**Fragment**

**静态创建**

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

tools:context="com.usher.fragment.MainActivity">

<fragment

android:id="@+id/myfragment"

android:name="com.usher.fragment.MyFragment"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

</LinearLayout>

public class MyFragment extends Fragment {

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

View v = inflater.inflate(R.layout.item\_fragment, container, false);

return v;

}

}

在Activity中声明该类，与普通的View对象一样

**动态创建**

fragment1 = new MyFragment();

//初始化FragmentManager对象

manager = getSupportFragmentManager();

//使用FragmentManager对象用来开启一个Fragment事务

FragmentTransaction transaction = manager.beginTransaction();

//默认显示fragment1

transaction.add(R.id.myframelayout, fragment1).commit();

**FragmentTransaction对象**【以下直接用transaction代替】，transaction的方法主要有以下几种：

transaction.add() 向Activity中添加一个Fragment

transaction.remove() 从Activity中移除一个Fragment，如果被移除的Fragment没有添加到回退栈（回退栈后面会详细说），这个Fragment实例将会被销毁

transaction.replace() 使用另一个Fragment替换当前的，实际上就是remove()然后add()的合体

transaction.hide() 隐藏当前的Fragment，仅仅是设为不可见，并不会销毁

transaction.show() 显示之前隐藏的Fragment

detach() 会将view从UI中移除,和remove()不同,此时fragment的状态依然由FragmentManager维护

attach() 重建view视图，附加到UI上并显示

ransatcion.commit() 提交事务

注意：在add/replace/hide/show以后都要commit其效果才会在屏幕上显示出来

**什么是Fragment的回退栈？**

Fragment的回退栈是用来保存每一次Fragment事务发生的变化 如果你将Fragment任务添加到回退栈，当用户点击后退按钮时，将看到上一次的保存的Fragment。一旦Fragment完全从后退栈中弹出，用户再次点击后退键，则退出当前Activity

**如何处理运行时配置发生变化**

因为当屏幕发生旋转，Activity发生重新启动，默认的Activity中的Fragment也会跟着Activity重新创建；这样造成当旋转的时候，本身存在的Fragment会重新启动，然后当执行Activity的onCreate时，又会再次实例化一个新的Fragment，这就是出现的原因。

那么如何解决呢：

通过检查onCreate的参数Bundle savedInstanceState就可以判断，当前是否发生Activity的重新创建

默认的savedInstanceState会存储一些数据，包括Fragment的实例

所以，我们简单改一下代码，判断只有在savedInstanceState==null时，才进行创建Fragment实例

MainActivity.class文件

现在无论进行多次旋转都只会有一个Fragment实例在Activity中，现在还存在一个问题，就是重新绘制时，Fragment发生重建，原本的数据如何保持？ 和Activity类似，Fragment也有onSaveInstanceState的方法，在此方法中进行保存数据，然后在onCreate或者onCreateView或者onActivityCreated进行恢复都可以。

**Fragment与Activity之间的通信**

Fragment依附于Activity存在，因此与Activity之间的通信可以归纳为以下几点：

如果你Activity中包含自己管理的Fragment的引用，可以通过引用直接访问所有的Fragment的public方法

如果Activity中未保存任何Fragment的引用，那么没关系，每个Fragment都有一个唯一的TAG或者ID,可以通过getFragmentManager.findFragmentByTag()或者findFragmentById()获得任何Fragment实例，然后进行操作

Fragment中可以通过getActivity()得到当前绑定的Activity的实例，然后进行操作。

**一个优美的代码**

FragmentOne.class文件

public class FragmentOne extends Fragment implements OnClickListener {

private Button mBtn;

//设置按钮点击的回调

public interface FOneBtnClickListener {

void onFOneBtnClick();

}

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.fragment\_one, container, false);

mBtn = (Button) view.findViewById(R.id.id\_fragment\_one\_btn);

mBtn.setOnClickListener(this);

return view;

}

//交给宿主Activity处理，如果它希望处理

@Override

public void onClick(View v) {

if (getActivity() instanceof FOneBtnClickListener) {

((FOneBtnClickListener) getActivity()).onFOneBtnClick();

}

}

}

FragmentTwo.class文件

public class FragmentTwo extends Fragment implements OnClickListener {

private Button mBtn ;

private FTwoBtnClickListener fTwoBtnClickListener ;

public interface FTwoBtnClickListener {

void onFTwoBtnClick();

}

//设置回调接口

public void setfTwoBtnClickListener(FTwoBtnClickListener fTwoBtnClickListener) {

this.fTwoBtnClickListener = fTwoBtnClickListener;

}

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.fragment\_two, container, false);

mBtn = (Button) view.findViewById(R.id.id\_fragment\_two\_btn);

mBtn.setOnClickListener(this);

return view ;

}

@Override

public void onClick(View v) {

if(fTwoBtnClickListener != null) {

fTwoBtnClickListener.onFTwoBtnClick();

}

}

}

MainActivity.class文件

public class MainActivity extends Activity implements FOneBtnClickListener,

FTwoBtnClickListener {

private FragmentOne mFOne;

private FragmentTwo mFTwo;

private FragmentThree mFThree; //FragmentThree代码参考上一个例子中的代码

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

requestWindowFeature(Window.FEATURE\_NO\_TITLE);

setContentView(R.layout.activity\_main);

If (savedInstanceState == null)

{

mFOne = new FragmentOne();

FragmentManager fm = getFragmentManager();

FragmentTransaction tx = fm.beginTransaction();

tx.add(R.id.id\_content, mFOne, "ONE");

tx.commit();

}

}

//FragmentOne 按钮点击时的回调

@Override

public void onFOneBtnClick() {

if (mFTwo == null) {

mFTwo = new FragmentTwo();

mFTwo.setfTwoBtnClickListener(this);

}

FragmentManager fm = getFragmentManager();

FragmentTransaction tx = fm.beginTransaction();

tx.replace(R.id.id\_content, mFTwo, "TWO");

tx.addToBackStack(null);

tx.commit();

}

//FragmentTwo按钮点击时的回调

@Override

public void onFTwoBtnClick() {

if (mFThree == null) {

mFThree = new FragmentThree();

}

FragmentManager fm = getFragmentManager();

FragmentTransaction tx = fm.beginTransaction();

tx.hide(mFTwo);

tx.add(R.id.id\_content, mFThree, "THREE");

//tx.replace(R.id.id\_content, fThree, "THREE");

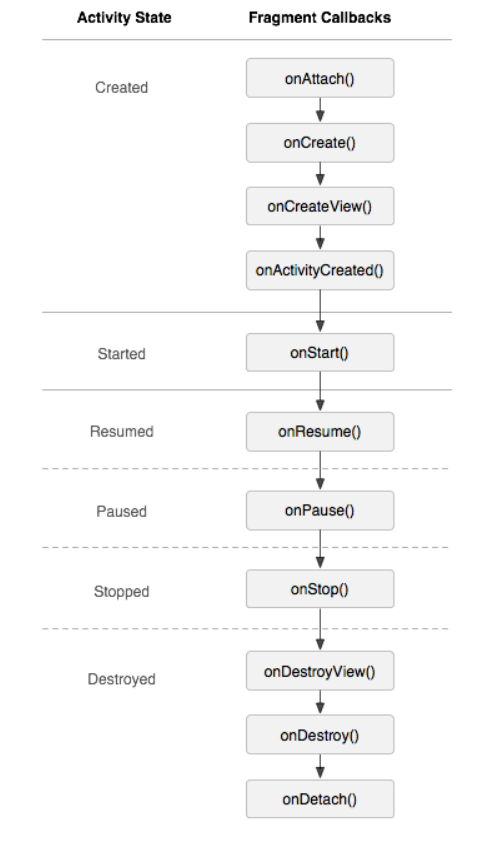
tx.addToBackStack(null);

tx.commit();

}

}

代码重构结束，与开始的效果一模一样。上面两种通信方式都是值得推荐的，随便选择一种自己喜欢的。这里再提一下：虽然Fragment和Activity可以通过getActivity与findFragmentByTag或者findFragmentById，进行任何操作，甚至在Fragment里面操作另外的Fragment，但是没有特殊理由是绝对不提倡的。Activity担任的是Fragment间类似总线一样的角色，应当由它决定Fragment如何操作。另外虽然Fragment不能响应Intent打开，但是Activity可以，Activity可以接收Intent，然后根据参数判断显示哪个Fragment。



Fragment比Activity多了几个生命周期的回调方法

onAttach(Activity) 当Fragment与Activity发生关联的时候调用

onCreateView(LayoutInflater, ViewGroup, Bundle) 创建该Fragment的视图

onActivityCreated(Bundle) 当Activity的onCreated方法返回时调用

onDestroyView() 与onCreateView方法相对应，当该Fragment的视图被移除时调用

onDetach() 与onAttach方法相对应，当Fragment与Activity取消关联时调用

PS：注意：除了onCreateView，其他的所有方法如果你重写了，必须调用父类对于该方法的实现