有类型的数据

2. 创建 Observer对象 ， 该对象具有 onChanged() 方法，在 LiveData的数据改变时 ， 会调用 onChanged()方法，进行相应的处理工作 。可以将 Observer 放置到 activity、fragment 中
3. 利用 observe()方法 将 Observer 和 LiveData 联系起来。
   * observe()接收一个 LifecycleOwner对象
   * 可以使用 observeForeve(Observer) 注册一个没使用 LifecycleOwner 的 Observer ， 这种场景中，该Observer会认为是一直存活的。
   * 使用 removeObserver(Observer) 可以移除这些观察者

创建LiveData对象

2、创建LiveData对象

1. LiveData 能用来包裹所有数据，包括实现了 Collections 的对象，例如List
2. LiveData 通常存储在 ViewModel 之中, 并通过 get方法 来获取

```
public class UserViewModel extends ViewModel {
    private MutableLiveData<String> mName;
    private MutableLiveData<Integer> mAge;

    public MutableLiveData<String> getName() {
        if(mName == null){
            mName = new MutableLiveData<>();
        }
        return mName;
    }

    public MutableLiveData<Integer> getAge() {
        if(mAge == null){
            mAge = new MutableLiveData<>();
        }
        return mAge;
    }
}
```

3、为什么将LiveData放置到ViewModel中，而不放到activity或者fragment中?

1. 避免fragment和activity的代码臃肿
2. 将 LiveData 和特定的activity/fragment解耦，能够在configuration改变的时候，LiveData依然存活。

观察LiveData对象

4、在App组件的哪个生命周期适合观察LiveData对象？为什么?

1. app组件的 onCreate() 方法
2. 不适合在 onResume() 等方法中，可能会调用多次

> 3. 能确保组件能尽可能快的展示出数据。只要app组件处于启动状态(STARTED)就会立即接收到 `LiveData对象` 中的数据---前提是已经监听了LiveData

5、监听LiveData实例

```
public class DataBindingActivity extends AppCompatActivity {

    ActivityDatabindingLayoutBinding mBinding;
    User mUser;
    private UserViewModel mUserViewModel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // DataBinding
        // xxx

        //  1. 创建用户信息的ViewModel
        mUserViewModel = ViewModelProviders.of(this).get(UserViewModel.class);
        //  2. 创建更新UI的观察者
        Observer<String> nameObserver = new Observer<String>() {
            @Override
            public void onChanged(@Nullable String s) {
                // 利用DataBinding更新: 用户账号
                mUser.setAccount(s);
                mBinding.setUser(mUser);
            }
        };
        //  3. 注册观察者
        mUserViewModel.getAccount().observe(this, nameObserver);
    }
}
```

6、ViewModelProviders为什么找不到?

> 1. 引用的版本太老了，需要新的Lifecyle扩展库(目前可以用的最新版)
>   android.arch.lifecycle

```
// ViewModel and LiveData
implementation "android.arch.lifecycle:extensions:1.1.1"
```

> androidx

```
// 引入lifecycle
def lifecycle_version = "2.0.0"

// ViewModel and LiveData
implementation "androidx.lifecycle:lifecycle-extensions:$lifecycle_version"
```

更新LiveData对象

7、MutableLiveData类自动提供 `setValue(T)`、`postValue(T)` 用于更新值

8、更新LiveData对象实例

```
button.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        String anotherName = "John Doe";
        model.getCurrentName().setValue(anotherName);
    }
});
```

> 1. 调用 `setValue()`或者`postValue()` 都会调用所有观察者的 `onChanged()` 方法

Room使用LiveData

9、Room数据持久化库，支持 `observable`查询

> 1. 该查询能返回 `LiveData`对象
> 2. Observable查询是 `DAO-Database Access Object` 的一部分
> 3. Room自动生成所有更新 `LiveData`对象 所需要的代码(当数据库更新的时候)
> 4. 查询操作是在后台线程执行异步操作

扩展的LiveData

1、扩展LiveData

```
package com.hao.architecture;

import android.arch.lifecycle.LiveData;

import java.math.BigDecimal;

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

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

    private SimplePriceListener listener = new SimplePriceListener() {
        @Override
        public void onPriceChanged(BigDecimal price) {
            // 更新LiveData并且通知所有活跃的观察者
            setValue(price);
        }
    };

    @Override
    protected void onActive() {
        // 具有活跃的观察者时调用
        stockManager.requestPriceUpdates(listener);
    }

    @Override
    protected void onInactive() {
        // 没有任何活跃的观察者时调用
        stockManager.removeUpdates(listener);
    }
}
```

> Fragment中使用

```
public class MyFragment extends Fragment {
    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        LiveData<BigDecimal> myPriceListener = ...;
        myPriceListener.observe(this, price -> {
            // Update the UI.
        });
    }
}
```

2、LiveData作为生命周期感知组件，可以在多个Activity中共享: 使用单例

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

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

    //xxx

}
```

> 多个Fragment中使用：LiveData仅仅当一个或者多个LifecycleOwner处于活跃状态时，才会和系统服务连接。

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

转换LiveData

1、有的时候需要在分发LiveData的数值到观察者前进行处理，可以利用 `Transformation.map()`

```
LiveData<User> userLiveData = ...;
LiveData<String> userName = Transformations.map(userLiveData, user -> {
    user.name + " " + user.lastName
});
```

创造新的transformations

2、可以使用 `MediatorLiveData`

合并多个LiveData

1、MediatorLiveData是LiveData的子类

> 允许merge多个LiveData源

2、LiveData merge的使用场景

> 1. 如果具有一个LiveData对象，可以根据本地数据库或者网络数据进行更新
> 2. 就可以将两种LiveData来源添加到MediatorLiveData对象中
>   1. 和本地数据库关联的LiveData
>   2. 和网络数据关联的LiveData
> 3. Activity只需要观察 `MediatorLiveData` 对象，就能接收到来自两个数据源的更新

LiveData与Lifecycle

1、为什么LiveData能作为生命感知组件

```
mUserViewModel.getUserListLiveData().observe(ArchActivity.this, new Observer<List<User>>()
        {
            @Override
            public void onChanged(@Nullable List<User> users)
            {
                for (User user : users)
                {
                    Log.d("mvvm", "name = " + user.getName() + " age = " + user.getAge());
                }
            }
        });
```

> 1. `LiveData的observe()` 会将 `Observer观察者` 包装成 `LifecycleObserver`
> 2. 让 `Activity、Fragment` 这些LifecycleOwner对这些观察者进行注册
> 3. 当Activity、Fragment的生命周期改变时，去通知LiveData，作相应处理

2、为什么LiveData只会在观察者的生命周期处于活跃状态时，才去通知观察者更新UI

> 1. LiveData对Activity、Fragment的生命周期进行了感知
> 2. 当LifecycleOwner的生命周期改变时，会通知LiveData

参考资料

1. LiveData