LRH1993 fragment详解 e5853fe on 12 Sep 2017

1 contributor

687 lines (506 sloc) | 24.7 KB

一、目录

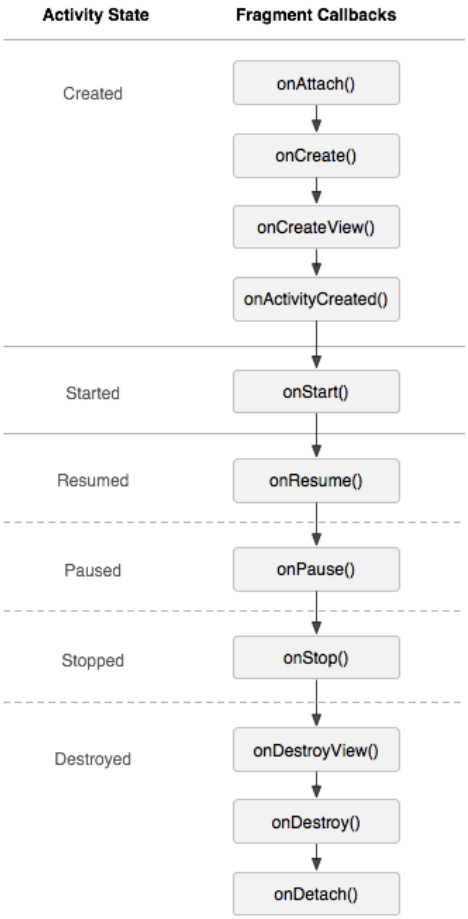
二、Fragment详解

1. 什么是Fragment ？

你可以简单的理解为，Fragment是显示在Activity中的Activity。它可以显示在Activity中，然后它也可以显示出一些内容。因为它拥有自己的生命周期，可以接受处理用户的事件，并且你可以在一个Activity中动态的添加，替换，移除不同的Fragment，因此对于信息的展示具有很大的便利性。

2. Fragment的生命周期

因为Fragment是依附于Activity存在的，因此它的生命周期收到Activity的生命周期影响



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

3. Fragment的使用方式

静态使用Fragment

步骤：

① 创建一个类继承Fragment，重写onCreateView方法，来确定Fragment要显示的布局

② 在Activity中声明该类，与普通的View对象一样

代码演示

MyFragment对应的布局文件item_fragment.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:background="@color/colorAccent"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:src="@mipmap/ic_launcher" />

</RelativeLayout>
```

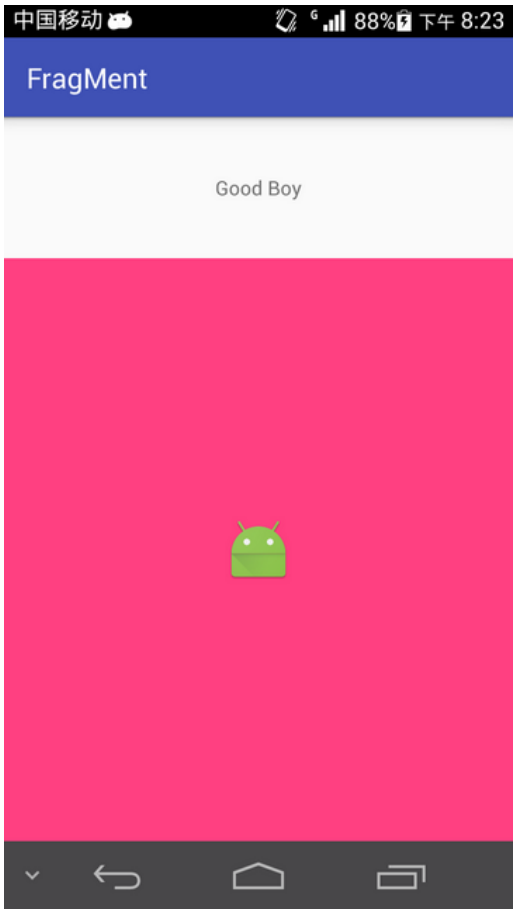
继承Frgmanet的类MyFragment【请注意导包的时候导v4的Fragment的包】

```
public class MyFragment extends Fragment {
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        /*
        * 参数1：布局文件的id
        * 参数2：容器
        * 参数3：是否将这个生成的View添加到这个容器中去
        * 作用是将布局文件封装在一个View对象中，并填充到此Fragment中
        * */
        View v = inflater.inflate(R.layout.item_fragment, container, false);
        return v;
    }
}
```

Activity对应的布局文件

```
<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">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="100dp"
        android:gravity="center"
        android:text="Good Boy" />

    <fragment
        android:id="@+id/myfragment"
        android:name="com.usher.fragment.MyFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

运行效果图



动态使用Fragment

实现点击不同的按钮，在Activity中显示不同的Fragment

代码演示

Fragment对应的布局文件item_fragment.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:background="@color/colorAccent" //背景红色
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:src="@mipmap/ic_launcher" />

</RelativeLayout>
```

继承Frgmanet的类MyFragment

```
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;
    }
}
```

Fragment2对应的布局文件item_fragment2.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:background="@color/colorPrimary" //背景蓝色
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:src="@mipmap/ic_launcher" />

</RelativeLayout>
```

继承Fragment2的类

```
public class MyFragment extends Fragment {
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.item_fragment2, container, false);
        return v;
    }
}
```

MainActivity对应的布局文件

```
<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">

    <Button
        android:id="@+id/bt_red"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Red" />

    <Button
```

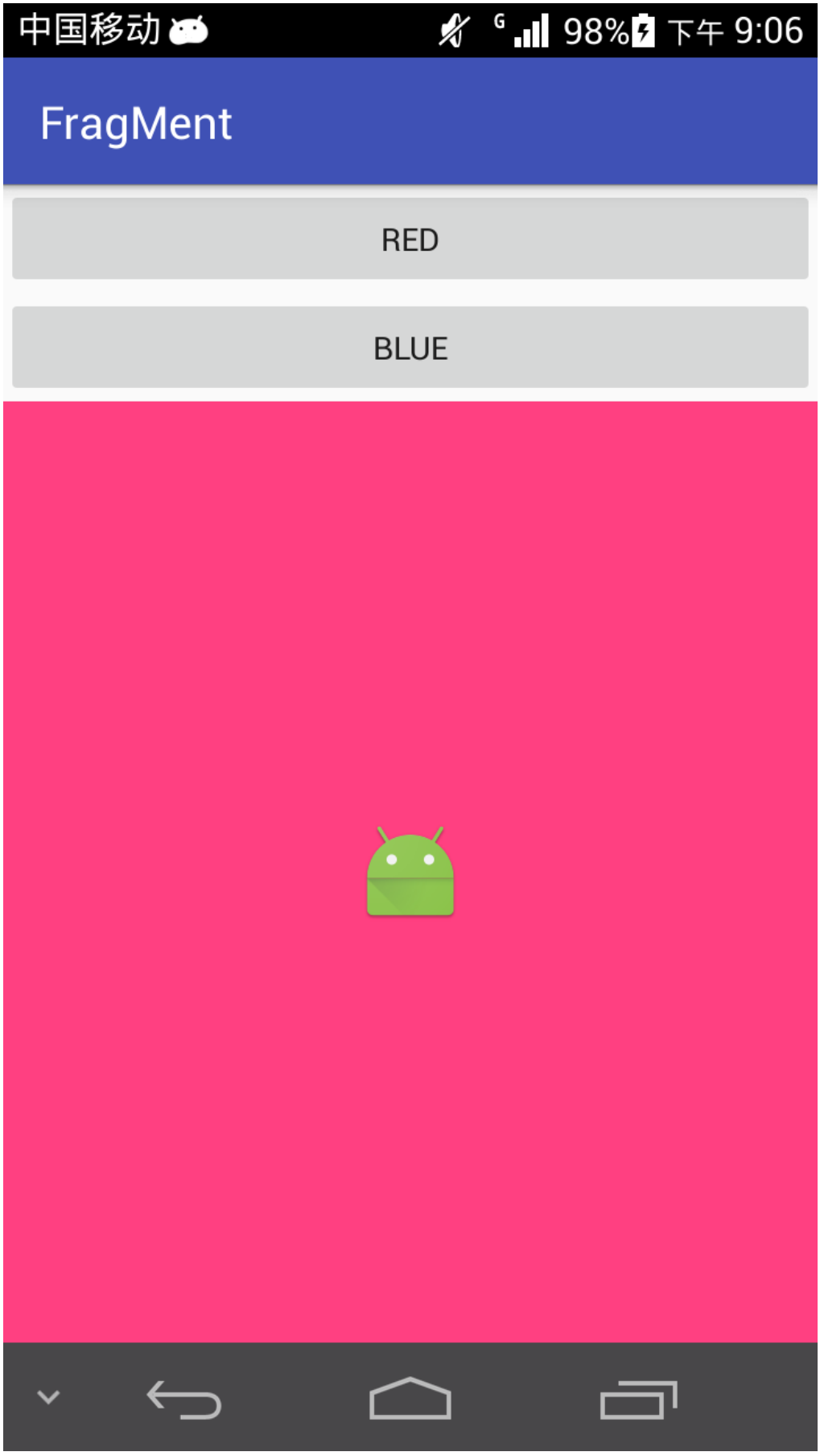
```
        android:id="@+id/bt_blue"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Blue" />

    <FrameLayout
        android:id="@+id/myframelayout"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

MainActivity类

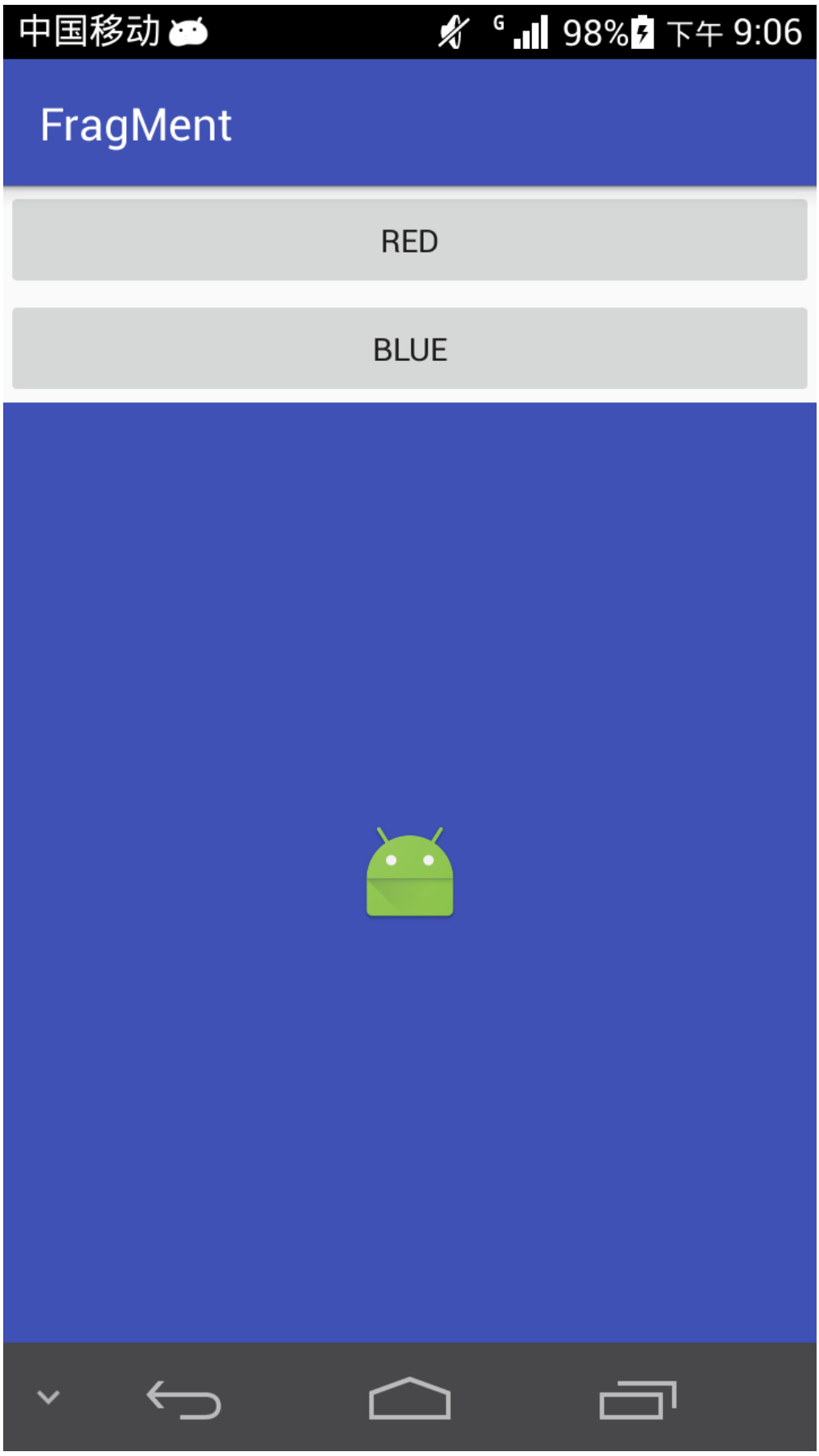
```
public class MainActivity extends AppCompatActivity {

    private Button bt_red;
    private Button bt_blue;
    private FragmentManager manager;
    private MyFragment fragment1;
    private MyFragment2 fragment2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        initView();

        fragment1 = new MyFragment();
        fragment2 = new MyFragment2();

        //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();

        //对bt_red设置监听
        bt_red.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                FragmentTransaction transaction = manager.beginTransaction();
                transaction.replace(R.id.myframelayout, fragment1).commit();
            }
        });

        //对bt_blue设置监听
        bt_blue.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                FragmentTransaction transaction = manager.beginTransaction();
                transaction.replace(R.id.myframelayout, fragment2).commit();
            }
        });
    }

    private void initView() {
        bt_red = (Button) findViewById(R.id.bt_red);
        bt_blue = (Button) findViewById(R.id.bt_blue);
    }

}
```

显示效果

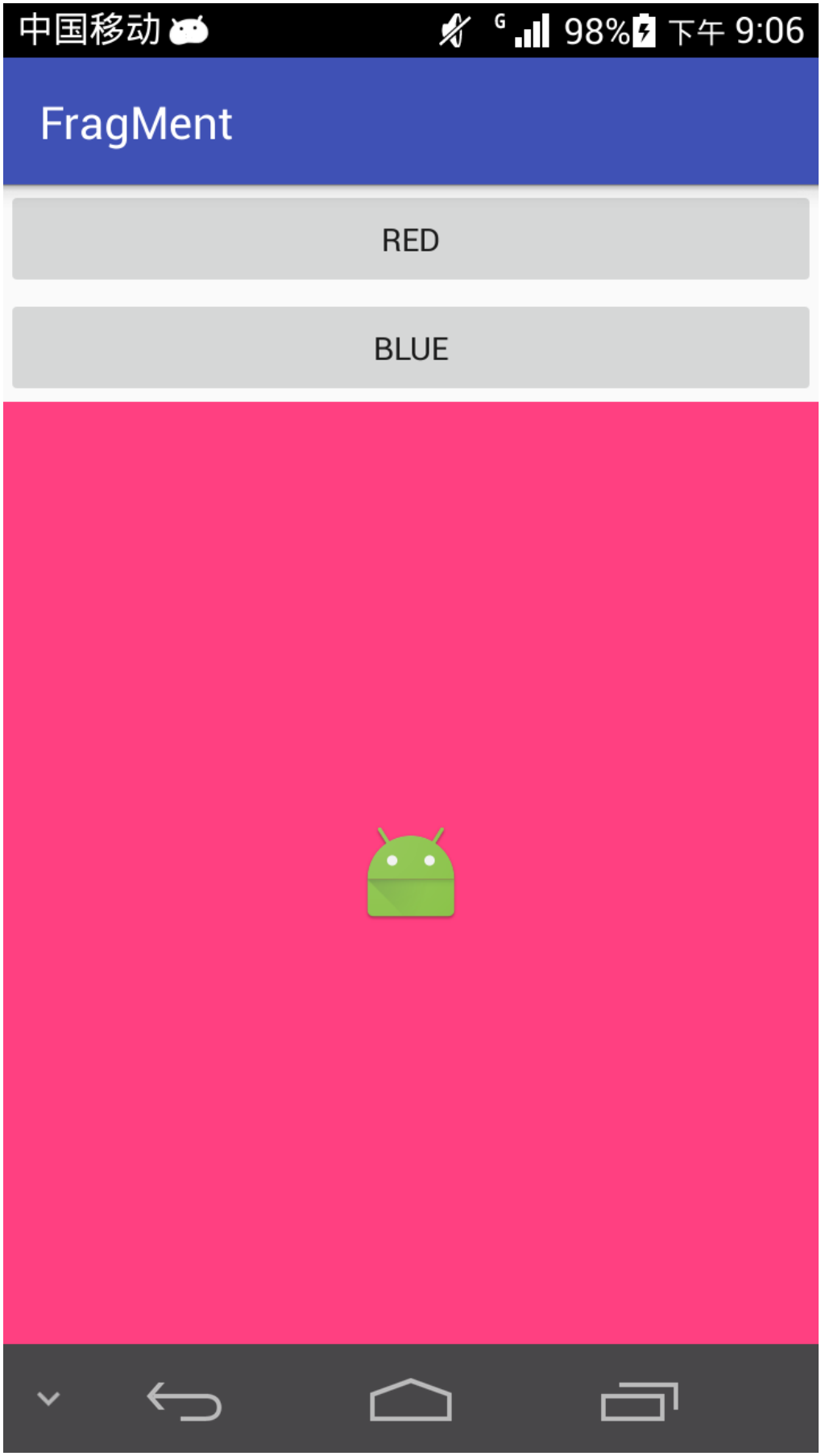
默认显示



点击BLUE按钮时



点击RED按钮时



以上代码我写的比较臃肿但是比较容易看明白：

① 在Acitivity对应的布局中写上一个FramLayout控件，此空间的作用是当作Fragment的容器，Fragment通过FrameLayout显示在Acitivity里，这两个单词容易混淆，请注意

② 准备好你的Fragment，然后再Activity中实例化，v4包的Fragment是通过getSupportFragmentManager()方法新建Fragment管理器对象，此处不讨论app包下的Fragment

③ 然后通过Fragment管理器对象调用beginTransaction()方法，实例化FragmentTransaction对象，有人称之为事务

④ 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

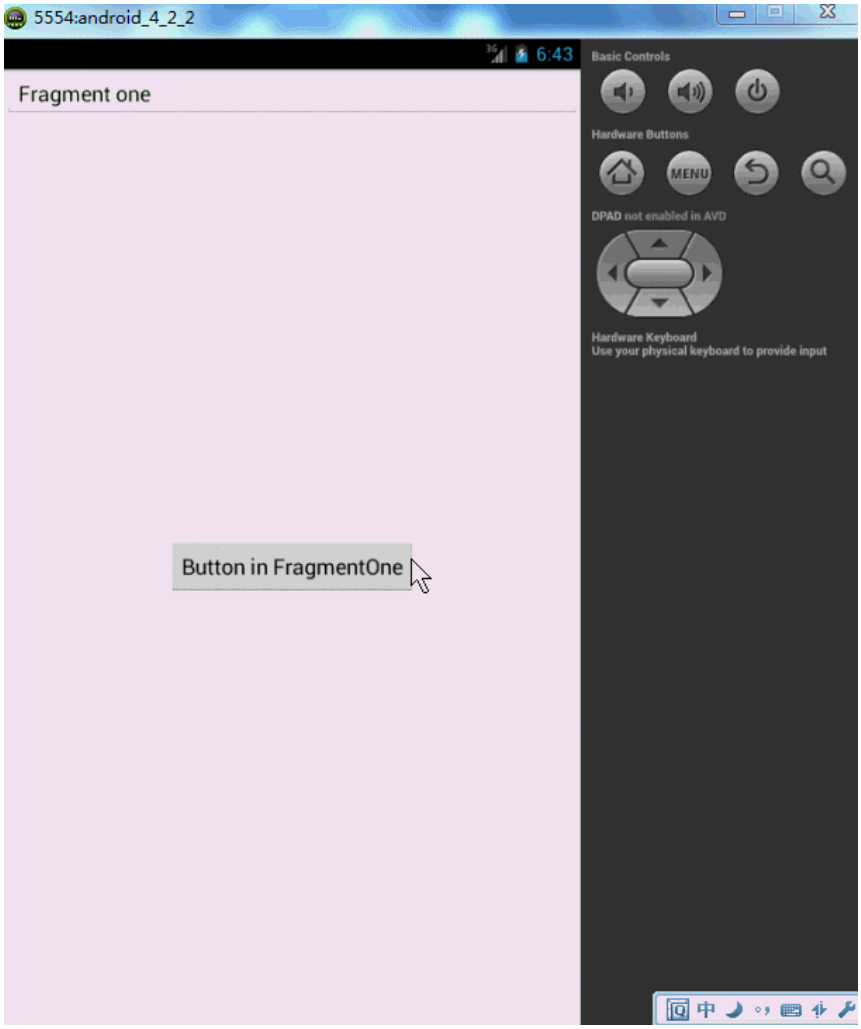
注意：在add/replace/hide/show以后都要commit其效果才会在屏幕上显示出来

4. 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

  那么这句话要怎么理解？

  首先来看一下这个东西：



- 首先显示第一个FragmentOne页面有一个Button in FragmentOne，上面有个输入框显示的是Fragment One
- 然后输入change，点击Button in FragmentOne，然后显示第二个Fragment，里面有一个Button in FragmentTwo，一个输入框显示Fragment Two
- 输入change，点击按钮，显示第三个Fragment，上面有个Button in FragmentThree，点击按钮显示出一个Toast
- 【注意】点击返回键，跳转到前一个FragmentTwo，这个时候可以看到上面的输入框中显示的是Fragment Two change，也就是说保留了我们离开这个Fragment时候他所呈现的状态

- 【注意】再点击返回键，跳转到FragmentOne，但是这个时候我们可以看到上面的输入框中只有Fragment One，并没有change这几个字母

那么原因是什么？

这里先要学习一个方法：FragmentTransaction.addToBackStack(String)【把当前事务的变化情况添加到回退栈】

代码如下：

MainActivity的布局文件

```
<RelativeLayout 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" >

    <FrameLayout
        android:id="@+id/id_content"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
    </FrameLayout>

</RelativeLayout>
```

MainActivity.java文件【这里添加的是app包下的Fragment，推荐v4包下的】

```
public class MainActivity extends Activity {

    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_main);

        FragmentManager fm = getFragmentManager();
        FragmentTransaction tx = fm.beginTransaction();
        tx.add(R.id.id_content, new FragmentOne(),"ONE");
        tx.commit();
    }

}
```

FragmentOne.class文件

```
public class FragmentOne extends Fragment implements OnClickListener {

    private Button mBtn;

    @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;
    }

    @Override
    public void onClick(View v) {
        FragmentTwo fTwo = new FragmentTwo();
        FragmentManager fm = getFragmentManager();
        FragmentTransaction tx = fm.beginTransaction();
        tx.replace(R.id.id_content, fTwo, "TWO");
        tx.addToBackStack(null);
        tx.commit();
    }

}
```

Fragment的点击事件里写的是replace方法，相当于remove和add的合体，并且如果不添加事务到回退栈，前一个Fragment实例会被销毁。这里很明显，我们调用tx.addToBackStack(null)将当前的事务添加到了回退栈，所以FragmentOne实例不会被销毁，但是视图层次依然会被销毁，即会调用onDestoryView和onCreateView。所以【请注意】，当之后我们从FragmentTwo返回到前一个页面的时候，视图层仍旧是重新按照代码绘制，这里仅仅是是实例没有销毁，因此显示的页面中没有change几个字。

FragmentTwo.class文件

```
public class FragmentTwo extends Fragment implements OnClickListener {

    private Button mBtn ;

    @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)  {
        FragmentThree fThree = new FragmentThree();
        FragmentManager fm = getFragmentManager();
        FragmentTransaction tx = fm.beginTransaction();
        tx.hide(this);
        tx.add(R.id.id_content , fThree, "THREE");
        //tx.replace(R.id.id_content, fThree, "THREE");
        tx.addToBackStack(null);
        tx.commit();
    }

}
```

这里点击时，我们没有使用replace，而是先隐藏了当前的Fragment，然后添加了FragmentThree的实例，最后将事务添加到回退栈。这样做的目的是为了给大家提供一种方案：如果不希望视图重绘该怎么做，请再次仔细看效果图，我们在FragmentTwo的EditText填写的内容，用户点击返回键回来时，内容还在。

FragmentThree.class文件

```
public class FragmentThree extends Fragment implements OnClickListener {

    private Button mBtn;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_three, container, false);
        mBtn = (Button) view.findViewById(R.id.id_fragment_three_btn);
        mBtn.setOnClickListener(this);
        return view;
    }

    @Override
    public void onClick(View v) {
        Toast.makeText(getActivity(), " i am a btn in Fragment three",
                Toast.LENGTH_SHORT).show();
    }

}
```

如果你还是不明白请仔细将上面的代码反复敲几遍

5. 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。

6. Fragment与Activity通信的优化

因为要考虑Fragment的重复使用，所以必须降低Fragment与Activity的耦合，而且Fragment更不应该直接操作别的Fragment，毕竟Fragment操作应该由它的管理者Activity来决定。

实现与上一个代码案例一模一样的功能与效果

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();
        }
    }

}
```

可以看到，现在的FragmentOne不和任何Activity耦合，任何Activity都可以使用，并且我们声明了一个接口，来回调其点击事件，想要重写其点击事件的Activity实现此接口即可，可以看到我们在onClick中首先判断了当前绑定的Activity是否实现了该接口，如果实现了则调用。

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();
        }
```

```
    }

}
```

与FragmentOne极其类似，但是我们提供了setListener这样的方法，意味着Activity不仅需要实现该接口，还必须显示调用mFTwo.setfTwoBtnClickListener(this)。

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);

        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。

7. 如何处理运行时配置发生变化

- 在Activity的学习中我们都知道，当屏幕旋转时，是对屏幕上的视图进行了重新绘制。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，用脚趾头都明白...横屏和竖屏显示的不一样肯定是进行了重新绘制视图的操作。所以，不断的旋转就不断绘制，这是一种很耗费内存资源的操作，那么如何来进行优化？

  代码分析：

Fragment的class文件

```
public class FragmentOne extends Fragment {

    private static final String TAG = "FragmentOne";

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        Log.e(TAG, "onCreateView");
        View view = inflater.inflate(R.layout.fragment_one, container, false);
        return view;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        Log.e(TAG, "onCreate");
    }

    @Override
    public void onDestroyView() {
        // TODO Auto-generated method stub
        super.onDestroyView();
        Log.e(TAG, "onDestroyView");
    }

    @Override
    public void onDestroy() {
        // TODO Auto-generated method stub
        super.onDestroy();
        Log.e(TAG, "onDestroy");
    }

}
```

然后你多次翻转屏幕都会打印如下log

```
07-20 08:18:46.651: E/FragmentOne(1633): onCreate
07-20 08:18:46.651: E/FragmentOne(1633): onCreate
07-20 08:18:46.651: E/FragmentOne(1633): onCreate
07-20 08:18:46.681: E/FragmentOne(1633): onCreateView
07-20 08:18:46.831: E/FragmentOne(1633): onCreateView
07-20 08:18:46.891: E/FragmentOne(1633): onCreateView
```

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

```
public class MainActivity extends Activity {

    private static final String TAG = "FragmentOne";
    private FragmentOne mFOne;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_main);

        Log.e(TAG, savedInstanceState+"");
```

```
        if(savedInstanceState == null) {
            mFOne = new FragmentOne();
            FragmentManager fm = getFragmentManager();
            FragmentTransaction tx = fm.beginTransaction();
            tx.add(R.id.id_content, mFOne, "ONE");
            tx.commit();
        }
    }

}
```

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。