

Android开发

Jetpack架构组件学习(1)——LifeCycle的使用

2021-11-19 / 0 评论 / 0 点赞 / 594 阅读 / 6,526 字

要看本系列其他文章,可访问此链接[Jetpack架构学习 | Stars-One的杂货小窝](#)

最近有时间了,准备入坑Jetpack架构,第一篇就学个简单的 `LifeCycle` ,可以帮助开发者创建可感知生命周期的组件。

介绍

为什么需要 `LifeCycle` 组件?

在很多情况下,我们需要在 `Activity` 的相关生命周期中进行相关的初始化操作,比如上一节说到的EventBus,需要在 `OnCreate()` 和 `onDestroy()` 方法中进行绑定和解绑,我们可以使用此组件来简化操作(下面的例子即是使用 `LifeCycle` 去简化 `EventBus` 绑定 `Activity` 的操作)

`LifeCycle` 的本质原理就是观察者模式,其中有类 `LifecycleOwner` (被观察者)和接口 `LifecycleObserver` (观察者),通过观察者模式,即可实现对页面生命周期进行监听,从而减低代码耦合度

这里提下:

`Support Library 26.1.0` 及其以后的版本(含androidx), `Activity` 和 `Fragment` 已经实现了 `LifecycleOwner` 接口

所以,我们可以直接在Activity和Fragment中使用 `getLifecycle()` 方法来获取 `lifecycle` 对象,来添加观察者监听。

我是直接使用了androidx版本,所以基本使用是无需导入额外的依赖

由于 `Activity` 和 `Fragment` 实现了 `LifecycleOwner` 接口,所以,使用我们只需要创建我们的观察者对象,与 `Activity` 进行绑定即可


基本使用

下面以 `EventBus` 的初始化来讲解下使用方法

1.实现LifecycleObserver接口

前面也说到, `Lifecycle` 提供了一个接口类 `LifecycleObserver` ,这个接口里没有任何的方法,我们需要定义个类去实现它,即可让该类成为一个 `Lifecycle` 观察者类

我们需要让一个类去实现此接口,由于 `EventBus` 需要绑定 `Activity` 对象,所以我们需要类提供一个构造方法来接收 `Activity` 对象



```
1 class EventBusInitiator(val context: Context) : LifecycleObserver {
2     @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
3     fun register() {
4         EventBus.getDefault().register(context)
5     }
6
7     @OnLifecycleEvent(Lifecycle.Event.ON_DESTROY)
8     fun unregister() {
9         EventBus.getDefault().unregister(context)
10    }
11 }
```

那我们实现了接口类,如何让Lifecycle回调我们的方法呢?这里Lifecycle设计就和EventBus有些异曲同工之妙了,也是提供注解的方式,从而Activity在进入对应的生命周期会回调参数

`OnLifecycleEvent` 注解里的参数只接收对应的事件,事件有以下7种类型

- `ON_CREATE` 匹配Activity的onCreate事件
- `ON_START` 匹配Activity的onStart事件
- `ON_RESUME` 匹配Activity的onResume事件
- `ON_PAUSE` 匹配Activity的onStop事件
- `ON_STOP` 匹配Activity的onStop事件
- `ON_DESTROY` 匹配Activity的onDestroy事件
- `ON_ANY` 可以匹配任何的事件(每次Activity的生命周期发生变化,都会调用一次)

前6种就是和Activity中的生命周期保持对应,最后一种有点迷,不知道具体是在哪种情况下使用的

PS:在这篇文章中提到[Android 架构组件之 LifeCycle详解 - SegmentFault 思否](#),有两种方式用来实现LifeCycle的监听

- 实现DefaultLifecycleObserver接口, 然后重写里面生命周期方法;
- 直接实现LifecycleObserver接口, 然后通过注解的方式来接收生命周期的变化;

Lifecycle.java文档中是建议使用第一种方式,随着Java8成为主流, 注解的方式会被弃用。

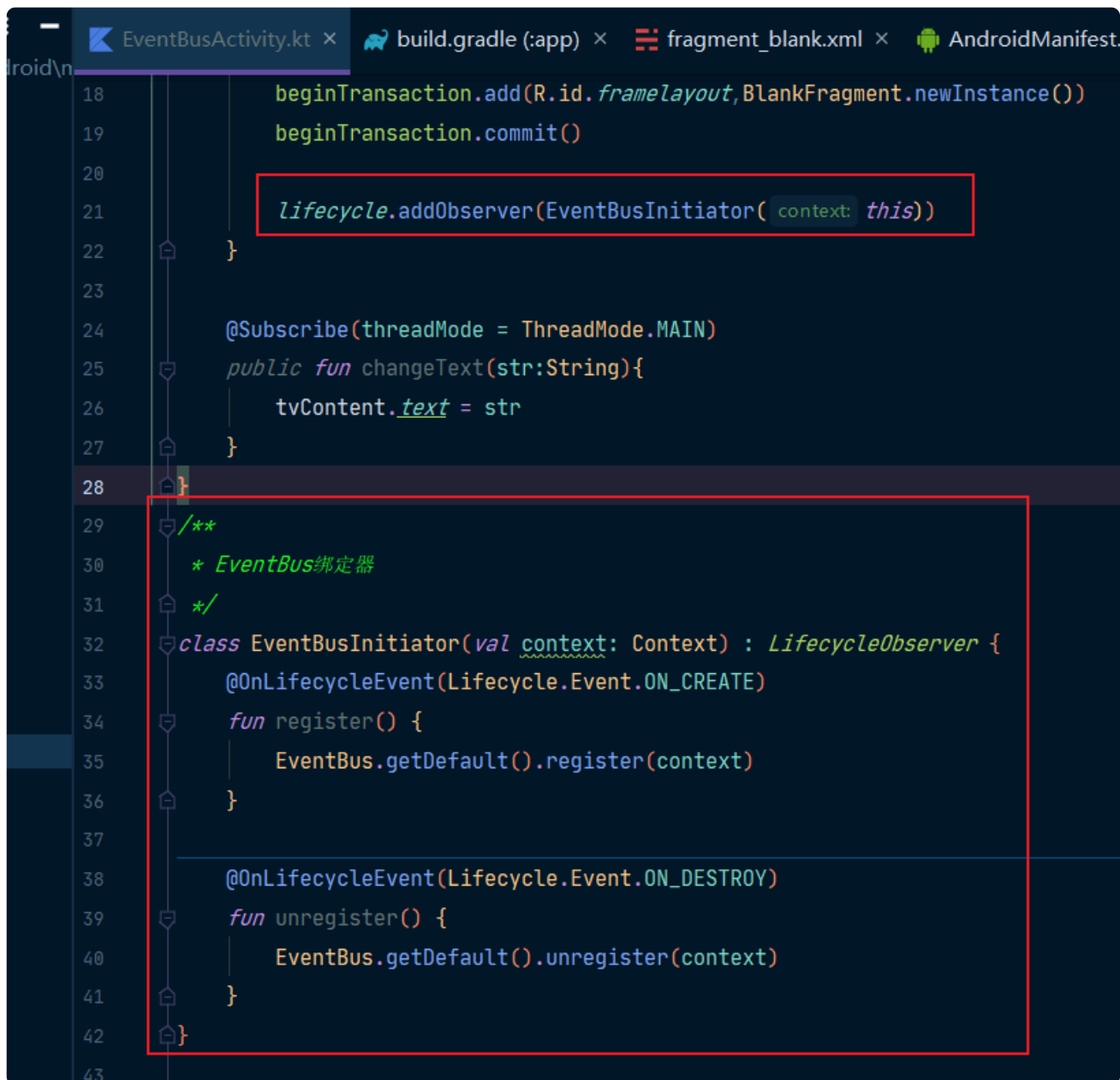
这里我也没细找文档去研究,也不知道是不是这回事,就我个人感觉使用注解的方式会比较舒服,智者见智仁者见仁吧

2.Activity注册观察者对象

在上面实现 `LifecycleObserver` 接口后,我们就有个类,当Activity生命周期发生改变的话,就会将事件分发给类中,前提是我们得让Activity注册这个类的对象,代码如下所示

```
1  override fun onCreate(savedInstanceState: Bundle?) {  
2      super.onCreate(savedInstanceState)  
3      setContentView(R.layout.activity_event_bus)  
4  
5      lifecycle.addObserver(EventBusInitiator(this))  
6  }
```

PS:如果是Java,是通过 `getLifecycle()` 方法获取LifeCycle对象



效果与之前的例子一样,点击按钮即可改变文字

补充-判断当前页面状态

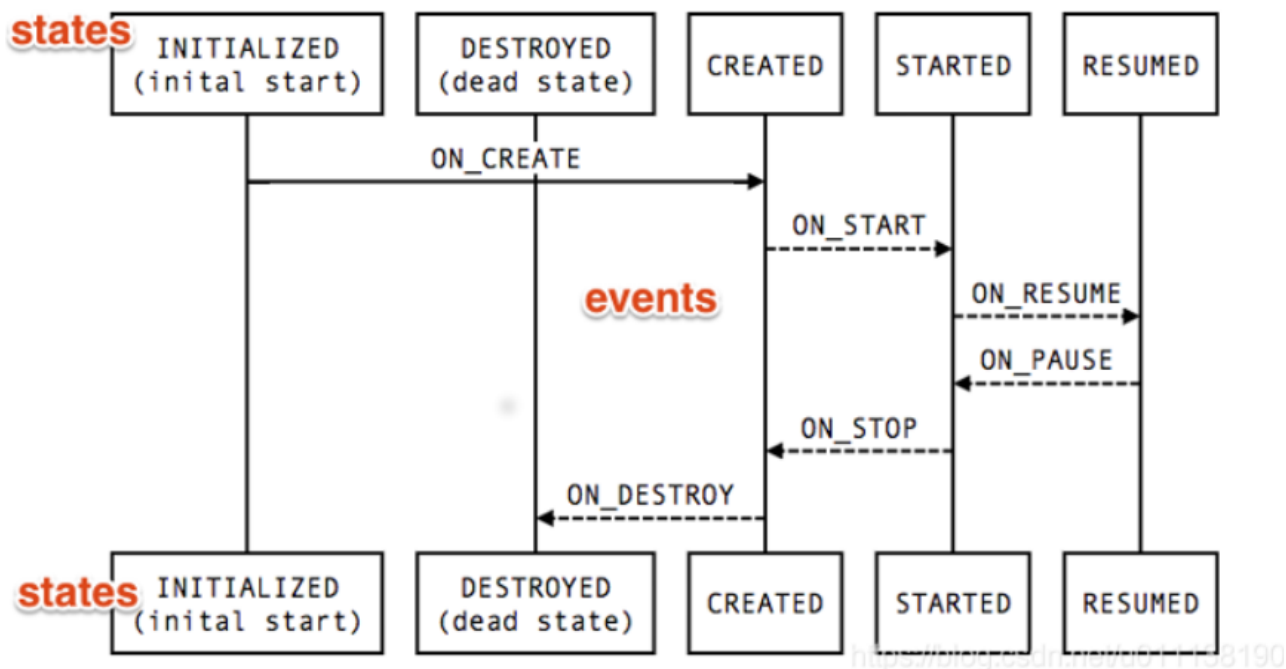
Lifecycle中的状态有以下5种类型:

- `Lifecycle.State.CREATED`
- `Lifecycle.State.DESTROYED`
- `Lifecycle.State.RESUMED`
- `Lifecycle.State.STARTED`
- `Lifecycle.State.INITIALIZED`

`INITIALIZED` 状态表示的是初始化完成,但 `onCreate()` 周期未开始的状态

而之后 `CREATED` 状态,标明是 `onCreate()` 周期已结束的状态,后面的依次类推

可以参考下面的 `Lifecycle` 状态改变和事件的解析图:



判断的话我们可以使用 `Lifecycle` 对象的 `currentState` 状态,如下代码:

```
1 //直接一个判断就完事了
2 if (lifecycle.currentState == Lifecycle.State.INITIALIZED) {
3
4 }
5
6 //状态至少要是CREATED
7 lifecycle.currentState.isAtLeast(Lifecycle.State.CREATED)
```

进阶使用

进阶使用中,需要引入依赖

```
1 implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
```

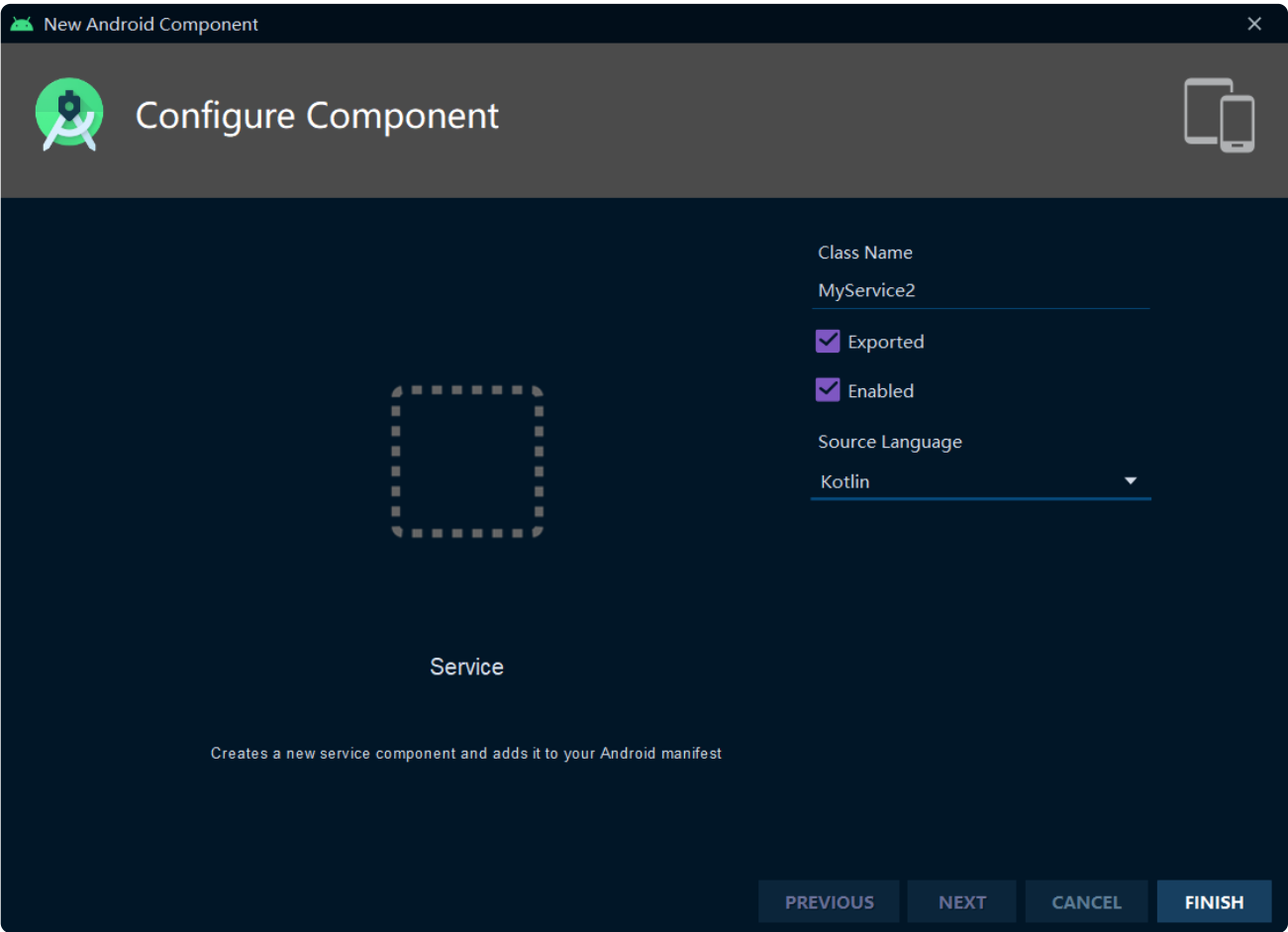
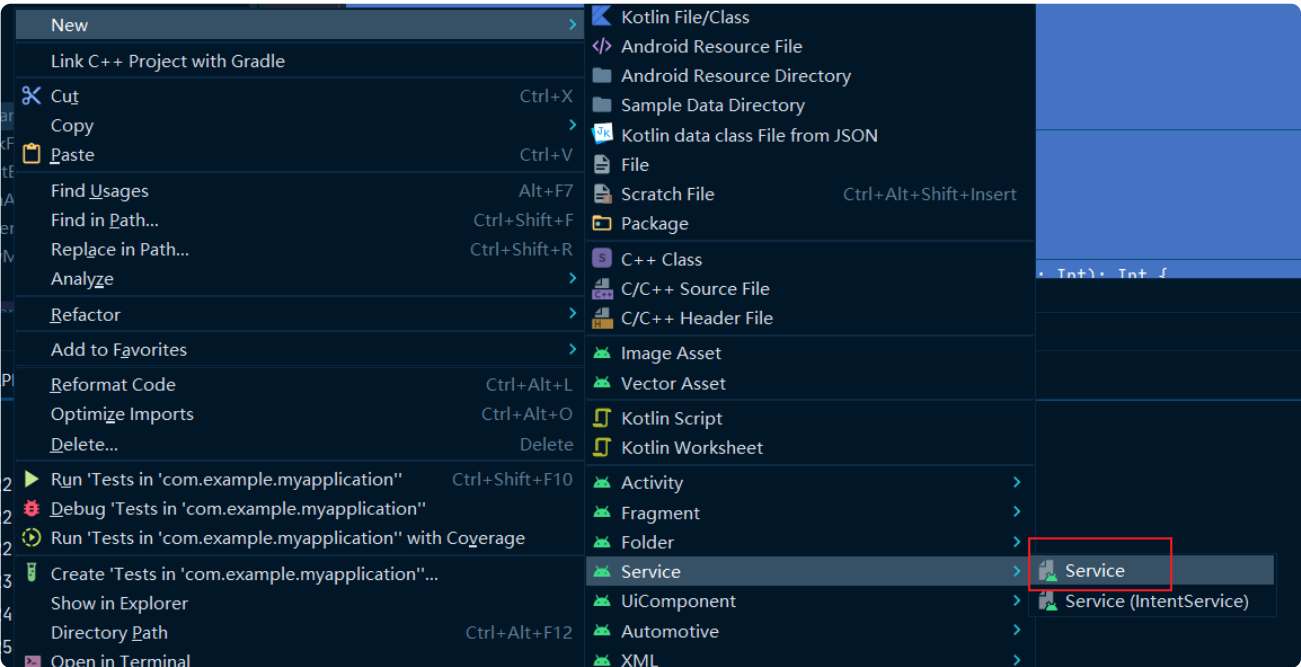
LifecycleService的使用

上述的只讲了Activity和Fragment的使用方法,如果我们想要Service中想要生命周期解耦,可以使用

`LifecycleService`, 其父类是Service,只不过里面已经实现了LifecycleOwner接口,所以用法是基本一

样的

我们使用Android Studio快速新建一个Service



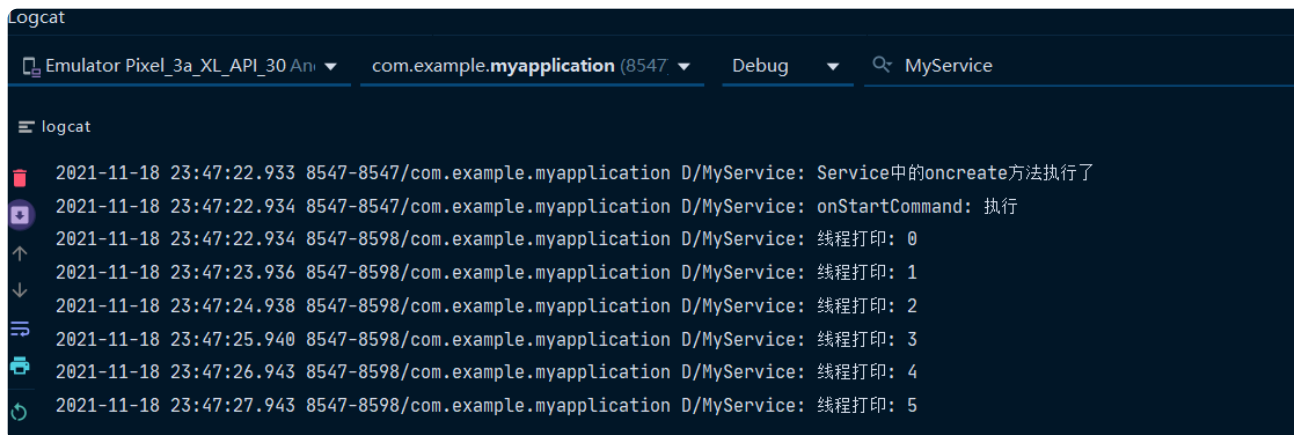
然后修改代码为下述代码:



```
1  class MyService : LifecycleService() {
2
3      val TAG = "MyService"
4
5      init {
6          lifecycle.addObserver(MyServiceObserver())
7      }
8
9      override fun onStartCommand(intent: Intent?, flags: Int, s
10         //开启服务就开启个线程
11         Log.d(TAG, "onStartCommand: 执行")
12         thread {
13             for (i in 0..5) {
14                 Log.d(TAG, "线程打印: $i")
15                 Thread.sleep(1000)
16             }
17         }
18         return super.onStartCommand(intent, flags, startId)
19     }
20 }
21
22 }
23
24 class MyServiceObserver() : LifecycleObserver {
25
26     @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
27     fun create() {
28         Log.d("MyService", "Service中的oncreate方法执行了");
29     }
30 }
31
```

Service的生命周期和上述Activity的例子一样,这里不再赘述,日志打印如图所示

PS:记得在Activity中使用startService开启服务哦



想起来,之前还没写过Service的学习笔记😂 之后有空补上

ProcessLifecycleOwner的使用

除了上面的情况,有时候我们需要判断下APP是处于前台或后台,这个时候就可以使用

`ProcessLifecycleOwner` ,具体用法如下

需要我们自定义一个application,然后在其中注册观察者即可

```
1 class MyApplication: Application() {
2     override fun onCreate() {
3         super.onCreate()
4         ProcessLifecycleOwner.get().lifecycle.addObserver(MyAp
5     }
6 }
7
8 class MyApplicationObserver : LifecycleObserver{
9     //在应用程序的整个生命周期中只会被调用一次
10    @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
11    fun onCreate(){
12        Log.e("application","Application onCreate方法执行了");
13    }
14
15
16    //在应用程序在前台出现时被调用
17    @OnLifecycleEvent(Lifecycle.Event.ON_START)
18    fun onStart(){
19        Log.e("application","Application onStart方法执行了");
20    }
21
22    //在应用程序在前台出现时被调用
23    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
```



```
24 fun onResume(){
25     Log.e("application","Application onResume方法执行了");
26 }
27
28 //在应用程序退出到后台时被调用
29 @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)
30 fun onPause(){
31     Log.e("application","Application onPause方法执行了");
32 }
33
34 //在应用程序退出到后台时被调用
35 @OnLifecycleEvent(Lifecycle.Event.ON_STOP)
36 fun onStop(){
37     Log.e("application","Application onStop方法执行了");
38 }
39
40
41 //永远不会被调用，系统不会分发调用ON_DESTROY事件
42 @OnLifecycleEvent(Lifecycle.Event.ON_DESTROY)
43 fun onDestroy(){
44     Log.e("application","Application onDestroy方法执行了");
45 }
46 }
```

PS:记得在AndroidManifest中使用我们的这个自定义Application

```
3      package="com.example.myapplication">
4
5      <uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE" />
6      <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
7      <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
8
9      <application
10         android:name=".MyApplication"
11         android:allowBackup="true"
12         android:icon="@mipmap/ic_launcher"
13         android:label="My Application"
14         android:requestLegacyExternalStorage="true"
15         android:roundIcon="@mipmap/ic_launcher_round"
16         android:supportsRtl="true"
17         android:theme="@style/Theme.MyApplication">
18         <activity android:name=".EventBusActivity">
19             <intent-filter>
20                 <action android:name="android.intent.action.MAIN" />
21
22                 <category android:name="android.intent.category.LAUNCHER" />
23             </intent-filter>
24         </activity>
25         <activity android:name=".ViewModelActivity">
26
27         </activity>
```

日志输出如下图所示:

```
logcat
2021-11-19 00:02:21.087 9108-9108/com.example.myapplication E/application: Application onCreate方法执行了
2021-11-19 00:02:21.315 9108-9108/com.example.myapplication E/application: Application onStart方法执行了
2021-11-19 00:02:21.327 9108-9108/com.example.myapplication E/application: Application onResume方法执行了
2021-11-19 00:03:08.683 9108-9108/com.example.myapplication E/application: Application onPause方法执行了
2021-11-19 00:03:08.741 9108-9108/com.example.myapplication E/application: Application onStop方法执行了
```

退出APP才会打印出来

- [Android架构之LifeCycle组件_冬瓜闯世界的博客-CSDN博客](#)
- [Android 架构组件之 LifeCycle详解 - SegmentFault 思否](#)
- [Android官方架构组件Lifecycle:生命周期组件详解&原理分析 - 简书](#)