第 4 章　手机平板要兼顾，探究碎片

当今是移动设备发展非常迅速的时代，不仅手机已经成为了生活必需品，就连平板电脑也变得越来越普及。平板电脑和手机最大的区别就在于屏幕的大小，一般手机屏幕的大小会在 3 英寸到 5 英寸之间，而一般平板电脑屏幕的大小会在 7 英寸到 10 英寸之间。屏幕大小差距过大有可能会让同样的界面在视觉效果上有较大的差异，比如一些界面在手机上看起来非常美观，但在平板电脑上看起来就可能会有控件被过分拉长、元素之间空隙过大等情况。

作为一名专业的 Android 开发人员，能够同时兼顾到手机和平板的开发是我们必须要做到的事情。Android 自 3.0 版本开始引入了碎片的概念，它可以让界面在平板上更好地展示，下面我们就来一起学习一下。

4.1　碎片是什么

碎片（Fragment）是一种可以嵌入在活动当中的 UI 片段，它能让程序更加合理和充分地利用大屏幕的空间，因而在平板上应用的非常广泛。虽然碎片对你来说应该是个全新的概念，但我相信你学习起来应该毫不费力，因为它和活动实在是太像了，同样都能包含布局，同样都有自己的生命周期。你甚至可以将碎片理解成一个迷你型的活动，虽然这个迷你型的活动有可能和普通的活动是一样大的。

那么究竟要如何使用碎片才能充分地利用平板屏幕的空间呢？想象我们正在开发一个新闻应用，其中一个界面使用 ListView 展示了一组新闻的标题，当点击了其中一个标题，就打开另一个界面显示新闻的详细内容。如果是在手机中设计，我们可以将新闻标题列表放在一个活动中，将新闻的详细内容放在另一个活动中，如图 4.1 所示。

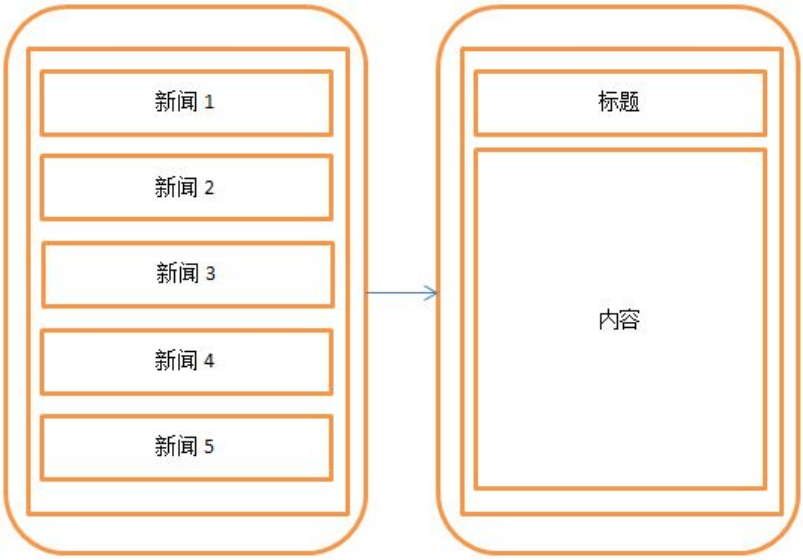

图　4.1

可是如果在平板上也这么设计，那么新闻标题列表将会被拉长至填充满整个平板的屏幕，而新闻的标题一般都不会太长，这样将会导致界面上有大量的空白区域，如图 4.2 所示。

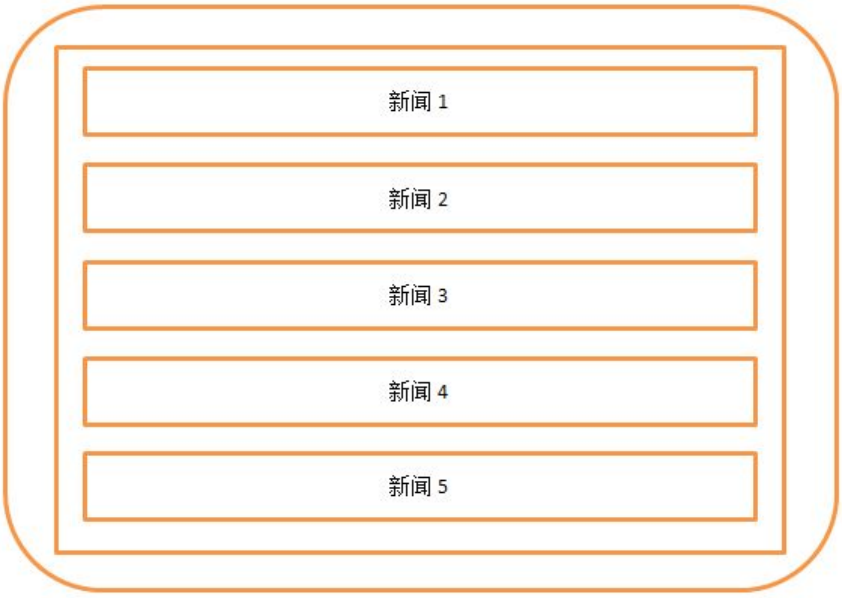

图　4.2

因此，更好的设计方案是将新闻标题列表界面和新闻详细内容界面分别放在两个碎片中，然后在同一个活动里引入这两个碎片，这样就可以将屏幕空间充分地利用起来了，如图 4.3 所示。

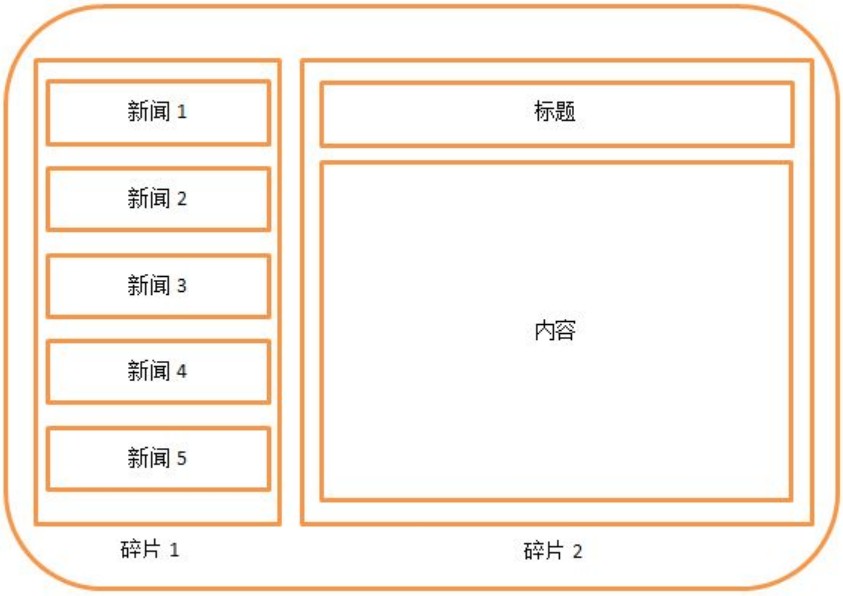

图 4.3

4.2 碎片的使用方式

介绍了这么多抽象的东西，也是时候应该学习一下碎片的具体用法了。你已经知道，碎片通常都是在平板开发中才会使用的，因此我们首先要做的就是新建一个平板电脑的模拟器。由于 4.0 系统的平板模拟器好像存在 bug，这里我就新建一个 4.2 系统的平板模拟器，如图 4.4 所示。

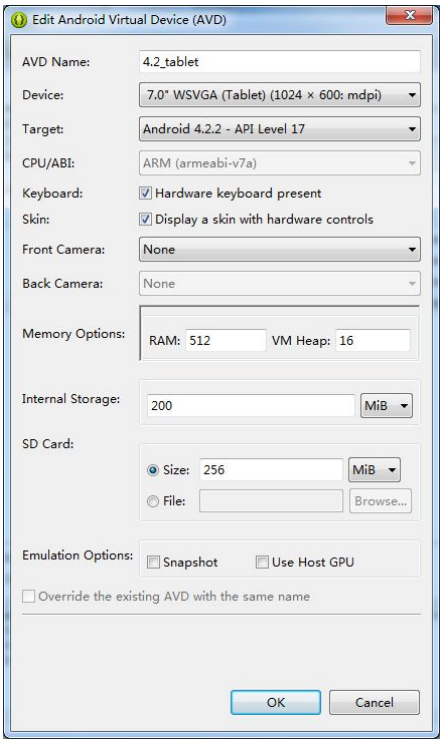

图　4.4

现在启动这个平板模拟器，效果如图 4.5 所示。

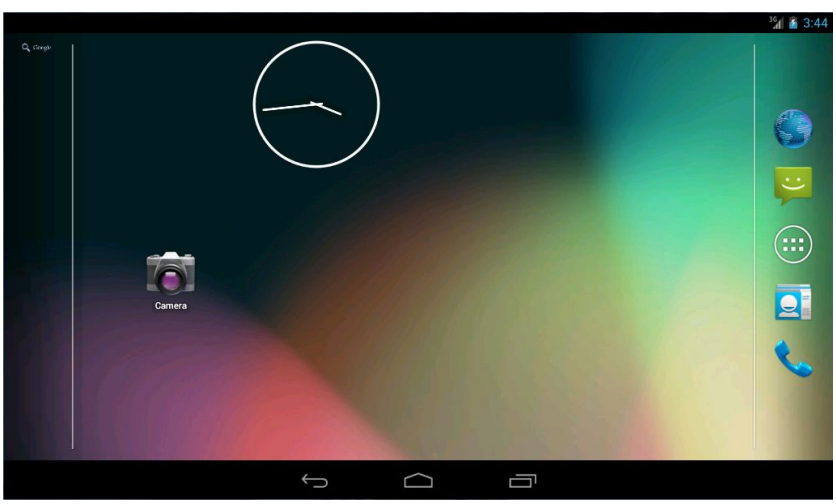

图　4.5

好了，准备工作都完成了，接着新建一个 FragmentTest 项目，然后开始我们的碎片探索之旅吧。

4.2.1 碎片的简单用法

这里我们准备先写一个最简单的碎片示例来练练手，在一个活动当中添加两个碎片，并让这两个碎片平分活动空间。

新建一个左侧碎片布局 left_fragment.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Button"
        />

</LinearLayout>
```

这个布局非常简单，只放置了一个按钮，并让它水平居中显示。然后新建右侧碎片布局 right_fragment.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#00ff00"
    android:orientation="vertical" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textSize="20sp"
        android:text="This is right fragment"
        />

</LinearLayout>
```

可以看到，我们将这个布局的背景色设置成绿色，并放置了一个 TextView 用于显示一段文本。接着新建一个 LeftFragment 类，继承自 Fragment。注意，这里可能会有两个不同包下的 Fragment 供你选择，建议使用 android.app.Fragment，因为我们的程序是面向 Android 4.0 以上系统的，另一个包下的 Fragment 主要是用于兼容低版本的 Android 系统。LeftFragment 的代码如下所示：

```
public class LeftFragment extends Fragment {

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

}
```

这里仅仅是重写了 Fragment 的 onCreateView()方法，然后在这个方法中通过 LayoutInflater 的 inflate()方法将刚才定义的 left_fragment 布局动态加载进来，整个方法简单明了。接着我们用同样的方法再新建一个 RightFragment，代码如下所示：

```
public class RightFragment extends Fragment {

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

}
```

基本上代码都是相同的，相信已经没有必要再做什么解释了。接下来修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <fragment
        android:id="@+id/left_fragment"
```

```
        android:name="com.example.fragmenttest.LeftFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <fragment
        android:id="@+id/right_fragment"
        android:name="com.example.fragmenttest.RightFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

</LinearLayout>
```

可以看到，我们使用了<fragment>标签在布局中添加碎片，其中指定的大多数属性你都是熟悉的，只不过这里还需要通过 android:name 属性来显式指明要添加的碎片类名，注意一定要将类的包名也加上。

这样最简单的碎片示例就已经写好了，现在运行一下程序，效果如图 4.6 所示。

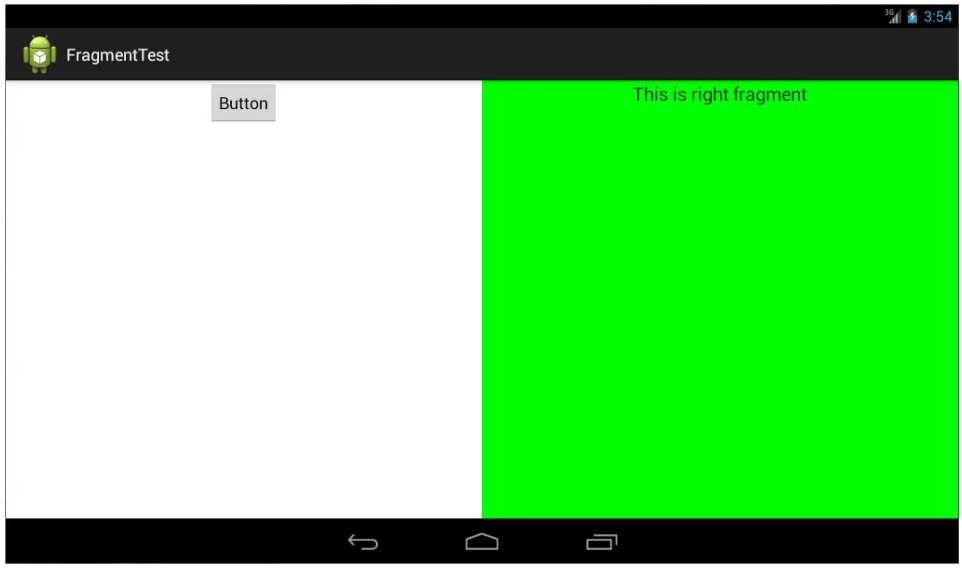

图 4.6

正如我们所期待的一样，两个碎片平分了整个活动的布局。不过这个例子实在是太简单了，在真正的项目中很难有什么实际的作用，因此我们马上来看一看，关于碎片更加高级的使用技巧。

4.2.2　动态添加碎片

在上一节当中，你已经学会了在布局文件中添加碎片的方法，不过碎片真正的强大之处在于，它可以在程序运行时动态地添加到活动当中。根据具体情况来动态地添加碎片，你就可以将程序界面定制得更加多样化。

我们还是在上一节代码的基础上继续完善，新建 another_right_fragment.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ffff00"
    android:orientation="vertical" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textSize="20sp"
        android:text="This is another right fragment"
        />

</LinearLayout>
```

这个布局文件的代码和 right_fragment.xml 中的代码基本相同，只是将背景色改成了黄色，并将显示的文字改了改。然后新建 AnotherRightFragment 作为另一个右侧碎片，代码如下所示：

```
public class AnotherRightFragment extends Fragment {

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

}
```

代码同样非常简单，在 onCreateView()方法中加载了刚刚创建的 another_right_fragment

布局。这样我们就准备好了另一个碎片，接下来看一下如何将它动态地添加到活动当中。修改 activity_main.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        android:id="@+id/left_fragment"
        android:name="com.example.fragmenttest.LeftFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <FrameLayout
        android:id="@+id/right_layout"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" >

        <fragment
            android:id="@+id/right_fragment"
            android:name="com.example.fragmenttest.RightFragment"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />

    </FrameLayout>

</LinearLayout>
```

可以看到，现在将右侧碎片放在了一个 FrameLayout 中，还记得这个布局吗？在上一章中我们学过，这是 Android 中最简单的一种布局，它没有任何的定位方式，所有的控件都会摆放在布局的左上角。由于这里仅需要在布局里放入一个碎片，因此非常适合使用 FrameLayout。

之后我们将在代码中替换 FrameLayout 里的内容，从而实现动态添加碎片的功能。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.button:
            AnotherRightFragment fragment = new AnotherRightFragment();
            FragmentManager fragmentManager = getFragmentManager();
            FragmentTransaction transaction = fragmentManager.
beginTransaction();
            transaction.replace(R.id.right_layout, fragment);
            transaction.commit();
            break;
        default:
            break;
        }
    }

}
```

可以看到，首先我们给左侧碎片中的按钮注册了一个点击事件，然后将动态添加碎片的逻辑都放在了点击事件里进行。结合代码可以看出，动态添加碎片主要分为 5 步。

1. 创建待添加的碎片实例。
2. 获取到 FragmentManager，在活动中可以直接调用 getFragmentManager()方法得到。
3. 开启一个事务，通过调用 beginTransaction()方法开启。
4. 向容器内加入碎片，一般使用 replace()方法实现，需要传入容器的 id 和待添加的碎片实例。
5. 提交事务，调用 commit()方法来完成。

这样就完成了在活动中动态添加碎片的功能，重新运行程序，可以看到和之前相同的界面，然后点击一下按钮，效果如图 4.7 所示。

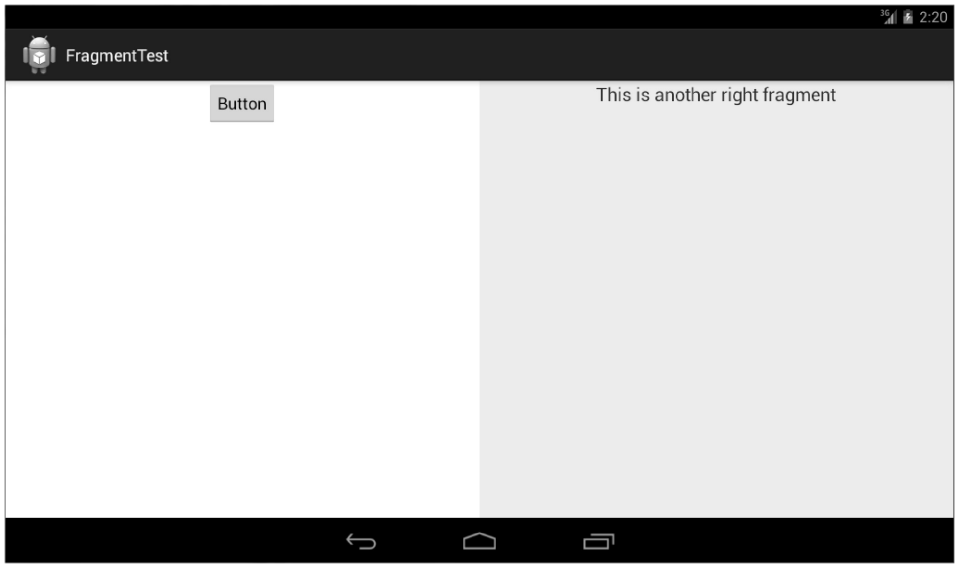

图 4.7

4.2.3 在碎片中模拟返回栈

在上一小节中，我们成功实现了向活动中动态添加碎片的功能，不过你尝试一下就会发现，通过点击按钮添加了一个碎片之后，这时按下 Back 键程序就会直接退出。如果这里我们想模仿类似于返回栈的效果，按下 Back 键可以回到上一个碎片，该如何实现呢？

其实很简单，FragmentTransaction 中提供了一个 addToBackStack()方法，可以用于将一个事务添加到返回栈中，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {
    ……
    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.button:
            AnotherRightFragment fragment = new AnotherRightFragment();
            FragmentManager fragmentManager = getFragmentManager();
            FragmentTransaction transaction = fragmentManager.
beginTransaction();
            transaction.replace(R.id.right_layout, fragment);
            transaction.addToBackStack(null);
            transaction.commit();
```

```
                break;
            default:
                break;
            }
        }
    }
```

这里我们在事务提交之前调用了 FragmentTransaction 的 addToBackStack()方法，它可以接收一个名字用于描述返回栈的状态，一般传入 null 即可。现在重新运行程序，并点击按钮将 AnotherRightFragment 添加到活动中，然后按下 Back 键，你会发现程序并没有退出，而是回到了 RightFragment 界面，再次按下 Back 键程序才会退出。

4.2.4　碎片和活动之间进行通信

虽然碎片都是嵌入在活动中显示的，可是实际上它们的关系并没有那么亲密。你可以看出，碎片和活动都是各自存在于一个独立的类当中的，它们之间并没有那么明显的方式来直接进行通信。如果想要在活动中调用碎片里的方法，或者在碎片中调用活动里的方法，应该如何实现呢？

为了方便碎片和活动之间进行通信，FragmentManager 提供了一个类似于 findViewById()的方法，专门用于从布局文件中获取碎片的实例，代码如下所示：

```
RightFragment rightFragment = (RightFragment) getFragmentManager()
        .findFragmentById(R.id.right_fragment);
```

调用 FragmentManager 的 findFragmentById()方法，可以在活动中得到相应碎片的实例，然后就能轻松地调用碎片里的方法了。

掌握了如何在活动中调用碎片里的方法，那在碎片中又该怎样调用活动里的方法呢？其实这就更简单了，在每个碎片中都可以通过调用 getActivity()方法来得到和当前碎片相关联的活动实例，代码如下所示：

```
MainActivity activity = (MainActivity) getActivity();
```

有了活动实例之后，在碎片中调用活动里的方法就变得轻而易举了。另外当碎片中需要使用 Context 对象时，也可以使用 getActivity()方法，因为获取到的活动本身就是一个 Context 对象了。

这时不知道你心中会不会产生一个疑问，既然碎片和活动之间的通信问题已经解决了，那么碎片和碎片之间可不可以进行通信呢？

说实在的，这个问题并没有看上去的复杂，它的基本思路非常简单，首先在一个碎片中可以得到与它相关联的活动，然后再通过这个活动去获取另外一个碎片的实例，这样也就实现了不同碎片之间的通信功能，因此这里我们的答案是肯定的。

经验值：+3000　升级！（由菜鸟升级至鸟）　目前经验值：11405

级别：鸟

赢得宝物：战胜平板兽。拾取平板兽掉落的宝物，九成新平板兽皮 Android 战袍一套、强力体力恢复剂 2 瓶、平板兽鄙视丸 1 颗、全自动新款卡宾枪一支（配有可拆卸代码银弹接口）、全新代码银弹弹夹一个（含 70 枚代码银弹）。平板兽是一种除了在神界，其他地方不仅不可能存在，而且连想都不敢想的小型食肉类动物。平板兽的名称得自于它奇怪的体型，它看起来完全就是一块平板，大约一米见方，3 公分厚，长有四条极粗但弹跳力极好的小短腿，平板上下各长有两只眼睛，除了它的体型，平板兽另一让人极其不可思议的地方是它的密度奇高，一米见方的大小，却可重达 5 吨。平板兽的攻击方式是奋力跳到对方身上，瞬间把对方压垮，它的主要捕食对象是立体牛，一种看起来特别立体的牛，当然，所有的牛看起来都是立体的，但立体牛看起来比立体更加立体，所以叫立体牛。这里我先插一句，可能有看官会奇怪，看你一路上收集了不少恢复剂和怒吼丸了，怎么从来没见你服用过，其实我每次战役都有服用，只是没有说而已，如果不吃药，我早挂了，怎么可能搞定这么多神界的英雄。接着我再来介绍一下鄙视丸，鄙视丸是用平板兽每年自行脱落的老眼球做的，因为平板兽有两只眼睛总是朝着天上，一天到晚被太阳晒，所以坏得特别快，每年都要换一次眼，新眼发育好后，老眼会自动脱落，被新眼所替代。而脱落下来的老眼，被人们收集后，配以多种草药后进行熬制，草药主要由高傲草和牛 B 花组成，以加强药效。鄙视丸名字不咋地，但效果惊人，可以在短时间内增加使用者的智商和情商，当你面对强大的对手或难题时，发现自己搞不定了，此时赶紧服下一颗鄙视丸，然后死死盯住对方，很快你就会发现你面对的其实是个白痴。此时，对手如果想战胜你，唯一的办法就是服用更多的鄙视丸。

4.3　碎片的生命周期

和活动一样，碎片也有自己的生命周期，并且它和活动的生命周期实在是太像了，我相信你很快就能学会，下面我们马上就来看一下。

4.3.1　碎片的状态和回调

还记得每个活动在其生命周期内可能会有哪几种状态吗？没错，一共有运行状态、暂停状态、停止状态和销毁状态这四种。类似地，每个碎片在其生命周期内也可能会经历这几种状态，只不过在一些细小的地方会有部分区别。

1. 运行状态

当一个碎片是可见的，并且它所关联的活动正处于运行状态时，该碎片也处于运行状态。

2. 暂停状态

当一个活动进入暂停状态时（由于另一个未占满屏幕的活动被添加到了栈顶），与它相关联的可见碎片就会进入到暂停状态。

3. 停止状态

当一个活动进入停止状态时，与它相关联的碎片就会进入到停止状态。或者通过调用 FragmentTransaction 的 remove()、replace()方法将碎片从活动中移除，但有在事务提交之前调用 addToBackStack()方法，这时的碎片也会进入到停止状态。总的来说，进入停止状态的碎片对用户来说是完全不可见的，有可能会被系统回收。

4. 销毁状态

碎片总是依附于活动而存在的，因此当活动被销毁时，与它相关联的碎片就会进入到销毁状态。或者通过调用 FragmentTransaction 的 remove()、replace()方法将碎片从活动中移除，但在事务提交之前并没有调用 addToBackStack()方法，这时的碎片也会进入到销毁状态。

结合之前的活动状态，相信你理解起来应该毫不费力吧。同样地，Fragment 类中也提供了一系列的回调方法，以覆盖碎片生命周期的每个环节。其中，活动中有的回调方法，碎片中几乎都有，不过碎片还提供了一些附加的回调方法，那我们就重点来看下这几个回调。

1. onAttach()

当碎片和活动建立关联的时候调用。

2. onCreateView()

为碎片创建视图（加载布局）时调用。

3. onActivityCreated()

确保与碎片相关联的活动一定已经创建完毕的时候调用。

4. onDestroyView()

当与碎片关联的视图被移除的时候调用。

5. onDetach()

当碎片和活动解除关联的时候调用。

碎片完整的生命周期示意图可参考图 4.8，图片源自 Android 官网。

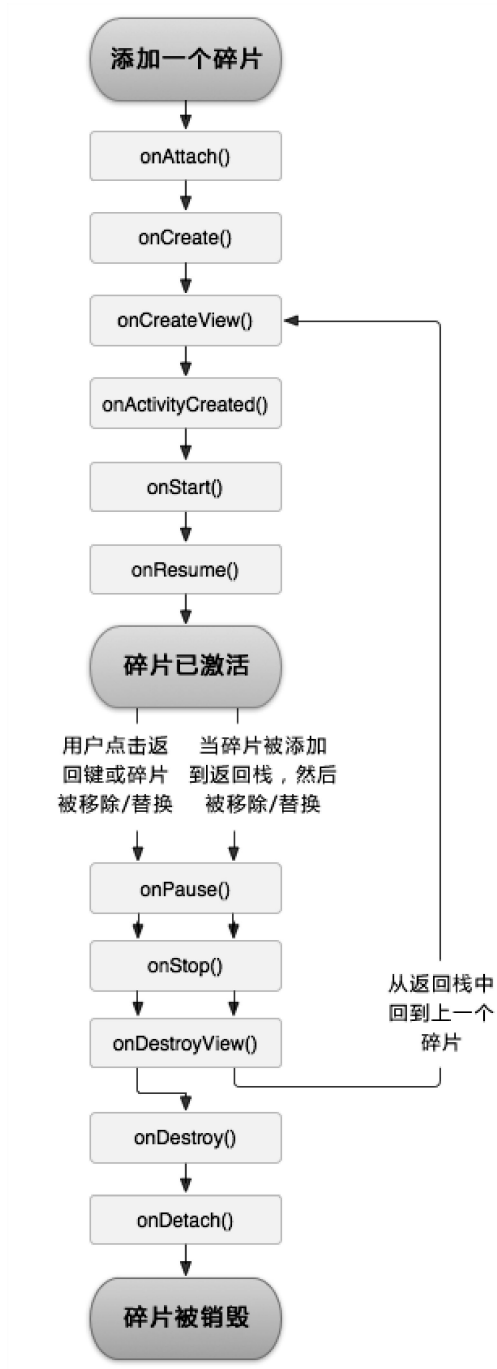
添加一个碎片
onAttach()
onCreate()
onCreateView()
onActivityCreated()
onStart()
onResume()
碎片已激活
用户点击返回键或碎片被移除/替换
当碎片被添加到返回栈，然后被移除/替换
onPause()
onStop()
onDestroyView()
从返回栈中回到上一个碎片
onDestroy()
onDetach()
碎片被销毁

图 4.8

4.3.2　体验碎片的生命周期

为了让你能够更加直观地体验碎片的生命周期，我们还是通过一个例子来实践一下。例子很简单，仍然是在 FragmentTest 项目的基础上改动的。

修改 RightFragment 中的代码，如下所示：

```
public class RightFragment extends Fragment {

    public static final String TAG = "RightFragment";

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        Log.d(TAG, "onAttach");
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate");
    }

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

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        Log.d(TAG, "onActivityCreated");
    }

    @Override
    public void onStart() {
        super.onStart();
        Log.d(TAG, "onStart");
    }
```

```
    @Override
    public void onResume() {
        super.onResume();
        Log.d(TAG, "onResume");
    }

    @Override
    public void onPause() {
        super.onPause();
        Log.d(TAG, "onPause");
    }

    @Override
    public void onStop() {
        super.onStop();
        Log.d(TAG, "onStop");
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        Log.d(TAG, "onDestroyView");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy");
    }

    @Override
    public void onDetach() {
        super.onDetach();
        Log.d(TAG, "onDetach");
    }

}
```

我们在 RightFragment 中的每一个回调方法里都加入了打印日志的代码，然后重新运行程序，这时观察 LogCat 中的打印信息，如图 4.9 所示。

Tag	Text
RightFragment	onAttach
RightFragment	onCreate
RightFragment	onCreateView
RightFragment	onActivityCreated
RightFragment	onStart
RightFragment	onResume

图　4.9

可以看到，当 RightFragment 第一次被加载到屏幕上时，会依次执行 onAttach()、onCreate()、onCreateView()、onActivityCreated()、onStart()和 onResume()方法。然后点击 LeftFragment 中的按钮，此时打印信息如图 4.10 所示。

Tag	Text
RightFragment	onPause
RightFragment	onStop
RightFragment	onDestroyView

图　4.10

由于 AnotherRightFragment 替换了 RightFragment，此时的 RightFragment 进入了停止状态，因此 onPause()、onStop()和 onDestroyView()方法会得到执行。当然如果在替换的时候没有调用 addToBackStack()方法，此时的 RightFragment 就会进入销毁状态，onDestroy()和 onDetach()方法就会得到执行。

接着按下 Back 键，RightFragment 会重新回到屏幕，打印信息如图 4.11 所示。

Tag	Text
RightFragment	onActivityCreated
RightFragment	onStart
RightFragment	onResume

图　4.11

由于 RightFragment 重新回到了运行状态，因此 onActivityCreated()、onStart()和 onResume()方法会得到执行。注意此时 onCreate()和 onCreateView()方法并不会执行，因为我们借助了 addToBackStack()方法使得 RightFragment 和它的视图并没有销毁。

再次按下 Back 键退出程序，打印信息如图 4.12 所示。

Tag	Text
RightFragment	onPause
RightFragment	onStop
RightFragment	onDestroyView
RightFragment	onDestroy
RightFragment	onDetach

图 4.12

依次会执行 onPause()、onStop()、onDestroyView()、onDestroy()和 onDetach()方法，最终将活动和碎片一起销毁。这样碎片完整的生命周期你也体验了一遍，是不是理解得更加深刻了？

另外值得一提的是，在碎片中你也是可以通过 onSaveInstanceState()方法来保存数据的，因为进入停止状态的碎片有可能在系统内存不足的时候被回收。保存下来的数据在 onCreate()、onCreateView()和 onActivityCreated()这三个方法中你都可以重新得到，它们都含有一个 Bundle 类型的 savedInstanceState 参数。具体的代码我就不在这里给出了，如果你忘记了该如何编写可以参考 2.4.5 小节。

经验值：+2000　　目前经验值：13405

级别：鸟

赢得宝物：战胜碎片周期兽。拾取碎片周期兽掉落的宝物，一只巨大的袜子。碎片周期兽和活动周期兽同属周期科周期属，按说智力应该差不多，但事实上差别巨大，活动周期兽完全是一介文盲武夫，而碎片周期兽则识文断字、饱读诗书，平日里喜欢演习书法，正常的碎片周期兽总是用右后蹄来写字，而这只大袜子也正是穿在这只蹄上。

4.4 动态加载布局的技巧

虽然动态添加碎片的功能很强大，可以解决很多实际开发中的问题，但是它毕竟只是在一个布局文件中进行一些添加和替换操作。如果程序能够根据设备的分辨率或屏幕大小在运行时来决定加载哪个布局，那我们可发挥的空间就更多了。因此本节我们就来探讨一下 Android 中动态加载布局的技巧。

4.4.1 使用限定符

如果你经常使用平板电脑，应该会发现很多的平板应用现在都采用的是双页模式（程序会在左侧的面板上显示一个包含子项的列表，在右侧的面板上显示内容），因为平板电脑的屏幕足够大，完全可以同时显示下两页的内容，但手机的屏幕一次就只能显示一页的内容，

因此两个页面需要分开显示。

那么怎样才能在运行时判断程序应该是使用双页模式还是单页模式呢？这就需要借助限定符（Qualifiers）来实现了。我们通过一个例子来学习一下它的用法，修改 FragmentTest 项目中的 activity_main.xml 文件，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <fragment
        android:id="@+id/left_fragment"
        android:name="com.example.fragmenttest.LeftFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

这里将多余的代码都删掉，只留下一个左侧碎片，并让它充满整个父布局。接着在 res 目录下新建 layout-large 文件夹，在这个文件夹下新建一个布局，也叫做 activity_main.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <fragment
        android:id="@+id/left_fragment"
        android:name="com.example.fragmenttest.LeftFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <fragment
        android:id="@+id/right_fragment"
        android:name="com.example.fragmenttest.RightFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3" />

</LinearLayout>
```

可以看到，layout/activity_main 布局只包含了一个碎片，即单页模式，而 layout-large/activity_main 布局包含了两个碎片，即双页模式。其中 large 就是一个限定符，那些屏幕被认为是 large 的设备就会自动加载 layout-large 文件夹下的布局，而小屏幕的设备则还是会加载 layout 文件夹下的布局。

然后将 MainActivity 中按钮点击事件的代码屏蔽掉，并在平板模拟器上重新运行程序，效果如图 4.13 所示。

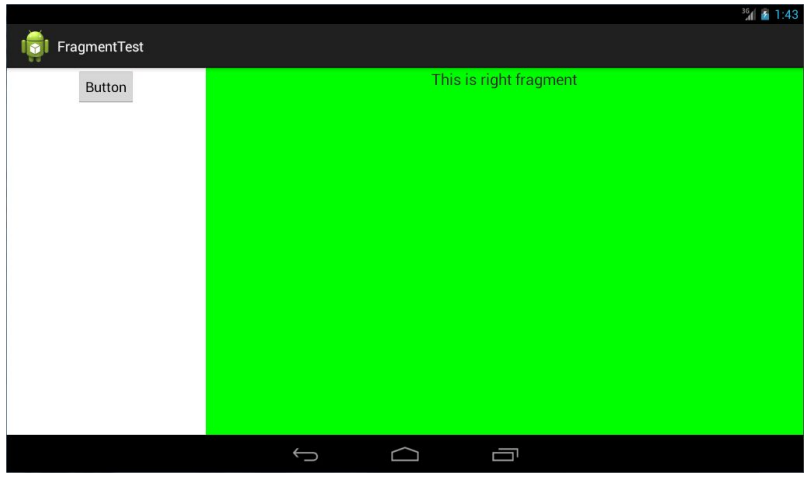

图 4.13

再启动一个手机模拟器，并在这个模拟器上重新运行程序，效果如图 4.14 所示。

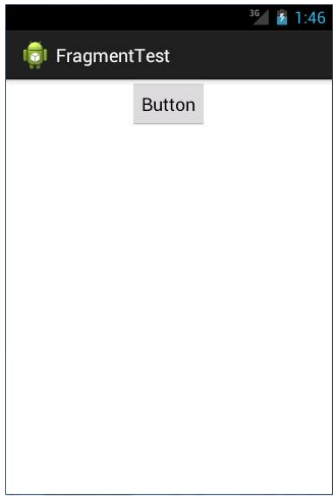

图 4.14

这样我们就实现了在程序运行时动态加载布局的功能。

Android 中一些常见的限定符可以参考下表。

屏幕特征	限定符	描述
大小	small	提供给小屏幕设备的资源
	normal	提供给中等屏幕设备的资源
	large	提供给大屏幕设备的资源
	xlarge	提供给超大屏幕设备的资源
分辨率	ldpi	提供给低分辨率设备的资源（120dpi 以下）
	mdpi	提供给中等分辨率设备的资源（120dpi 到 160dpi）
	hdpi	提供给高分辨率设备的资源（160dpi 到 240dpi）
	xhdpi	提供给超高分辨率设备的资源（240dpi 到 320dpi）
方向	land	提供给横屏设备的资源
	port	提供给竖屏设备的资源

4.4.2　使用最小宽度限定符

在上一小节中我们使用 large 限定符成功解决了单页双页的判断问题，不过很快又有一个新的问题出现了，large 到底是指多大呢？有的时候我们希望可以更加灵活地为不同设备加载布局，不管它们是不是被系统认定为“large”，这时就可以使用最小宽度限定符（Smallest-width Qualifier）了。

最小宽度限定符允许我们对屏幕的宽度指定一个最小指（以 dp 为单位），然后以这个最小值为临界点，屏幕宽度大于这个值的设备就加载一个布局，屏幕宽度小于这个值的设备就加载另一个布局。

在 res 目录下新建 layout-sw600dp 文件夹，然后在这个文件夹下新建 activity_main.xml 布局，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <fragment
        android:id="@+id/left_fragment"
        android:name="com.example.fragmenttest.LeftFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
```

```
        android:layout_weight="1" />

    <fragment
        android:id="@+id/right_fragment"
        android:name="com.example.fragmenttest.RightFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3" />

</LinearLayout>
```

这就意味着，当程序运行在屏幕宽度大于 600dp 的设备上时，会加载 layout-sw600dp/activity_main 布局，当程序运行在屏幕宽度小于 600dp 的设备上时，则仍然加载默认的 layout/activity_main 布局。

需要注意一点，最小宽度限定符是在 Android 3.2 版本引入的，由于这里我们最低兼容的系统版本是 4.0，所以可以放心地使用它。

4.5 碎片的最佳实践——一个简易版的新闻应用

现在你已经将关于碎片的重要知识点都掌握得差不多了，不过在灵活运用方面可能还有些欠缺，因此又该进入我们本章的最佳实践环节了。

前面有提到过，碎片很多时候都是在平板开发当中使用的，主要是为了解决屏幕空间不能充分利用的问题。那是不是就表明，我们开发的程序都需要提供一个手机版和一个 Pad 版呢？确实有不少公司都是这么做的，但是这样会浪费很多的人力物力。因为维护两个版本的代码成本很高，每当增加什么新功能时，需要在两份代码里各写一遍，每当发现一个 bug 时，需要在两份代码里各修改一次。因此今天我们最佳实践的内容就是，教你如何编写同时兼容手机和平板的应用程序。

还记得在本章开始的时候提到过的一个新闻应用吗？现在我们就将运用本章中所学的知识来编写一个简易版的新闻应用，并且要求它是可以同时兼容手机和平板的。新建好一个 FragmentBestPractice 项目，然后开始动手吧！

第一步我们要先准备好一个新闻的实体类，新建类 News，代码如下所示：

```
public class News {

    private String title;

    private String content;
```

```
    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

}
```

News 类的代码还是比较简单的，title 字段表示新闻标题，content 字段表示新闻内容。接着新建一个 news_item.xml 布局，用于作为新闻列表中子项的布局：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/news_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:singleLine="true"
        android:ellipsize="end"
        android:textSize="18sp"
        android:paddingLeft="10dp"
        android:paddingRight="10dp"
        android:paddingTop="15dp"
        android:paddingBottom="15dp"
        />

</LinearLayout>
```

这段代码也非常简单，只是在 LinearLayout 中放入了一个 TextView 用于显示新闻的标题。仔细观察 TextView，你会发现其中有几个属性是我们之前没有学过的。android:padding 表示给控件的周围加上补白，这样不至于让文本内容会紧靠在边缘上。android:singleLine 设置为 true 表示让这个 TextView 只能单行显示。android:ellipsize 用于设定当文本内容超出控件宽度时，文本的缩略方式，这里指定成 end 表示在尾部进行缩略。

接下来需要创建新闻列表的适配器，让这个适配器继承自 ArrayAdapter，并将泛型指定为 News 类。新建类 NewsAdapter，代码如下所示：

```
public class NewsAdapter extends ArrayAdapter<News> {

    private int resourceId;

    public NewsAdapter(Context context, int textViewResourceId, List<News>
objects) {
        super(context, textViewResourceId, objects);
        resourceId = textViewResourceId;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        News news = getItem(position);
        View view;
        if (convertView == null) {
            view = LayoutInflater.from(getContext()).inflate(resourceId, null);
        } else {
            view = convertView;
        }
        TextView newsTitleText = (TextView) view.findViewById(R.id.news_title);
        newsTitleText.setText(news.getTitle());
        return view;
    }

}
```

可以看到，在 getView()方法中，我们获取到了相应位置上的 News 类，并让新闻的标题在列表中进行显示。

这样基本就把新闻列表部分的代码编写完了，接下来我们看一下如何编写新闻内容部分的代码。新建布局文件 news_content_frag.xml，代码如下所示：

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

    <LinearLayout
        android:id="@+id/visibility_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:visibility="invisible" >

        <TextView
            android:id="@+id/news_title"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:padding="10dp"
            android:textSize="20sp" />

        <ImageView
            android:layout_width="match_parent"
            android:layout_height="1dp"
            android:scaleType="fitXY"
            android:src="@drawable/spilt_line" />

        <TextView
            android:id="@+id/news_content"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:padding="15dp"
            android:textSize="18sp" />
    </LinearLayout>

    <ImageView
        android:layout_width="1dp"
        android:layout_height="match_parent"
        android:layout_alignParentLeft="true"
        android:scaleType="fitXY"
        android:src="@drawable/spilt_line_vertical" />

</RelativeLayout>
```

新闻内容的布局主要可以分为两个部分，头部显示完整的新闻标题，正文部分显示新闻内容，中间使用一条细线分隔开。这里的细线是利用 ImageView 显示了一张很窄的图片来实现的，将 ImageView 的 android:scaleType 属性设置为 fitXY，表示让这张图片填充满整个控件的大小。

然后再新建一个 NewsContentFragment 类，继承自 Fragment，代码如下所示：

```
public class NewsContentFragment extends Fragment {

    private View view;

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

    public void refresh(String newsTitle, String newsContent) {
        View visibilityLayout = view.findViewById(R.id.visibility_layout);
        visibilityLayout.setVisibility(View.VISIBLE);
        TextView newsTitleText = (TextView) view.findViewById (R.id.news_title);
        TextView newsContentText = (TextView) view.findViewById(R.id.news_
content);
        newsTitleText.setText(newsTitle); // 刷新新闻的标题
        newsContentText.setText(newsContent); // 刷新新闻的内容
    }

}
```

首先在 onCreateView()方法里加载了我们刚刚创建的 news_content_frag 布局，这个没什么好解释的。接下来又提供了一个 refresh()方法，这个方法就是用于将新闻的标题和内容显示在界面上的。可以看到，这里通过 findViewById()方法分别获取到新闻的标题和内容控件，然后将方法传递进来的参数设置进去。

接着要创建一个在活动中使用的新闻内容布局，新建 news_content.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
```

```
    <fragment
        android:id="@+id/news_content_fragment"
        android:name="com.example.fragmentbestpractice.NewsContentFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        />

</LinearLayout>
```

这里我们充分发挥了代码的复用性，直接在布局中引入了 NewsContentFragment，这样也就相当于把 news_content_frag 布局的内容自动加了进来。

然后新建 NewsContentActivity，作为显示新闻内容的活动，代码如下所示：

```
public class NewsContentActivity extends Activity {

    public static void actionStart(Context context, String newsTitle,
            String newsContent) {
        Intent intent = new Intent(context, NewsContentActivity.class);
        intent.putExtra("news_title", newsTitle);
        intent.putExtra("news_content", newsContent);
        context.startActivity(intent);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.news_content);
        String newsTitle = getIntent().getStringExtra("news_title");
// 获取传入的新闻标题
        String newsContent = getIntent().getStringExtra("news_content");
// 获取传入的新闻内容
        NewsContentFragment newsContentFragment = (NewsContentFragment)
getFragmentManager()
                .findFragmentById(R.id.news_content_fragment);
        newsContentFragment.refresh(newsTitle, newsContent);
// 刷新NewsContentFragment界面
    }

}
```

可以看到，在 onCreate()方法中我们通过 Intent 获取到了传入的新闻标题和新闻内容，然后调用 FragmentManager 的 findFragmentById()方法得到了 NewsContentFragment 的实例，接着调用它的 refresh()方法，并将新闻的标题和内容传入，就可以把这些数据显示出来了。注意这里我们还提供了一个 actionStart()方法，还记得它的作用吗？如果忘记的话就再去阅读一遍 2.6.3 节吧。

接下来还需要再创建一个用于显示新闻列表的布局，新建 news_title_frag.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <ListView
        android:id="@+id/news_title_list_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
    </ListView>

</LinearLayout>
```

这个布局的代码就非常简单了，里面只有一个 ListView。不过想必你已经猜到了，这个布局并不是给活动使用的，而是给碎片使用的，因此我们还需要创建一个碎片来加载这个布局。新建一个 NewsTitleFragment 类，继承自 Fragment，代码如下所示：

```
public class NewsTitleFragment extends Fragment implements
OnItemClickListener {

    private ListView newsTitleListView;

    private List<News> newsList;

    private NewsAdapter adapter;

    private boolean isTwoPane;

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        newsList = getNews(); // 初始化新闻数据
```

```
        adapter = new NewsAdapter(activity, R.layout.news_item, newsList);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.news_title_frag, container, false);
        newsTitleListView = (ListView) view.findViewById(R.id.news_title_
list_view);
        newsTitleListView.setAdapter(adapter);
        newsTitleListView.setOnItemClickListener(this);
        return view;
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        if (getActivity().findViewById(R.id.news_content_layout) != null) {
            isTwoPane = true; // 可以找到news_content_layout布局时，为双页模式
        } else {
            isTwoPane = false; // 找不到news_content_layout布局时，为单页模式
        }
    }

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position,
            long id) {
        News news = newsList.get(position);
        if (isTwoPane) {
            // 如果是双页模式，则刷新NewsContentFragment中的内容
            NewsContentFragment newsContentFragment = (NewsContentFragment)
                    getFragmentManager().findFragmentById(R.id.news_
content_fragment);
            newsContentFragment.refresh(news.getTitle(), news.getContent());
        } else {
            // 如果是单页模式，则直接启动NewsContentActivity
            NewsContentActivity.actionStart(getActivity(), news.getTitle(),
news.getContent());
        }
    }
```

```
    private List<News> getNews() {
        List<News> newsList = new ArrayList<News>();
        News news1 = new News();
        news1.setTitle("Succeed in College as a Learning Disabled Student");
        news1.setContent("College freshmen will soon learn to live with a
        roommate, adjust to a new social scene and survive less-than-stellar
        dining hall food. Students with learning disabilities will face these
        transitions while also grappling with a few more hurdles.");
        newsList.add(news1);
        News news2 = new News();
        news2.setTitle("Google Android exec poached by China's Xiaomi");
        news2.setContent("China's Xiaomi has poached a key Google executive
        involved in the tech giant's Android phones, in a move seen as a coup
        for the rapidly growing Chinese smartphone maker.");
        newsList.add(news2);
        return newsList;
    }

}
```

这个类的代码有点长，我来重点解释一下。根据碎片的生命周期，我们知道，onAttach()方法会首先执行，因此在这里做了一些数据初始化的操作，比如调用 getNews()方法获取几条模拟的新闻数据，以及完成 NewsAdapter 的创建。然后在 onCreateView()方法中加载了 news_title_frag 布局，并给新闻列表的 ListView 注册了点击事件。接下来在 onActivityCreated()方法中，我们通过是否能够找到一个 id 为 news_content_layout 的 View 来判断当前是双页模式还是单页模式，这个 id 为 news_content_layout 的 View 只在双页模式中才会出现，在稍后的布局里你将会看到。然后在 ListView 的点击事件里我们就可以判断，如果当前是单页模式，就启动一个新的活动去显示新闻内容，如果当前是双页模式，就更新新闻内容碎片里的数据。

剩下工作就非常简单了，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <fragment
        android:id="@+id/news_title_fragment"
        android:name="com.example.fragmentbestpractice.NewsTitleFragment"
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        />

</LinearLayout>
```

上述代码表示，在单页模式下，只会加载一个新闻标题的碎片。然后新建 layout-sw600dp 文件夹，在这个文件夹下再新建一个 activity_main.xml 文件，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <fragment
        android:id="@+id/news_title_fragment"
        android:name="com.example.fragmentbestpractice.NewsTitleFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <FrameLayout
        android:id="@+id/news_content_layout"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3" >

        <fragment
            android:id="@+id/news_content_fragment"
            android:name="com.example.fragmentbestpractice.
NewsContentFragment"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
    </FrameLayout>

</LinearLayout>
```

可以看出，在双页模式下我们同时引入了两个碎片，并将新闻内容的碎片放在了一个 FrameLayout 布局下，而这个布局的 id 正是 news_content_layout。因此，能够找到这个 id 的时候就是双页模式，否则就是单面模式。

最后再将 MainActivity 稍作修改，把标题栏去除掉，代码如下所示：

```
public class MainActivity extends Activity {

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

}
```

这样我们所有的编写工作就已经完成了，赶快来运行一下吧！首先在手机模拟器上运行，效果如图 4.15 所示。

图 4.15

可以看到两条新闻的标题，并且超出屏幕部分的文字是在尾部使用省略号代替的。

然后点击第二条新闻，会启动一个新的活动来显示新闻的内容，效果如图 4.16 所示。

图　4.16

接下来将程序在平板模拟器上运行，同样点击第二条新闻，效果如图 4.17 所示。

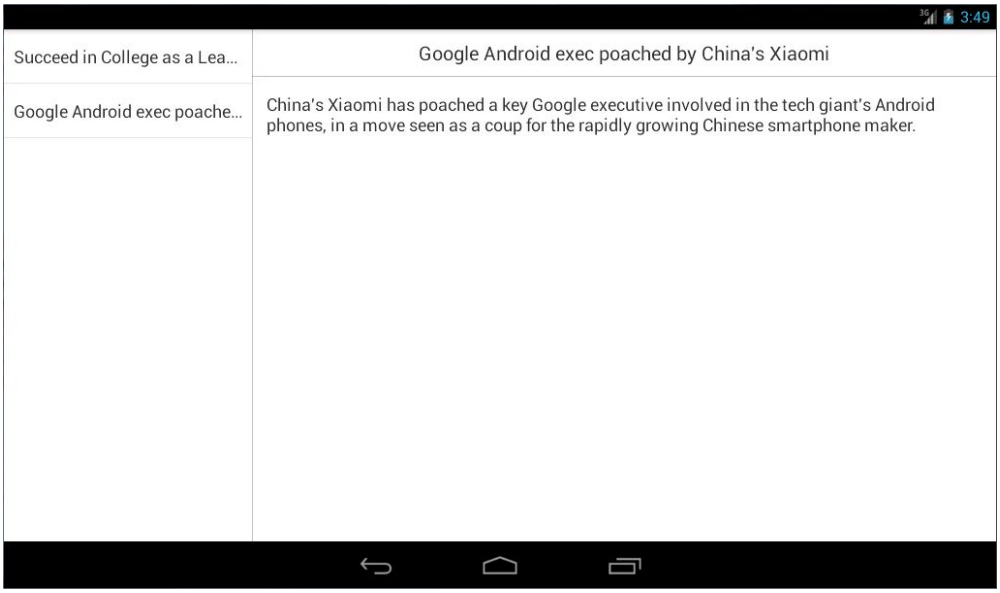

图　4.17

怎么样？同样的一份代码，在手机和平板上运行却分别是两种完全不同的效果，说明我们程序的兼容性已经写得相当不错了。通过这个例子，我相信你对碎片的理解一定又加深了很多，现在就让我们一起来总结一下吧。

4.6 小结与点评

你应该可以感觉到，上一节中我们开发的新闻应用，代码复杂度还是有点高的，比起只需要兼容一个终端的应用，我们要考虑的东西多了很多。不过开发的过程中多付出一些，在以后的代码维护中就可以轻松很多。因此，有时候提前的付出还是很值得的。

我们再来回顾一下本章所学的内容吧，首先你了解了碎片的基本概念以及使用场景，接着通过几个实例掌握了碎片的常见用法，随后又学习了碎片生命周期的相关内容以及动态加载布局的技巧，最后在本章的最佳实践部分将前面所学的内容综合运用了一遍，相信你已经将碎片相关知识点都牢记在心，并可以较为熟练地应用了。

本章其实是有一个里程碑式的纪念意义的，因为到这里为止，我们已经将 Android UI 相关的重要知识点全部讲完了。后面将不会再系统性地介绍 UI 方面的知识，而是将结合前面所学的 UI 知识来更好地讲解相应章节的内容。那么我们下一章将要学习什么呢？还记得在第一章里介绍过的 Android 四大组件吧，目前我们只掌握了活动这一个组件，那么下一章就来学习广播接收器吧。跟上脚步，准备继续前进！

经验值：+3500　　目前经验值：16905

级别：鸟

赢得宝物：在大战 300 回合，双方把所有下三烂的手段都用完后，我终于战胜了碎片最佳实践兽。拾取碎片最佳实践兽掉落的宝物，一块当下在神界特别流行的 iPad，由神界设计新秀小乔主导设计。