Android移动开发课程

实验指导书-Fragment部分

【实验目的】

初步了解Android 的 Notification 组件

【实验设计】

本次实验包括三个验证实验。

其中验证实验为个人实验，已经提供源代码、操作步骤、实验指导视频

由于NetWork涉及网络通讯，因此本实验提供了一个Jsp编写的服务器，源代码已经给出，可以在Tomcat下运行。

要求：

按照实验步骤完成，

以个人为单位提交，提交实验报告一份，实验报告需要回答指导书中问题。

自选实验为小组实验，建议2-4人组队，提供了参考选题和参考资料。

要求：

以小组为单位提交，提交实验报告一份、源代码一份、可以执行的APK文件一个。

【实验内容】

Fragment 表示 FragmentActivity 中的行为或界面的一部分。应用可以在一个 Activity 中组合多个fragment，从而构建多窗格界面，并在多个 Activity 中重复使用某个fragment。fragment具有自己的生命周期，能接收自己的输入事件，并且可以在 Activity 运行时添加或移除片段，fragment必须始终托管在 Activity 中，其生命周期直接受宿主 Activity 生命周期的影响。

本处实验包括三个验证性实验。

验证实验一：了解碎片的生命周期

本实验已经提供了源代码，源代码参见（exmaple包下的ExpFragment1和Exp1Activity两个文件）

1. 了解Fragment的生命周期

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。

打开实验一，查看Logcat（过滤条件：Level为Info，关键词为Exp1，注：关键词为Exp1F可以只关注ExpFragment1的行为; 关键词为Exp1A可以只关心Exp1Activity的行为）。

自行设计实验，验证以下生命周期执行过程。

Fragment生命周期执行流程：

1、Fragment创建：
onAttach()->onCreate()->onCreateView()->onActivityCreated()->onStart()->onResume()；

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

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

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）；

6、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。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()。

参考链接：https://juejin.im/post/6844903517065314317

验证实验二：了解Fragment的事务管理

本实验已经提供了源代码，源代码参见（exmaple包下的ExpFragment1、ExpFragment2和Exp2Activity两个文件）

1. 如要管理 Activity 中的片段，需要使用 FragmentManager，在Activity中调用getSupportFragmentManager()可以获取Fragment。 可使用 FragmentManager 执行的操作包括：

通过 findFragmentById()（针对在 Activity 布局中提供界面的片段）或

findFragmentByTag()（针对提供或不提供界面的片段）获取 Activity 中存在的片段。

通过 popBackStack()（模拟用户发出的返回命令）使片段从返回栈中弹出。

通过 addOnBackStackChangedListener() 注册侦听返回栈变化的侦听器。

2. 了解FragmentTransaction

通过getSupportFragmentManager().beginTransaction()可以获得一个FragmentTransaction对象。在事务结束时必须调用commit。调用 commit() 不会立即执行事务，而是在 Activity 的界面线程（"主"线程）可执行该操作时，再安排该事务在线程上运行。

3. 了解FragmentTransaction的add方法

点击"为Left添加ExpFragment1"按钮，查看Logcat（过滤条件：Level为Info，关键词为Exp[2|[1F]],记得把Regex勾选）；点击返回键， 查看Logcat，对比context；多次点击"为Left添加ExpFragment1"按钮，查看Logcat。

4. 了解FragmentTransaction的replace方法

点击"替换Left为ExpFragment1"按钮，查看Logcat（过滤条件：Level为Info，关键词为Exp[2|[1F]],记得把Regex勾选）；点击返回键， 查看Logcat，对比context；多次点击"为Left添加ExpFragment1"按钮，然后点击"替换Left为ExpFragment1"按钮，查看Logcat；点击返回键，查看Logcat

5. 了解FragmentTransaction的addToBackStack(null)方法

点击"替换Left为ExpFragment1\n不入返回栈"按钮，然后点击返回键，查看Logcat（过滤条件：Level为Info，关键词为Exp[2|[1F]],记得把Regex勾选）。与点击"替换Left为ExpFragment1"按钮后点击返回做对比。

6 了解利用FragmentTransaction同时提交多个事务

如果事务添加多个更改，并调用 addToBackStack()，则调用 commit() 前应用的所有更改都将作为单一事务添加到返回栈，并且返回按钮会将它们一并撤消。向 FragmentTransaction 添加更改的顺序无关紧要。

点击"替换Left为ExpFragment1\n替换Right为ExpFragment1"按钮，查看Logcat（过滤条件：Level为Info，关键词为Exp[2|[1F]],记得把Regex勾选）。点击返回键，查看Logcat。

本次验证实验提交内容： 截取Logcat、 关于这些Logcat的分析

附录： 关于fragment导致的命名冲突，参考：
https://blog.csdn.net/beta4/article/details/49362547?utm_source=blogxgwz5

验证实验三：了解fragment间的通讯

1. 了解Fragment和所属Activity的通讯

了解getActivity方法获得所属Activity的实例。

点击"测试与Activity的通讯"按钮，查看Logcat（过滤条件：Level为Info，关键词为Exp[3|[2F]],记得把Regex勾选）。

2. 了解Fragment通过回调与Activity通讯。

点击"测试通过回调与Activity的通讯"按钮，查看界面。

3. 从 Fragment 1.3.0-alpha04 开始，每个 FragmentManager 都会实现FragmentResultOwner。这意味着 FragmentManager 可以充当 Fragment 结果的集中存储区。此更改通过设置 Fragment 结果并监听这些结果，而不要求 Fragment 直接引用彼此，让单独的 Fragment 相互通信。这是今年（2020）新发布的版本，需要添加androidx.fragment:fragment:1.3.0-alpha08。

点击"测试与Fragment的通讯",查看界面。

4. 在父级 Fragment 和子级 Fragment 之间传递结果

点击"测试与ChildFragment的通讯"按钮，查看页面。

参考：https://developer.android.com/jetpack/androidx/releases/fragment

参考：https://www.codenong.com/js773a70ee288b/

参考：https://developer.android.com/training/basics/fragments/pass-data-between

拓展：了解ViewModel、LiveData、MVVM设计模式