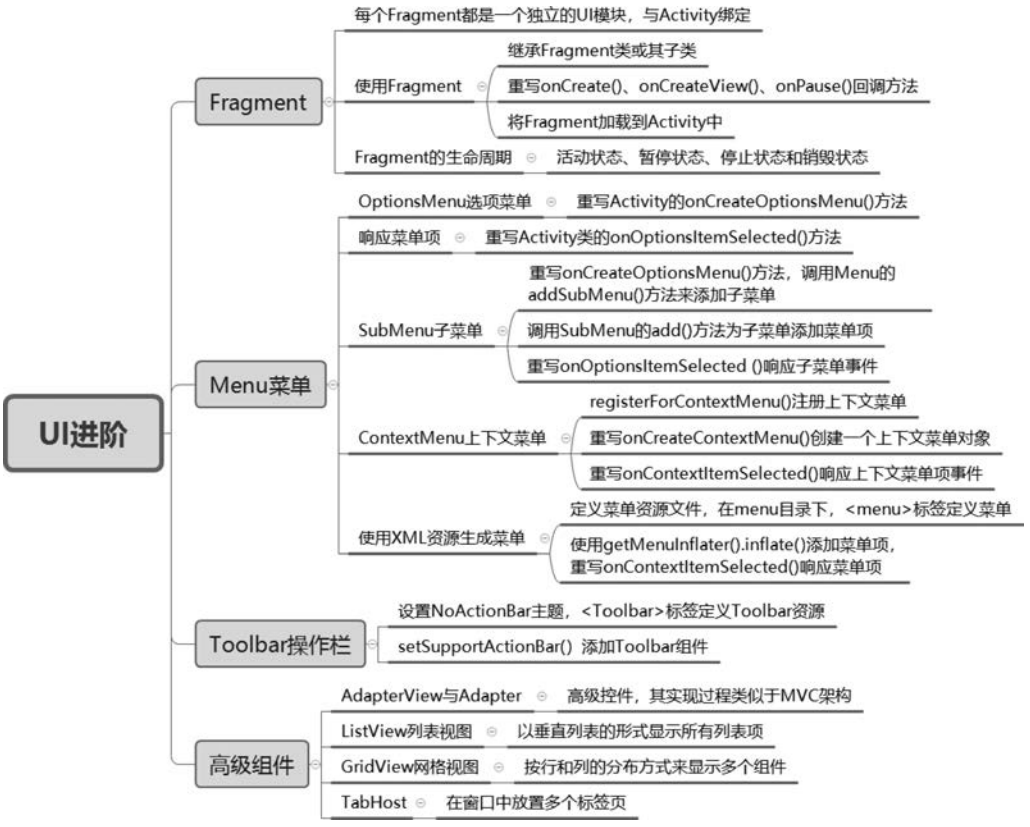
第5章 UI进阶

本章思维导图



本章目标

- 能够熟练使用 Fragment 动态设计 UI 界面；
- 能够熟练使用 Menu 和 Toolbar 组件；
- 能够熟练使用 AdapterView、ListView 和 GridView；
- 掌握 TabHost 组件的使用。

5.1 Fragment

Android 从 3.0 版本开始引入 Fragment(碎片)，允许将 Activity 拆分成多个完全独立封装的可重用的组件，每个组件拥有自己的生命周期和 UI 布局。使用 Fragment 为不同型号、尺寸、分辨率的设备提供统一的 UI 设计方案，Fragment 最大的优点是让开发者更加灵活地根据屏幕大小(包括小屏幕的手机、大屏幕的平板电脑)来创建相应的 UI 界面。

以新闻列表为例，当针对小屏幕手机开发时，开发者通常需要编写两个 Activity，分别是 Activity A 和 Activity B，其中，Activity A 用于显示所有的新闻列表，列表内容为新闻的标题；Activity B 用于显示新闻的详细信息；当用户单击某个新闻标题时，由 Activity A 启动 Activity B，并显示该标题所对应的新闻内容，两个 Activity 界面如图 5-1 所示。

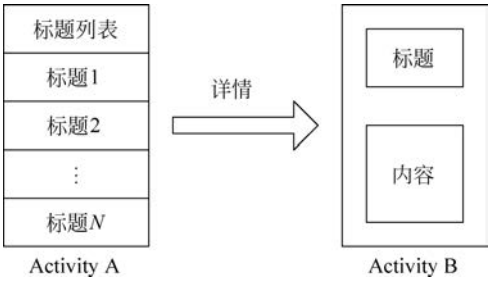


图 5-1 手机上显示新闻列表

当针对平板电脑开发时，使用 Fragment A 来显示标题列表，使用 Fragment B 来显示新闻的详细内容，将这两个 Fragment 在同一个 Activity 中并排显示；当单击 Fragment A 中的新闻标题时，通过 Fragment B 来显示该标题对应的新闻内容，显示效果如图 5-2 所示。每个 Fragment 都有自己的生命周期和相应的响应事件，通过切换 Fragment 同样可以实现显示效果的切换。

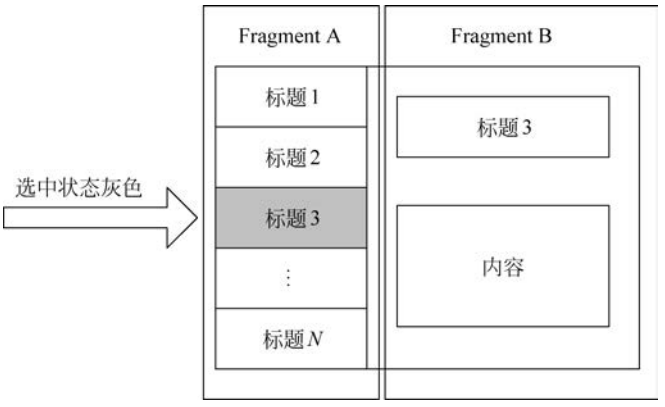


图 5-2 平板电脑上显示新闻列表及内容

每个 Fragment 都是独立的模块，并与其所绑定的 Activity 紧密联系在一起，Fragment 通常会被封装成可重用的模块。对于一个界面允许有多个 UI 模块的设备(如平板电脑等)，Fragment 拥有更好的适应性和动态构建 UI 的能力，在 Activity 中可以动态地添加、删

除或更换 Fragment。

由于 Fragment 具有独立的布局，能够进行事件响应，且具有自身的生命周期和行为，所以开发人员还可以在多个 Activity 中共用一个 Fragment 实例，即当程序运行在大屏设备时启动一个包含多个 Fragment 的 Activity，当程序运行在小屏设备时启动一个包含少量 Fragment 的 Activity。同样以新闻列表为例，对程序进行如下设置：当检测到程序运行在大屏设备时，启动 Activity A，并将标题列表和新闻内容所对应的两个 Fragment 都放在 Activity A 中；当检测到程序运行在小屏设备时，依然启动 Activity A，但此时 Activity A 中只包含一个标题列表 Fragment，当用户单击某个新闻标题时，Activity A 将启动 Activity B，再通过 Activity B 加载新闻内容所对应的 Fragment。

5.1.1 使用 Fragment

创建 Fragment 的过程与 Activity 类似，自定义的 Fragment 必须继承 Fragment 类或其子类。Fragment 的继承体系如图 5-3 所示。

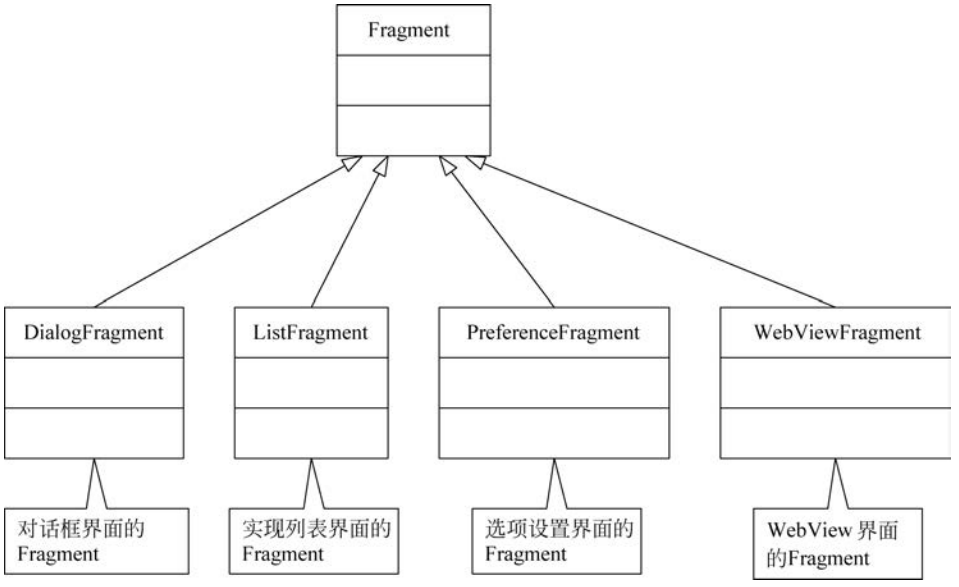


图 5-3 Fragment 的继承体系

与 Activity 类似，同样需要实现 Fragment 中的回调方法，如 onCreate()、onCreateView()、onStart()和 onResume()等。

通常在创建 Fragment 时，需要实现以下三个方法。

- onCreate()：系统在创建 Fragment 对象时调用此方法，用于初始化相关的组件，例如，一些在暂停或者停止时依然需要保留的组件。
- onCreateView()：系统在第一次绘制 Fragment 对应的 UI 时调用此方法，该方法将返回一个 View，如果 Fragment 未提供 UI 则返回 null。当 Fragment 继承自 ListFragment 时，onCreateView()方法默认返回一个 ListView。
- onPause()：当用户离开 Fragment 时首先调用此方法；当用户无须返回时，可以通过该方法来保存相应的数据。

Fragment 不能独立运行，必须嵌入 Acitivity 中使用，因此 Fragment 的生命周期与其所在的 Activity 密切相关。

将 Fragment 加载到 Activity 中主要有以下两种方式。

- 把 Fragment 添加到 Activity 的布局文件中。
- 在 Activity 的代码中动态添加 Fragment。

在上述两种方式中，第一种方式虽然简单但灵活性不够。如果把 Fragment 添加到 Activity 的布局文件中，就会使得 Fragment 及其视图与 Activity 的视图绑定在一起，在 Activity 的生命周期中，无法灵活地切换 Fragment 视图。因此在实际开发过程中，多采用第二种方式。相对而言，第二种方式要比第一种方式复杂，但也是唯一一种可以在运行时控制 Fragment 的方式，可以动态地添加、删除或替换 Fragment 实例。

1. 创建 Fragment

下述代码演示了 Fragment 的基本用法。屏幕分为左右两部分，通过单击屏幕左侧的按钮，在右侧动态地显示 Fragment。其布局文件代码如下。

【案例 5-1】 fragment_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" ...
    android:orientation="horizontal">
    <LinearLayout android:id="@+id/left" android:layout_width="0dp"
        android:layout_height="match_parent" android:layout_weight="1"
        android:background="#FFFFFF" android:orientation="vertical" >
        <Button android:id="@+id/displayBtn" android:text="显示" .../>
    </LinearLayout>
    <LinearLayout android:id="@+id/right" android:layout_width="0dp"
        android:layout_height="match_parent" android:layout_weight="3"
        android:background="#D3D3D3" android:orientation="vertical" >
    </LinearLayout>
</LinearLayout>
```

上述代码较为简单，定义了一个 id 为“left”的 LinearLayout 和一个 id 名为“right”的 LinearLayout，将整个布局分为左右两部分：左边占 1/4，右边占 3/4。其中，id 为“right”的 LinearLayout 是一个包含 Fragment 的容器。

接下来创建 FragmentDemoActivity，用于显示上面的布局效果，代码如下。

【案例 5-2】 FragmentDemoActivity.java

```
public class FragmentDemoActivity extends AppCompatActivity {
    //展示内容 Button
    Button displayBtn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.fragment_main);
        displayBtn = (Button) findViewById(R.id.displayBtn);
        displayBtn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
                //TODO
            }
        });
    }
}
```

上述代码中，声明并初始化名为“displayBtn”的 Button 组件，并为其添加了 OnClickListener 事件监听器，此处仅作演示，事件处理部分暂未实现。运行上述代码，界面效果如图 5-4 所示。

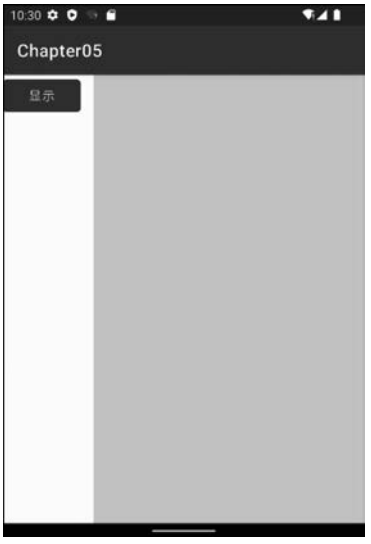


图 5-4 fragment_main.xml 界面效果

当用户单击“显示”按钮时，需要在屏幕右侧动态地显示内容，此处通过动态地添加 Fragment 来实现。在工程中创建一个名为 fragment_right.xml 的文件，用于显示右侧的内容，代码如下。

【案例 5-3】 fragment_right.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" ...
    android:orientation="vertical" >
    <TextView android:id="@+id/textView1" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:singleLine="false"
        android:text="新闻内容新闻内容新闻内容新闻内容新闻内容新闻内容新闻内容" />
    <Button android:id="@+id/frgBtn" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="show" />
</LinearLayout>
```

上述代码较为简单，定义了两个组件：TextView 组件用于显示普通的文本；Button 按钮用于演示 Fragment 中的事件处理机制。

下述代码定义一个 Fragment 类，代码实现如下。

【案例 5-4】 RightFragment.java

```
public class RightFragment extends Fragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
    //重写 onCreateView()方法 ①
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        //获取 view 对象 ②
        View view = inflater.inflate(R.layout.fragment_right, null);
        //从 view 容器中获取组件③
        Button button = (Button) view.findViewById(R.id.frgBtn);
        button.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(getActivity(),
                        "我是 Fragment", Toast.LENGTH_SHORT).show();
            }
        });
```

```
            return view;
        }
        @Override
        public void onPause() {
            super.onPause();
        }
    }
```

上述代码需要注意以下几点。

- 标号①处重写了 onCreateView()方法，该方法返回的 View 对象将作为该 Fragment 显示的 View 组件，当 Fragment 绘制界面组件时将会回调该方法。
- 标号②处通过 LayoutInflater 对象的 inflate()方法加载 fragment_right.xml 布局文件，并返回对应的 View 容器对象，其他组件对象都是从该 View 对象中获取。
- 标号③处从 View 容器中获取 Button 对象，并为该对象添加 OnClickListener 事件监听器，然后实现相应的事件处理功能。

修改案例 5-2 FragmentDemoActivity.java 中 displayBtn 按钮的事件处理方法，当用户单击“显示”按钮时，右侧动态显示 Fragment 对象对应的布局，所修改的代码如下。

```
...
displayBtn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        //步骤 1: 获得一个 FragmentTransaction 的实例
        FragmentManager fragmentManager = getFragmentManager();
        FragmentTransaction transaction = fragmentManager
                .beginTransaction();
        //步骤 2: 用 add()方法加上 Fragment 的对象 rightFragment
        RightFragment rightFragment = new RightFragment();
        transaction.add(R.id.right, rightFragment);
        //步骤 3: 调用 commit()方法使得 FragmentTransaction 实例的改变生效
        transaction.commit();
    }
});
...
```

图 5-5　动态显示 Fragment 对象

再次运行 FragmentDemoActivity 并单击“显示”按钮时，显示右侧的 Fragment；单击 show 按钮时，弹出“我是 Fragment”提示信息，界面效果如图 5-5 所示。

2. 管理 Fragment

通过 FragmentManager 可实现管理 Fragment 对象的管理。在 Activity 中可以通过 getFragmentManager()方法来获取 FragmentManager 对象。

FragmentManager 能够完成以下几方面的操作。

- 通过 findFragmentById()或 findFragmentByTag()方法获取 Activity 中已存在的 Fragment 对象。
- 通过 popBackStack()方法将 Fragment 从 Activity 的回退栈中弹出（模拟用户按 Back 键）。
- 通过 addOnBackStackChangedListerner()方法注册一个侦听器以监视回退栈的变化。

当需要添加、删除或替换 Fragment 对象时，需要借助于 FragmentTransaction 对象来实现，FragmentTransaction 用于实现 Activity 对 Fragment 的操作，例如，添加或删除 Fragment 操作。

Fragment 的最大特点是根据用户的输入可以灵活地对 Fragment 进行添加、删除、替换及执行其他操作。开发人员可以把每个事务保存在 Activity 的回退栈中，使得用户能够在 Fragment 之间进行导航(与在 Activity 之间导航相同)。

针对一组 Fragment 的变化称为一个事务，事务通过 FragmentTransaction 来执行，而 FragmentTransaction 对象需要通过 FragmentManager 来获取，示例代码如下。

【示例】 获取 FragmentTransaction 对象

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

事务是指在同一时刻执行的一组动作，要么一起成功，要么同时失败。事务可以用 add()、remove()、replace()等方法构成，最后使用 commit()方法来提交事务。

在调用 commit()方法之前，可以使用 addToBackStack()方法把事务添加到一个回退栈中，这个后退栈属于所对应的 Activity。当用户按下回退键时，就可以返回到 Fragment 执行事务之前的状态。

注意 FragmentTransaction 被称作 Fragment 事务，与数据库事务类似，Fragment 事务代表了 Activity 对 Fragment 执行的多个改变操作。

下述代码演示了如何用一个 Fragment 代替另一个 Fragment，并在回退栈中保存被代替的 Fragment 的状态。

【示例】 使用 FragmentTransaction

```
//创建一个新的 Fragment 对象
Fragment newFragment = new ExampleFragment();
//通过 FragmentManager 获取 Fragment 事务对象
FragmentTransaction transaction
        = getFragmentManager().beginTransaction();
//通过 replace()方法把 fragment_container 替换成新的 Fragment 对象
transaction.replace(R.id.fragment_container,newFragment);
//添加到回退栈
transaction.addToBackStack(null);
//提交事务
transaction.commit();
```

上述代码中，ExampleFragment 类是一个自定义的 Fragment 子类。通过 replace()方法使用 newFragment 代替组件 R.id.fragment_container 所指向的 ViewGroup 中包含的 Fragment 对象。然后调用 addToBackStack()方法，将被代替的 Fragment 放入回退栈。当用户按 Backspace 键时，回退到事务提交之前的状态，即界面重新展示原来的 Fragment 对象。如果向事务中添加了多个动作，例如，多次调用了 add()、remove()方法之后又调用了 addToBackStack()方法，那么在 commit()之前调用的所有方法都被作为一个事务。当用户按回退键时，所有的动作都会回滚。

事务中动作的执行顺序可以随意，但需要注意以下两点。

- 程序的最后必须调用 commit()方法。
- 如果程序中添加了多个 Fragment 对象，则显示的顺序跟添加顺序一致(即后添加的覆盖之前的)。如果在执行的事务中有删除 Fragment 对象的动作，而且没有调用 addToBackStack()方法，那么当事务提交时被删除的 Fragment 就会被销毁。反之，那些 Fragment 就不会被销毁，而是处于停止状态，当用户返回时，这些 Fragment 将会被恢复。

注意 调用 commit()方法后，事务并不会马上提交，而是会在 Activity 的 UI 线程(主线程)中等待直到线程能执行的时候才执行，不过可以在 UI 线程中调用 executePendingTransactions()方法来立即执行事务。但一般不需要这样做，除非有其他线程在等待事务的执行。

3. 与 Activity 通信

Fragment 的实现是独立于 Activity 的，可以用于多个 Activity 中；而每个 Activity 允许包含同一个 Fragment 类的多个实例。在 Fragment 中，通过调用 getActivity()方法可以获得其所在的 Activity 实例，然后使用 findViewById()方法查找 Activity 中的组件，示例代码如下。

【示例】 Fragment 获取其所在的 Activity 中的组件

```
View listView = getActivity().findViewById(R.id.list);
```

在 Activity 中还可以通过 FragmentManager 的 findFragmentById()等方法来查找其所包含的 Frament 实例，示例代码如下。

【示例】 Activity 获取指定 Frament 实例

```
ExampleFragment fragment = (ExampleFragment)getFragmentManager()
                .findFragmentById(R.id.example_fragment);
```

有时需要 Fragment 与 Activity 共享事件，通常的做法是在 Fragment 中定义一个回调接口，然后在 Activity 中实现该回调接口。

下面以新闻列表为例，在 Activity 中包含两个 Fragment：FragmentA 用于显示新闻标题，FragmentB 用于显示标题对应的内容。在 FragmentA 中，用户单击某个标题时通知 Activity，然后 Activity 再通知 FragmentB，此时 FragmentB 就会显示该标题所对应的新闻内容。在 FragmentA 中定义 OnNewsSelectedListener 接口，代码如下。

【示例】 在 Fragment 中定义回调接口

```
public static class FragmentA extends ListFragment {
...
    //Activity必须实现下面的接口
    public interface OnNewsSelectedListener{
            //传递当前被选中的标题的id
                public void onNewsSelected(long id);
        }
...
}
```

然后在 Activity 中实现 OnNewsSelectedListener 接口，并重写 onNewsSelected()方法来通知 FragmentB。当 Fragment 添加到 Activity 中时，会调用 Fragment 的 onAttach()方法，在该方法中检查 Activity 是否实现了 OnNewsSelectedListener 接口，并对传入的 Activity 实例进行类型转换，代码如下。

【示例】 使用 onAttach()方法检查 Activity 是否实现回调接口

```
public static class FragmentA extends ListFragment {
    OnNewsSelectedListener mListener;
    …
    @Override
    public void onAttach(Activity activity){
        super.onAttach(activity);
        try{
            mListener = (OnNewsSelectedListener)activity;
        }catch(ClassCastException e){
            throw new ClassCastException(activity.toString()
                    + "必须继承接口 OnNewsSelectedListener");
        }
    }
    …
}
```

上述代码中，如果 Activity 没有实现该接口，FragmentB 会抛出 ClassCastException 异常。mListener 成员变量用于保存 OnNewsSelectedListener 的实例，FragmentA 通过调用 mListener 的方法实现与 Activity 共享事件。由于 FragmentA 继承自 ListFragment 类，所以每次选中列表项时，就会调用 FragmentA 的 onListItemClick()方法，在 onListItemClick()方法中调用 onNewsSelected()方法实现与 Activity 的共享事件，示例代码如下。

【示例】 Fragment 与 Activity 共享事件

```
public static class FragmentA extends ListFragment {
    OnNewsSelectedListener mListener;
    …
    @Override
            public void onListItemClick(ListView l,View v,int position,long id){
                    mListener.onNewsSelected(id);
            }
    …
}
```

上述代码中，onListItemClick()方法中的参数 id 是列表中被选项的 ID，Fragment 通过该 ID 实现从程序的某个存储单元中取得标题的内容。

注意 在数据传递时，也可以直接把数据从 FragmentA 传递给 FragmentB，不过该方式降低了 Fragment 的可重用的能力。现在的处理方式只需要把发生的事件告诉宿主，由宿主决定如何处置，以便 Fragment 的重用性更好。

5.1.2 Fragment 的生命周期

Fragment 的生命周期与 Activity 的生命周期类似，也具有以下几个状态。

- 活动状态——当前 Fragment 位于前台时，用户可见并且可以获取焦点。

- 暂停状态——其他 Activity 位于前台，该 Fragment 仍然可见，但不能获取焦点。
- 停止状态——该 Fragment 不可见，失去焦点。
- 销毁状态——该 Fragment 被完全删除或该 Fragment 所在的 Activity 结束。

Fragment 的生命周期及相关回调方法如图 5-6 所示。

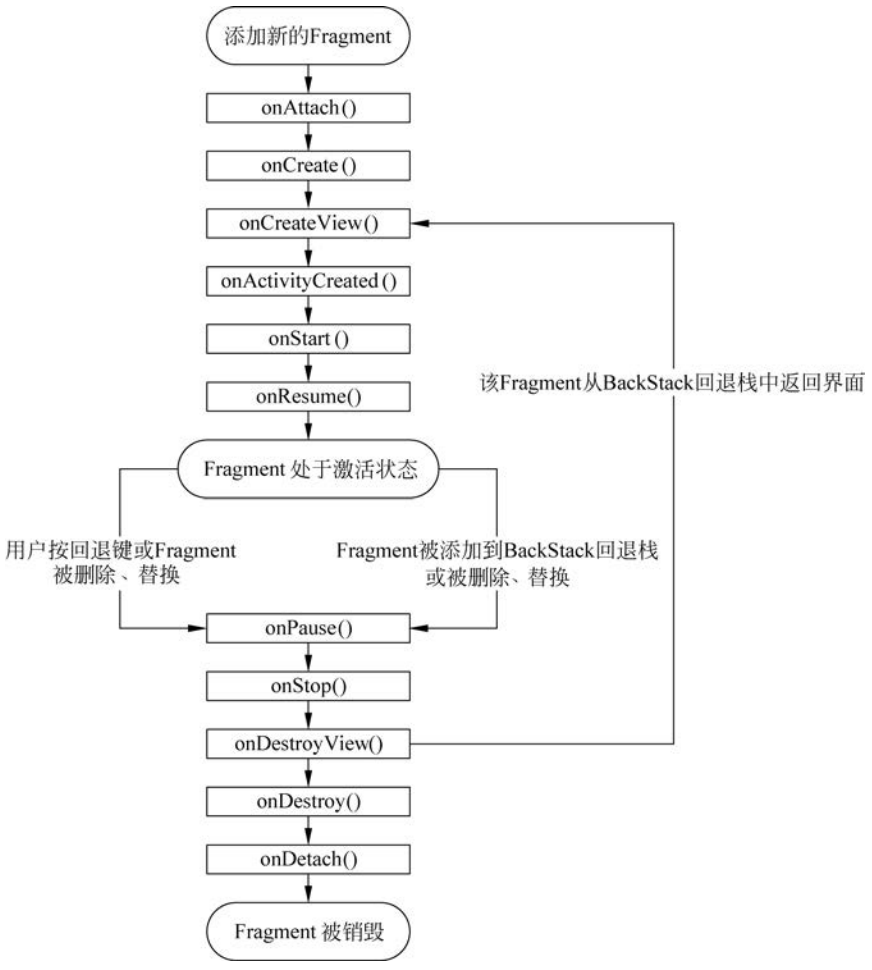


图 5-6　Fragment 的生命周期及相关回调方法

Fragment 生命周期中的方法说明如表 5-1 所示。

表 5-1　Fragment 生命周期中的方法说明

序号	方　法	功 能 描 述
1	onAttach()	当一个 Fragment 对象关联到一个 Activity 时被调用
2	onCreate()	初始化创建 Fragment 对象时被调用
3	onCreateView()	当 Activity 获得 Fragment 的布局时调用此方法，Fragment 在其中创建自己的界面
4	onActivityCreated()	当 Activity 对象完成自己的 onCreate()方法时调用
5	onStart()	Fragment 对象在 UI 界面可见时调用
6	onResume()	Fragment 对象的 UI 可以与用户交互时调用

续表

序号	方　法	功 能 描 述
7	onPause()	Fragment 对象可见，但不可交互，由 Activity 对象转为 onPause 状态时调用
8	onStop()	有组件完全遮挡，或者宿主 Activity 对象转为 onStop 状态时调用
9	onDestroyView()	Fragment 对象清理 View 资源时调用，即移除 Fragment 中的视图
10	onDestroy()	Fragment 对象完成对象清理 View 资源时调用
11	onDetach()	当 Fragment 被从 Activity 中删掉时调用

上述方法中，当一个 Fragment 被创建的时候执行方法 1～4；当 Fragment 创建完毕并呈现到前台时，执行方法 5～6；当该 Fragment 从可见状态转换为不可见状态时，执行方法 7～8；当该 Fragment 被销毁(或者持有该 Fragment 的 Activity 被销毁)时，执行方法 9～11；此外，在方法 3～5 过程中，可以使用 Bundle 对象保存一个 Fragment 的对象。

无论是在布局文件中包含 Fragment，还是在 Activity 中动态添加 Fragment，Fragment 必须依存于 Activity，因此 Activity 的生命周期会直接影响到 Fragment 的生命周期。Fragment 和 Activity 两者生命周期之间的关系如图 5-7 所示。

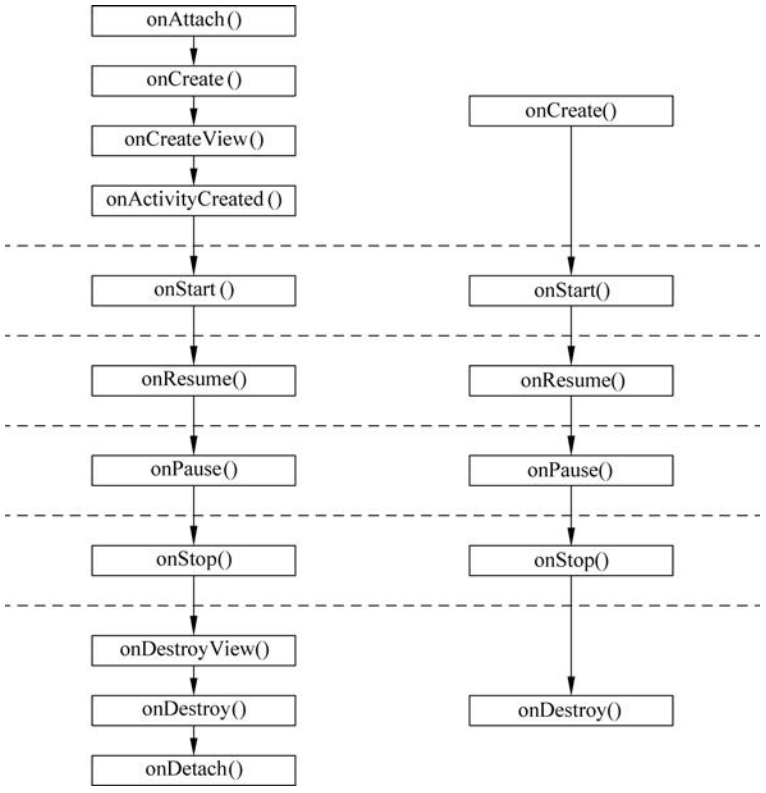


图 5-7　Activity 与 Fragment 生命周期对比

Activity 直接影响其所包含的 Fragment 的生命周期，所以对 Activity 生命周期中的某个方法调用时，也会产生对 Fragment 相应的方法调用。例如，当 Activity 的 onPause()方

法被调用时，其中包含的所有 Fragment 的 onPause()方法都将被调用。

在生命周期中，Fragment 的回调方法要比 Activity 多，多出的方法主要用于与 Activity 的交互，例如，onAttach()、onCreateView()、onActivityCreated()、onDestroyView()和 onDetach()方法。

当 Activity 进入运行状态时（即 running 状态），才允许添加或删除 Fragment。因此，只有当 Activity 处于 resumed 状态时，Fragment 的生命周期才能独立运转，其他阶段依赖于 Activity 的生命周期。

为了使读者更好地理解 Fragment 的生命周期，下面分别介绍静态方式和动态方式。

1. 静态方式

静态方式是指将 Fragment 组件在布局文件中进行布局。Fragment 的生命周期会随其所在的 Activity 的生命周期而发生变化，生命周期的方法调用过程如下。

- 当首次展示布局页面时，其生命周期方法调用的顺序是：onAttach()→onCreate()→onCreateView()→onActivityCreated()→onStart()→onResume()。
- 当关闭手机屏幕或者手机屏幕变暗时，其生命周期方法调用的顺序是：onPause()→onStop()。
- 当对手机屏幕解锁或者手机屏幕变亮时，其生命周期方法调用的顺序是：onStart()→onResume()。
- 当对 Fragment 所在屏幕按回退键时，其生命周期方法调用的顺序是：onPause()→onStop()→onDestroyView()→onDestroy()→onDetach()。

2. 动态方式

当使用 FragmentManager 动态地管理 Fragment 并且涉及 addToBackStack 时，其生命周期的展现显得有些复杂。

动态方式主要通过重写 Fragment 生命周期的方法，然后在 Activity 代码中动态使用 Fragment。例如，定义两个 Fragment 分别为 FragmentA 和 FragmentB，在其生命周期的各个方法中打印（Log 输出方式）相关信息来验证方法的调用顺序。定义 FragmentA 的代码如下。

【案例 5-5】 FragmentA.java

```
public class FragmentA extends Fragment {
    private static final String TAG = FragmentA.class.getSimpleName();
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        Log.i(TAG, "onAttach");
    }
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.i(TAG, "onCreate");
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
```

```
        Log.i(TAG, "onCreateView");
        return inflater.inflate(R.layout.fragment_a, null, false);
    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        Log.i(TAG, "onViewCreated");
        super.onViewCreated(view, savedInstanceState);
    }
    @Override
    public void onDestroy() {
        Log.i(TAG, "onDestroy");
        super.onDestroy();
    }
    @Override
    public void onDetach() {
        Log.i(TAG, "onDetach");
        super.onDetach();
    }
    @Override
    public void onDestroyView() {
        Log.i(TAG, "onDestroyView");
        super.onDestroyView();
    }
    @Override
    public void onStart() {
        Log.i(TAG, "onStart");
        super.onStart();
    }
    @Override
    public void onStop() {
        Log.i(TAG, "onStop");
        super.onStop();
    }
    @Override
    public void onResume() {
        Log.i(TAG, "onResume");
        super.onResume();
    }
    @Override
    public void onPause() {
        Log.i(TAG, "onPause");
        super.onPause();
    }
    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        Log.i(TAG, "onActivityCreated");
        super.onActivityCreated(savedInstanceState);
    }
}
```

定义 FragmentB 的代码如下。

【案例 5-6】 FragmentB.java

```
public class FragmentB extends Fragment {
    private static final String TAG = FragmentB.class.getSimpleName();
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        Log.i(TAG, "onAttach");
    }
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.i(TAG, "onCreate");
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        Log.i(TAG, "onCreateView");
        return inflater.inflate(R.layout.fragment_b, null, false);
    }
    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        Log.i(TAG, "onViewCreated");
        super.onViewCreated(view, savedInstanceState);
    }
    @Override
    public void onDestroy() {
        Log.i(TAG, "onDestroy");
        super.onDestroy();
    }
    @Override
    public void onDetach() {
        Log.i(TAG, "onDetach");
        super.onDetach();
    }
    @Override
    public void onDestroyView() {
        Log.i(TAG, "onDestroyView");
        super.onDestroyView();
    }
    @Override
    public void onStart() {
        Log.i(TAG, "onStart");
        super.onStart();
    }
    @Override
    public void onStop() {
        Log.i(TAG, "onStop");
        super.onStop();
    }
    @Override
    public void onResume() {
```

```
        Log.i(TAG, "onResume");
        super.onResume();
    }
    @Override
    public void onPause() {
        Log.i(TAG, "onPause");
        super.onPause();
    }
    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        Log.i(TAG, "onActivityCreated");
        super.onActivityCreated(savedInstanceState);
    }
}
```

在 Activity 中调用 FragmentA 和 FragmentB，代码如下。

【案例 5-7】 FragmentLifecircleActivity.java

```
public class FragmentLifecircleActivity extends AppCompatActivity
        implements View.OnClickListener {
    //声明 Fragment 管理器 ①
    private FragmentManager fragmentManager;
    //声明变量
    private Button fragABtn;
    private Button fragBBtn;
    //Fragments
    private FragmentA fragmentA;
    private FragmentB fragmentB;
    //Fragment 名称列表
    private String[] fragNames = {"FragmentA","FragmentB"};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_frg_life);
        //初始化 Fragment 管理器 ②
        fragmentManager = getFragmentManager();
        //初始化组件
        fragABtn = (Button) findViewById(R.id.fragABtn);
        fragBBtn = (Button) findViewById(R.id.fragBBtn);
        //设置事件监听器 ③
        fragABtn.setOnClickListener(this);
        fragBBtn.setOnClickListener(this);
    }
    //单击事件监听 ④
    @Override
    public void onClick(View v) {
        FragmentTransaction fragmentTransaction
                = fragmentManager.beginTransaction();
        switch (v.getId()) {
            case R.id.fragABtn:
                if (fragmentA == null) {
                    fragmentA = new FragmentA();
                    fragmentTransaction.replace(R.id.frag_container,
                        fragmentA, fragNames[0]);
```

```
                    //把 FragmentA 对象添加到 BackStack 回退栈中
                    //fragmentTransaction.addToBackStack(fragNames[0]);
                } else {
                    Fragment fragment
                        = fragmentManager.findFragmentByTag(fragNames[0]);
                    //替换 Fragment
                    fragmentTransaction.replace(R.id.frag_container,
                        fragment, fragNames[0]);
                }
                break;
            case R.id.fragBBtn:

                if (fragmentB == null) {
                    fragmentB = new FragmentB();
                    fragmentTransaction.replace(R.id.frag_container,
                        fragmentB, fragNames[1]);
                    //把 FragmentB 对象添加到 BackStack 回退栈中
                    //fragmentTransaction.addToBackStack(fragNames[1]);
                } else {
                    Fragment fragment
                        = fragmentManager.findFragmentByTag(fragNames[1]);
                    //替换 Fragment
                    fragmentTransaction.replace(R.id.frag_container,
                        fragment, fragNames[1]);
                }
                break;
            default:
                break;
        }
        fragmentTransaction.commit();
    }
}
```

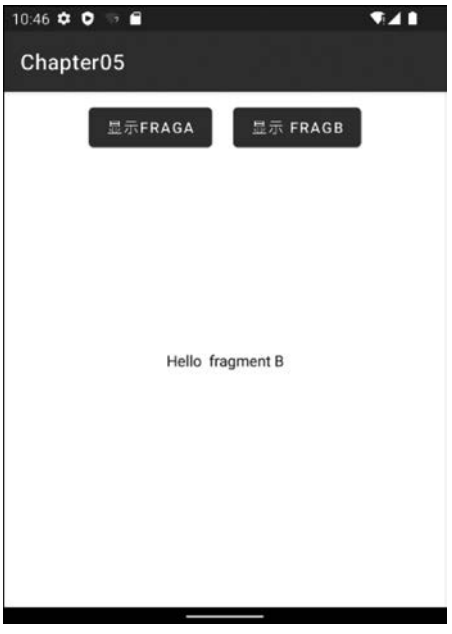


图 5-8　运行 FragmentLifecircleActivity 代码

上述代码需要说明以下几点。

- 标号①分别声明了 FragmentManager、Button 类型的变量属性。
- 标号②处用于对标号①处所声明的属性变量进行初始化，使其可以进行后续的业务逻辑操作。
- 标号③处用于对 fragABtn、fragBBtn 对象注册监听器，来监听用户单击时触发的事件。
- 标号④处重写了 OnClickListener 监听器中的 onClick()方法，用于处理单击事件发生时根据按钮的 id 来决定相应的处理逻辑功能。

运行 FragmentLifecircleActivity 代码时，产生的效果如图 5-8 所示。

当第一次单击“显示 FRAGA”按钮时，实际执行的代码如下。

```
...
fragmentA = new FragmentA();
fragmentTransaction.replace(R.id.frag_container, fragmentA, fragNames[0]);
...
fragmentTransaction.commit();
```

在 Logcat 控制台中打印的日志如下。

```
... I/FragmentA: onAttach
... I/FragmentA: onCreate
... I/FragmentA: onCreateView
... I/FragmentA: onViewCreated
... I/FragmentA: onActivityCreated
... I/FragmentA: onStart
... I/FragmentA: onResume
```

由上述打印日志可知，FragmentA 的生命周期和在布局文件中静态设置的表现完全一致，此处不再赘述。当继续单击“显示 FRAGB”按钮时，在 Logcat 控制台打印的日志如下。

```
... I/FragmentA: onPause
... I/FragmentA: onStop
... I/FragmentA: onDestroyView
... I/FragmentA: onDestroy
... I/FragmentA: onDetach
... I/FragmentB: onAttach
... I/FragmentB: onCreate
... I/FragmentB: onCreateView
... I/FragmentB: onViewCreated
... I/FragmentB: onActivityCreated
... I/FragmentB: onStart
... I/FragmentB: onResume
```

由上述打印结果可知，FragmentA 从运行状态到销毁状态所调用方法的顺序为：onPause()→onStop()→onDestroyView()→onDestroy()→onDetach()。此时，FragmentA 已经由 FragmentManager 进行销毁，取而代之的是 FragmentB 对象。如果此时按回退键，FragmentB 所调用的方法与 FragmentA 调用的顺序一样。在添加 Fragment 过程中如果没有调用 addToBackStack()方法进行保存，那么使用 FragmentManager 更换 Fragment 时，不会保存 Fragment 的状态。

如果取消 FragmentLifecircleActivity 中的 addToBackStack()部分的代码注释，如下。

```
//把 FragmentA 对象添加到 BackStack 回退栈中
fragmentTransaction.addToBackStack(fragNames[0]);
      ...
//把 FragmentB 对象添加到 BackStack 回退栈中
fragmentTransaction.addToBackStack(fragNames[1]);
```

重新运行 FragmentLifecircleActivity，然后单击“显示 FRAGA”按钮，在 Logcat 控制台打印的日志如下。

```
... I/FragmentA: onAttach
... I/FragmentA: onCreate
```

```
... I/FragmentA: onCreateView
... I/FragmentA: onViewCreated
... I/FragmentA: onActivityCreated
... I/FragmentA: onStart
... I/FragmentA: onResume
```

由上述日志可以得知：此时 FragmentA 所调用的方法与没有添加 addToBackStack() 方法时没有任何区别。

然后继续单击“显示 FRAGB”按钮，在 Logcat 控制台打印的日志如下。

```
... I/FragmentA: onPause
... I/FragmentA: onStop
... I/FragmentA: onDestroyView
... I/FragmentB: onAttach
... I/FragmentB: onCreate
... I/FragmentB: onCreateView
... I/FragmentB: onViewCreated
... I/FragmentB: onActivityCreated
... I/FragmentB: onStart
... I/FragmentB: onResume
```

由上述日志可以得知：FragmentA 生命周期方法只是调用了 onDestroyView()，而没有调用 onDestroy()和 onDetach()方法，即 FragmentA 界面虽然被销毁，但 FragmentManager 并没有完全销毁 FragmentA，FragmentA 仍然存在并保存在 FragmentManager 中。

继续单击“显示 FRAGA”按钮，使用 FragmentA 来替换当前显示的 FragmentB，此时实际上执行的代码如下。

```
Fragment fragment = fragmentManager.findFragmentByTag(fragNames[0]);
//替换 Fragment
fragmentTransaction.replace(R.id.frag_container,fragment, fragNames[0]);
```

此时 Logcat 控制台打印的日志为如下。

```
... I/FragmentA: onCreateView
... I/FragmentA: onViewCreated
... I/FragmentA: onActivityCreated
... I/FragmentA: onStart
... I/FragmentA: onResume
... I/FragmentB: onPause
... I/FragmentB: onStop
... I/FragmentB: onDestroyView
```

由上述日志可以得知：使用 FragmentA 替换 FragmentB 的方法调用顺序与使用 FragmentB 替换 FragmentA 时的调用顺序一致，其作用只是销毁视图，但依然保留了 Fragment 的状态。此时 FragmentA 直接调用 onCreateView()方法重新创建视图，并使用上次被替换时的 Fragment 状态。

注意 Fragment 的生命周期对于初学者有些难度，希望读者通过实践、Log 观察的方式对本节有关生命周期的案例运行并认真比对，深刻地理解 Fragment 生命周期的原理和机制，对后期 Fragment 的进一步使用会有很大的帮助。

5.2 Menu 和 Toolbar

Menu(菜单)和 ToolBar(活动条)都是在 Android 应用开发过程中必不可少的元素。Menu 在桌面应用中使用十分广泛,几乎所有的桌面应用都有菜单。而由于受到手机屏幕大小的制约,菜单在手机应用中的使用减少了很多,但为了增强用户的体验仍然在手机应用中提供了菜单功能。

5.2.1 Menu 菜单

Android 中提供的菜单有如下几种。

- 选项菜单(Option Menu):是最常规的菜单,通过单击 Android 设备的菜单栏来启动。
- 子菜单:单击子菜单会弹出悬浮窗口来显示子菜单项,子菜单不支持嵌套,即子菜单中只能包含菜单项而不能再包含其他子菜单。
- 上下文菜单:长按视图控件时所弹出的菜单。在 Windows 中右击时弹出的菜单也是上下文菜单。
- 图标菜单:带 icon 的菜单项。
- 扩展菜单:选项菜单最多只能显示 6 个菜单项,当超过 6 个时第 6 个菜单项会被系统替换为一个“更多”子菜单,显示不出来的菜单项都作为“更多”菜单的子菜单项。

注意 子菜单项、上下文菜单项、扩展菜单项均无法显示图标。

在 Android 中,android.view.Menu 接口代表一个菜单,用来管理各种菜单项。在开发过程中,一般不需要自己创建菜单,因为每个 Activity 默认都自带了一个菜单,只要为菜单添加菜单项及相关事件处理即可。MenuItem 类代表菜单中的菜单项,SubMenu 代表子菜单,两者均位于 android.view 包中。Menu、MenuItem 和 SubMenu 三者的关系如图 5-9 所示。

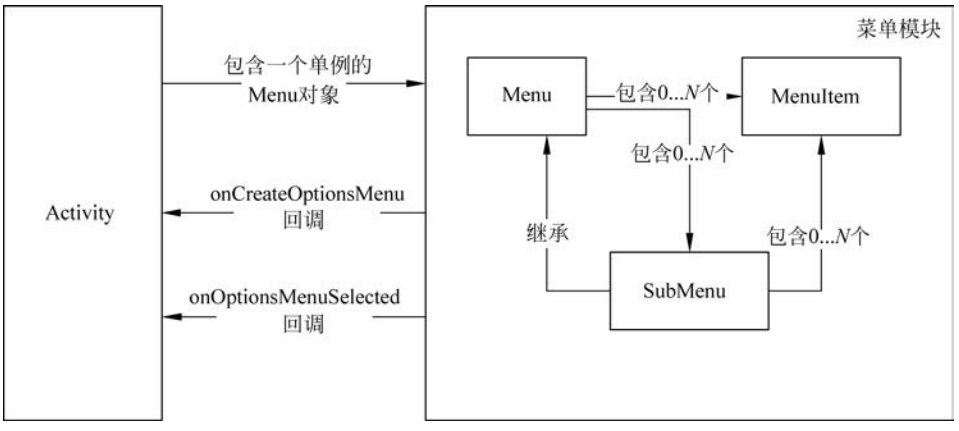


图 5-9 Menu、SubMenu 和 MenuItem 三者的关系

每个 Activity 都包含一个菜单;在菜单中又可以包含多个菜单项和子菜单;由于子菜单实现了 Menu 接口,所以子菜单本身也是菜单,其中可以包含多个菜单项;通常系统创建菜单的方法主要有以下两种。

- onCreateOptionsMenu()：创建选项菜单。
- onCreateContextMenu()：创建上下文菜单。

而 OnCreateOptionsMenu()和 OnOptionsMenuSelected()方法是 Activity 中提供的两个回调方法，分别用于创建菜单项和响应菜单项的单击事件。

下面介绍如何创建菜单项、菜单项分组及菜单事件的处理方法。

1. Option Menu 选项菜单

前面介绍过 Android 的 Activity 中已经封装了 Menu 对象，并提供了 onCreateOptionsMenu()回调方法供开发人员对菜单进行初始化，该方法只会在选项菜单第一次显示时被调用；如果需要动态改变选项菜单的内容，可以使用 onPrepareOptionsMenu()方法来实现。初始化菜单内容的代码如下。

【案例 5-8】 MenuDemoActivity.java

```
public class MenuDemoActivity extends AppCompatActivity{

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
    public boolean onCreateOptionsMenu(Menu menu) {
        //调用父类方法来加入系统菜单
        super.onCreateOptionsMenu(menu);
        //添加菜单项
        menu.add("菜单项 1");
        menu.add("菜单项 2");
        menu.add("菜单项 3");
        //如果希望显示菜单，请返回 true
        return true;
    }
}
```

图 5-10　菜单项效果

上述代码重写了 Activity 的 onCreateOptionsMenu()方法，在该方法中获得系统提供的 Menu 对象，然后通过 Menu 对象的 add()方法向菜单中添加菜单项。运行上述代码并单击右侧的图标时，效果如图 5-10 所示。

添加菜单项时，除了使用 add(CharSequence title)方法，还可以使用以下两种方法。

- add(int resId)——使用资源文件中的文本来设置菜单项的内容，例如 add(R.string.menu1)，其中，R.string.menu1 对应的是在 res\string.xml 中定义的文本。
- add(int groupId,int itemId,int order,CharSequence title)——该方法的参数 groupId 表示组号，开发人员可以给菜单项进行分组，以便快速地操作同一组菜单；参数 itemId 为菜单项指定唯一的 ID，该项用户可以自己指定，也可以让系统来自动分配，在响应菜单时通过 ID 来判断被单击的

菜单；参数 order 表示菜单项显示顺序的编号，编号小的显示在前面；参数 title 用于设置菜单项的内容。

下面使用多个参数的 add()方法实现菜单项的添加，代码如下。

【案例 5-9】 MenuDemoActivity.java

```
...
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        //添加 4 个菜单项,分成两组
        int group1 = 1;
        int gourp2 = 2;
        menu.add(group1, 1, 1, "菜单项 1");
        menu.add(group1, 2, 2, "菜单项 2");
        menu.add(gourp2, 3, 3, "菜单项 3");
        menu.add(gourp2, 4, 4, "菜单项 4");
        //显示菜单
        return true;
    }
```

上述代码运行效果与图 5-10 类似，此处不再演示。对菜单项分组之后，使用 Menu 接口中提供的方法对菜单按组进行操作，该常用的方法如下。

- removeGroup(int group)——用于删除一组菜单。
- setGroupVisible(int group, boolean visible)——用于设置一组菜单是否可见。
- setGroupEnabled(int group, boolean enabled)——用于设置一组菜单是否可单击。
- setGroupCheckable(int group,boolean checkable,boolean exclusive)——用于设置一组菜单的勾选情况。

2. 响应菜单项

Android 常用的菜单响应方式是通过重写 Activity 类的 onOptionsItemSelected()方法来响应菜单项事件。当菜单项被单击时，Android 会自动调用该方法，并传入当前所单击的菜单项，其核心代码如下。

【案例 5-10】 MenuDemoActivity.java

```
...
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
     switch (item.getItemId()) {
        case 1:
            Toast.makeText(this, "菜单项 1", Toast.LENGTH_SHORT).show();
            break;
        case 2:
            Toast.makeText(this, "菜单项 2", Toast.LENGTH_SHORT).show();
            break;
        case 3:
            Toast.makeText(this, "菜单项 3", Toast.LENGTH_SHORT).show();
            break;
        case 4:
            Toast.makeText(this, "菜单项 4", Toast.LENGTH_SHORT).show();
```

```
                break;
            }
            return super.onOptionsItemSelected(item);
        }
```

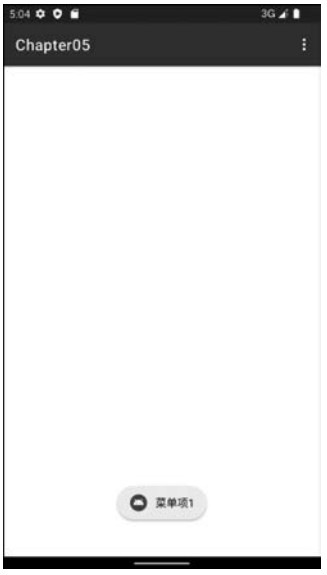


图 5-11　单击菜单项效果

上述代码通过重写 onOptionsItemSelected()方法来响应菜单事件，为方便代码演示，此处将菜单项 ID 直接编码在程序中。运行上述代码，并单击“菜单项 1”按钮时，效果如图 5-11 所示。

3. SubMenu 子菜单

子菜单是一种组织式菜单项，被大量地运用在 Windows 和其他操作系统的 GUI 设计中。Android 同样支持子菜单，开发人员可以通过 addSubMenu()方法来创建子菜单。创建子菜单的步骤如下。

(1) 重写 Activity 类的 onCreateOptionsMenu()方法，调用 Menu 的 addSubMenu()方法来添加子菜单。

(2) 调用 SubMenu 的 add()方法为子菜单添加菜单项。

(3) 重写 Activity 类的 onOptionsItemSelected()方法，以响应子菜单的单击事件。

下述代码演示创建子菜单的过程。

【案例 5-11】 SubMenuDemoActivity.java

```
public class SubMenuDemoActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);
    }
    //初始化菜单
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        //添加子菜单
        SubMenu subMenu = menu.addSubMenu(0, 2, Menu.NONE, "基础操作");
        //为子菜单添加菜单项
        //"重命名"菜单项
        MenuItem renameItem = subMenu.add(2, 201, 1, "重命名");
        //"分享"菜单项
        MenuItem shareItem = subMenu.add(2, 202, 2, "分享");
        //"删除"菜单项
        MenuItem delItem = subMenu.add(2, 203, 3, "删除");
        return true;
    }
    //根据菜单执行相应内容
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case 201:
            Toast.makeText(getApplicationContext(), "重命名...",
```

```
                    Toast.LENGTH_SHORT).show();
            Break;
        case 202:
            Toast.makeText(getApplicationContext(),
                    "分享...", Toast.LENGTH_SHORT).show();
            Break;
        case 203:
            Toast.makeText(getApplicationContext(),
                    "删除...", Toast.LENGTH_SHORT).show();
            Break;
    }
    return true;
}
```

上述代码中，通过 addSubmenu()方法为 menu 菜单添加了 SubMenu 子菜单；使用 add()方法为子菜单连续添加了三个菜单项；在子菜单中添加菜单项的方式和在菜单中添加菜单项的方式完全相同。此外，通过 MenuItem 的 setIcon()方法为子菜单的每个菜单项设置相应的图标；运行代码并单击右侧的⋮图标后，界面效果如图 5-12 所示。

然后单击“基础操作”菜单项，弹出与之对应的子菜单项，效果界面如图 5-13 所示。

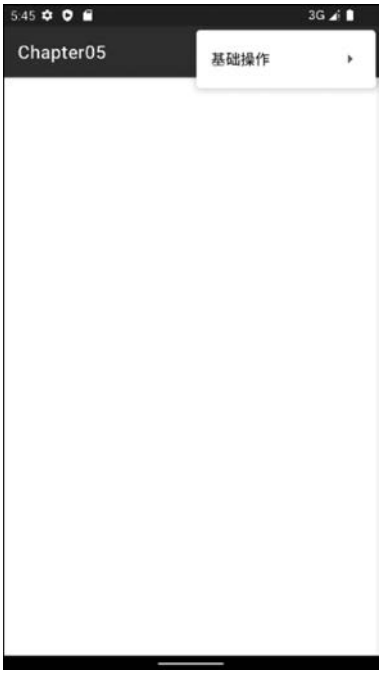


图 5-12 子菜单弹出

图 5-13 子菜单项

与图 5-13 的菜单项相比，图 5-12 中显示的“基础操作”菜单项视觉效果较差；接下来使用 SubMenu 的 setIcon()方法为“基础操作”子菜单及子菜单项添加图标，代码如下。

```
//添加子菜单
SubMenu subMenu = menu.addSubMenu(0, 2, Menu.NONE, "基础操作");
subMenu.setIcon(android.R.drawable.ic_menu_manage);
```

```
//添加子菜单项
//"重命名"菜单项
MenuItem renameItem = subMenu.add(2, 201, 1, "重命名");
renameItem.setIcon(android.R.drawable.ic_menu_edit);
//"分享"菜单项
MenuItem shareItem = subMenu.add(2, 202, 2, "分享");
shareItem.setIcon(android.R.drawable.ic_menu_share);
//"删除"菜单项
MenuItem delItem = subMenu.add(2, 203, 3, "删除");
delItem.setIcon(android.R.drawable.ic_menu_delete);
```

重新运行修改后的 SubMenuDemoActivity，并单击图标后，界面效果如图 5-14 所示。

图 5-14　为子菜单增加图标

在 Menu 中可以包含多个 SubMenu，SubMenu 可以包含多个 MenuItem，但 SubMenu 不能包含 SubMenu，即子菜单不能嵌套。例如，下面的语句在运行时会报错。

```
subMenu.addSubMenu("子菜单嵌套"); //编译时通过,运
                                  //行时报错
```

上面的语句虽然能够通过编译，但在运行时会报错。

4. ContextMenu 上下文菜单

在 Windows 操作系统中，用户能够在文件上右击来执行“打开”“复制”“剪切”等操作，右击所弹出的菜单就是上下文菜单。在手机中经常通过长按某个视图元素来弹出上下文菜单。

上下文菜单是通过调用 ContextMenu 接口中的方法来实现的。ContextMenu 接口继承了 Menu 接口，如图 5-15 所示，因此可以像操作选项菜单一样为上下文菜单增加菜单项。上下文菜单与选项菜单最大的不同是：选项菜单的拥有者是 Activity，而上下文菜单的拥有者是 Activity 中的 View 对象。每个 Activity 有且只有一个选项菜单，并为整个 Activity 服务。而一个 Activity 通常拥有多个 View 对象，根据需要为某些特定的 View 对象提供上下文菜单，通过调用 Activity 的 registerForContextMenu()方法将某个上下文菜单注册到指定的 View 对象上。

虽然 ContextMenu 对象的拥有者是 View 对象，但是需要使用 Activity 的 onCreateContextMenu()方法来生成 ContextMenu 对象，该方法的签名如下。

【语法】

```
onCreateContextMenu(ContextMenu menu, View v,
                    ContextMenu.ContextMenuInfo menuInfo)
```

上述方法与 onCreateOptionsMenu(Menu menu)方法相似，两者的不同之处在于：onCreateOptionsMenu()只在用户第一次按菜单键时被调用，而 onCreateContextMenu()会

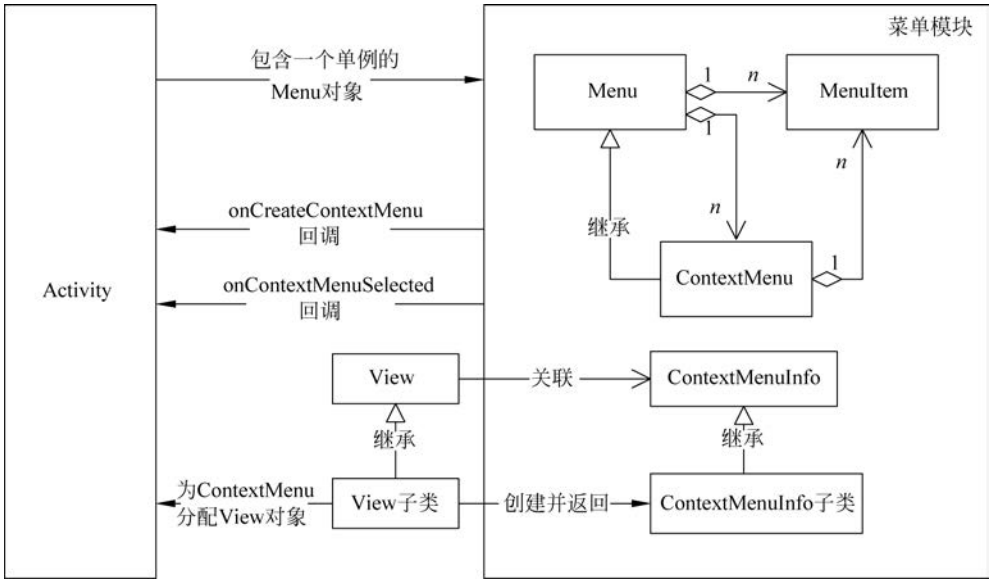


图 5-15 Menu、ContextMenu 关系示意图

在用户每一次长按 View 组件时被调用，并且需要为该 View 注册上下文菜单对象。

注意 在图 5-15 中 ContextMenuInfo 接口的实例作为 onCreateContextMenu()方法的参数。该接口实例用于视图元素需要向上下文菜单传递一些信息，例如，该 View 对应 DB 记录的 ID 等，此时需要使用 ContextMenuInfo。当需要传递额外信息时，需要重写 getContextMenuInfo()方法，并返回一个带有数据的 ContextMenuInfo 实现类对象。限于篇幅，此处不再赘述。

创建上下文菜单的步骤如下。

(1) 通过 registerForContextMenu()方法为 ContextMenu 分配一个 View 对象。

(2) 通过 onCreateContextMenu()创建一个上下文对象。

(3) 重写 onContextItemSelected()方法实现子菜单的单击事件的响应处理。

【案例 5-12】 ContextMenuDemoActivity.java

```
public class ContextMenuDemoActivity extends AppCompatActivity {
    Button contextMenuBtn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_contextmenu);
        //显示列表
        contextMenuBtn = (Button) findViewById(R.id.contextMenuBtn);
        //(1)为按钮注册上下文菜单,长按按钮则弹出上下文菜单
        this.registerForContextMenu(contextMenuBtn);
    }
    //(2)生成上下文菜单
    @Override
    public void onCreateContextMenu(ContextMenu menu, View v,
```

```
            ContextMenuInfo menuInfo) {
        //观察日志确定每次是否重新调用
        Log.d("ContextMenuDemoActivity", "被创建...");
        menu.setHeaderTitle("文件操作");
        //为上下文添加菜单项
        menu.add(0, 1, Menu.NONE, "发送");
        menu.add(0, 2, Menu.NONE, "重命名");
        menu.add(0, 3, Menu.NONE, "删除");
    }
    //(3)响应上下文菜单项
    @Override
    public boolean onContextItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case 1:
            Toast.makeText(this, "发送...", Toast.LENGTH_SHORT).show();
            break;
        case 2:
            Toast.makeText(this, "重命名...", Toast.LENGTH_SHORT).show();
            break;
        case 3:
            Toast.makeText(this, "删除...", Toast.LENGTH_SHORT).show();
            break;
        default:
            return super.onContextItemSelected(item);
        }
        return true;
    }
}
```

图 5-16　ContextMenu 效果

上述代码中，首先在 onCreate()方法中加载了 activity_contextmenu.xml 视图文件，该文件位于 res\layout 文件夹下，其中只包含一个按钮组件，读者可自行查看；然后为按钮注册上下文菜单；接下来通过 onCreateContextMenu()回调方法为系统创建的 ContextMenu 对象添加菜单项；最后通过 onContextItemSelected()方法实现菜单项事件处理。

运行上述代码并长按“上下文菜单”按钮时，系统会弹出上下文菜单，效果如图 5-16 所示。

注意　在运行程序时，通过 Logcat 的输出信息发现：每次弹出上下文菜单时都会调用 onCreateContextMenu()方法。

5. 使用 XML 资源生成菜单

前面介绍的常用菜单，都是通过硬编码方式添加菜单项，Android 为开发人员提供了一种更加方便的菜单生成方式，即通过 XML 文件来加载和响应菜单，

此种方式易于维护，可读性更强。

使用 XML 资源生成菜单项的步骤如下。

(1) 在 res 目录中创建 menu 子目录。

(2) 在 menu 子目录中创建一个 Menu Resource file(XML 文件)，文件名可以随意，Android 会自动为其生成资源 ID，例如，R. menu. context_menu 对应 menu 目录的 context_menu. xml 资源文件；在该 XML 文件中可以提供 menu 所需的菜单项。

(3) 使用 XML 文件的资源 ID(如 R. menu. context_menu)，在 Activity 中将 XML 文件中所定义的菜单元素添加到 menu 对象中。

(4) 通过判断菜单项对应的资源 ID(如 R. id. item_send)，来实现相应的事件处理。

下面将工程中的 ContextMenuDemoActivity 类文件进行复制，并改名为 XMLContextMenuDemoActivity。接下来，在 XMLContextMenuDemoActivity 中使用 XML 资源来生成菜单。

1) 定义菜单资源文件

在 res 目录下创建 menu 子目录，在 menu 目录下创建一个 XML 资源文件，并命名为 context_menu. xml，代码如下。

【案例 5-13】 context_menu. xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< menu xmlns:android = "http://schemas. android. com/apk/res/android">
    < group android:id = "@ + id/group1" >
        < item android:id = "@ + id/item_send" android:title = "发送"/>
        < item android:id = "@ + id/item_rename" android:title = "重命名"/>
        < item android:id = "@ + id/item_del" android:title = "删除"/>
    </group >
</menu >
```

在 context_menu. xml 文件中针对 XMLContextMenuDemoActivity 所定义的菜单项进行重写，并为每个菜单项分配了一个可读性较强的 ID。

2) 使用 MenuInflater 添加菜单项

Inflater 为 Android 建立了从资源文件到对象的桥梁，MenuInflater 把 XML 菜单资源转换为对象并将其添加到 menu 对象中。在 XMLContextMenuDemoActivity 中重写 onCreateContextMenu( ) 方法，并使用 Activity 的 getMenuInflater( ) 方法可以获取 MenuInflater 对象，然后将 XML 文件中所定义的菜单元素添加到 menu 对象中，代码如下。

```
//(2)生成上下文菜单
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
        ContextMenuInfo menuInfo) {
    Log. d("ContextMenuDemoActivity", "被创建...");
    menu. setHeaderTitle("文件操作");
    getMenuInflater(). inflate(R. menu. context_menu, menu);
}
```

3) 响应菜单项

接下来重写 XMLContextMenuDemoActivity 类的 onContextItemSelected( )方法实现菜单项的事件处理功能，代码如下。

```
    //(3)响应上下文菜单项
    @Override
    public boolean onContextItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case R.id.item_send:
            Toast.makeText(this, "发送...", Toast.LENGTH_SHORT).show();
            break;
        case R.id.item_rename:
            Toast.makeText(this, "重命名...", Toast.LENGTH_SHORT).show();
            break;
        case R.id.item_del:
            Toast.makeText(this, "删除...", Toast.LENGTH_SHORT).show();
            break;
        default:
            return super.onContextItemSelected(item);
        }
        return true;
    }
```

上述代码演示了使用 XML 资源文件生成菜单的优势。Android 不仅为 context_menu.xml 文件生成了资源 ID，还为文件中 group、menu 和 item 等元素来自动生成相应的 ID（与布局文件中所定义的 ID 相同）。菜单项 ID 的创建与管理全部由 Android 系统来完成，无须开发人员花费心思进行定义。运行 XMLContextMenuDemoActivity，效果与图 5-16 完全相同。

使用 XML 生成菜单是在 Android 中创建菜单的推荐方式。实际上，开发人员在代码中对菜单项或分组等操作都能在 XML 资源文件中完成，下面简单介绍一些比较常见的操作。

（1）资源文件实现子菜单。

通过在 item 元素中嵌套 menu 子元素来实现子菜单，代码如下。

```
<item android:title="系统设置">
    <menu>
        <item android:id="@+id/mi_display_setting"android:title="显示设置"/>
        <item android:id="@+id/mi_network_setting"android:title="网络设置"/>
        <!—其他菜单项 -->
    </menu>
</item>
```

（2）为菜单项添加图标。

```
<item android:id="@+id/mi_exit" android:title="退出"
    android:icon="@drawable/exit"/>
```

（3）设置菜单项的可选策略。

使用 android:checkableBehavior 设置一组菜单项的可选策略，可选值为 none、all 或 single。

```
<group android:id="..." android:checkableBehavior="all">
    <!-- 菜单项 -->
</group>
```

（4）使用 android:checked 设置特定菜单项。

```
<item android:id="..." android:title="sometitle" android:checked="true"/>
```

(5) 设置菜单项可用/不可用。

```
< item android:id = "..." android:title = "sometitle" android:enabled = "false"/>
```

(6) 设置菜单项可见/不可见。

```
< item android:id = "..." android:title = "sometitle" android:visible = "false"/>
```

5.2.2 Toolbar 操作栏

Toolbar 是从 Android 5.0 开始推出的一个 Material Design 风格的导航组件,Google 非常推荐人们使用 Toolbar 来作为 Android 客户端的导航栏,以此来取代之前的 Actionbar。Actionbar 需要要固定在 Activity 的顶部,与 Actionbar 相比,Toolbar 明显要灵活,Toolbar 可以放到界面的任意位置。除此之外,在设计 Toolbar 时,Google 也为开发者预留了许多可定制修改的余地,例如,设置导航栏图标、设置 App 的 Logo 图标、支持设置标题和子标题、支持添加一个或多个自定义组件、支持 Action Menu 等。

Toolbar 继承自 ViewGroup 类,Toolbar 的常用方法如表 5-2 所示。

表 5-2 Toolbar 常用方法

方 法	功 能 描 述
setTitle(int resId)	设置标题
setSubtitle(int resId)	设置子标题
setTitleTextColor(int color)	设置标题字体颜色
setSubtitleTextColor(int color)	设置子标题字体颜色
setNavigationIcon(Drawable icon)	设置导航栏的图标
setLogo(Drawable drawable)	设置 Toolbar 的 Logo 图标

1. Toolbar 的简单应用

首先在 res\values\themes.xml 文件中,对< style >元素的 parent 属性进行设置,使用 MaterialComponents.DayNight 中的 NoActionBar 主题,从而去除 ActionBar 提供的操作栏,代码如下。

【语法】

```
< style name = "Theme.Chapter05"
        parent = "Theme.MaterialComponents.DayNight.NoActionBar">
```

然后在 Activity 对应的布局文件中添加 androidx.appcompat.widget 包中的 Toolbar 组件,语法如下。

【语法】

```
< androidx.appcompat.widget.Toolbar android:id = "@ + id/my_toolbar" .../>
```

接下来,在 Activity 的 onCreate()方法中,使用 setSupportActionBar()方法将 Toolbar 设置为 Activity 的操作栏,其语法如下。

【语法】 显示 Toolbar 组件

```
Toolbar toolbar = (Toolbar) findViewById(R.id.my_toolbar);
setSupportActionBar(toolbar);
```

以上各步操作对应的完整代码如下。

【案例 5-14】 toolbar.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/relativeLayoutContainer" ...>
    <androidx.appcompat.widget.Toolbar
        android:id="@+id/my_toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary" />
</RelativeLayout>
```

【案例 5-15】 ToolbarActivity.java

```
public class ToolbarActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.toolbar);
        Toolbar toolbar = (Toolbar) findViewById(R.id.my_toolbar);
        setSupportActionBar(toolbar);
    }
}
```

运行 ToolbarActivity，结果如图 5-17 所示。

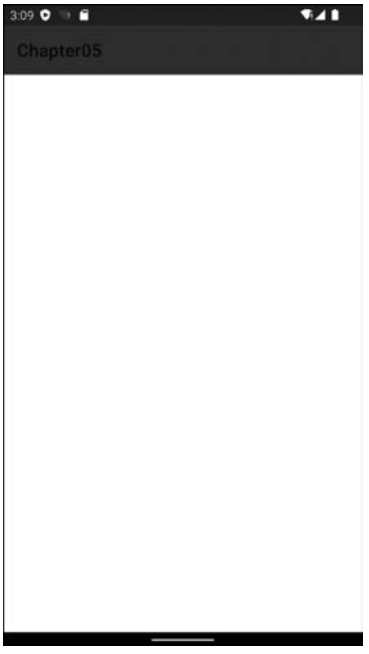


图 5-17 Toolbar 的简单使用

2. Toolbar 的综合应用

在图 5-17 中只显示了应用程序的名称；除此之外，Toolbar 还可以包含导航按钮、应用的 Logo、标题和子标题、若干个自定义 View 及动作菜单等元素，代码如下。

【案例 5-16】 toolbar.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"...
    android:id="@+id/relativeLayoutContainer">
    <androidx.appcompat.widget.Toolbar android:id="@+id/my_toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary" >
        <TextView android:id="@+id/toolbar_title"
            android:text="自定义"
            android:textColor="#fff" android:textSize="21sp" .../>
    </androidx.appcompat.widget.Toolbar>
</RelativeLayout>
```

在上述代码中，在 Toolbar 组件中添加一个 TextView 组件，然后在 Activity 中通过 ID 来获取该 TextView 组件，并为其添加相应的事件处理。

下面在 res\menu 目录中创建一个 XML 布局文件 **menu_tool_demo.xml**，代码如下。

【案例 5-17】 menu_tool_demo.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< menu xmlns:android = "http://schemas.android.com/apk/res/android"...>
    < item android:id = "@ + id/toolbar_action" android:icon = "@mipmap/ic_search"
        android:title = "Action1"app:showAsAction = "ifRoom"/>
    < item android:id = "@ + id/action_item1" android:title = "item1"
        app:showAsAction = "never" />
    < item android:id = "@ + id/action_item2" android:title = "item2"
        app:showAsAction = "never" />
</menu >
```

上述代码用于设置 Toolbar 右侧导航栏的内容。

接下来,在 Activity 中设置 Toolbar 的标题及字体颜色、应用的图标、导航按钮图标等特征,代码如下。

【案例 5-18】 ToolbarActivity.java

```
public class ToolbarActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.toolbar);
        Toolbar toolbar = (Toolbar) findViewById(R.id.my_toolbar);
        toolbar.setTitle("ToolbarDemo");
        setSupportActionBar(toolbar);
        //显示应用的 Logo 并设置图标
        getSupportActionBar().setLogo(R.mipmap.ic_launcher);
        //显示标题和子标题并设置颜色
        toolbar.setTitleTextColor(Color.WHITE);
        toolbar.setSubtitle("Android 基础");
        toolbar.setSubtitleTextColor(Color.WHITE);
        //显示导航按钮图标
        toolbar.setNavigationIcon(R.mipmap.ic_drawer_home);
    }
    //显示 Menu 菜单按钮
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_toolbar_demo, menu);
        return true;
    }
}
```

上述代码中,getSupportActionBar()是用于获取已设定的 Toolbar 组件,并通过 setTitle()、setSubtitle()等方法对 Toolbar 进一步进行设置。

运行 ToolbarActivity,结果如图 5-18 所示。

注意 上述代码中,Toolbar 的 setTitle()方法需要在 setSupportActionBar()方法之前调用,否则无效。

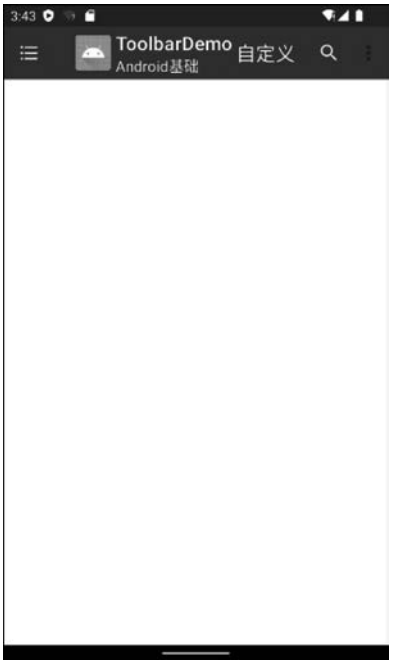


图 5-18　Toolbar 的综合应用

5.3　高级组件

5.3.1　AdapterView 与 Adapter

Java EE 中提供了一种架构模式：MVC(Model View Controller)架构，即模型-视图-控制器三层架构。MVC 架构的实现原理：数据模型用于存放数据，利用控制器将数据显示在视图中。在 Android 中提供了一种高级组件 AdapterView，其实现过程类似于 MVC 架构。AdapterView 之所以称为高级组件，是因为该组件的使用方式与其他组件不同，不仅需要在界面中使用 AdapterView，还需要通过适配器为其添加所需的数据或组件。

(1) 控制层：在 AdapterView 实现的过程中，Adapter 适配器承担了控制层的角色，通过 Adapter 可以将数据源中的数据以某种样式(例如 XML 文件)呈现到视图中。

(2) 视图层：AdapterView 充当了 MVC 中的视图层，用于将前端显示和后端数据分离，其内容一般是包含多项相同格式资源的列表。

(3) 模型层：将数据源当作模型层，其中包括数组、XML 文件等形式的数据。

1. AdapterView 组件

AdapterView 组件是一组重要的组件。AdapterView 本身是一个抽象类，其所派生的子类的使用方式十分相似，但显示特征有所不同。AdapterView 具有以下特征。

- AdapterView 继承了 ViewGroup，其本质上是容器。
- AdapterView 可以包括多个“列表项”，并将“列表项”以合适的形式显示出来。
- AdapterView 所显示的“列表项”是由 Adapter 提供的，通过 AdapterView 的 setAdapter()方法来设置 Adapter 适配器。

AdapterView 及其子类的继承关系如图 5-19 所示。

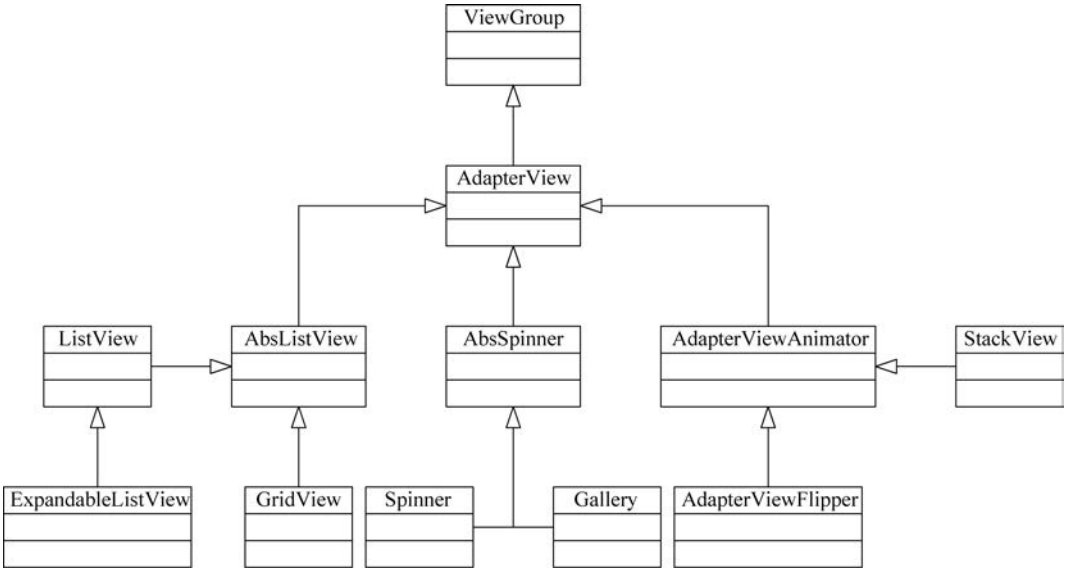


图 5-19 AdapterView 及其子类的继承关系

由图 5-19 可以看出，从 AdapterView 派生出以下三个子类：AbsListView、AbsSpinner 和 AdapterViewAnimator；这些子类依然是抽象类，在实际运用时需要使用这些类的子类，如 GridView、ListView、Spinner 等，具体如下。

- ListView——列表类型。
- Spinner——下拉列表，用于为用户提供选择。
- Gallery——缩略图，已经被 ScrollView 和 ViewPicker 所取代，但有时也会用到，多用于将子项以中心锁定、水平滚动的列表。
- GridView——网格图，以表格形式显示资源，并允许左右滑动。

注意 通常将 ListView、GridView、Spinner 和 Gallery 等 AdapterView 子类作为容器，然后使用 Adapter 为容器提供“列表项”，AdapterView 负责采用合适的方式显示这些列表项。

2. Adapter 组件

Adapter 是一个接口，ListAdapter 和 SpinnerAdapter 是 Adapter 的子接口。其中，ListAdapter 为 AbsListView 提供列表项；而 SpinnerAdapter 为 AbsSpinner 提供列表项。Adapter 接口、子接口以及实现类的关系如图 5-20 所示。

大多数 Adapter 实现类都继承自 BaseAdapter 类，而 BaseAdapter 类实现了 ListAdapter 和 SpinnerAdapter 接口，因此 BaseAdapter 及其子类可以为 AbsListView 和 AbsSpinner 提供列表项。

Adapter 的常用子接口及实现类介绍如下。

- ListAdapter 接口继承于 Adapter 接口，是 ListView 和 List 数据集合之间的桥梁。ListView 组件能够显示由 ListAdapter 所包装的任何数据。

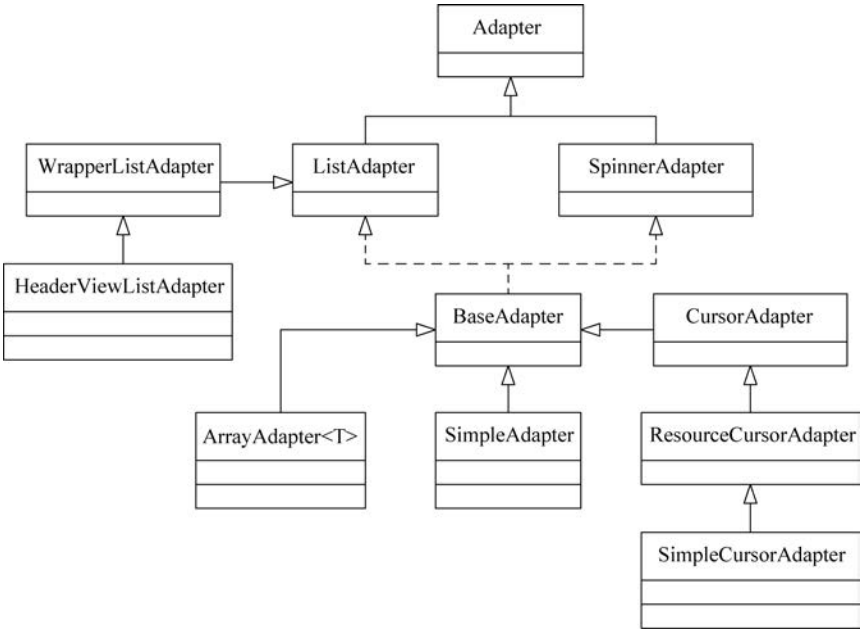


图 5-20　Adapter 接口、子接口以及实现类的关系

- BaseAdapter 抽象类是一个能够在 ListView 和 Spinner 中使用的 Adapter 类的父类。提供扩展 BaseAdapter 可以对各列表项进行最大限度的定制。
- SimpleCursorAdapter 类适用于简单的纯文字型 ListView，需要将 Cursor 字段和 View 中列表项的 ID 对应起来，如需要实现更复杂的 UI 可以通过重写其方法来实现。
- ArrayAdapter 类是简单易用的 Adapter，通常用于将数组或 List 集合包装成多个列表项。
- SimpleAdapter 类是一种简单 Adapter，可以将静态数据在 View 组件中显示。开发人员可以把 List 集合中的数据封装为一个 Map 泛型的 ArrayList。ArrayList 的列表项与 List 集合中的数据相对应。SimpleAdapter 功能强大，使用较为广泛。

注意　Adapter 对象扮演着桥梁的角色，通过桥梁连接着 AdapterView 和所要显示的数据。Adapter 提供了一个连通数据项的途径，将数据集呈现到 View 中。

5.3.2　ListView 列表视图

ListView 列表视图是以垂直列表的形式显示所有列表项，在手机应用中使用比较广泛。ListView 通常具有以下两个职责。

- 将数据填充到布局，以列表的方式来显示数据。
- 处理用户的选择、单击等操作。

通常创建 ListView 有以下两种方式。

- 直接使用 ListView 进行创建。
- 使用 Activity 继承 ListActivity，实现 ListView 对象的获取。

ListView 常用的 XML 属性如表 5-3 所示。

表 5-3 ListView 常用的 XML 属性

XML 属性	功 能 描 述
android:divider	设置列表的分隔条(既可以用颜色分隔,也可以用 Drawable 分隔)
android:dividerHeight	用来指定分隔条的高度
android:entries	指定一个数组资源(例如 List 集合或 String 集合),Android 将根据该数组资源生成 ListView
android:footerDividersEnabled	默认为 true;当设为 false 时,ListView 将不会在各个 footer 之间绘制分隔条
android:headerDividersEnabled	默认为 true;当设为 false 时,ListView 将不会在各个 header 之间绘制分隔条

ListView 从 AbsListView 中继承的 XML 属性如表 5-4 所示。

表 5-4 AbsListView 常用的 XML 属性

XML 属性	功 能 描 述
android:cacheColorHint	用于设置该列表的背景始终以单一、固定的颜色绘制,可以优化绘制过程
android:choiceMode	为视图指定选择的行为,可选的类型如下。 • none:不显示任何选中项。 • singleChoice:允许单选。 • multipleChoice:允许多选。 • multipleChoiceModal:允许多选
android:drawSelectorOnTop	默认为 false;如果为 true,选中的列表项将会显示在上面
android:fastScrollEnabled	用于设置是否允许使用快速滚动滑块;如果设为 true,则将会显示滚动图标,并允许用户拖动该滚动图标进行快速滚动
android:listSelector	设置选中项显示的可绘制对象,可以是图片或者颜色属性
android:scrollingCache	设置在滚动时是否使用绘制缓存,默认为 true。如果为 true,则将使滚动显示更快速,但会占用更多内存
android:smoothScrollbar	默认该属性为 true,列表会使用更精确的基于条目在屏幕上的可见像素高度的计算方法。如果适配器需要绘制可变高度的选项,此时应该设为 false
android:stackFromBottom	设置 GridView 或 ListView 是否将列表项从底部开始显示
android:textFilterEnabled	设置是否对列表项进行过滤;当设为 true 时,列表会对结果进行过滤
android:transcriptMode	设置该组件的滚动模式,该属性支持如下值。 • disabled:关闭滚动,默认值。 • normal:当新条目添加进列表中并且已经准备好显示的时候,列表会自动滑动到底部以显示最新条目。 • alwaysScroll:列表会自动滑动到底部,无论新条目是否已经准备好显示

如果想对 ListView 的外观、行为进行定制,需要将 ListView 作为 AdapterView 来使用,通过 Adapter 来控制每个列表的外观和行为。

在 ListView 中,每个 Item 子项既可以是一个字符串,也可以是一个组合控件。通常而言,使用 ListView 需要完成以下步骤。

(1) 准备 ListView 所要显示的数据。

（2）使用数组或 List 集合存储数据。

（3）创建适配器，作为列表项数据源。

（4）将适配器对象添加到 ListView，并进行展示。

对于简单的 List 列表，直接使用 ArrayAdapter 将数据显示到 ListView 中。如果列表中的内容比较复杂，就需要使用自定义布局来实现 List 列表。接下来分别演示并介绍 ListView 的使用场景。

1. 通过继承 ListActivity 实现 ListView

通过继承 ListActivity 类可以实现 ListView。ListActivity 是 Android 中常用的布局组件之一，通常用于显示可以滚动的列表项。ListActivity 默认布局是由一个位于屏幕中心的全屏列表构成（默认 ListView 占满全屏），该 ListView 组件本身默认的 ID 为@id/android:list，所以在 onCreate()方法中不需要调用 setContentView()方法进行设置布局，而且直接调用 getListView()即可以获取系统默认的 ListView 组件并进行使用。

下述代码通过继承 ListActivity 实现一个简单的 ListView 组件。

【案例 5-19】 ListViewSimpleDemoActivity.java

```
public class ListViewSimpleDemoActivity extends ListActivity {
    //数据源列表
    private String[] mListStr = { "姓名：张三", "性别：男", "年龄：25",
            "居住地：青岛","邮箱：ZhangSan@163.com" };
    ListView mListView = null;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //获取系统默认的 ListView 组件
        mListView = getListView();
        setListAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, mListStr));
        mListView.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                    int position, long id) {
                Toast.makeText(ListVewSimpleDemoActivity.this, "您选择了"
                        + mListStr[position], Toast.LENGTH_LONG).show();
            }
        });
        //设置 ListView 作为显示
        super.onCreate(savedInstanceState);
    }
}
```

上述代码中，ListViewSimpleDemoActivity 继承了 ListActivity 类，使用 ListActivity 中默认的布局及 ListView 组件，因此无须定义该 Activity 的布局文件，也无须调用 setContentView()方法设置布局，直接调用 getListView()获取系统默认的 ID 为@id/android:list 的 ListView 组件即可。除此之外，代码中定义了一个 ArrayAdapter 对象，并通过 ListActivity 的 setListAdapter()方法将其设为 ListView 的适配器对象。

运行上述代码，界面效果如图 5-21 所示。

如果需要在 ListActivity 中显示其他组件，如文本框和按钮等组件，可以采用如下步骤。

图 5-21 简单的 ListView 视图

(1) 先定义 Activity 的布局文件，在布局 UI 界面时先增加其他组件，再添加一个 ListView 组件用于展示数据。

(2) 在 Activity 中通过 setContentView()方法来添加布局对象。

创建 ListActivity 的布局文件，代码如下。

【案例 5-20】 listview_demo.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android" ...>
    <!-- 添加按钮 -->
    < LinearLayout android:layout_width = "match_parent"
        android:layout_height = "wrap_content" >
        < EditText android:id = "@ + id/addTxt" android:layout_width = "212dp"
            android:layout_height = "wrap_content" >
        </EditText >
        < Button android:id = "@ + id/addBtn" android:layout_width = "83dp"
            android:layout_height = "wrap_content" android:text = "添加" >
        </Button >
    </LinearLayout >
    <!-- 自定义的 ListView -->
    < ListView android:id = "@id/android:list " android:layout_width = "match_parent"
        android:layout_height = "0dip" android:layout_weight = "1"
        android:drawSelectorOnTop = "false" />
</LinearLayout >
```

注意 通过继承 ListActivity 来实现 ListView 时，当用户也定义了一个 ID 为@id/android:list 的 ListView，与 ListActivity 中的默认 ListView 组件 ID 一致，则使用 setContentView()方法可以指定用户定义的 ListView 作为 ListActivity 的布局，否则会使用系统提供的 ListView 作为 ListActivity 的布局。

下述代码创建一个 Activity 来加载自定义的 XML 布局文件。

【案例 5-21】 ListViewDemoActivity.java

```
public class ListViewDemoActivity extends ListActivity {
    //数据源列表
    private String[] mListStr = { "姓名：张三", "性别：男", "年龄：25",
          "居住地：青岛","邮箱：ZhangSan@163.com" };
    ListView mListView = null;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //设置 Activity 的布局
        setContentView(R.layout.listview_demo);
        //获取 ID 为 android:list 的 ListView 组件
        mListView = getListView();
        setListAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, mListStr));
        mListView.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                    int position, long id) {
                Toast.makeText(ListVewDemoActivity.this,
                        "您选择了" + mListStr[position],
                            Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

运行上述代码，结果如图 5-22 所示。

2. 在 AppCompatActivity 中使用自定义的 ListView

首先，修改 listview_demo.xml 文件，将 ListView 组件中的 ID 修改为用户自定义的字段，代码如下。

【案例 5-22】 listview_demo.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    //省略
    <ListView android:id="@+id/listview" android:layout_width="match_parent"
        android:layout_height="0dip" android:layout_weight="1"
        android:drawSelectorOnTop="false" />
</LinearLayout>
```

下面创建一个 Activity 来加载上述自定义的 XML 布局文件，代码如下。

图 5-22 继承 ListActivity 实现自定义 ListView

【案例 5-23】 ListViewDemoActivity.java

```
public class ListViewDemoActivity extends AppCompatActivity{
    //数据源列表
    private String[] mListStr = { "姓名：张三", "性别：男", "年龄：25",
"居住地：青岛","邮箱：zhangsan@163.com" };
    ListView mListView = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //设置 Activity 的布局
        setContentView(R.layout.listview_demo);
        //获取 ID 为 listview 的 ListView 组件
        mListView = (ListView) findViewById(R.id.listview);
        mListView.setAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, mListStr));
        mListView.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                                    int position, long id) {
                Toast.makeText(ListVewDemoActivity.this,
                        "您选择了" + mListStr[position],
                        Toast.LENGTH_LONG).show();
            }
        });
    }}
```

上述代码中，ListViewDemoActivity 与 ListViewSimpleDemoActivity 基本相同，不同之处在于：ListViewSimpleDemoActivity 没有调用 setContentView()方法，而 ListViewDemoActivity 使用 listview_demo.xml 布局文件来渲染整个布局。

运行 ListVewDemoActivity，界面效果如图 5-23 所示。

图 5-23　ListView 视图

3. 复杂 ListView 的使用

前面介绍的两个例子都只展示文本行，在实际应用中图文混排也是较常见的，即在行中既包括文字又包括图片。图文混排功能需要用户根据需求来自定义 Adapter 适配器。通常实现图文混排的步骤如下。

（1）定义行选项的布局格式。

（2）自定义一个 Adapter，并重写其中的关键方法，如 getCount()、getView()等方法。

（3）注册列表选项的单击事件。

（4）创建 Activity 并加载对应的布局文件。

下述代码通过上述步骤来完成一个图文混排的列表案例。新建行选项的布局文件 item.xml，代码如下。

【案例 5-24】 item.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< RelativeLayout xmlns:android = "http://schemas.android.com/apk/res/android"...>
    < TextView android:id = "@ + id/itemTxt".../>
    < ImageView android:id = "@ + id/itemImg" android:layout_alignParentRight = "true"
        android:layout_marginRight = "10dp" .../>
</RelativeLayout >
```

上述代码主要定义一个 TextView 和一个 ImageView，用于显示列表的每行中的文本和图片。

然后，创建一个自定义的 TextImageAdapter，代码如下。

【案例 5-25】 TextImageAdapter.java

```
public class TextImageAdapter extends BaseAdapter {
    private Context mContext;
    //展示的文字
    private List < String > texts;
    //展示的图片
    private List < Integer > images;
    public TextImageAdapter(Context context, List < String > texts,
            List < Integer > images) {
```

```
        this.mContext = context;
        this.texts = texts;
        this.images = images;
    }
    /**
     * 元素的个数
     */
    public int getCount() {
        return texts.size();
    }
    public Object getItem(int position) {
        return null;
    }
    public long getItemId(int position) {
        return 0;
    }
    //用以生成在 ListView 中展示的一个 View 元素
    public View getView(int position, View convertView, ViewGroup parent) {
        //优化 ListView
        if (convertView == null) {
            convertView = LayoutInflater.from(mContext)
                    .inflate(R.layout.item,null);
            ItemViewCache viewCache = new ItemViewCache();
            viewCache.mTextView = (TextView) convertView
                    .findViewById(R.id.itemTxt);
            viewCache.mImageView = (ImageView) convertView
                    .findViewById(R.id.itemImg);
            convertView.setTag(viewCache);
        }
        ItemViewCache cache = (ItemViewCache) convertView.getTag();
        //设置文本和图片,然后返回这个 View,用于 ListView 的 Item 的展示
        cache.mTextView.setText(texts.get(position));
        cache.mImageView.setImageResource(images.get(position));
        return convertView;
    }
    //元素的缓冲类,用于优化 ListView
    private class ItemViewCache {
        public TextView mTextView;
        public ImageView mImageView;
    }
}
```

上述代码中创建了 TextImageAdapter 类,用于进行数据的适配与展示,其中该类继承了 BaseAdapter,由于 BaseAdapter 已经实现了 Adapter 的大部分方法,因此在 TextImageAdapter 中只需要实现所需要的部分即可,例如,getCount()方法和 getView()方法;getCount()方法用于返回 ListView 中文本元素的数量,getView()方法用于生成所要展示的 View 对象。在 ListView 中,每添加一个 View 就会调用一次 Adapter 的 getView()方法,所以有必要对该方法进行优化,上面例子中通过自定义 ItemViewCache 类实现了部分优化。

创建 Activity 的布局文件,代码如下。

【案例 5-26】 listview_image.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"...>
< ListView android:id = "@ + id/list_image".../>
</LinearLayout >
```

接下来创建一个用于展现图文混排的 Activity，并注册单击选择项的事件，代码如下。

【案例 5-27】 ListVewImageDemoActivity.java

```
public class ListVewImageDemoActivity extends AppCompatActivity {
    //展示的文字
    private String[ ] texts = new String[ ]{"樱花","小鸡","坚果"};
    //展示的图片
    private int[ ] images = new int[ ]{R.drawable.cherry_blossom,
            R.drawable.chicken,R.drawable.chestnut};
    ListView mListView = null;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //设置 ListView 作为显示
        super.onCreate(savedInstanceState);
        //设置 Activity 布局
        setContentView(R.layout.listview_image);
        //获取 ID 为 list_image 的 ListView 组件
        mListView = (ListView)findViewById(R.id.list_image);
        //加载适配器
        TextImageAdapter adapter = new TextImageAdapter(this, texts, images);
        mListView.setAdapter(adapter);
        mListView.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView <?> parent, View view,
                    int position, long id) {
                Toast.makeText(ListVewImageDemoActivity.this,
                        "您选择了" + texts[position], Toast.LENGTH_LONG).show();
            }
        });}
}
```

上述代码中，首先通过 ListActivity 提供的 getListView()方法来获取 ListView 对象，然后创建一个 TextImageAdapter 适配器对象，并作为 setListAdapter()方法的传入参数，从而把 ListView 和 Adapter 对象进行绑定；最后通过定义内部监听器来实现 ListView 中选择项的单击事件。

运行上述代码，界面运行效果如图 5-24 所示。

注意 上述几个案例主要介绍了 ListView 的常用功能，限于篇幅还有很多功能没能介绍，例如 ListView 的分隔部分、headView、footView 及 ListView 的分页等，请参考其他资料。

图 5-24 图文混排效果

5.3.3 GridView 网格视图

GridView 用于按行和列的分布方式来显示多个组件。GridView 与 ListView 拥有相同的父类 AbsListView，因此两者有许多相同之处，唯一的区别在于：ListView 只显示一列，GridView 可以显示多列。从这个角度看，ListView 可以视为一个特殊的 GridView，当 GridView 只显示一列时，GridView 就变成了 ListView。GridView 也需要通过 Adapter 来提供显示数据。

GridView 常用的 XML 属性如表 5-5 所示。

表 5-5 GridView 常用的 XML 属性

XML 属性	功 能 描 述
android:numColumns	设置列数，可以设置自动，如 auto_fit
android:columnWidth	设置每一列的宽度
android:stretchMode	设置拉伸模式。 • none：拉伸被禁用，不允许被拉伸。 • spacingWidth：列与列之间的间距会被拉伸；因此使用该拉伸模式时，必须指定 columnWidth，而指定 horizontalSpacing 就会使 columnWidth 无效；每列的宽度相等，只需要指定 numColumns 和 horizontalSpacing 属性。 • spacingWidthUniform：每列的间距均被拉伸。当拉伸被禁用时不可以被拉伸
android:verticalSpacing	设置各个元素之间的垂直边距
android:horizontalSpacing	设置各个元素之间的水平边距

在使用 GridView 时，一般都需要为其指定 numColumns 属性，否则 numColumns 默认为 1；当 numColumns 属性设置为 1 时，意味着该 GridView 只有 1 列，此时功能与

ListView 相同。在实际开发中，创建 GridView 的过程与 ListView 相似，步骤如下。

（1）在布局文件中使用< GridView >元素来定义 GridView 组件。

（2）自定义一个 Adapter，并重写其中的关键方法，如 getCount()、getView()等方法。

（3）注册列表选项的单击事件。

（4）创建 Activity 并加载对应的布局文件。

下面通过一个简单示例演示 GridView 的用法。新建布局文件 gridview_demo. xml，代码如下。

【案例 5-28】 gridview_demo. xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< GridView xmlns:android = "http://schemas.android.com/apk/res/android"
    android:id = "@ + id/gridview"
    android:layout_width = "fill_parent"
    android:layout_height = "fill_parent"
    android:columnWidth = "90dp"
    android:gravity = "center"
    android:horizontalSpacing = "10dp"
    android:numColumns = "auto_fit"
    android:stretchMode = "columnWidth"
    android:verticalSpacing = "10dp" >
</GridView >
```

上述代码比较简单，整个布局文件中只有一个 GridView。通过 columnWidth 属性设置列宽为 90dp；将属性 numColumns 设为 auto_fit，Android 会自动计算手机屏幕的大小以决定每行展示几个元素；将属性 stretchMode 设为 columnWidth 则根据列宽自动缩放；horizontalSpacing 属性用于定义列之间的间隔；verticalSpacing 用于定义行之间的间隔。

然后，自定义一个 Adapter 适配器，用于适配 GridView，代码如下。

【案例 5-29】 ImageAdapter. java

```
public class ImageAdapter extends BaseAdapter {
    private Context mContext;
    //一组 Image 的 Id
    private int[] mThumbIds;
    public ImageAdapter(Context context) {
        this.mContext = context;
    }
    @Override
    public int getCount() {
        return mThumbIds.length;
    }
    @Override
    public Object getItem(int position) {
        return mThumbIds[position];
    }
    @Override
    public long getItemId(int position) {
        return 0;
    }
    @Override
```

```
    public View getView(int position, View convertView, ViewGroup parent) {
        //定义一个 ImageView,显示在 GridView 里
        ImageView imageView;
        if (convertView == null) {
            imageView = new ImageView(mContext);
            imageView.setLayoutParams(new GridView.LayoutParams(200, 200));
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8, 8, 8, 8);
        } else {
            imageView = (ImageView) convertView;
        }
        imageView.setImageResource(mThumbIds[position]);
        return imageView;
    }
}
```

上述代码中采用了自定义 Adapter 的方式,与前面自定义 Adapter 的方式相似,以“九宫格”的方式展示图片,每幅图片大小为 200×200px。

最后,创建 GridViewDemoActivity,并加载相应的布局文件,用于显示使用 GridView 布局的界面,代码如下。

【案例 5-30】 GridViewDemoActivity.java

```
public class GridViewDemoActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.gridview_demo);
        GridView gridView = (GridView)findViewById(R.id.gridview);
        ImageAdapter imageAdapter = new ImageAdapter(this,mThumbIds);
        gridView.setAdapter(imageAdapter);
        //单击 GridView 元素的响应
        gridView.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                    int position, long id) {
                //弹出单击的 GridView 元素的位置
                Toast.makeText(GridViewDemoActivity.this,mThumbIds[position],
                        Toast.LENGTH_SHORT).show();
            }
        });
    }
    //展示图片
    private int[] mThumbIds = {
        R.drawable.flg_1, R.drawable.flg_2,
        R.drawable.flg_3, R.drawable.flg_4,
        R.drawable.flg_5, R.drawable.flg_6,
        R.drawable.flg_7, R.drawable.flg_8,
        R.drawable.flg_9
    };
}
```

上述代码中,定义了一组国旗图片,并通过 setOnItemClickListener()方法实现了 gridView 中的图片单击事件,当单击一个图片时会显示该图片所存储的位置。

运行上述代码，界面运行效果如图 5-25 所示。

图 5-25　九宫格

5.3.4　TabHost

TabHost 可以很方便地在窗口中放置多个选项卡，每个选项卡所显示的区域与其外部容器大小相同，通过叠放选项卡可以在容器中放置更多组件。TabHost 是一种比较实用的组件，在应用中比较常见，例如，手机的通话记录中包括“未接电话”“已接电话”等选项卡。

在使用 TabHost 时，通常需要与 TabWidget、TabSpec 组件结合使用，具体功能如下。

- TabWidget 组件用于显示 TabHost 中上部和下部的按钮，单击按钮时切换选项卡。
- TabSpec 代表选项卡界面，通过将 TabSpec 添加到 TabHost 中实现选项卡的添加。

TabHost 仅仅是一个简单的容器，通过以下方法来创建、添加选项卡。

- newTabSpec(String tag)方法用于创建选项卡。
- addTab(tabSpec)方法用于添加选项卡。

使用 TabHost 有两种形式：继承 TabActivity 和不继承 TabActivity。

1. 继承 TabActivity 使用 TabHost

当继承 TabActivity 时，使用 TabHost 的步骤如下。

(1) 定义布局——在 XML 文件中使用 TabHost 组件，并在其中定义一个 FrameLayout 选项卡内容。

(2) 创建 TabActivity——用于显示选项卡组件的 Activity，需要继承 TabActivity。

(3) 获取组件——通过 getTabHost()方法获取 TabHost 对象。

(4) 创建选项卡——通过 TabHost 来创建一个选项卡。

下述代码通过一个简单示例演示 TabHost 的用法。新建布局文件 tabhost_demo1.xml，代码如。

【案例 5-31】 tabhost_demo1.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< TabHost xmlns:android = "http://schemas.android.com/apk/res/android"
    android:id = "@android:id/tabhost"...>
    < LinearLayout android:orientation = "vertical" ...>
        < TabWidget android:id = "@android:id/tabs"
            android:orientation = "horizontal" .../>
            < FrameLayout android:id = "@android:id/tabcontent"
                android:layout_weight = "1" ...>
                < LinearLayout android:id = "@ + id/content1"
                    android:orientation = "vertical" ...>
                    < TextView android:text = "内容 1" .../>
                </LinearLayout >
                < LinearLayout android:id = "@ + id/content2"
                    android:orientation = "vertical" ...>
                     < TextView android:text = "内容 2" .../>
                </LinearLayout >
                < LinearLayout android:id = "@ + id/content3"
                    android:orientation = "vertical" ...>
                      < TextView android:text = "内容 3" .../>
                </LinearLayout >
            </FrameLayout >
    </LinearLayout >
</TabHost >
```

上述布局文件解释如下。

- 布局文件中的根元素为 TabHost，其中其 ID 必须引用 Android 系统自带的 ID，即 android:id=@android:id/tabhost。
- 使用 TabHost 一定要有 TabWidget 和 FramLayout 两个控件。
- TabWidget 必须使用系统 ID，即@android:id/tabs。
- FrameLayout 作为标签内容的基本框架，也必须使用系统 ID，即@android:id/tabcontent。

接下来创建 TabHostDemo1Activity，代码如下。

【案例 5-32】 TabHostDemo1Activity.java

```
public class TabHostDemo1Activity extends TabActivity{
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
      setContentView(R.layout.tabhost_demo1);
        TabHost tabHost = getTabHost();
        //添加第 1 个标签
        TabSpec page1 = tabHost.newTabSpec("tab1")//创建新标签
                .setIndicator("标签 1")//设置标签内容
                .setContent(R.id.content1);
        tabHost.addTab(page1);
        //添加第 2 个标签
        TabSpec page2 = tabHost.newTabSpec("tab2")
                .setIndicator("标签 2")
```

```
                .setContent(R.id.content2);
        tabHost.addTab(page2);
        //添加第 3 个标签
        TabSpec page3 = tabHost.newTabSpec("tab3")
                .setIndicator("标签 3")
                .setContent(R.id.content3);
        tabHost.addTab(page3);
    }
}
```

上述代码解释如下。

- 通过调用从 TabActivity 继承而来的 getTabHost()方法来获取布局文件中的 TabHost 组件。
- 调用 TabHost 组件的 newTabSpec(tag)方法创建一个选项卡，其中，参数 tag 是一个字符串，即选项卡的唯一标识。
- 使用 TabHost. TabSpec 的 setIndicator()方法来设置新选项卡的名称。
- 使用 TabHost. TabSpec 的 setContent()方法来设置选项卡的内容，可以是视图组件、Activity 或 Fragment。
- 使用 tabHost. add(tag)方法将选项卡添加到 TabHost 组件中，其中，传入的 tag 参数是选项卡的唯一标识。

运行上述代码，界面效果如图 5-26 所示。

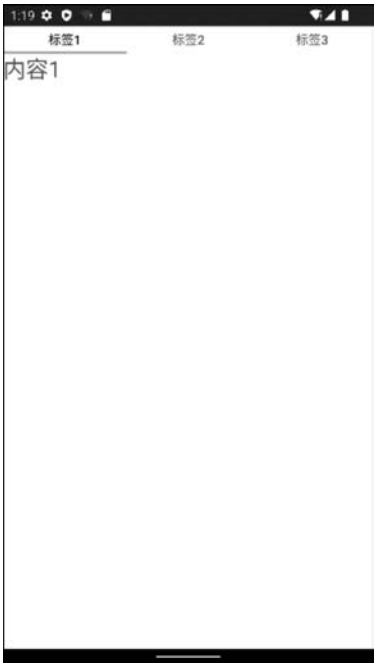


图 5-26　继承 TabActiviy 使用 TabHost

在实际应用中，有时会改变选项卡标签的高度，在代码中通过 getTabWidget()方法来获取 TabWidget 对象，然后使用该对象的 getChildAt()方法来获得指定的标签，最后对该标签中内容的位置进行设置，代码如下。

【示例】 改变选项卡标签的高度

```
TabWidget mTabWidget = tabHost.getTabWidget();
for (int i = 0; i < mTabWidget.getChildCount(); i++) {
    //设置选项卡的高度
    mTabWidget.getChildAt(i).getLayoutParams().height = 50;
    //设置选项卡的宽度
    mTabWidget.getChildAt(i).getLayoutParams().width = 60;
}
```

2. 不继承 TabActivity 使用 TabHost

当不继承 TabActivity 时，使用 TabHost 的步骤如下。

(1) 定义布局——在 XML 文件中使用 TabHost 组件。

(2) 创建 TabActivity——用于显示选项卡组件的 Activity，需要继承 TabActivity。

(3) 获取组件——通过 findViewById()方法获取 TabHost 对象。

(4) 创建选项卡——通过 TabHost 来创建一个选项卡。

新建布局文件 tabhost_demo2.xml,代码如下。

【案例 5-33】 tabhost_demo2.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"...>
    < TabHost android:id = "@ + id/tabHost" android:layout_weight = "1" ...>
        < LinearLayout android:orientation = "vertical" ...>
            < TabWidget android:id = "@android:id/tabs" ...></TabWidget >
            < FrameLayout android:id = "@android:id/tabcontent"
                android:layout_weight = "1" ...>
                < LinearLayout android:id = "@ + id/content_1"
                    android:orientation = "vertical" ...>
                    < TextView android:id = "@ + id/textView" android:textSize = "25sp"
                        android:text = "内容 1" .../>
                </LinearLayout >
                < LinearLayout android:id = "@ + id/content_2"
                    android:orientation = "vertical"...>
                    < TextView android:id = "@ + id/textView2"
                        android:textSize = "25sp" android:text = "内容 2".../>
                </LinearLayout >
                < LinearLayout android:id = "@ + id/content_3"
                    android:orientation = "vertical" ...>
                    < TextView android:id = "@ + id/textView3"
                        android:textSize = "25sp" android:text = "内容 3" .../>
                </LinearLayout >
            </FrameLayout >
        </LinearLayout >
    </TabHost >
</LinearLayout >
```

布局文件 tabhost_demo2.xml 与 tabhost_demo1.xml 相比,TabHost 组件的 ID 是用户自定义的 ID,不再使用 Android 系统自带的 ID。

然后,创建 TabHostDemo2Activity,代码如下。

【案例 5-34】 TabHostDemo2Activity.java

```
public class TabHostDemo2Activity extends AppCompatActivity{
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.tabhost_demo2);
        TabHost tabHost = (TabHost) findViewById(R.id.tabHost);
        tabHost.setup();
        tabHost.addTab(tabHost.newTabSpec("tab1").setIndicator("标签 1")
                .setContent(R.id.content_1));
        tabHost.addTab(tabHost.newTabSpec("tab2").setIndicator("标签 2")
                .setContent(R.id.content_2));
        tabHost.addTab(tabHost.newTabSpec("tab3").setIndicator("标签 3")
                .setContent(R.id.content_3));
        //设置选项卡的高度和宽度
        TabWidget mTabWidget = tabHost.getTabWidget();
        for (int i = 0; i < mTabWidget.getChildCount(); i++) {
```

```
                //设置选项卡的高度
                mTabWidget.getChildAt(i).getLayoutParams().height = 80;
                //设置选项卡的宽度
                mTabWidget.getChildAt(i).getLayoutParams().width = 60;
            }
        }
    }
```

与代码 TabHostDemo1Activity 相比，获取 TabHost 组件不再使用 getTabHost()方法，而是使用 findViewById()方法来获取。使用 addTab()方法来添加选项卡，使用 newTabSpec(tag)方法创建一个选项卡。运行上述代码，界面如图 5-27 所示。

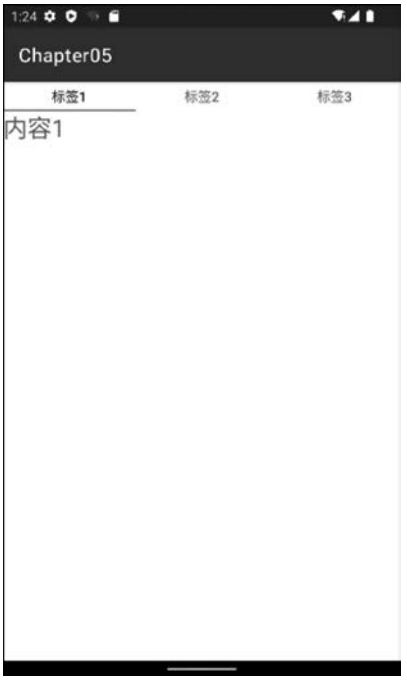


图 5-27　不继承 TabActivity 使用 TabHost

注意　TabActivity 在 Android 3.0 以后已过时，推荐使用“不继承 TabActivity 的方式”使用 TabHost。

小结

（1）Fragment 允许将 Activity 拆分成多个完全独立封装的可重用的组件，每个组件拥有自己的生命周期和 UI 布局。

（2）创建 Fragment 需要实现三个方法：onCreate()、onCreateView()和 onPause()。

（3）Fragment 的生命周期与 Activity 的生命周期相似，具有以下状态：活动状态、暂停状态、停止状态和销毁状态。

（4）Android 中提供的菜单有选项菜单、子菜单、上下文菜单和图标菜单等。

(5) 在 Android 中提供了一种高级控件,其实现过程就类似于 MVC 架构,该控件就是 AdapterView。

(6) ListView(列表视图)是手机应用中使用非常广泛的组件,以垂直列表的形式显示所有列表项。

(7) GridView 用于在界面上按行、列分布的方式显示多个组件。

(8) GridView 与 ListView 拥有相同的父类 AbsListView,且都是列表项,两者唯一的区别在于:ListView 只显示一列,GridView 可以显示多列。

习题

1. 在 Android 中使用 Menu 时可能需要重写的方法有_____。(多选)

A. onCreateOptionsMenu()　　B. onCreateMenu()

C. onOptionsItemSelected()　　D. onItemSelected()

2. 自定义 Adapter 需要重写下列方法中的_____。(多选)

A. getCount()　　B. getItem()

C. getItemId()　　D. getView()

3. 下面自定 style 的方式正确的是_____。

A.

```
<resources>
        <style name="myStyle">
            <item name="android:layout_width">match_parent</item>
         </style>
</resources>
```

B.

```
<style name="myStyle">
        <item name="android:layout_width">match_parent</item>
</style>
```

C.

```
<resources>
    <item name="android:layout_width">match_parent</item>
</resources>
```

D.

```
<resources>
    <style name="android:layout_width">match_parent</style>
</resources>
```

4. 通过使用 ListView 来完成用户的列表功能,并且在每个 item 上显示用户的删除和更新按钮,同时通过测试数据的方式实现对应的业务功能。

5. 使用 GridView 网格显示 3 行 3 列图片。