Android Activity Fragment Lifecycle

deepwaterooo

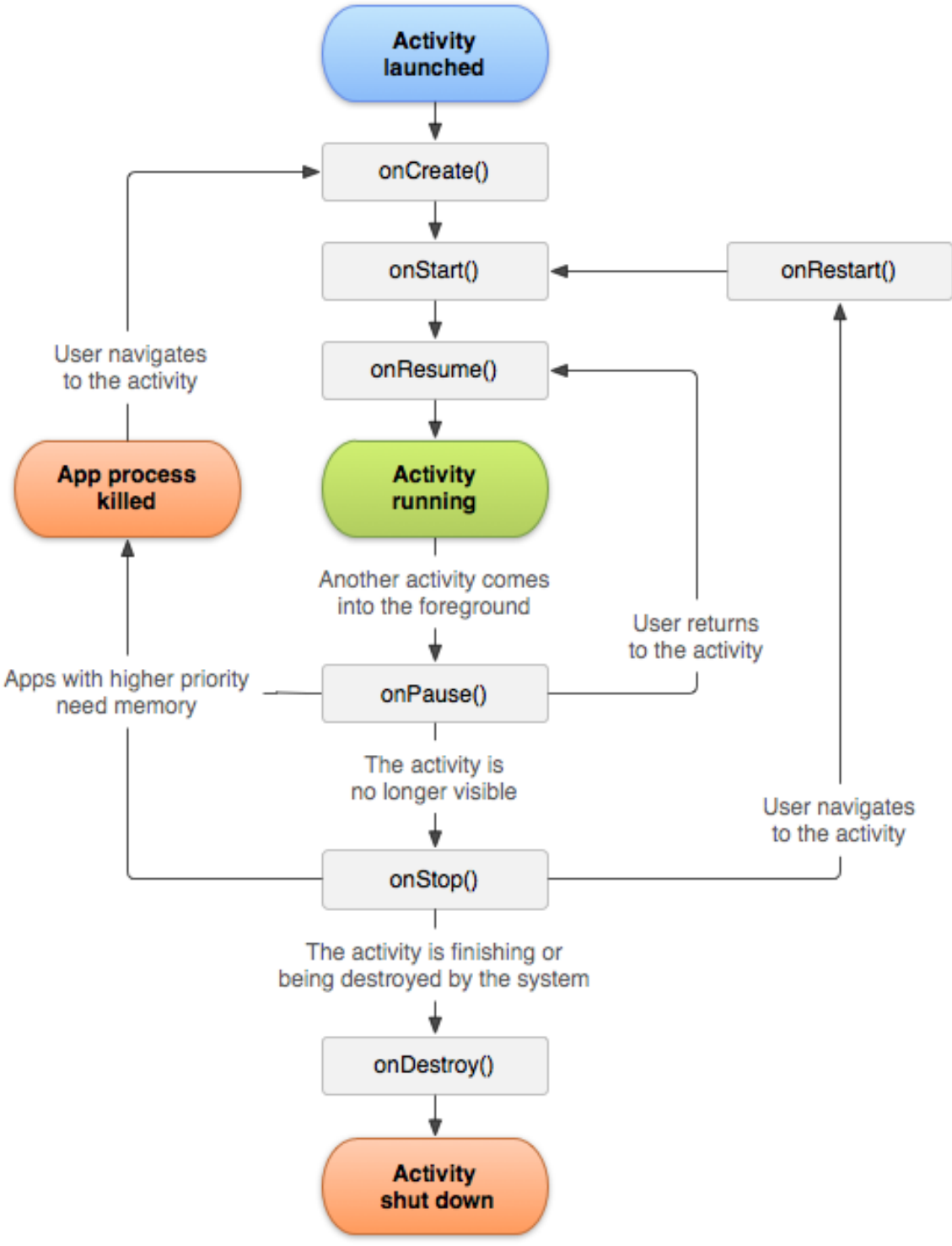
2019 年 5 月 4 日

目录

1 Activity 与 Fragment 生命周期探讨

- https://www.jianshu.com/p/1b3f829810a1

1.1 Activity 生命周期探讨



1.1.1 Activity 生命周期相关函数说明

- (1) onCreate() 是 activity 正在被创建，也就是说此时的 UI 操作不会更新 UI，比如 setText() 操作，所以此时在子线程调用 setText() 不会报线程错误。详解可见 Android 子线程更新 View 的探索, 在这个方法内我们可以做一些初始化工作。

- (2) onRestart() 需要注意的是：activity 正在重新启动，一般情况下，activity 从不可见状态到可见状态，

onRestart() 才会被调用，但是一定要注意的是一般来说这是用户行为导致 activity 不可见的时候，此时变为可见的时候才会调用 onRestart(), 这里所说的用户行为就是用户按 home 键，或者进入“新”的 activity。这样的操作会使 activity 先执行 onPause(), 后执行 onStop()，这样回到这个 activity 会调用 onRestart()。为什么我这里强调说用户行为导致的不可见状态，等下我会说。。。。

- (3) onStart() 的时候，activity 才可见，但是没有出现在前台，无法与用户交互
- (4) onResume() 的时候，activity 已经可见，并且出现在前台开始活动，与 onStart() 相比，activity 都已经可见，但是 onStart() 的时候 activity 还在后台，onResume() 才显示在前台
- (5) onPause() 主要注意的是：此时的 activity 正在被停止，接下来马上调用 onStop()。特殊情况下快速回到该 activity，onStop() 不会执行，会去执行 onResume()。
  - 一般在这个生命周期内做存储数据、停止动画工作，但不能太耗时。
  - 为什么特殊强调呢，因为该 activity 的 onPause() 执行完了，才回去执行新的 activity 的 onResume()，一旦耗时，必然会拖慢新的 activity 的显示。
- (6) onStop()：此时的 activity 即将停止。在这里可以做稍微重量级的操作，同样也不能耗时。
- (7) onDestroy()：此时的 activity 即将被回收，在这里会做一些回收工作和最终资源释放。

1.1.2 他人总结

- https://juejin.im/post/5a18f58651882531bb6c82e2
- 1. 打开一个全新的 activityA：
  - onCreate() —-> onStart() —-> onResume()
- 2. 从 activity A —-> activity B（全屏）：
  - activity A 先执行: onPause()
  - 然后 activity B 执行: onCreate() —-> onStart() —-> onResume()
  - activity A 再执行: onStop()
- 3. 从 activity A —-> activity B（非全屏）：
  - activity A 先执行: onPause()
  - 然后 activity B 执行: onCreate() —-> onStart() —-> onResume()
  - **activity A 不会执行 onStop()**
- 4. activity B（全屏）返回到 activity A：
  - activity B 先执行: onPause()
  - activity A: onRestart —-> onStart() —-> onResume()
  - activity B 再执行: onStop() —-> onDestory()
- 5. activity B（非全屏）返回到 activity A
  - activity B 先执行: onPause()
  - activity A: onResume()
  - activity B 再执行: onStop() —-> onDestory()

- 6. activity B 返回到 activity A：
  - 如果 activityA 已经被销毁，activityA 会重新创建，执行: onCreate() —-> onStart() —-> onResume()
  - activityB 的流程不变
- 7. activity A 按 home 键退居后台：
  - 同 2 的流程: onPause()
- 8. 再从 home 返回到 activity A
  - 同 4 的流程: onRestart —-> onStart() —-> onResume()

1.1.3 Activity 注意事项

- Activity 中所有和状态相关的回调函数：

- 在这里我会特别提出一个 point，就是异常情况下 activity 被杀死，而后被重新创建的情况。

图 1: 异常情况下 activity 的重建过程

- 这张图非常重要，可以帮我们解决异常情况下 activity 如何正常回复的问题
- 当系统停止 activity 时，它会调用 onSaveInstanceState()(过程 1)，如果 activity 被销毁了，但是需要创建同样的实例，系统会把过程 1 中的状态数据传给 onCreate() 和 onRestoreInstanceState()，所以我们要在 onSaveInstanceState() 内做保存参数的动作，在 onRestoreInstanceState() 做获取参数的动作。

```
1  // Save Activity State
2  static final String STATE_SCORE = "playerScore";
3  static final String STATE_LEVEL = "playerLevel";
4  @Override
5  public void onSaveInstanceState(Bundle savedInstanceState) {
6      // Save the user's current game state
7      savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
8      savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);
```

```
    // Always call the superclass so it can save the view hierarchy state
    super.onSaveInstanceState(savedInstanceState);
}
```

- 获取参数操作：

```
// onCreate() 方法
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Always call the superclass first

    // Check whether we're recreating a previously destroyed instance
    if (savedInstanceState != null) {
        // Restore value of members from saved state
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
    } else {
        // Probably initialize members with default values for a new instance
    }
}
```

- 也可以

```
// onRestoreInstanceState() 方法
public void onRestoreInstanceState(Bundle savedInstanceState) {
    // Always call the superclass so it can restore the view hierarchy
    super.onRestoreInstanceState(savedInstanceState);

    // Restore state members from saved instance
    mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
    mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
}
```

1.1.4 一些特殊的方法

1. onWindowFocusChanged()
   - 在 Activity 窗口获得或失去焦点时被调用并且当 Activity 被创建时是在 onResume 之后被调用，当 Activity 被覆盖或者退居后台或者当前 Activity 退出时，它是在 onPause 之后被调用（在这个方法中可以 view 已经绘制完成，可以获取 view 的宽高等属性）
2. onSaveInstanceState()
   - (1) 在 Activity 被覆盖或退居后台之后，系统资源不足将其杀死，此方法会被调用；
   - (2) 在用户改变屏幕方向时，此方法会被调用；
   - (3) 在当前 Activity 跳转到其他 Activity 或者按 Home 键回到主屏，自身退居后台时，此方法会被调用。
   - 第一种情况我们无法保证什么时候发生，系统根据资源紧张程度去调度；
   - 第二种是屏幕翻转方向时，系统先销毁当前的 Activity，然后再重建一个新的，调用此方法时，我们可以保存一些临时数据；

- 第三种情况系统调用此方法是为了保存当前窗口各个 View 组件的状态。
- onSaveInstanceState 的调用顺序是在 onstop 之前。（android3.0 之前：在 onPause 之前调用，在 3.0 之后，在 onPause 之后调用）

3. onRestoreInstanceState()

- 有的人说这个方法和 onSaveInstanceState 是一对，其实不然，
- (1) 在 Activity 被覆盖或退居后台之后，系统资源不足将其杀死，然后用户又回到了此 Activity，此方法会被调用；
- (2) 在用户改变屏幕方向时，重建的过程中，此方法会被调用。
- 我们可以重写此方法，以便可以恢复一些临时数据。
- onRestoreInstanceState 的调用顺序是在 onStart 之后。
- 在当前 Activity 跳转到其他 Activity 或者按 Home 键回到主屏，自身退居后台时：onRestoreInstanceState 不会调用，但是 onSaveInstanceState 会调用，这点就是区别

1.1.5 最后的总结

- 当 Activity 被系统撤销后重新建立时，保存以及恢复数据的函数调用顺序是：
  - onSaveInstanceState(保存数据) —-> onCreate(恢复数据 allstate) —-> onRestoryInstanceState(恢复数据 HierarchyState)
- 如果要取消切换屏幕方法重建 activity，可以配置 configChanges 属性：当支持的最小 sdk 版本大于 android4.0 需要设置这个属性）

```
android:configChanges="keyboardHidden|orientation|screenSize（当支持的最小 sdk 版本大于 and
```

1.2 Fragment 生命周期探讨

- Fragment 基本类，生命周期如下：

```
void onAttach(Context context)
void onCreate(Bundle savedInstanceState)
View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
void onActivityCreated(Bundle savedInstanceState)
void onStart()
void onStop()
void onResume()
void onPause()
void onDestroyView()
void onDestroy()
void onDetach()
```

1.2.1 首先需要提出的一些 points：

- Fragment 是直接从 Object 继承的，而 Activity 是 Context 的子类。因此我们可以得出结论：Fragment 不是 Activity 的扩展。但是与 Activity 一样，在我们使用 Fragment 的时候我们总会扩展 Fragment(或者是她的子类)，并可以通过子类更改她的行为。

- 使用 Fragment 时，必要构建一个无参构造函数，系统会默认带。但一但写有参构造函数，就必要构建无参构造函数。一般来说我们传参数给 Fragment，会通过 bundle，而不会用构造方法传，代码如下：

```
public static MyFragment newInstance(int index){
    MyFragment mf = new MyFragment();
    Bundle args = new Bundle();
    args.putInt("index",index);
    mf.setArguments(args);
    return mf;
}
```

1.2.2 生命周期

- onAttach()：onAttach() 回调将在 Fragment 与其 Activity 关联之后调用。需要使用 Activity 的引用或者使用 Activity 作为其他操作的上下文，将在此回调方法中实现。
  - 将 Fragment 附加到 Activity 以后，就无法再次调用 setArguments()–除了在最开始，无法向初始化参数添加内容。
- onCreate(Bundle savedInstanceState)：此时的 Fragment 的 onCreate() 回调时，该 fragmet 还没有获得 Activity 的 onCreate() 已完成的通知，所以不能将依赖于 Activity 视图层次结构存在性的代码放入此回调方法中。在 onCreate() 回调方法中，我们应该尽量避免耗时操作。此时的 bundle 就可以获取到 activity 传来的参数

```
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Bundle args = getArguments();
        if (args != null) {
            mLabel = args.getCharSequence("label", mLabel);
        }
    }
```

- onCreateView()

```
onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
```

  - 其中的 Bundle 为状态包与上面的 bundle 不一样。
  - 不要将视图层次结构附加到传入的 ViewGroup 父元素中，该关联会自动完成。如果在此回调中将碎片的视图层次结构附加到父元素，很可能会出现异常。
  - 这句话什么意思呢？就是不要把初始化的 view 视图主动添加到 container 里面，以为这会系统自带，所以 inflate 函数的第三个参数必须填 false，而且不能出现 container.addView(v) 的操作。

```
View v = inflater.inflate(R.layout.hello_world, container, false);
```

- onActivityCreated()
  - onActivityCreated() 回调会在 Activity 完成其 onCreate() 回调之后调用。在调用 onActivityCreated() 之前，Activity 的视图层次结构已经准备好了，这是在用户看到用户界面之前你可对用户界面执行的最后调整的地方。

  - 如果 Activity 和她的 Fragment 是从保存的状态重新创建的，此回调尤其重要，也可以在这里确保此 Activity 的其他所有 Fragment 已经附加到该 Activity 中了

- Fragment 与 Activity 相同生命周期调用：接下来的 onStart(), onResume(), onPause(), onStop() 回调方法将和 Activity 的回调方法进行绑定，也就是说与 Activity 中对应的生命周期相同，因此不做过多介绍。
- onDestroyView(): 该回调方法在视图层次结构与 Fragment 分离之后调用。
- onDestroy()：不再使用 Fragment 时调用。(备注：Fragment 仍然附加到 Activity 并任然可以找到，但是不能执行其他操作)
- onDetach()：Fragme 生命周期最后回调函数，调用后，Fragment 不再与 Activity 绑定，释放资源。

1.2.3 Fragment 每个生命周期方法的意义、作用

- onAttach()
  - 执行该方法时，Fragment 与 Activity 已经完成绑定，该方法有一个 Activity 类型的参数，代表绑定的 Activity，这时候你可以执行诸如 mActivity = activity 的操作。
- onCreate()
  - 初始化 Fragment。可通过参数 savedInstanceState 获取之前保存的值。
- onCreateView()
  - 初始化 Fragment 的布局。加载布局和 findViewById 的操作通常在此函数内完成，但是不建议执行耗时的操作，比如读取数据库数据列表。
- onActivityCreated()
  - 执行该方法时，与 Fragment 绑定的 Activity 的 onCreate 方法已经执行完成并返回，在该方法内可以进行与 Activity 交互的 UI 操作，所以在该方法之前 Activity 的 onCreate 方法并未执行完成，如果提前进行交互操作，会引发空指针异常。
- onStart()
  - 执行该方法时，Fragment 由不可见变为可见状态。
- onResume()
  - 执行该方法时，Fragment 处于活动状态，用户可与之交互。
- onPause()
  - 执行该方法时，Fragment 处于暂停状态，但依然可见，用户不能与之交互。
- onSaveInstanceState()
  - 保存当前 Fragment 的状态。该方法会自动保存 Fragment 的状态，比如 EditText 键入的文本，即使 Fragment 被回收又重新创建，一样能恢复 EditText 之前键入的文本。
- onStop()
  - 执行该方法时，Fragment 完全不可见。
- onDestroyView()

  - 销毁与 Fragment 有关的视图，但未与 Activity 解除绑定，依然可以通过 onCreateView 方法重新创建视图。通常在 ViewPager+Fragment 的方式下会调用此方法。

- onDestroy()
  - 销毁 Fragment。通常按 Back 键退出或者 Fragment 被回收时调用此方法。

- onDetach()
  - 解除与 Activity 的绑定。在 onDestroy 方法之后调用。

- setUserVisibleHint()
  - 设置 Fragment 可见或者不可见时会调用此方法。在该方法里面可以通过调用 getUserVisibleHint() 获得 Fragment 的状态是可见还是不可见的，如果可见则进行懒加载操作。

1.2.4 Fragment 生命周期执行流程

- 1、Fragment 创建
  - setUserVisibleHint() —-> onAttach() —-> onCreate() —-> onCreateView() —-> onActivityCreated() —-> onStart() —-> onResume()

- 2、Fragment 变为不可见状态（锁屏、回到桌面、被 Activity 完全覆盖）
  - onPause() —-> onSaveInstanceState() —-> onStop()

- 3、Fragment 变为部分可见状态（打开 Dialog 样式的 Activity）
  - onPause() —-> onSaveInstanceState()

- 4、Fragment 由不可见变为活动状态
  - onStart() —-> OnResume()

- 5、Fragment 由部分可见变为活动状态
  - onResume()

- 6、Fragment 退出
  - onPause() —-> onStop() —-> onDestroyView() —-> onDestroy() —-> onDetach()
  -（注意退出不会调用 onSaveInstanceState 方法，因为是人为退出，没有必要再保存数据）

- 7、Fragment 被回收又重新创建
  - 被回收执行: onPause() —-> onSaveInstanceState() —-> onStop() —-> onDestroyView() —-> onDestroy() —-> onDetach()
  - 重新创建执行: onAttach() —-> onCreate() —-> onCreateView() —-> onActivityCreated() —-> onStart() —-> onResume() —-> setUserVisibleHint()

- 横竖屏切换
  - 与 Fragment 被回收又重新创建一样。

1.2.5 onHiddenChanged 的回调时机

- 当使用 add()+show()，hide() 跳转新的 Fragment 时，旧的 Fragment 回调 onHiddenChanged()，不会回调 onStop() 等生命周期方法，而新的 Fragment 在创建时是不会回调 onHiddenChanged()，这点要切记。

1.2.6 FragmentPagerAdapter+ViewPager 的注意事项

- 1、使用 FragmentPagerAdapter+ViewPager 时，切换回上一个 Fragment 页面时（已经初始化完毕），不会回调任何生命周期方法以及 onHiddenChanged()，只有 setUserVisibleHint(boolean isVisibleToUser) 会被回调，所以如果你想进行一些懒加载，需要在这里处理。
- 2、在给 ViewPager 绑定 FragmentPagerAdapter 时，new FragmentPagerAdapter(fragmentManager) 的 FragmentManager，一定要保证正确，如果 ViewPager 是 Activity 内的控件，则传递 getSupportFragmentManager()，如果是 Fragment 的控件中，则应该传递 getChildFragmentManager()。只要记住 ViewPager 内的 Fragments 是当前组件的子 Fragment 这个原则即可。
- 3、你不需要考虑在“内存重启”的情况下，去恢复的 Fragments 的问题，因为 FragmentPagerAdapter 已经帮我们处理啦。

1.2.7 setUserVisibleHint() 不调用的问题

- 通常情况下都是因为 PagerAdapter 不是 FragmentPagerAdapter 造成的，FragmentPagerAdapter 内部实现了对 setUserVisibleHint() 方法的调用，所以需要懒加载的结构最好使用 FragmentPagerAdapter +Fragment 的结构，少用 PagerAdapter。

1.2.8 Fragment 注意事项

- 在使用 Fragment 时，我发现了一个金矿，那就是 setRetainInstance() 方法, 此方法可以有效地提高系统的运行效率，对流畅性要求较高的应用可以适当采用此方法进行设置。
- Fragment 有一个非常强大的功能–就是可以在 Activity 重新创建时可以不完全销毁 Fragment，以便 Fragment 可以恢复。在 onCreate() 方法中调用 setRetainInstance(true/false) 方法是最佳位置。当 Fragment 恢复时的生命周期如上图所示，注意图中的红色箭头。当在 onCreate() 方法中调用了 setRetainInstance(true) 后，Fragment 恢复时会跳过 onCreate() 和 onDestroy() 方法，因此不能在 onCreate() 中放置一些初始化逻辑.

2 Android Fragment 生命周期图

- http://www.cnblogs.com/purediy/p/3276545.html
- Fragment 与 Activity 生命周期对比图：

2.1 生命周期分析

2.1.1 当一个 fragment 被创建的时候 (它会经历以下状态)

- onAttach()
- onCreate()
- onCreateView()
- onActivityCreated()

2.1.2 当这个 fragment 对用户可见的时候

- onStart()
- onResume()

2.1.3 当这个 fragment 进入“后台模式”的时候

- onPause()
- onStop()

2.1.4 当这个 fragment 被销毁了（或者持有它的 activity 被销毁了）

- onPause()
- onStop()
- onDestroyView()
- onDestroy() // 本来漏掉类这个回调，感谢 xiangxue336 提出。
- onDetach()

2.1.5 就像 activity 一样，在以下的状态中，可以使用 Bundle 对象保存一个 fragment 的对象。

- onCreate()
- onCreateView()
- onActivityCreated()

2.1.6 fragments 的大部分状态都和 activity 很相似，但 fragment 有一些新的状态。

- onAttached() – 当 fragment 被加入到 activity 时调用（在这个方法中可以获得所在的 activity）。
- onCreateView() – 当 activity 要得到 fragment 的 layout 时，调用此方法，fragment 在其中创建自己的 layout(界面)。
- onActivityCreated() – 当 activity 的 onCreated() 方法返回后调用此方法
- onDestroyView() – 当 fragment 中的视图被移除的时候，调用这个方法。
- onDetach() – 当 fragment 和 activity 分离的时候，调用这个方法。
- Notes:
  - 一旦 activity 进入 resumed 状态（也就是 running 状态），你就可以自由地添加和删除 fragment 了。
  - 因此，只有当 activity 在 resumed 状态时，fragment 的生命周期才能独立的运转，其它时候是依赖于 activity 的生命周期变化的。