第 5 章　布局和控件进阶

用户在使用 Android 应用时，往往特别注重导航的清晰和易用性，以及操作的流畅性。导航一般分为整个应用的导航和界面内部导航两种，如抽屉式导航、选项卡式导航、固定数量内部导航和任意数量内部导航等。而流畅性一般通过综合使用 Fragment、ViewPager 等实现。通过本章的学习，读者了解 Fragment 的产生和生命周期，掌握静态和动态地使用 Fragment，了解 Toolbar，使用 DialogFragment 创建 LoginFragment。最后综合使用 Fragment，构建 Android 项目开发框架。

本章的学习目标
- 重点
  - （1）Fragment 的使用
  - （2）Toolbar 和 DialogFragment 的使用
- 难点
  - （1）FragmentAdapter 的使用
  - （2）Fragment 导航

【项目导学】

Fragment 是 Android 的布局利器，本章综合使用 Fragment、ViewPage 等制作仿微信导航，如图 5-1 所示。最后实现一个包含固定数量和任意数量的内导航的综合实例，如图 5-2 所示。学习本章内容后，可以直接构建本书综合项目底部导航、Fragment、内导航等布局，添加 Toolbar 和 Dialog 等。

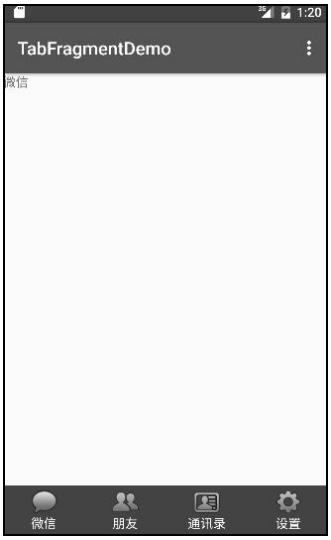

图 5-1　Tab 导航

图 5-2　综合实例

5.1 Fragment

5.1.1 Fragment 的产生与介绍

Android 运行在各种各样的设备中，如小屏幕的手机，超大屏的平板电脑甚至电视。针对屏幕尺寸的差距，很多情况下，都是先针对手机开发一套 App，然后复制一份，修改布局以适应平板电脑等大屏幕设备。Fragment 的出现，就解决了 App 可以同时适应手机和平板电脑及电视的问题。可以把 Fragment 当成 Activity 的一个界面的一个组成部分，甚至 Activity 的界面可以完全由不同的 Fragment 组成。更重要的是，Fragment 拥有自己的生命周期和接收、处理用户的事件，Activity 中就不用写很多控件的事件处理的代码了。除此之外，还可以动态地添加、替换和移除某个 Fragment。

5.1.2 Fragment 的生命周期

Fragment 必须依存于 Activity 而存在，因此 Activity 的生命周期会直接影响到 Fragment 的生命周期。图 5-3 很好地说明了两者生命周期的关系。

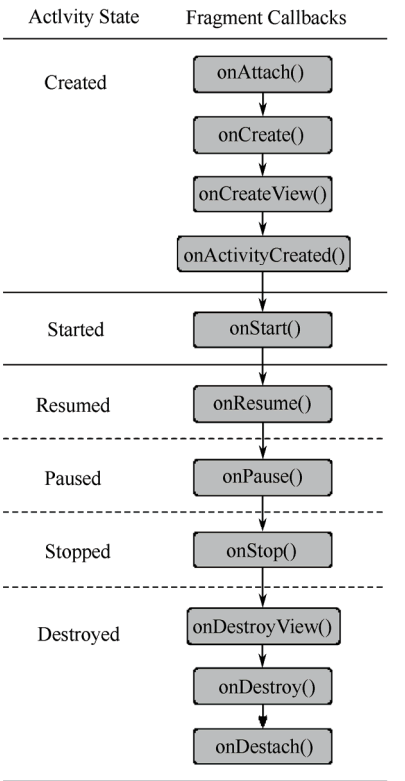

图 5-3 Fragment 生命周期

表 5-1 是 Fragment 生命周期的回调方法。

【说明】 除了 onCreateView，其他的所有方法如果重写了，必须调用父类对于该方法的实现。

表 5-1　回调方法

回调方法	描述
onAttach(Activity)	当 Fragment 与 Activity 发生关联时调用
onCreateView(LayoutInflater, ViewGroup,Bundle)	创建该 Fragment 的视图
onActivityCreated(Bundle)	当 Activity 的 onCreate 方法返回时调用
onDestoryView()	与 onCreateView 相对应，当该 Fragment 的视图被移除时调用
onDetach()	与 onAttach 相对应，当 Fragment 与 Activity 关联被取消时调用

5.1.3　静态使用 Fragment

这是使用 Fragment 最简单的一种方式，把 Fragment 当成普通的控件，直接写在 Activity 的布局文件中。步骤如下：

（1）创建 Fragment 类，继承 Fragment，重写 onCreateView，使用 inflate 加载 Fragment 的布局。

创建 Fragment 有两种方式，一是手工创建，另一种是使用 Fragment 向导。手工创建时，分别创建类文件和布局文件。使用向导创建时，选择“File”→“New”→“Fragment”→“Fragment(Blank)”命令创建空的 Fragment，只需要填写 Fragment 类名和布局文件名，根据需要勾选“Include fragment factory methods?”和“Include interface callbacks?”，一般情况下，都不勾选此两项。

（2）在 Activity 的布局文件中声明此 Fragment，视为普通 View。以下代码就是在 Activity 的布局文件中声明 MasterFragment。

```
<fragment
    android:id="@+id/master_fragment_id"
    android:name="cn.edu.neusoft.simplefragmentdemo.MasterFragment"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```

【例 5-1】 SimpleFragmentDemo 实例，使用 Fragment 向导创建 MasterFragment 和 DetailFragment。在 Activity 的布局文件中，声明这两个 Fragment 作为 Activity 的视图，其中 MasterFragment 用于标题布局，DetailFragment 用于内容布局，如图 5-4 和图 5-5 所示，其中图 5-5 为向下滚动后的效果，也就是标题布局不变，内容布局改变。

图 5-4　静态 Fragment1

图 5-5　静态 Fragment2

实现步骤如下：

（1）创建新项目：项目名称 SimpleFragmentDemo。

（2）MasterFragment 的布局文件

主要包括两个 TextView 和一个 ImageView，用来显示标题、图片和介绍。

```
<RelativeLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    xmlns:tools="http://schemas.Android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="cn.edu.neusoft.simplefragmentdemo.MasterFragment"
    android:background="#FFB6C1"
    android:padding="5dp">
    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:id="@+id/master_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="宝宝百天照系列"
        android:textSize="16sp"/>
    <ImageView
        android:id="@+id/master_img"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/master_text"
        android:src="@drawable/i1"
        android:layout_margin="5dp"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/master_text"
        android:layout_toRightOf="@+id/master_img"
        android:text="宝宝一百天了，给他（她）换上几套新装吧。Cute 宝贝们闪亮登场。"
        android:textSize="14sp"/>
</RelativeLayout>
```

（3）MasterFragment.Java

这部分使用 inflate 适配布局。实际应用中，也可以添加动态获取信息，绑定到布局控件中。

```
public class MasterFragment extends Fragment {
    public MasterFragment() {
        // Required empty public constructor
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
```

```
        return inflater.inflate(R.layout.fragment_master, container, false);
    }
}
```

（4）DetailFragment 的布局文件

DetailFragment 布局使用 ListView。

```
<FrameLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    xmlns:tools="http://schemas.Android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="cn.edu.neusoft.simplefragmentdemo.DetailFragment">
    <!-- TODO: Update blank fragment layout -->
    <ListView
        android:id="@+id/news_list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"></ListView>
</FrameLayout>
```

（5）DetailFragment.Java

在 onCreateView 中获取 ListView 使用如下代码：

```
View view = inflater.inflate(R.layout.fragment_detail, container, false);
news_list = (ListView)view.findViewById(R.id.news_list);
```

完整代码如下：

```
public class DetailFragment extends Fragment {


    private ListView news_list;
    public DetailFragment() {
        // Required empty public constructor
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View view =  inflater.inflate(R.layout.fragment_detail, container, false);
        news_list = (ListView)view.findViewById(R.id.news_list);
        SimpleAdapter adapter = new SimpleAdapter(getActivity(),getData(),R.layout.listview_item,new
String[]{"news_title","news_info","news_thumb"},
                new int[]{R.id.news_title,R.id.news_info,R.id.news_thumb});
        news_list.setAdapter(adapter);
        return view;
    }
    private List<Map<String, Object>> getData() {
        List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
        Map<String, Object> map = new HashMap<String, Object>();
        map.put("news_title","毡帽系列");
        map.put("news_info","此系列服装有点 cute，像不像小车夫。");
```

```
            map.put("news_thumb",R.drawable.i1);
            list.add(map);
            map = new HashMap<String, Object>();
            map.put("news_title","蜗牛系列");
            map.put("news_info","宝宝变成了小蜗牛，爬啊爬啊爬啊。");
            map.put("news_thumb",R.drawable.i2);
            list.add(map);
            map = new HashMap<String, Object>();
            map.put("news_title","小蜜蜂系列");
            map.put("news_info","小蜜蜂，嗡嗡嗡，飞到西，飞到东。");
            map.put("news_thumb",R.drawable.i3);
            list.add(map);
            map = new HashMap<String, Object>();
            map.put("news_title","毡帽系列");
            map.put("news_info","此系列服装有点 cute，像不像小车夫。");
            map.put("news_thumb",R.drawable.i4);
            list.add(map);
            map = new HashMap<String, Object>();
            map.put("news_title","蜗牛系列");
            map.put("news_info","宝宝变成了小蜗牛，爬啊爬啊爬啊。");
            map.put("news_thumb",R.drawable.i5);
            list.add(map);
            map = new HashMap<String, Object>();
            map.put("news_title","小蜜蜂系列");
            map.put("news_info","小蜜蜂，嗡嗡嗡，飞到西，飞到东。");
            map.put("news_thumb",R.drawable.i6);
            list.add(map);
            return list;
        }
}
```

（6）MainActivity

MainActivity 没有特别的语法加入，主要就是调用 setContentView(R.layout.activity_main)加载布局视图。

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

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
            }
        });
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

（7）MainActivity 的布局文件

activity_main.xml 布局文件中，直接使用两个静态的 Fragment，同时使用 android:name 属性，将 master_fragment_id 匹配 MasterFragment，将 detail_fragment_id 匹配 DetailFragment。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    xmlns:app="http://schemas.Android.com/apk/res-auto"
    xmlns:tools="http://schemas.Android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context="cn.edu.neusoft.simplefragmentdemo.MainActivity"
    tools:showIn="@layout/activity_main">
    <fragment
        android:id="@+id/master_fragment_id"
        android:name="cn.edu.neusoft.simplefragmentdemo.MasterFragment"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
```

```
    <fragment
        android:id="@+id/detail_fragment_id"
        android:name="cn.edu.neusoft.simplefragmentdemo.DetailFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@id/master_fragment_id"/>
</RelativeLayout>
```

（8）运行该项目，查看结果。

将 Fragment 当成普通的 View 一样声明在 Activity 的布局文件中，然后所有控件的事件处理等代码都由各自的 Fragment 去处理，代码的可读性、复用性及可维护性有很大提升。

5.1.4 动态使用 Fragment

Fragment 不仅提供静态的使用，也可以动态加载和使用。

实现动态加载，需要先了解 Fragment 事务。熟悉数据库的读者都知道，事务指的就是一种原子性、不可拆分的操作。所谓的 Fragment 事务就是对 Fragment 进行添加、移除、替换或执行其他动作，提交给 Activity 的每一个变化，这就是 Fragment 事务。

Fragment 是 UI 模块，显然在一个 Activity 中可以包含多个模块，所以 Android 提供了 FragmentManage 类来管理 Fragment，FragmentTransaction 类来管理事务。对 Fragment 的动态加载就是先将添加、移除等操作提交到事务，然后通过 FragmentManage 完成的。

通过 FragmentManager.beginTransaction()可以开始一个事务。在事务中，可以对 Fragment 进行的操作以及对应的方法如下：

- 添加，add()；
- 移除，remove()；
- 替换，replace()；
- 提交事务，commit()。

上面几个是比较常用的，还有 attach()、detach()、hide()、addToBackStack()等方法。

这里需要注意的是，Fragment 以 ID 或 Tag 作为唯一标识，所以 remove 和 replace 的参数是 Fragment，这个 Fragment 与目标 Fragment 一致。在下面的示例里，使用了一个栈记录所有添加的 Fragment，然后在移除时使用。

【例 5-2】下面结合经典的底部导航、ViewPage 和 Fragment，介绍如何构建 Fragment 切换应用。图 5-6 是默认 Fragment 的效果，图 5-7 是切换 Fragment 的效果。

实现步骤如下：

（1）创建新项目：项目名称 TabFragmentDemo。

（2）导入素材图片

创建布局之前，复制素材图片到资源目录，如图 5-8 所示。

（3）创建 bottom 布局

底部导航采用简单的线性布局，也可以使用 RadioGroup 来完成。

使用向导创建 bottom 布局文件 Layout/bottom.xml，如图 5-9 所示。布局采用两层线性布局的方式，其中内部的线性布局设置 android:layout_width="0dp"和 android:layout_weight="1"两个属性，使 4 个控件均匀分布。

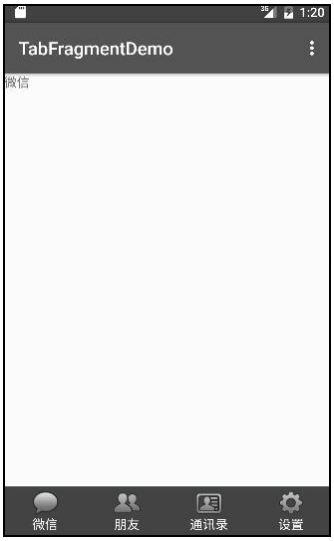

图 5-6　动态 Fragment1

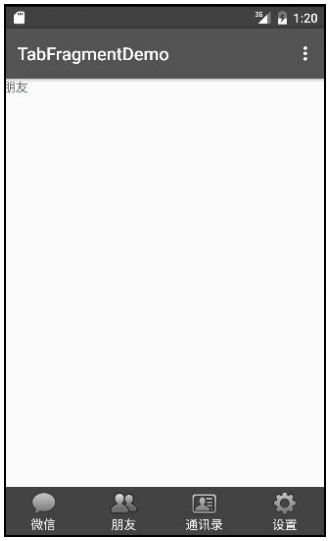

图 5-7　动态 Fragment2

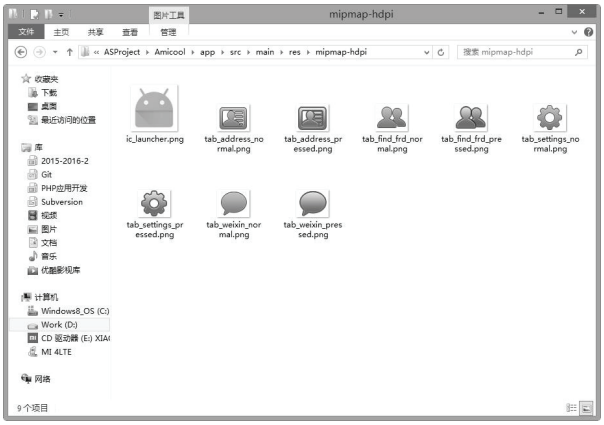

图 5-8　复制素材图片

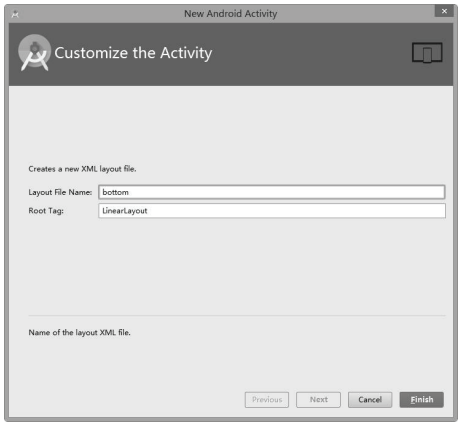

图 5-9　创建 bottom 布局

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    android:layout_width="match_parent"
    android:layout_height="55dp"
    android:gravity="center"
    android:background="@color/material_blue_grey_800">
    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:id="@+id/id_tab_wechat"
        android:gravity="center"
        android:orientation="vertical">
        <ImageButton
            android:id="@+id/id_tab_wechat_img"
```

```
            android:clickable="false"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@mipmap/tab_weixin_pressed"
            android:background="#00000000"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="#ffffff"
            android:text="微信"/>
    </LinearLayout>
    <LinearLayout
        android:id="@+id/id_tab_friend"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:gravity="center"
        android:orientation="vertical">
        <ImageButton
            android:id="@+id/id_tab_friend_img"
            android:clickable="false"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@mipmap/tab_find_frd_normal"
            android:background="#00000000"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="#ffffff"
            android:text="朋友"/>
    </LinearLayout>
    <LinearLayout
        android:id="@+id/id_tab_contact"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:gravity="center"
        android:orientation="vertical">
        <ImageButton
            android:id="@+id/id_tab_contact_img"
            android:clickable="false"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@mipmap/tab_address_normal"
```

```
            android:background="#00000000"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="#ffffff"
            android:text="通讯录"/>
    </LinearLayout>
    <LinearLayout
        android:id="@+id/id_tab_setting"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:gravity="center"
        android:orientation="vertical">
    <ImageButton
        android:id="@+id/id_tab_setting_img"
        android:clickable="false"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/tab_settings_normal"
        android:background="#00000000"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#ffffff"
        android:text="设置"/>
    </LinearLayout>
</LinearLayout>
```

（4）在 content_main.xml 包含 bottom.xml

首先将布局修改为线性，然后增加 ViewPager，最后包含 bottom.xml。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    xmlns:app="http://schemas.Android.com/apk/res-auto"
    xmlns:tools="http://schemas.Android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context=".MainActivity"
    tools:showIn="@layout/app_bar_main"
    android:orientation="vertical">
    <Android.support.v4.view.ViewPager
        android:id="@+id/id_viewpager"
        android:layout_width="match_parent"
        android:layout_height="0dp"
```

```
        android:layout_weight="1">
    </Android.support.v4.view.ViewPager>
    <include layout="@layout/bottom"/>
</LinearLayout>
```

（5）创建 Fragment

使用向导创建 WechatFragment，如图 5-10 所示。

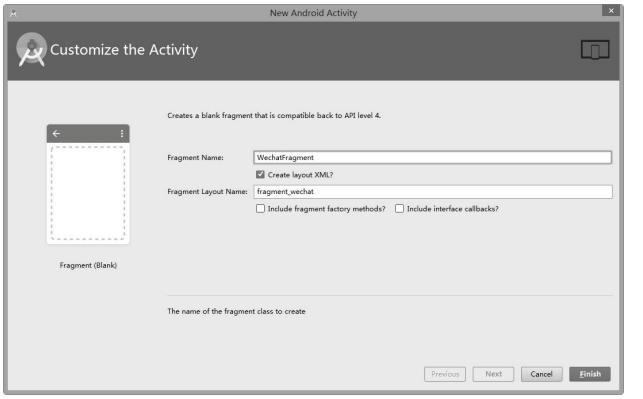

图 5-10　创建 WechatFragment

（6）单击事件

底部导航有 4 个控件，使用 switch 分别进行处理，调用 selectTab，切换到响应 Fragment。

```
View.OnClickListener onClickListener = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //先将 4 个 ImageButton 置为灰色
        resetImgs();

        //根据单击的 Tab 切换不同的页面及设置对应的 ImageButton 为绿色
        switch (v.getId()) {
            case R.id.id_tab_wechat:
                selectTab(0);
                break;
            case R.id.id_tab_friend:
                selectTab(1);
                break;
            case R.id.id_tab_contact:
                selectTab(2);
                break;
            case R.id.id_tab_setting:
                selectTab(3);
                break;
        }
    }
};
```

（7）管理 Fragment

① 声明变量

```
//声明 ViewPager
private ViewPager mViewPager;
//适配器
private FragmentPagerAdapter mAdapter;
//装载 Fragment 的集合
private List<Fragment> mFragments;

//4 个 Tab 对应的布局
private LinearLayout mTabWechat;
private LinearLayout mTabFriend;
private LinearLayout mTabContact;
private LinearLayout mTabSetting;

//4 个 Tab 对应的 ImageButton
private ImageButton mImgWechat;
private ImageButton mImgFriend;
private ImageButton mImgContact;
private ImageButton mImgSetting;
```

② 初始化

调用 3 个自定义方法，方法在下面的步骤中给出。

```
initViews();//初始化控件
initEvents();//初始化事件
initDatas();//初始化数据
```

③ 初始化控件

使用 findViewById 获取每个控件。

```
private void initViews() {
    mViewPager = (ViewPager) findViewById(R.id.id_viewpager);

    mTabWechat = (LinearLayout) findViewById(R.id.id_tab_wechat);
    mTabFriend = (LinearLayout) findViewById(R.id.id_tab_friend);
    mTabContact = (LinearLayout) findViewById(R.id.id_tab_contact);
    mTabSetting = (LinearLayout) findViewById(R.id.id_tab_setting);

    mImgWechat = (ImageButton) findViewById(R.id.id_tab_wechat_img);
    mImgFriend = (ImageButton) findViewById(R.id.id_tab_friend_img);
    mImgContact = (ImageButton) findViewById(R.id.id_tab_contact_img);
    mImgSetting = (ImageButton) findViewById(R.id.id_tab_setting_img);

}
```

④ 初始化事件

InitEvents 使用 setOnClickListener 为每个 ImageButton 控件设置监听事件 onClickListener。

onClickListener 是（5）中定义的单击事件。

```
private void initEvents() {
    //设置 4 个 Tab 的单击事件
    mTabWechat.setOnClickListener(onClickListener);
    mTabFriend.setOnClickListener(onClickListener);
    mTabContact.setOnClickListener(onClickListener);
    mTabSetting.setOnClickListener(onClickListener);
}
```

⑤ 初始化数据

首先定义 ArrayList 类型的 mFragments，添加 4 个 Fragment 对象；然后初始化适配器，重载 getItem、getCount 方法；最后设置 ViewPager 的切换监听 addOnPageChangeListener，重载 onPageSelected 方法。在 onPageSelected 中调用 selectTab 和 resetImg 方法进行导航按钮图片的处理。

```
private void initDatas() {
    mFragments = new ArrayList<>();
    //将 4 个 Fragment 加入集合中
    mFragments.add(new WechatFragment());
    mFragments.add(new FriendFragment());
    mFragments.add(new ContactFragment());
    mFragments.add(new SettingFragment());

    //初始化适配器
    mAdapter = new FragmentPagerAdapter(getSupportFragmentManager()) {
        @Override
        public Fragment getItem(int position) {//从集合中获取对应位置的 Fragment
            return mFragments.get(position);
        }

        @Override
        public int getCount() {//获取集合中 Fragment 的总数
            return mFragments.size();
        }

    };
    //设置 ViewPager 的适配器
    mViewPager.setAdapter(mAdapter);
    //设置 ViewPager 的切换监听
    mViewPager.addOnPageChangeListener(new ViewPager.OnPageChangeListener() {
        @Override
        //页面滚动事件
        public void onPageScrolled(int position, float positionOffset, int positionOffsetPixels) {

        }

        //页面选中事件
```

```
        @Override
        public void onPageSelected(int position) {
            //设置 position 对应的集合中的 Fragment
            mViewPager.setCurrentItem(position);
            resetImgs();
            selectTab(position);
        }

        @Override
        //页面滚动状态改变事件
        public void onPageScrollStateChanged(int state) {

        }
    });
}

    private void selectTab(int i) {
    //根据单击的 Tab 设置对应的 ImageButton 为绿色
    switch (i) {
        case 0:
            mImgWechat.setImageResource(R.mipmap.tab_weixin_pressed);
            break;
        case 1:
            mImgFriend.setImageResource(R.mipmap.tab_find_frd_pressed);
            break;
        case 2:
            mImgContact.setImageResource(R.mipmap.tab_address_pressed);
            break;
        case 3:
            mImgSetting.setImageResource(R.mipmap.tab_settings_pressed);
            break;
    }
    //设置当前单击的 Tab 所对应的页面
    mViewPager.setCurrentItem(i);
}

//将 4 个 ImageButton 设置为灰色
private void resetImgs() {
    mImgWechat.setImageResource(R.mipmap.tab_weixin_normal);
    mImgFriend.setImageResource(R.mipmap.tab_find_frd_normal);
    mImgContact.setImageResource(R.mipmap.tab_address_normal);
    mImgSetting.setImageResource(R.mipmap.tab_settings_normal);
}
```

（8）运行该项目，查看结果。

5.2 Toolbar 和对话框

5.2.1 Toolbar

Actionbar 是在 Android3.0 推出的一个标识应用程序和用户位置的窗口功能，并且给用户提供操作和导航模式。在大多数情况下，当开发者需要突出展现用户行为或全局导航的 Activity 中使用 Actionbar，因为 Actionbar 能够使应用程序为用户提供一致的界面，并且系统能够很好地根据不同的屏幕配置来适应操作栏的外观。开发者能够用 Actionbar 的对象的 API 来控制操作栏的行为和可见性。

Toolbar 是在 Android 5.0 开始推出的一个 Material Design 风格的导航控件，Google 公司非常推荐大家使用 Toolbar 来作为 Android 客户端的导航栏，以此来取代 Actionbar 。

与 Actionbar 相比，Toolbar 明显要灵活得多。它不像 Actionbar 那样，一定要固定在 Activity 的顶部，而是可以放到界面的任意位置。除此之外，在设计 Toolbar 时，Google 也留给了开发者很多可定制修改的余地，这些可定制修改的属性在 API 文档中都有详细介绍，如：

- 设置导航栏图标；
- 设置 App 的 logo；
- 支持设置标题和子标题；
- 支持添加一个或多个的自定义控件；
- 支持 Action Menu。

【例 5-3】 下面的例子为 Toolbar 添加搜索和分享图标，效果如图 5-15 的 Toolbar 所示。

实现步骤如下：

（1）添加图标

在 res/menu/main.xml 中添加搜索和分享，设置 title 和 showAsAction，其中 ifRoom 是有空间则显示，否则隐藏。

```
<item
    android:id="@+id/ab_search"
    android:orderInCategory="80"
    android:title="action_search"
    app:actionViewClass="Android.support.v7.widget.SearchView"
    app:showAsAction="ifRoom"/>
<item
    android:id="@+id/action_share"
    android:orderInCategory="90"
    android:title="action_share"
    app:actionProviderClass="Android.support.v7.widget.ShareActionProvider"
    app:showAsAction="ifRoom"/>
```

全部布局代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:Android="http://schemas.Android.com/apk/res/Android"
    xmlns:app="http://schemas.Android.com/apk/res-auto">
<item
```

```
    android:id="@+id/ab_search"
    android:orderInCategory="80"
    android:title="action_search"
    app:actionViewClass="Android.support.v7.widget.SearchView"
    app:showAsAction="ifRoom"/>
<item
    android:id="@+id/action_share"
    android:orderInCategory="90"
    android:title="action_share"
    app:actionProviderClass="Android.support.v7.widget.ShareActionProvider"
    app:showAsAction="ifRoom"/>
<item
    android:id="@+id/action_settings"
    android:orderInCategory="100"
    android:title="action_settings"
    app:showAsAction="never"/>
</menu>
```

（2）添加菜单事件

重载 onOptionsItemSelected 方法，使用 getItemId 获取菜单控件，然后使用 switch，为不同的菜单控件设置不同的事件。这里使用 Toast 显示简单的信息。

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();
    switch (id) {
        case R.id.action_settings:
            Toast.makeText(MainActivity.this, "action_settings", Toast.LENGTH_LONG).show();
            break;
        case R.id.action_share:
            Toast.makeText(MainActivity.this, "action_share", Toast.LENGTH_LONG).show();
            break;
        default:
            break;
    }
    //return super.onOptionsItemSelected(item);
    return true;
}
```

（3）运行该项目，查看结果。

5.2.2 DialogFragment

在 Android 应用开发中，程序与用户交互的方式会直接影响到用户的使用体验，而对话框又是与用户交互必不可少的部分。我们经常会需要在界面上弹出一个对话框，让用户单击

对话框的某个按钮、选项，或者是输入一些文本，从而知道用户做了什么操作，或是下达了什么指令。

DialogFragment 在 Android 3.0 时被引入。它是一种特殊的 Fragment，用于在 Activity 的内容之上展示一个模态的对话框。典型地用于展示警告框、输入框、确认框等。在 DialogFragment 产生之前，创建对话框一般采用 AlertDialog 和 Dialog。Google 公司不推荐直接使用 Dialog 创建对话框。

使用 DialogFragment 来管理对话框，当旋转屏幕和按下后退键时可以更好地管理其生命周期，它和 Fragment 有着基本一致的生命周期。且 DialogFragment 也允许开发者把 Dialog 作为内嵌的组件进行重用，类似 Fragment（可以在大屏幕和小屏幕显示出不同的效果）。

【例 5-4】 下面用弹出登录对话框为例，展示如何使用 DialogFragment，登录对话框效果如图 5-11 所示，对话框响应如图 5-12 所示。

图 5-11　登录对话框

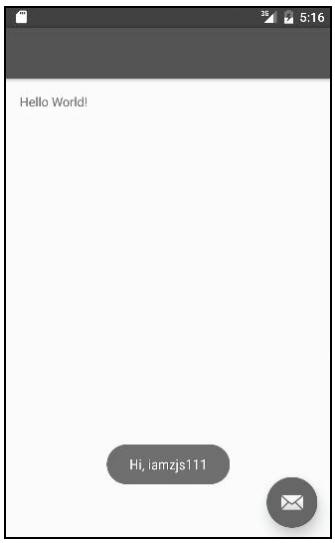

图 5-12　对话框响应

实现步骤如下：

（1）创建新项目：项目名称 DialogDemo。

（2）创建 LoginFragment

布局中使用 TextView 和 EditText 控件。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    android:id="@+id/login_dialog"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical">
    <TextView
        android:layout_width="300dp"
        android:layout_height="50dp"
        android:text="登录"
        android:layout_marginTop="15dp"
```