Fragment是什么？

Fragment 表示应用界面中可重复使用的一部分。Fragment 定义和管理自己的布局，具有自己的生命周期，并且可以处理自己的输入事件。Fragment 不能独立存在，而是必须由 Activity 或另一个 Fragment 托管。Fragment 的视图层次结构会成为宿主的视图层次结构的一部分，或附加到宿主的视图层次结构。

Fragment是Activity里面用来区分不同功能的一个个“碎片”，大家都知道面向对象的五大原则，Android程序同样也应该遵循这些原则，但是考虑移动应用是直接面向用户的，所以我们总会遇到一个页面中装着各种各样一点联系都没有的功能的情况。

在这种情况下，我们的代码也应当遵循“高聚合，低耦合”的原则，也就是说，一个页面里的一个功能应当专注于它自己，同时和其他功能之间应当是无感知的；另外的，页面本身（也即是Activity），也应当对其中的功能仅保持最低限度的了解，这样才能增强代码的灵活性和可扩展性。

Fragment用来做什么？

Fragment就是用来将同一个页面里面有着不同功能的部分解藕的。

Fragment有什么优点？

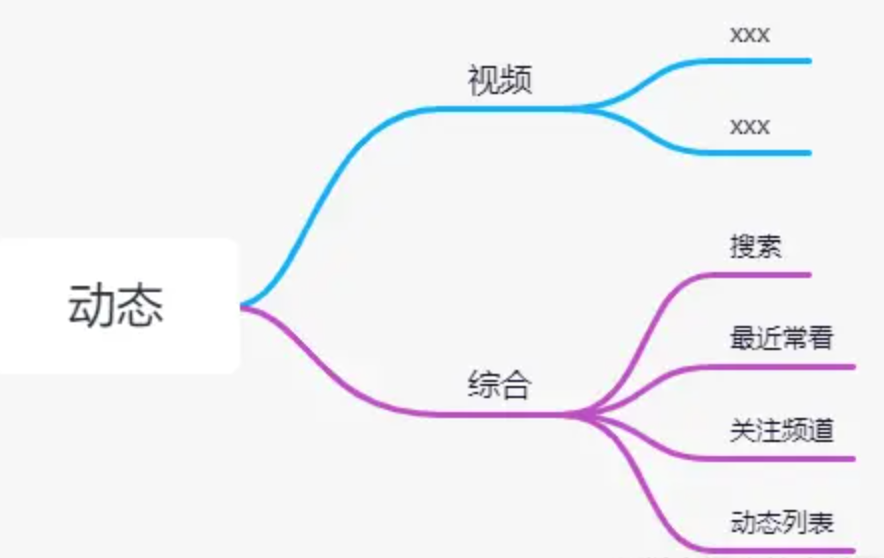
Fragment拥有自己的生命周期，让你能够更好地根据不同的生命周期来控制其中的具体逻辑；

Fragment是官方支持，这一方面说明它会随着Android的版本更新增加功能，同时越来越稳定；另一方面则是说明它是一个全局通用的“组件”，你可以将不同的Fragment拼装成一个页面，也可以随时将他们搬走，很方便地让它们在另一个页面当中生效；

Fragment很“轻”，你可以使用它来实现“单Activity架构”；

能够完美地为你的App中的不同功能进行解耦，提供良好的灵活性和可复用性。

多Fragment最佳实践



如果要在一个实现动态功能的Fragment中实现如上图的功能，页面结构可以如下实现：

1、动态功能的Fragment中，由顶部的两个Tab组成，ViewPager + Fragment。

2、由于视频和综合的页面相似，我们这边分析综合Fragment的结构，底下的滚动页面，一个RecyclerView（或者NestedScrollView + RecyclerView）全搞定。

第1步：在动态功能的Fragment中加载视频、动态的Tab。

public class MomentsFragment extends Fragment {

@Nullable

@Override

public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {

// return inflater.inflate()...

return super.onCreateView(inflater, container, savedInstanceState);

}

@Override

public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

ViewPager tabPager = view.findViewById(R.id.view\_pager);

FragmentPagerAdapter adapter = new MomentsFragmentAdaper();

// 视频 Tab

adapter.addFragment(new VideoFragment());

// 综合 Tab

adapter.addFragment(new CommonFragment());

tabPager.setAdapter(adapter);

}

}

1. 对于综合功能的Fragment，有几种实现方案：
2. 使用RecyclerView配合RecyclerView.Adapter将综合功能分为几个Type。

public class CommonPageAdapter extends RecyclerView.Adapter {

private static final int TYPE\_VIDEO = 0;

private static final int TYPE\_SEARCH = 1;

private static final int TYPE\_RECENT = 2;

private static final int TYPE\_CHANNEL = 3;

@Override

public int getItemViewType(int position) {

int type = TYPE\_VIDEO;

switch (position) {

case 0:

type = TYPE\_SEARCH;

break;

case 1:

type = TYPE\_RECENT;

break;

case 2:

type = TYPE\_CHANNEL;

break;

}

return type;

}

// 此处省略三千行

}

1. 写好几个View，然后动态添加。
2. 自定义容器作为工厂作为解耦，然后提供getView，通过传递的Type返回不同的View，然后通过动态添加。

分析以上三种方案，第一种方案代码多，可维护性差，不符合单一职责规范；第二种和第三种将不同功能封装到单独的View中，符合解耦，但是但是不满足项目架构的MVP、MVVM的结构，需要自己管理生命周期等回调。如果还需要让这些不同的模块相互通信，那么他们之间还需要相互引用，维护工作量极大。

public class CommonFragment extends Fragment {

private ContainerOrPresenter searchPresenter;

private ContainerOrPresenter recentPresenter;

private ContainerOrPresenter channelPresenter;

private ContainerOrPresenter videoPresenter;

@Override

public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

searchPresenter = new ContainerOrPresenter();

recentPresenter = new ContainerOrPresenter();

channelPresenter= new ContainerOrPresenter();

videoPresenter= new ContainerOrPresenter();

// 假设两个 Presenter 需要互相通信。

searchPresenter.setRecentPresenter(recentPresenter);

}

@Override

public void onResume() {

super.onResume();

// 模拟繁琐的生命周期控制

searchPresenter.onResume(this);

recentPresenter.onResume(this);

channelPresenter.onResume(this);

videoPresenter.onResume(this);

}

}

1. 多Fragment+ViewModel

多Fragment将不同模块进行解耦，符合单一职责原则，Fragment的生命周期交给系统维护，ViewModel处理不同模块之间的数据通信。

修改后的实现动态功能的Fragment的页面结构可以如下实现：

1、动态功能的Fragment中，由顶部的两个Tab组成，ViewPager + Fragment。

2、由于视频和综合的页面相似，我们这边分析综合Fragment的结构，底下的滚动页面：NestedScrollView + 4个Fragment。



我们无需在意滚动页面里面的Fragment装的到底是什么，我们只需要简单的在外层Fragment里面根据需要，new Fragment()，然后 add 即可，不同的部分所要实现的不同功能，我们可以全部放在不同的Fragment里面，而不同的Fragment之间我们通过ViewModel来通信即可。

public class MomentsFragment extends Fragment {

private MomentsViewModel momentsViewModel;

private FragmentActivity hostActivity;

@Override

public void onAttach(@NonNull Context context) {

super.onAttach(context);

hostActivity = (FragmentActivity) context;

}

@Override

public void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

// 这里 of 方法传入的参数可以是Fragment也可以是Activity

// 如果此 Fragment 需要和Activity或者和Activity下的其他Fragments通讯，建议传入Activity作为参数

// 值得注意的是，传入Activity创建的ViewModel会一直存续到Activity销毁，所以注意内存泄露问题

momentsViewModel = ViewModelProviders.of(hostActivity).get(MomentsViewModel.class);

momentsViewModel.newMessageLiveData.observe(this, new Observer<Integer>() {

@Override

public void onChanged(Integer integer) {

// 这里模拟网络请求回传了“动态”tab右上角的新消息数量

}

});

}

@Override

public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

ViewPager tabPager = view.findViewById(R.id.view\_pager);

// 这里传入的应该是 getChildFragmentManager，不做过多解释

MomentsFragmentAdapter adapter = new MomentsFragmentAdapter(

getChildFragmentManager(), FragmentPagerAdapter.BEHAVIOR\_RESUME\_ONLY\_CURRENT\_FRAGMENT);

adapter .add(new VideoFragment());

adapter .add(new VideoFragment());

tabPager.setAdapter(adapter);

}

@Override

public void onResume() {

super.onResume();

// 这里模拟进行了一个网络请求

momentsViewModel.requestNewMessage();

}

@Override

public void onDestroy() {

super.onDestroy();

// 避免内存泄露

momentsViewModel.newMessageLiveData.removeObservers(this);

}

}

1. 多Fragment+ViewModel+ARouter

上面我们看到在实现动态功能的Fragment我们通过手动new的方式添加了两个Fragment。

这样就会造成Fragment之间的依赖。通过ARouter我们甚至可以不知道具体实现的Fragment类。如下所示：



@Route(path = Routes.ROUTE\_COMMON)

public class CommonFragment extends Fragment {

private CommonViewModel commonViewModel;

private FragmentActivity hostActivity;

@Override

public void onAttach(@NonNull Context context) {

super.onAttach(context);

hostActivity = (FragmentActivity) context;

}

@Override

public void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

// 这里 of 方法传入的参数可以是Fragment也可以是Activity

// 如果此 Fragment 需要和Activity或者和Activity下的其他Fragments通讯，建议传入Activity作为参数

// 值得注意的是，传入Activity创建的ViewModel会一直存续到Activity销毁，所以注意内存泄露问题

commonViewModel = ViewModelProviders.of(this).get(CommonViewModel.class);

}

@Override

public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

// 假设这个LinearLayout包裹在NestedScrollView里面，假设它叫“ll\_container”

// 借用ARouter来创建Fragment后，我们只需要四个路径，此容器Fragment的工作就全部完成啦~

FragmentManager manager = getChildFragmentManager();

FragmentTransaction transaction = manager.beginTransaction();

transaction.add(R.id.ll\_container, createFragmentByPath(Routes.ROUTE\_SEARCH), Routes.ROUTE\_SEARCH);

transaction.add(R.id.ll\_container, createFragmentByPath(Routes.ROUTE\_RECENT), Routes.ROUTE\_RECENT);

transaction.add(R.id.ll\_container, createFragmentByPath(Routes.ROUTE\_CHANNEL), Routes.ROUTE\_CHANNEL);

transaction.add(R.id.ll\_container, createFragmentByPath(Routes.ROUTE\_VIDEO\_LIST), Routes.ROUTE\_VIDEO\_LIST);

transaction.commit();

}

private Fragment createFragmentByPath(String path) {

return (Fragment) ARouter.getInstance().build(path).navigation();

}

}

这里创建ViewModel是为了实现父Fragment和其所有子Fragment之间的通信。

可以看到，我们从原本的一个大类负责所有的功能，逐渐变成了大类只做容器，功能细化拆分给不同的child fragment，从而提高灵活性和复用性。

这样做还有一个好处，那就是不同的页面现在可以交给不同的人来开发了，避免了冲突，提高了效率。

Fragment + ARouter 好处

1.解耦

2.传参优化&依赖注入

3.通过路径统一实现页面的多端统一

第一点不用说，我们原本想要创建一个Fragment，需要知道类名，现在知道路由地址path就可以了。

第二点是ARouter提供的功能，具体可以在ARouter的项目里查看，我们都知道如果给Fragment的构造函数添加参数来传参是个天坑，是个Crash的经典来源，使用ARouter可以帮我们简化许多工作。

第三点则是，当页面和路径对应起来以后，我们通过一个路径可以在安卓，ios，甚至是h5端都对应到同一个页面。理论上服务器只要下发一套这样的路径配置，我们就能把同一个页面通过不同的样式组合起来，或是轻松实现把首页的不同tab调换顺序。又比如服务器下发路径就可以在路径对应的页面上配置弹窗或者广告，悬浮窗等，十分灵活。最后还有，一个页面我们可以分别对应一个原生页和H5页，如果原生页路径找不到，这个路径（我们可以把它拼装成url)就可以直接降级打开一个web页面。

Fragment + ViewModel 的好处

1.实现Fragment与Activity，Fragment与Fragment之间的通信。

2.轻松实现MVVM模式。

当需要在多个 Fragment 之间或 Fragment 与其宿主 Activity 之间共享数据时，ViewModel 是理想的选择。ViewModel 对象可存储和管理界面数据。

Fragment 及其宿主 Activity 均可通过将 Activity 传入 ViewModelProvider 构造函数来使用 Activity 范围检索 ViewModel 的共享实例。ViewModelProvider 负责实例化 ViewModel 或检索它（如果已存在）。这两个组件都可以观察和修改此数据。

同一 Activity 中的两个或更多 Fragment 通常需要相互通信。例如，假设有一个 Fragment 显示一个列表，另一个 Fragment 允许用户对该列表应用各种过滤器。如果 Fragment 不直接通信（这意味着，它们不再独立），这种情况可能不容易实现。此外，这两个 Fragment 还必须处理另一个 Fragment 尚未创建或不可见的情况。

这很好实现，我们只需要 ViewModelProviders.of(hostActivity).get(CommonViewModel.class);即可，只要of传入的实例是同一个，那么这个方法我们就能得到相同的ViewModel。

这两个 Fragment 可以使用其 Activity 范围共享 ViewModel 来处理这种通信。通过以这种方式共享 ViewModel，Fragment 不需要相互了解，Activity 也不需要执行任何操作来促进通信。

同样的，我们只需要 ViewModelProviders.of(getParentFragment()).get(CommonViewModel.class);即可，只要of传入的实例是同一个，那么这个方法我们就能得到相同的ViewModel。