

本文首发于微信公众号「后厂技术官」

前言

在本系列的第4和第5篇文章中，介绍了LiveData的使用和原理，LiveData和ViewModel是一对好搭档，这篇文章我们一起来学习什么是ViewModel、ViewModel基本使用、ViewModel的原理。

1.什么是ViewModel

在说ViewModel前，我们需要解一下MVVM和DataBinding，在Android进阶三部曲第一部《Android进阶之光》中，我简单介绍了下MVVM和DataBinding。MVVM最早于2005年被微软的架构师John Gossman提出，在2015年Google I/O大会上发布了MVVM的支持库DataBinding，DataBinding遭到了一些开发者的诟病，主要是绑定数据时如果出现问题会很难排查，这样就没有一个合适的ViewModel规范来帮助开发者来实现MVVM。

在2017年的Google I/O大会上，谷歌推出了ViewModel组件，来规范化ViewModel层。

ViewModel顾名思义，是以感知生命周期的形式来存储和管理视图相关的数据。ViewModel主要有以下的特点：

- 1.当Activity被销毁时，我们可以使用onSaveInstanceState()方法恢复其数据，这种方法仅适用于恢复少量的支持序列化、反序列化的数据，不适用于大量数据，如用户列表或位图。而ViewModel不仅支持大量数据，还不需要序列化、反序列化操作。
- 2.Activity/Fragment（视图控制器）主要用于显示视图数据，如果它们也负责数据库或者网络加载数据等操作，那么一旦逻辑过多，会导致视图控制器臃肿，ViewModel可以更容易，更有效的将视图数据相关逻辑和视图控制器分离开来。
- 3.视图控制器经常需要一些时间才可能返回的异步调用，视图控制器需要管理这些调用，在合适的时候清理它们，以确保它们的生命周期不会大于自身，避免内存泄漏。而ViewModel恰恰可以避免内存泄漏的发生。

2.ViewModel基本使用

添加依赖









在[Android Jetpack架构组件（二）带你了解Lifecycle（使用篇）](#)这篇文章中讲过，一般情况只需要添加如下代码就可以：



```
implementation "android.arch.lifecycle:extensions:1.1.1"
```

由于Gradle默认支持依赖传递，添加这一句代码就依赖了很多库。如果不懂Gradle的依赖传递，可以查看

Android Gradle（二）签名配置和依赖管理这篇文章。

- ▶  Gradle: android.arch.core:common:1.1.1@jar
- ▶  Gradle: android.arch.core:runtime:1.1.1@aar
- ▶  Gradle: android.arch.lifecycle:common:1.1.1@jar
- ▶  Gradle: android.arch.lifecycle:extensions:1.1.1@aar
- ▶  Gradle: android.arch.lifecycle:livedata:1.1.1@aar
- ▶  Gradle: android.arch.lifecycle:livedata-core:1.1.1@aar
- ▶  Gradle: android.arch.lifecycle:runtime:1.1.1@aar
- ▶  Gradle: android.arch.lifecycle:viewmodel:1.1.1@aar

如果需要其他特性，比如kotlin的支持，再另行添加。

自定义ViewModel

继承ViewModel，实现自定义ViewModel。

● ● JAVA

```
import android.arch.lifecycle.LiveData;
import android.arch.lifecycle.MutableLiveData;
import android.arch.lifecycle.ViewModel;

public class MyViewModel extends ViewModel {
    private MutableLiveData<String> name;
    public LiveData<String> getName() {
        if (name == null) {
            name = new MutableLiveData<String>();
            addName();
        }
        return name;
    }
    private void addName() {
        name.setValue("Android进阶解密");
    }
}
```

getName方法中创建一个MutableLiveData，并通过MutableLiveData的setValue方法来更新数据。

使用ViewModel

然后就可以在Activity中使用MyViewModel了，如下所示。

JAVA



```
import android.arch.lifecycle.Observer;
import android.arch.lifecycle.ViewModelProviders;
import android.support.annotation.Nullable;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends AppCompatActivity {
    private static final String TAG = "MainActivity";

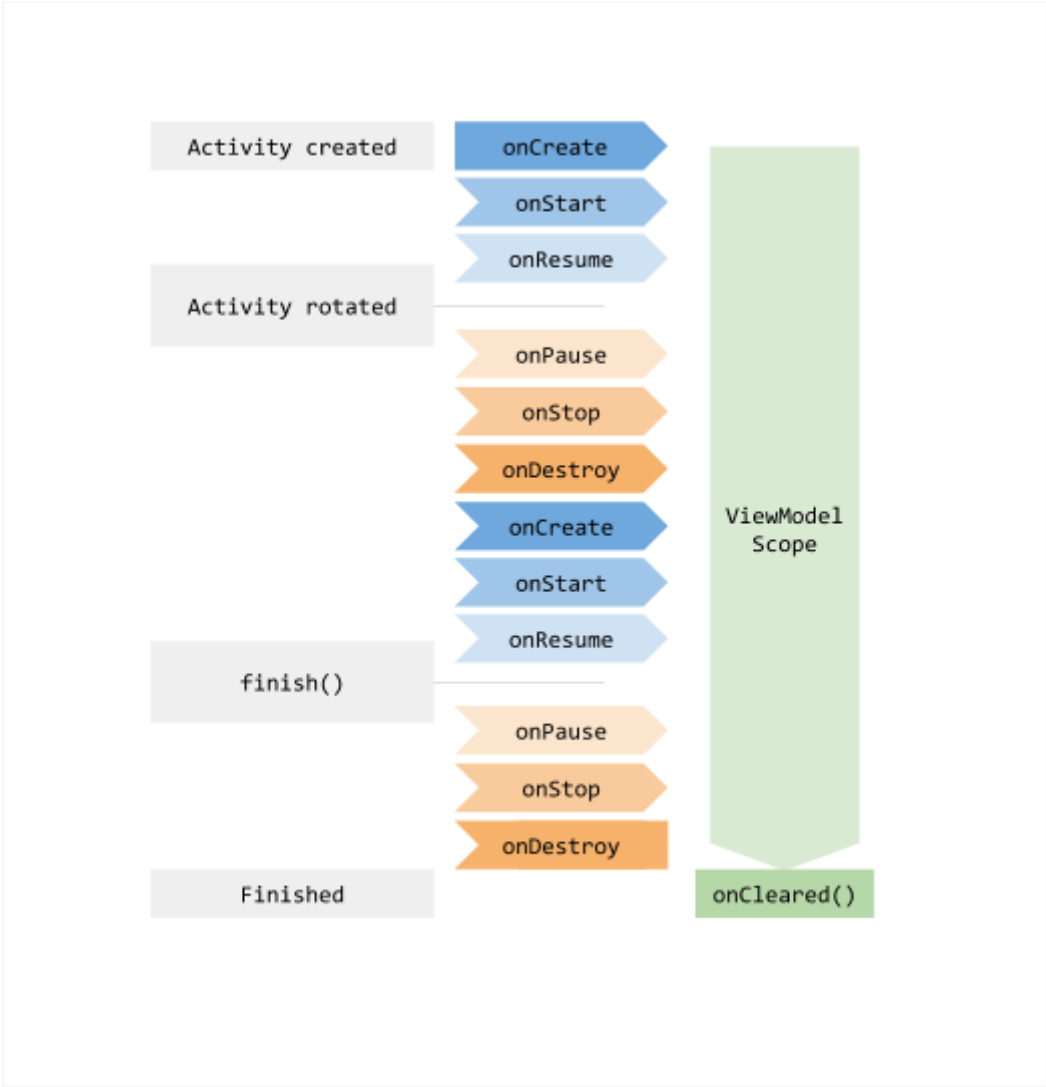
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        MyViewModel model = ViewModelProviders.of(this).get(MyViewModel.class);
        model.getName().observe(this, new Observer<String>() {
            @Override
            public void onChanged(@Nullable String s) {
                Log.d(TAG, "畅销书: "+s);
            }
        });
    }
}
```

通过ViewModelProviders就可以获得MyViewModel的实例，然后配合LiveData就可以观察Name的变化。打印结果为：

D/MainActivity: 书名为Android进阶解密

3.ViewModel的生命周期

在旋转设备屏幕时，Activity会被销毁重新创建，而ViewModel却不会这样，它的生命周期如下所示。



可以看到，Activity的生命周期不断变化，经历了被销毁重新创建，而ViewModel的生命周期没有发生变化。

4.ViewModel的原理

要讲解原理，我们需要先从一个点入手，那就是第2节例子中的：

● ● CODE

MyViewModel model = ViewModelProviders.of(this).get(MyViewModel.class);

因为我們是在Activity中调用的，因此this的值为Activity，我們还可以在Fragment中调用上面的方法，那么this的值为Fragment，因此ViewModelProviders.of()有多个构造方法，我們以在Activity中调用为例。

frameworks/support/lifecycle/extensions/src/main/java/androidx/lifecycle/ViewModelProviders.java

● ● JAVA

```

@NonNull
@MainThread
public static ViewModelProvider of(@NonNull FragmentActivity activity) {
    return of(activity, null);
}

@NonNull
@MainThread
public static ViewModelProvider of(@NonNull FragmentActivity activity,
    @Nullable Factory factory) {
    Application application = checkApplication(activity); //1
    if (factory == null) {
        factory = ViewModelProvider.AndroidViewModelFactory.getInstance(application);
    }
    return new ViewModelProvider(activity.getViewModelStore(), factory);
}

```

ViewModelProvider的of方法有多个构造方法，

注释1处内部会调用activity.getApplication()来返回该Activity对应的Application。

注释2处的代码来创建AndroidViewModelFactory实例。最后会新建一个ViewModelProvider，将AndroidViewModelFactory作为参数传进去，AndroidViewModelFactory的代码如下所示。

```

● ● JAVA
public static class AndroidViewModelFactory extends ViewModelProvider.N

private static AndroidViewModelFactory sInstance;
@NonNull
public static AndroidViewModelFactory getInstance(@NonNull Application application) {
    if (sInstance == null) {
        sInstance = new AndroidViewModelFactory(application);
    }
    return sInstance;
}
private Application mApplication;
public AndroidViewModelFactory(@NonNull Application application) {
    mApplication = application;
}

@NonNull
@Override
public <T extends ViewModel> T create(@NonNull Class<T> modelClass) {
    if (AndroidViewModel.class.isAssignableFrom(modelClass)) {
        //noinspection TryWithIdenticalCatches
        try {
            return modelClass.getConstructor(Application.class).newInstance(mApplication);
        } catch (Exception e) {
            return null;
        }
    }
    return super.create(modelClass);
}

```

```

    } catch (NoSuchMethodException e) {
        throw new RuntimeException("Cannot create an instance of " + modelClass.getName());
    } catch (IllegalAccessException e) {
        throw new RuntimeException("Cannot create an instance of " + modelClass.getName());
    } catch (InstantiationException e) {
        throw new RuntimeException("Cannot create an instance of " + modelClass.getName());
    } catch (InvocationTargetException e) {
        throw new RuntimeException("Cannot create an instance of " + modelClass.getName());
    }
}
return super.create(modelClass);
}
}

```

AndroidViewModelFactory是一个单例，讲这个类前，需要知道ViewModel类本身是一个抽象类，我们通过继承ViewModel，来实现自定义ViewModel，那么AndroidViewModelFactory的create方法的作用，就是通过反射生成ViewModel的实现类。

接着回头看ViewModelProvider.get方法。

frameworks/support/lifecycle/viewmodel/src/main/java/androidx/lifecycle/ViewModelProvider.java

```

● ● JAVA

@NonNull
@MainThread
public <T extends ViewModel> T get(@NonNull Class<T> modelClass) {
    String canonicalName = modelClass.getCanonicalName(); //1
    if (canonicalName == null) {
        throw new IllegalArgumentException("Local and anonymous classes have no canonical names");
    }
    return get(DEFAULT_KEY + ":" + canonicalName, modelClass); //2
}

@NonNull
@MainThread
public <T extends ViewModel> T get(@NonNull String key, @NonNull Class<T> modelClass) {
    ViewModel viewModel = mViewModelStore.get(key); //3

    if (modelClass.isInstance(viewModel)) {
        //noinspection unchecked
        return (T) viewModel;
    } else {
        //noinspection StatementWithEmptyBody
        if (viewModel != null) {
            // TODO: log a warning.
        }
    }
}

```

```

    }

    viewModel = mFactory.create(modelClass);
    mViewModelStore.put(key, viewModel);
    //noinspection unchecked
    return (T) viewModel;
}

```

注释1处得到类的名称，对这个名称进行字符串拼接，作为注释2处方法的参数，DEFAULT_KEY的值为：“androidx.lifecycle.ViewModelProvider.DefaultKey”。

因此，注释3处的key值实际上就是“androidx.lifecycle.ViewModelProvider.DefaultKey”+类名。根据这个key值从ViewModelStore获取ViewModel（ViewModel的实现类）。如果ViewModel能转换为modelClass类的对象，直接返回该ViewModel。否则会通过Factory创建一个ViewModel，并将其存储到ViewModelStore中。这里的Factory指的是AndroidViewModelFactory，它在ViewModelProvider创建时作为参数传进来。

到此为止，我们已经知道了ViewModel的实现类是如何创建的了。

当创建完ViewModel的实现类后，在第2小节我们还会调用如下代码。

CODE

```

MyViewModel model = ViewModelProviders.of(this).get(MyViewModel.class);
model.getName().observe(this, new Observer<String>() {
    @Override
    public void onChanged(@Nullable String s) {
        Log.d(TAG, "畅销书: "+s);
    }
});

```

model.getName()会返回一个MutableLiveData，接着调用了MutableLiveData的observe方法，这个在Android Jetpack架构组件（五）带你了解LiveData(原理篇) 这篇文章中讲过，就不再赘述。

文章作者：刘望舒

文章链接：<http://liuwangshu.cn/application/jetpack/6-viewmodel.html>

版权声明：本博客所有文章除特别声明外，均采用 [CC BY-NC-SA 4.0](#) 许可协议。转载请注明来自 [BATcoder - 刘望舒](#)！

Android Jetpack



