第3章 Activity、Fragment 和 Intent

本章要点

- Android 应用程序的生命周期为从启动到终止的全过程，由系统进行调度和控制。
- Android 应用的基本组件有 Activity（活动）、Service（服务）、BroadcastReceiver（广播接收器）、ContentProvider（数据提供器）、Intent（意图）等。
- Activity 的生命周期中存在启动状态、运行状态、暂停状态、停止状态、销毁状态 5 种状态。
- Activity 的生命周期可分为完全生命周期、可视生命周期和活动生命周期，每种生命周期中包含不同的回调方法。
- Fragment 不能独立存在，它必须嵌入到 Activity 中，Fragment 的生命周期被其所属的 Activity 生命周期控制。
- Intent 用于启动 Activity、Service 或者 BroadcastReceiver 等组件，并且是组件之间通信的重要媒介。
- Intent 对象包含 Component、Action、Data、Category、Extra 及 Flag 等 6 种属性。

Android 应用程序由 Activity、Service、BroadcastReceiver、ContentProvider 等组件构成，Activity 组件为用户提供可视化用户界面，它是 Android 应用程序中最常见、最基本的组件。本章介绍 Android 应用程序的生命周期，Android 应用的基本组件，Activity 的运行状态和生命周期，Fragment 的使用，Intent 的组成、调用和传递数据等内容。

3.1 Android 应用程序的生命周期

Android 应用程序生命周期指从启动到终止的全过程，应用程序的生命周期是由 Android 系统进行调度和控制，而不是由应用程序直接控制的。

Android 应用程序组件有其生命周期，指从创建到销毁的全过程，组件会在可见、不可见、活动、不活动等状态中不断变化，Activity 组件是 Android 应用生命周期的重要部分之一。

1. 进程

进程（Process）是程序的一次执行，进程由程序、数据和进程控制块构成，进程是一个可拥有资源的独立实体，又是一个可以独立调度的基本单位。

进程的执行过程包括创建（New）、就绪（Ready）、执行（Running）、阻塞（Blocked）、挂起（Suspend）、终止（Terminated）等状态。

在 Android 操作系统中，进程是应用程序的具体实现。组件运行的进程由

Androidmanifest 文件控制。组件标签< activity >、< service >、< receiver >、< provider >等包含一个 process 属性，这个属性可以设置组件运行的进程。< application >标签也包含 process 属性，用来设置程序中所有组件的默认进程。所有的组件在默认进程的主线程中实例化，系统对这些组件的调用从主线程中分离。

每个 Android 应用程序的进程都是由 Android 进行时独立管理的，每个 Android 的应用程序在自己的进程中运行。

Android 系统往往运行在资源受限的平台上，资源管理非常重要，因此，由 Android 系统管理资源。

Android 系统的进程优先级从高到低分别为：前台进程、可见进程、服务进程、后台进程、空进程。前台进程为高优先级、可见进程、服务进程为中优先级、后台进程、空进程为低优先级。

1）前台进程

前台进程是 Android 系统中最重要的进程，它是与用户进行交互的进程。

前台进程包括：

- 该进程拥有一个正在与用户交互的 Activity()(其 onResume()方法被调用)。
- 该进程拥有一个绑定到正与用户交互的 Activity 上的 Service。
- 该进程拥有一个前台运行并调用了 startForeground()方法的 Service。
- 该进程拥有一个正在执行的回调方法(如 onStart()、onCreate()、onDestroy())的 Service。
- 该进程拥有一个正在执行 onReceive()方法的 BroadcastReceiver 对象。

通常在任何时间点，只有很少前台进程存在。当出现资源不足时，也会“杀死”部分前台进程。

2）可见进程

可见进程是用户能够在屏幕上看见，但不能与用户进行交互，不响应界面事件的进程。

可见进程包括：

- 该进程拥有一个不在前台但为用户可见的 Activity(如调用了方法 onPause()之后)。
- 一个可见的 Activity 所绑定的 Service。

当出现无法维持前台进程运行等情况时，才会清除可见进程。

3）服务进程

包含已启动服务的进程称为服务进程。服务进程不可见，不与用户直接交互，但能在后台运行，提供用户需要的功能。

服务进程包括：

- 一个由 startService()方法启动的 Service。
- 支持正在处理的不需要可见界面运行的 Service。

当系统内存不足，不能维持前台进程和可见进程的运行时，才会清除服务进程。

4）后台进程

不包含任何已启动服务，而且没有用户可见的 Activity 的进程，即为后台进程。

后台进程包括：

- 该进程拥有一个当前不可见的 Activity(已调用了 onStop()方法)。
- 目前没有服务的 Service。

一般情况下，存在较多的后台进程，当系统资源紧张时，Android 将会使用 LRU 模式来清除最近最少使用的后台进程。

5) 空进程

空进程是不包含任何 Activity 组件，对用户没有任何作用的进程。

为了改善系统的整体性能，Android 通常在内存中保留生命周期结束了的应用，当系统资源紧张时，空进程首先被清除。

2. 线程

线程(Thread)是进程中的一个实体，是被系统独立调度的基本单位。线程基本上不拥有系统资源，只有一些在运行中必不可少的资源(如程序计数器、一组寄存器和栈)，但它可共享所属进程的全部资源。

引入线程的目的是为了减少程序并发执行时所付出的时空开销，使操作系统具有更好的并发性，提高系统运行的效率。线程具有许多传统进程所具有的特征，又称为轻量级进程(Light-Weight Process)，而把传统的进程称为重量级进程(Heavy-Weight Process)。

每个进程有一到多个线程运行在其中。进程中的所有组件都在 UI 线程中实例化，以保证应用程序是单线程的，除非应用程序又创建了自己的线程，例如网络连接、下载或其他费时操作。线程通过 Java 的 Thread 类创建。

3.2 Android 应用的基本组件

Android 应用程序由组件组成，并通过项目的 AndroidManifest.xml 将它们绑定在一起。

Android 应用中常用的基本组件有 Activity(活动)、Service(服务)、BroadcastReceiver(广播接收器)、ContentProvider(数据提供器)、Intent(意图)等，下面分别进行介绍。

3.2.1 Activity

Activity 用于提供可视化用户界面并与用户交互，它是最常用的组件，Activity 是应用程序的显示层，显示可视化的用户界面，并接收与用户交互所产生的界面事件。

一个 Activity 展现一个可视化用户界面，如果需要多个可视化用户界面，该 Android 应用会包含多个 Activity，尽管多个 Activity 在一起工作，但每个 Activity 是相对独立的，每个 Activity 都继承自 android.app.Activity 类。

例如，第一个 Android 应用项目 FirstAndroidApplication 中 MainActivity.java 的代码如下：

```
...
import android.app.Activity;
import android.view.View;
import android.view.ViewGroup;
```

```
…
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
…
    }
…
```

Activity 的显示内容由 View(视图)组件的对象提供,并定义在 res/layout 下的 XML 文件中,View 组件的对象包括文本框、多选框、单选框、按钮、菜单等。

通过 Activity 将指定的 View 显示出来,调用 Activity 的 setContentView()方法,例如上面代码中的 setContentView(R.layout.activity_main)方法。

3.2.2 Service

Service 是一个常用组件,需要继承 Service 类。Service 一般用于没有用户界面,又需要长时间在后台运行的应用,例如播放背景音乐或在网络上获取数据。

Service 与 Activity 有以下区别:Service 通常位于后台运行,它一般不需要与用户交互,也没有用户界面。Service 一般由 Activity 启动,但拥有自己独立的生命周期。Service 具有较长的生命周期,当启动它的 Activity 生命周期结束,Service 仍能继续运行,直到自己的生命周期结束。

Service 有两种启动方式:

(1) 使用 startService 方式启动。

(2) 使用 bindService 方式启动。

3.2.3 BroadcastReceiver

BroadcastReceiver 是另一个常用组件,用来接收广播消息,不包含任何用户界面,其监听的事件源是其他组件。

使用 BroadcastReceiver 组件接收广播信息,需要继承 BroadcastReceiver 类并重写 onReceive 方法。当其他组件通过 sendBroadcast()、sendStickyBroadcast()或 sendOrderBroadcast()方法发送广播消息时,如果通过 IntentFilter 过滤的 BroadcastReceiver 感兴趣,就会被接收。

BroadcastReceiver 注册方式有两种:

(1) 在 AndroidManifest.xml 中,在< receiver > </receiver >标签中设置。

(2) 在 Java 代码中,通过 Context.registReceiver()方法注册。

3.2.4 ContentProvider

ContentProvider 组件是 Android 系统提供的一种标准的共享数据的机制,用来管理和共享应用程序的数据存储。例如开发一个发送短信的程序,需要多个应用程序之间共享和

交换数据。

一般的使用方法是：一个应用程序使用 ContentProvider 暴露自己的数据，另一个应用程序使用 ContentResolver 访问数据。

3.2.5 Intent

Intent 是不同组件间通信的载体，是连接各个组件的桥梁。Intent 不仅可以用到不同组件之间的交互，还可以用到不同应用程序之间的交互。

Activity、Service、BroadcastReceiver 组件之间的通信都使用 Intent 作为通信的载体，但各个组件使用 Intent 的机制不同。

(1) 当需要启动一个 Activity 时，可调用 Context. startActivity( ) 或 Context. startActivityForResult()方法，这两个方法中的 Intent 参数封装了需要启动的目标 Activity 的信息。

(2) 当需要启动一个 Service 时，可调用 Context. startService()或 Context. bindService()方法，这两个方法中的 Intent 参数封装了需要启动的目标 Service 的信息。

(3) 当需要触发一个 BroadcastReceiver 时，可调用 sendBroadcast()、sendStickyBroadcast()或 sendOrderedBroadcast( )方法，这三个方法中的 Intent 参数封装了需要触发的目标 BroadcastReceiver 的信息。

3.3 Activity 的运行状态和生命周期

Activity 生命周期指 Activity 从启动到销毁的过程，下面介绍 Activity 的运行状态和生命周期。

3.3.1 Activity 的运行状态

Activity 的生命周期中存在五种状态：启动状态、运行状态、暂停状态、停止状态、销毁状态。

(1) 启动状态(Starting)：Activity 在屏幕的前台。

(2) 运行状态(Running)：Activity 可见，获得焦点，可与用户进行交互。Activity 启动后，随即进入运行状态。

(3) 暂停状态(Paused)：Activity 失去焦点，但仍可见，依然保持活力，但在系统内存极低时将被杀掉。

(4) 停止状态(Stopped)：Activity 失去焦点，不可见，此时 Activity 被另一个 Activity 完全覆盖，系统可以随时将其释放。

(5) 销毁状态(Destroyed)：系统将 Activity 从内存中删除，有两种方式，一种是要求该 Activity 结束，一种是直接被杀掉。

3.3.2 Activity 的生命周期

本节介绍 Activity 生命周期和回调方法。

1. Android 的回调机制

一个通用的程序架构具有完成整个应用的流程和功能，但在某个特定点需要一段业务相关的代码进行处理，例如 Activity 的 onCreate()、onPause()和 onStop()等回调方法，开发人员可以选择性地重写这些方法，通用的程序架构就会回调该方法进行相关的业务处理。

2. Activity 的回调方法

Activity 的回调方法有 onCreate()、onStart()、onResume()、onPause()、onStop()、onRestart()、onDestroy()、onSaveInstanceState()、onRestoreInstanceState()等，下面分别介绍。

(1) onCreate(Bundle)：

创建 Activity 时被回调，该方法只会被调用一次。如果 Activity 之前是被冻结状态，其状态由 Bundle 提供，接收参数为 null 或由 onSaveInstanceState()方法保存的状态信息。其后调用 onStart()或 onRestart()方法。

(2) onStart()：

启动 Activity 时被回调，当 Activity 对用户即将可见时被调用。

(3) onResume()：

恢复 Activity 时被回调，当 Activity 可以开始与用户进行交互之前被调用。

(4) onPause()：

暂停 Activity 时被回调，活动将进入后台时会运行该方法，当系统将要启动另一个 Activity 之前被调用。

(5) onStop()：

停止 Activity 时被回调，当 Activity 不再为用户可见时被调用。

(6) onRestart()：

重新启动 Activity 时被回调，在再次启动之前被调用。

(7) onDestroy()：

销毁 Activity 时被回调，在 Activity 销毁前被调用。

(8) onSaveInstanceState(Bundle)：

回调该方法让活动可以保存每个实例的状态。

(9) onRestoreInstanceState(Bundle)：

回调 onSaveInstanceState()方法保存的状态来重新初始化某个活动时调用该方法，其后紧跟的方法是 onResume()。

3. Activity 的生命周期

Activity 的生命周期如图 3.1 所示。

Activity 的生命周期可分为完全生命周期、可视生命周期和活动生命周期，每种生命周期中包含不同的回调方法，如图 3.2 所示。

1) 完全生命周期

完全生命周期是从 Activity 创建到销毁的全部过程，从调用 onCreate()开始到 onDestroy()结束。开发人员通常在 onCreate()中初始化 Activity 所能使用的全局资源和状态，并在 onDestroy()中释放这些资源。在一些极端的情况下，Android 系统不调用 onDestroy()，直接终止进程。

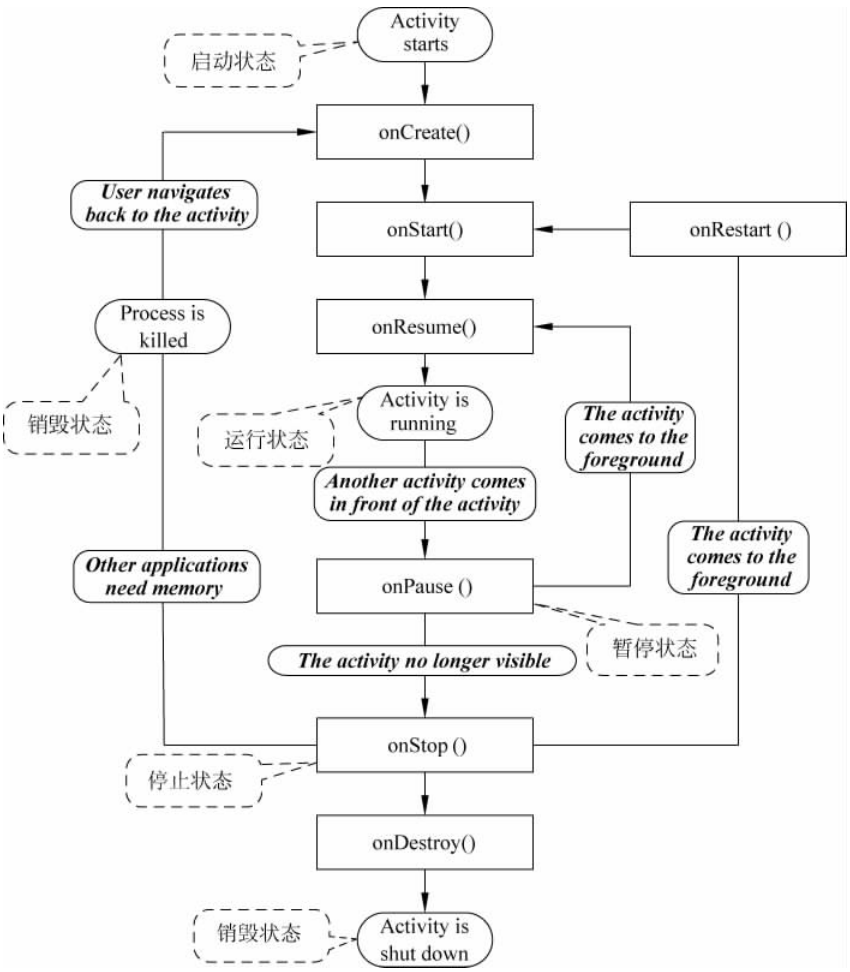

图 3.1　Activity 生命周期

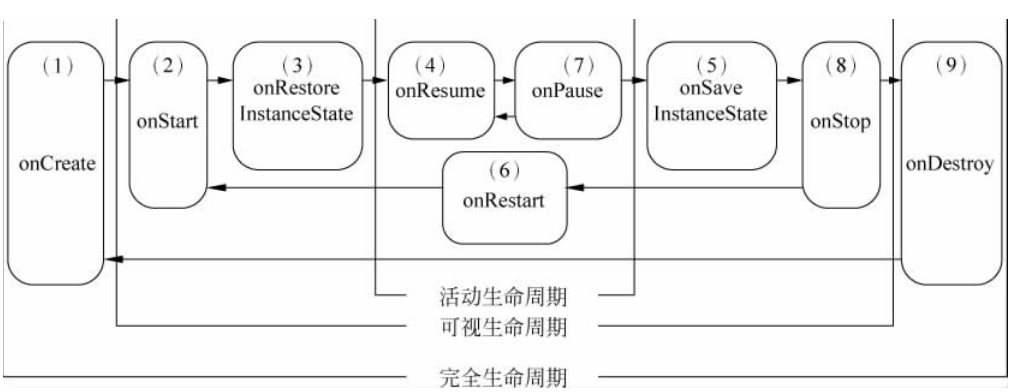

图 3.2　Activity 生命周期分类

2）可视生命周期

可视生命周期是 Activity 在界面上从可见到不可见的过程，从调用 onStart()开始到 onStop()结束。onStart()一般用来初始化或启动与更新界面相关的资源，onStop()一般用来暂停或停止一切与更新用户界面相关的线程、计时器和服务。onRestart()在 onSart()前被调用，用来在 Activity 从不可见变为可见的过程中，进行一些特定的处理过程。onStart()和 onStop()会被多次调用，使 Activity 不断地从可见到不可见，再从不可见到可见。

3）活动生命周期

活动生命周期是 Activity 在屏幕的最上层，并能够与用户交互的阶段，从调用 onResume()开始到 onPause()结束。在 Activity 的状态变换过程中 onResume()和 onPause()经常被调用，因此这两个回调方法中应使用简单、高效的轻量级代码。

【例 3.1】 为了更好地理解 Activity 生命周期和 Android 的回调机制，通过 LifeCycle 示例的演示进行说明和分析。

【解题思路】

通过在生命周期回调方法中添加“日志点”的方法进行调试，程序的运行结果将会显示在 LogCat 中。为了使显示结果易于观察和分析，在 LogCat 中设置过滤器 LifeTest，过滤方法选择 by Log Tag，过滤关键字为 ActivityLifeCycle。

【开发步骤和程序分析】

（1）在 Eclipse 中创建一个 LifeCycle 应用项目，包名为 com.application.lifecycle。

（2）在 src/com.application.lifecycle 包下的 lifecycle.java 文件中，加载 main.xml 布局文件，在生命周期回调方法中添加“日志点”。

在该文件中编辑代码如下：

```
1  package com.application.lifecycle;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.util.Log;
6  import android.view.View;
7  import android.widget.Button;
8
9
10     public class LifeCycle extends Activity {
11     private static String TAG = "ActivityLifeCycle";
12
13     @Override                    //生命周期开始，创建 Activity
14     public void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.main);
17         Log.i(TAG, " -- (1) onCreate()");
18
19         Button button = (Button)findViewById(R.id.btn_finish);
20         button.setOnClickListener(new View.OnClickListener() {
```

```
21            public void onClick(View view) {
22                finish();
23            }
24        });
25    }
26
27    @Override                    //启动 Activity
28    public void onStart() {
29        super.onStart();
30        Log.i(TAG, " -- (2) onStart()");
31    }
32
33    @Override                    //重新初始化 Activity
34    public void onRestoreInstanceState(Bundle savedInstanceState) {
35        super.onRestoreInstanceState(savedInstanceState);
36        Log.i(TAG, " -- (3) onRestoreInstanceState()");
37    }
38
39    @Override                    //恢复 Activity
40    public void onResume() {
41        super.onResume();
42        Log.i(TAG, " -- (4) onResume()");
43    }
44
45    @Override                    //让 Activity 保存实例的状态
46    public void onSaveInstanceState(Bundle savedInstanceState) {
47        super.onSaveInstanceState(savedInstanceState);
48        Log.i(TAG, " -- (5) onSaveInstanceState()");
49    }
50
51    @Override                    //重新启动 Activity
52    public void onRestart() {
53        super.onRestart();
54        Log.i(TAG, " -- (6) onRestart()");
55   }
56
57    @Override                    //暂停 Activity
58    public void onPause() {
59        super.onPause();
60        Log.i(TAG, " -- (7) onPause()");
61    }
62
63    @Override                    //停止 Activity
64    public void onStop() {
65        super.onStop();
```

```
66          Log.i(TAG, " -- (8) onStop()");
67      }
68
69      @Override                    //生命周期结束,销毁 Activity
70      public void onDestroy() {
71          super.onDestroy();
72          Log.i(TAG, " -- (9) onDestroy()");
73      }
74  }
```

【运行结果】

在 Eclipse 中启动模拟器,然后运行项目 LifeCycle,运行结果如图 3.3 所示。

图 3.3　LifeCycle 应用项目界面

(1) 演示完全生命周期。

启动项目 LifeCycle 后,单击项目界面中的"结束"按钮,LogCat 输出结果如图 3.4 所示。

图 3.4　Activity 完全生命周期回调方法次序

由图 3.4 可看出,回调方法的调用顺序如下:(1) onCreate→(2) onStart→(4)onResume→(7)onPause→(8)onStop→(9)onDestroy。

启动 Activity 时,系统首先调用 onCreate()分配资源,再调用 onStart()将 Activity 显示在屏幕上,然后调用 onResume()获取焦点,能够与用户进行交互,此时用户能够正常使用这个 Android 项目。

当用户单击"结束"按钮时,系统相继调用 onPause()、onStop()和 onDestroy(),释放资源并销毁进程。

(2) 演示可视生命周期。

正常启动 LifeCycle,再通过“Call 键”(拨号键)启动内置的拨号程序,然后通过“Back 键”(回退键)退出拨号程序,LifeCycle 重新显示在屏幕上,LogCat 输出结果如图 3.5 所示。

图 3.5 Activity 可视生命周期回调方法次序

由图 3.5 可看出,回调方法的调用顺序:(1)onCreate→(2)onStart→(4)onResume→(7) onPause→(5) onSaveInstanceState→(8) onStop→(6) onRestart→(2) onStart→(4)onResume。

Activity 启动时,回调方法的调用顺序仍为(1)onCreate→(2)onStart→(4)onResume。

当按下“Call 键”(拨号键)时,内置拨号程序被启动,原有的 Activity 被覆盖,系统首先调用 onPause(),再调用 onSaveInstanceState()保存 Activity 状态,最后调用 onStop()停止对不可见的 Activity 的更新。

当按下“Back 键”(回退键)时,退出拨号程序,系统调用 onRestart()恢复界面上需要更新的信息,再调用 onStart()和 onResume()重新显示 Activity,能够与用户进行交互。

3.4 Fragment 的使用

Fragment(片段)以 Activity 界面的一个组成部分出现。

3.4.1 Fragment 的生命周期

Fragment 有自己的生命周期,但它的生命周期受其所在的 Activity 生命周期控制,Fragment 不能独立存在,它必须嵌入到 Activity 中。当 Activity 暂停时,它拥有的所有的 Fragment 都暂停了;当 Activity 销毁时,它拥有的所有 Fragment 都被销毁;当 Activity 处于活动状态时(在 onResume()之后,onPause()之前),用户可以通过方法操作每个 Fragment。Fragment 的生命周期如图 3.6 所示。

Fragment 有以下特点:

- Fragment 总是作为 Activity 界面的组成部分。
- Fragment 有自己的生命周期,但它的生命周期被其所属的 Activity 生命周期控制。
- 在 Activity 运行过程中,可动态地添加、删除和替换 Fragment。

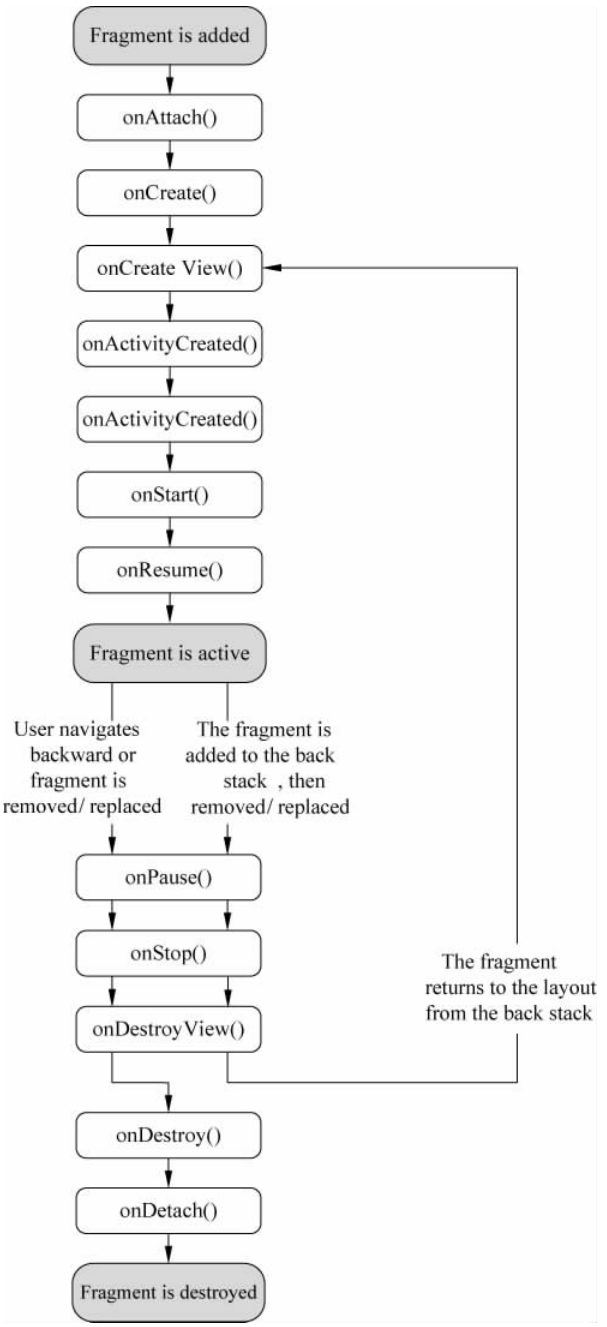

图 3.6　Fragment 生命周期

- 一个 Activity 中可同时出现多个 Fragment，一个 Fragment 也可在多个 Activity 中使用。
- Fragment 可以响应自己的输入事件。

1. Fragment 对象跟用户交互时需要回调的方法

1）onAttach(Activity)

当 Fragment 对象跟 Activity 关联时，调用该方法。

2）onCreate(Bundle)

当 Fragment 对象初始创建时，调用该方法。

3）onCreateView(LayoutInflater, ViewGroup, Bundle)

该方法用于创建和返回跟 Fragment 关联的 View 对象。

4）onActivityCreate(Bundle)

该方法会告诉 Fragment 对象，它所依附的 Activity 对象已经完成了 Activity. onCreate() 方法的执行。

5）onStart()

该方法会让 Fragment 对象显示给用户（在包含该 Fragment 对象的 Activity 被启动后）。

6）onResume()

该方法会让 Fragment 对象跟用户交互（在包含该 Fragment 对象的 Activity 被启动恢复后）。

2. Fragment 对象不再使用时需要回调的方法

1）onPause()

当 Fragment 对象所依附的 Activity 对象被挂起，或者在 Activity 中正在执行一个修改 Fragment 对象的操作，而导致 Fragment 对象不再跟用户交互时，系统会调用该方法。

2）onStop()

当 Fragment 对象所依附的 Activity 对象被终止，或者在 Activity 中正在执行一个修改 Fragment 对象的操作，而导致 Fragment 对象不再显示给用户时，系统会调用该方法。

3）onDestroyView()

该方法用于清除跟 Fragment 中的 View 对象关联的资源。

4）onDestroy()

当 Fragment 对象的状态被最终清理完成之后，要调用该方法。

5）onDetach()

当 Fragment 对象不再跟它依附的 Activity 关联的时候，该方法会立即被调用。

3.4.2 Fragment 的应用

Fragment 在应用中是一个模块化和可重用的组件，下面介绍向 Activity 中添加 Fragment 的方法、Fragment 常用的类和方法、Fragment 的子类等内容。

1. 向 Activity 中添加 Fragment 的方法

向 Activity 中添加 Fragment 有两种方法：一种是直接在布局文件中添加，另一种是当 Activity 运行时添加。

1）直接在布局文件中添加 Fragment

直接在布局文件中添加 Fragment，可以使用< fragment >标记实现，将 Fragment 作为

Activity 整个布局的一部分。

2）当 Activity 运行时添加 Fragment

当 Activity 运行时，也可以将 Fragment 添加到 Activity 的布局中，实现方法是获取一个 FragmentTransaction 的实例，然后使用 add()方法添加一个 Fragment，再调用 commit()方法提交事务。

2. Fragment 常用的类

（1）android.app.Fragment：用于定义 Fragment。

（2）android.app.FragmentManager：用于在 Activity 中操作 Fragment。通过调用 Activity 的 getFragmentManager()方法可以取得 FragmentManager 的实例。

（3）android.app.FragmentTransaction：对 Fragment 进行添加、移除、替换及执行其他动作。

在使用 FragmentTransaction 的方法前，首先需要取得 FragmentManager 的实例，再利用 FragmentManage 的 beginTransaction()方法开启一个事务，获取一个 FragmentTransaction 对象。

3. FragmentTransaction 的方法

（1）FragmentTransaction.add()：往 Activity 中添加一个 Fragment。

（2）FragmentTransaction.remove()：从 Activity 中移除一个 Fragment，如果被移除的 Fragment 没有添加到回退栈，这个 Fragment 实例将会被销毁。回退栈（back stack）由 Activity 管理，允许用户通过按下 Back 按键返回到前一个 Fragment 状态。

（3）FragmentTransaction.replace()：使用另一个 Fragment 替换当前的 Fragment。

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

（5）FragmentTransaction.show()：显示之前隐藏的 Fragment。

（6）FragmentTransaction.detach()：会将 View 从 UI 中移除，和 remove()不同，此时 Fragment 的状态依然由 FragmentManager 维护。

（7）FragmentTransaction.attach()：重建 View 视图，附加到 UI 上并显示。

（8）FragmentTransaction.commit()：提交一个事务。在一个事务开启到提交可以进行多个 Fragment 的添加、移除和替换等操作。需要注意，FragmentTransaction 的 commit()方法一定要在 Activity.onSaveInstance()方法之前调用。

4. Fragment 的子类

Fragment 有以下子类，可实现不同类型的 UI 面板。

（1）DialogFragment 类：显示一个浮动的对话框。

（2）ListFragment 类：显示一个由 Adapter 管理项目的列表，类似于 ListActivity，它提供一些方法来管理一个 ListView，使用 onListItemClick 回调来处理单击事件。

（3）PreferenceFragment 类：显示一个 Preference 对象的层次结构的列表，类似于 PreferenceActivity。

Android 引入 Fragment 的初衷是为了适应大屏幕的平板电脑，下面的例题介绍使用 Fragment 模拟平板电脑的显示。

【例 3.2】 模拟平板电脑划分为左右两个片段，分别显示 Java 概念列表和定义。

【解题思路】

在 Activity 界面左部的 Fragment 显示 Java 概念列表项，当单击某一列表项时，右部 Fragment 进行动态更新，显示对应的 Java 概念定义。

【开发步骤和程序分析】

（1）在 Eclipse 中创建一个 ActivityFragment 应用项目，包名为 com. application . activityfragment。

（2）设计布局。

布局文件为 activitytwopanes. xml 和 fragmentdetail. xml。

在 res/layout 目录下的 activitytwopanes. xml 文件中，左部添加一个 Fragment 元素，右部添加一个 FrameLayout 容器。

该文件编辑代码如下。

```
1  <?xml version = "1.0" encoding = "utf - 8"?>
2
3  <! -- 定义一个水平的 LinearLayout,并指定使用中等分隔条 -->
4  < LinearLayout
5      xmlns:android = "http://schemas.android.com/apk/res/android"
6      android:orientation = "horizontal"
7      android:layout_width = "match_parent"
8      android:layout_height = "match_parent"
9      android:layout_marginLeft = "16dp"
10     android:layout_marginRight = "16dp"
11     android:divider = "?android:attr/dividerHorizontal"
12     android:showDividers = "middle">
13
14     <! -- 添加一个 Fragment,位于左部 -->
15     < fragment
16        android:name = "com.application.activityfragment.FragmentListConcept"
17        android:id = "@ + id/concept_list "
18        android:layout_width = "0dp"
19        android:layout_height = "match_parent"
20        android:layout_weight = "1" />
21
22     <! -- 添加一个 FrameLayout 容器,位于右部 -->
23     < FrameLayout
24        android:id = "@ + id/concept_detail_container"
25        android:layout_width = "0dp"
26        android:layout_height = "match_parent"
27        android:layout_weight = "3" />
28 </LinearLayout >
```

在 res/layout 目录下的 fragmentdetail. xml 文件中包含两个文本框，上边的文本框用于显示概念名称，下边的文本框用于显示概念内容。

该文件编辑代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
<! -- 定义一个垂直分布的线性布局 -- >
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:orientation = "vertical">
    <! -- 定义一个 TextView 来显示概念名称,位于上边 -- >
    < TextView
        style = "?android:attr/textAppearanceLarge"
        android:id = "@ + id/concept_title"
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content"
        android:padding = "16dp"/>

    <! -- 定义一个 TextView 来显示概念内容,位于下边 -- >
    < TextView
        style = "?android:attr/textAppearanceMedium"
        android:id = "@ + id/concept_desc"
        android:layout_width = "match_parent"
        android:layout_height = "match_parent"
        android:padding = "16dp"/>
</LinearLayout >
```

(3) 在 com.application.activityfragment 包下的 ActivityFragmentConcept.java 文件中,加载 activitytwopane.xml 布局文件并实现 Callbacks 接口,该 Activity 左部的 Fragment 显示 Java 概念列表项,右部 Fragment 显示 Java 概念定义并进行动态更新。

该文件编辑代码如下:

```
package com.application.activityfragment;

import com.application.activityfragment.R;
import android.app.Activity;
mport android.os.Bundle;

//定义一个类 ActivityFragmentConcept 继承 Activity 类, 且实现 Callbacks 接口
public class ActivityFragmentConcept extends Activity implements
        FragmentListConcept.Callbacks
   {

    //重写 onCreate()方法
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        //加载/res/layout 目录下的 activitytwopanes.xml 布局文件
```

```
18              setContentView(R.layout.activitytwopanes);
19          }
20          //重写 onItemSelected()方法, 实现 Callbacks 接口必须实现的方法
21          @Override
22          public void onItemSelected(Integer id)
23          {
24              //创建 Bundle,准备向 Fragment 传入参数
25              Bundle arguments = new Bundle();
26              arguments.putInt(FragmentDetailConcept.ITEM_ID, id);
27              //创建 FragmentDetailConcept 对象
28              FragmentDetailConcept fragment = new FragmentDetailConcept();
29              //向 Fragment 传入参数
30              fragment.setArguments(arguments);
31              //使用 fragment 替换 concept_detail_container 容器当前显示的 Fragment
32              getFragmentManager().beginTransaction()
33                  .replace(R.id. concept_detail_container, fragment)
34                  .commit();
35          }
36  }
```

① 第 8 行至第 36 行定义一个类 ActivityFragmentConcept 继承 Activity 类,且实现 Callbacks 接口。

② 第 13 行至第 19 行重写 onCreate()方法,第 18 行加载/res/layout 目录下的 activitytwopanes.xml 布局文件。

③ 第 21 行至第 35 行重写 onItemSelected()方法,这是实现 Callbacks 接口必须实现的方法,第 25 行至第 26 行创建 Bundle,准备向 Fragment 传入参数,第 28 行为创建 FragmentDetailConcept 对象 fragment,第 30 行向 Fragment 传入参数,第 32 行至第 34 行使用 fragment 替换 concept_detail_container 容器当前显示的 Fragment。

(4) 下面的 Fragment 将会加载 fragmentdetail.xml 布局文件,构成 Activity 界面的右部,并根据传入的参数更新该部分。

在 com.application.activityfragment 包下的 FragmentDetailConcept.java 文件中,编辑代码如下:

```
1   package com.application.activityfragment;
2
3   import com.application.activityfragment.R;
4   import com.application.activityfragment.model.ConceptData;
5   import android.app.Fragment;
6   import android.os.Bundle;
7   import android.view.LayoutInflater;
8   import android.view.View;
9   import android.view.ViewGroup;
10     import android.widget.TextView;
11
```

```
12    //定义一个类 FragmentDetailConcept 继承 Fragment 类
13    public class FragmentDetailConcept extends Fragment
14    {
15        public static final String ITEM_ID = "item_id";
16        //保存该 Fragment 显示的 Concept 对象
17        ConceptData.Concept concept;
18
19        //重写 onCreate()方法
20        @Override
21        public void onCreate(Bundle savedInstanceState)
22        {
23            super.onCreate(savedInstanceState);
24            //如果启动该 Fragment 时包含了 ITEM_ID 参数
25            if (getArguments().containsKey(ITEM_ID))
26            {
27                concept = ConceptData.ITEM_MAP.get(getArguments()
28                    .getInt(ITEM_ID));
29            }
30        }
31
32        //重写 onCreateView()方法,该方法返回的 View 将作为 Fragment 显示的组件
33        @Override
34        public View onCreateView(LayoutInflater inflater,
35            ViewGroup container, Bundle savedInstanceState)
36        {
37            //加载/res/layout/目录下的 fragmentdetail.xml 布局文件
38            View rootView = inflater.inflate(R.layout.fragmentdetail,
39                    container, false);
40            if (concept != null)
41            {
42                //让 concept_title 文本框显示 concept 对象的 title 属性
43                ((TextView) rootView.findViewById(R.id.concept_title))
44                        .setText(concept.title);
45                //让 concept_desc 文本框显示 concept 对象的 desc 属性
46                ((TextView) rootView.findViewById(R.id.concept_desc))
47                    .setText(concept.desc);
48            }
49            return rootView;
50        }
51 }
```

① 第 13 行至第 51 行定义一个类 FragmentDetailConcept 继承 Fragment 类。

② 第 20 行至第 30 行重写 onCreate()方法。

③ 第 33 行至第 50 行重写 onCreateView()方法,该方法返回的 View 将作为 Fragment 显示的组件,第 38 行至第 39 行加载/res/layout/目录下的 fragmentdetail.xml 布局文件,

构成 Activity 界面的右部，第 43 行至第 44 行让 concept_title 文本框显示 concept 对象的 title 属性，第 46 行至第 47 行让 concept_desc 文本框显示 concept 对象的 desc 属性。

(5) 下面的 Fragment 开发了一个 ListFragment 的子类，调用 setListAdapter()方法设置 Adapter，构成 Activity 界面的左部的列表项。

在 com.application.activityfragment 包下的 FragmentListConcept.java 文件中，编辑代码如下：

```
1  package com.application.activityfragment;
2
3  import com.application.activityfragment.model.ConceptData;
4  import android.app.Activity;
5  import android.app.ListFragment;
6  import android.os.Bundle;
7  import android.view.View;
8  import android.widget.ArrayAdapter;
9  import android.widget.ListView;
10
11    //定义一个类 FragmentListConcept 继承 ListFragment 类
12    public class FragmentListConcept extends ListFragment
13    {
14        private Callbacks mCallbacks;
15        //定义一个回调接口 Callbacks,该 Fragment 所在 Activity 需要实现该接口
16        //该 Fragment 将通过该接口与它所在的 Activity 交互
17        public interface Callbacks
18        {
19            public void onItemSelected(Integer id);
20        }
21
22        //重写 onCreate()方法
23        @Override
24        public void onCreate(Bundle savedInstanceState)
25        {
26            super.onCreate(savedInstanceState);
27            //为该 ListFragment 设置 Adapter
28            setListAdapter(new ArrayAdapter<ConceptData.Concept>(getActivity(),
29                    android.R.layout.simple_list_item_activated_1,
30                    android.R.id.text1, ConceptData.ITEMS));
31        }
32
33        //重写 onAttach()方法,当该 Fragment 被添加、显示到 Activity 时,回调该方法
34        @Override
35        public void onAttach(Activity activity)
36        {
37            super.onAttach(activity);
```

```
38              //如果 Activity 没有实现 Callbacks 接口,抛出异常
39              if (!(activity instanceof Callbacks))
40              {
41                  throw new IllegalStateException(
42                          BookListFragment 所在的 Activity 必须实现 Callbacks 接口!);
43              }
44              //把该 Activity 当成 Callbacks 对象
45              mCallbacks = (Callbacks)activity;
46          }
47          //重写 onDetach()方法,当该 Fragment 从它所属的 Activity 中被删除时回调该方法
48          @Override
49          public void onDetach()
50          {
51              super.onDetach();
52              //将 mCallbacks 赋为 null
53              mCallbacks = null;
54          }
55          //重写 onListItemClick()方法,当用户单击某列表项时激发该回调方法
56          @Override
57          public void onListItemClick(ListView listView
58              , View view, int position, long id)
59          {
60              super.onListItemClick(listView, view, position, id);
61              //激发 mCallbacks 的 onItemSelected 方法
62              mCallbacks.onItemSelected(ConceptData
63                  .ITEMS.get(position).id);
64          }
65
66          public void setActivateOnItemClick(boolean activateOnItemClick)
67          {
68              getListView().setChoiceMode(
69                      activateOnItemClick ? ListView.CHOICE_MODE_SINGLE
70                              : ListView.CHOICE_MODE_NONE);
71          }
72      }
```

① 第 12 行至第 72 行定义一个类 FragmentListConcept 继承 ListFragment 类。

② 第 17 行至第 20 行重写 onCreate()方法,定义一个回调接口 Callbacks,该 Fragment 所在 Activity 需要实现该接口,该 Fragment 将通过该接口与它所在的 Activity 交互。

③ 第 23 行至第 31 行重写 onCreate()方法,第 28 行至第 30 行为该 ListFragment 设置 Adapter。

④ 第 34 行至第 46 行重写 onAttach()方法,重写 onDetach()方法,当该 Fragment 从它所属的 Activity 中被删除时回调该方法。

⑤ 第 56 行至第 64 行重写 onListItemClick()方法,当用户单击某列表项时激发该回调方法。

（6）下面的 ConceptData 类使用 List 集合和 Map 集合记录系统所包含的 Concept 对象。

在 com.application.activityfragment.model 包下的 ConceptData.java 文件中，编辑代码如下：

```
1  package com.application.activityfragment.model;
2
3  import java.util.ArrayList;
4  import java.util.HashMap;
5  import java.util.List;
6  import java.util.Map;
7
8  // 定义一个类 ConceptData
9  public class ConceptData
10   {
11       //定义一个内部类 Concept,作为系统的业务对象
12       public static class Concept
13       {
14
15           public Integer id;
16           public String title;
17           public String desc;
18
19           public Concept(Integer id, String title, String desc)
20           {
21               this.id = id;
22               this.title = title;
23               this.desc = desc;
24           }
25
26           @Override
27           public String toString()
28           {
29               return title;
30           }
31       }
32       //使用 List 集合记录系统所包含的 Concept 对象
33       public static List<Concept> ITEMS = new ArrayList<Concept>();
34       //使用 Map 集合记录系统所包含的 Concept 对象
35       public static Map<Integer, Concept> ITEM_MAP
36           = new HashMap<Integer, Concept>();
37
38       static
39       {
40           //使用静态初始化代码,将 Concept 对象添加到 List 集合、Map 集合中
41           addItem(new Concept(1, "类(Class)",
```

```
42                  "将数据和方法封装在一起的数据结构,用户定义一个类"
43                          + "实际上是定义一个新的数据类型."));
44          addItem(new Concept(2, "对象(Object)",
45                  "对象是类的实例,类是对象的模板."));
46          addItem(new Concept(3, "继承(Inheritance)",
47                  "继承可以实现代码的复用,被继承的类称为父类、基类或超类,"
48                          +"由继承而得的类称为子类或导出类.
49                                  子类继承父类的成员变量和成员方法,"
50                          + "可以修改父类的成员变量或重写父类的方法,"
51                          + "还可以添加新的成员变量和成员方法."));
52      }
53
54      private static void addItem(Concept concept)
55      {
56          ITEMS.add(concept);
57          ITEM_MAP.put(concept.id, concept);
58      }
59  }
```

① 第 9 行至第 59 行定义一个类 ConceptData。

② 第 33 行使用 List 集合记录系统所包含的 Concept 对象,第 35 行使用 Map 集合记录系统所包含的 Concept 对象。

③ 第 41 行至第 51 行使用静态初始化代码,将 Concept 对象添加到 List 集合、Map 集合中。

【运行结果】

在 Eclipse 中启动模拟器,然后运行项目 ActivityFragment,当左部 Fragment 的“继承(Inheritance)”列表项被选中,右部 Fragment 出现该条目的名称和内容,如图 3.7 所示。

图 3.7　ActivityFragment 运行结果

3.5 Intent 属性、过滤器和传递数据

无论是启动 Activity、启动 Service 或者启动 BroadcastReceiver 等某个组件，Android 使用统一的 Intent 来封装对某个组件的“启动意图”，以利于高层次的解耦。此外，Intent 还是组件之间通信的重要媒介。

3.5.1 Intent 属性

Intent 是连接应用程序的三个核心组件——Activity、Service 和 BroadcastReceiver 的桥梁，Intent 负责对应用中操作的动作、动作涉及数据及附加数据进行描述。

Intent 类定义在 android.content.Intent 包中，Intent 对象包含 Component、Action、Data、Category、Extra 及 Flag 等 6 种属性。

1. Component

Component(组件)属性用于指定 Intent 的目标组件，一般由相应组件的包名与类名组合而成。指定了 Component 属性值之后，Intent 的其他属性值都是可选的，此时该 Intent 就是一个显式 Intent。如果不指定 Component 属性值，则在 Androidmanifest 中，通过使用 IntentFilter 来找到一个与之匹配的目标组件，则该 Intent 就是个隐式 Intent。

通过 setComponent()方法设置组件属性，通过 setClass()、setClassName()方法设置将要启动的组件对应的类，通过 getComponent()方法读取组件名。

2. Action

Action(行动)属性用来指明要实施的动作是什么，其属性值是 Intent 即将触发动作名称的字符串。

Intent 定义了用大写字母和下画线组成的动作常量，如表 3.1 所示。

表 3.1 常用动作常量

常　量	含　义
ACTION_MAIN	应用程序入口
ACTION_VIEW	显示指定数据
ACTION_EDIT	编辑指定数据
ACTION_PICK	从列表中选择某项，并返回所选数据
ACTION_CALL	向指定用户打电话
ACTION_BATTERY_LOW	提示电池电量低
ACTION_SCREEN_ON	屏幕已开启

Action 属性可通过 setAction()方法来设置，通过 getAction()方法来读取。

3. Data

Data(数据)属性用于完成对 Intent 消息中数据的封装，描述 Intent 动作所操作数据的 URI(Uniform Resource Identifier，通用资源标识符)及 MIME(多用途互联网邮件扩展)。

Type(数据类型)属性用于指定 URI 对应的 MIME。

URI 格式为：scheme://host:port/path。

获取一个 URI 的语句格式为：

```
Uri uri = Uri.parse(<字符串>);
```

创建一个 Intent 对象的语句格式为：

```
Intent intent = new Intent(<动作>, <内容>);
```

代码：

```
Uri uri1 = Uri.parse(content://contacts/1);
Intent intent = new Intent(Intent.ACTION_VIEW, uri1);
```

说明：

在上面的代码中，uri1 是一个 Uri 变量，其值为：content://contacts/1，指向手机联系人信息集中的第一个联系人，创建的对象 intent 显示标识符为“1”的联系人的详细信息。

通过 setData()方法设置 URI，通过 getData ()方法读取 URI。

4. Category

Category(类别)属性用于描述目标组件额外的附加类别信息，其属性值是一个字符串。

一个 Intent 中可以包含多个 Category。如果没有设置 Category 属性值，Intent 会与在 Intent filter 中包含“android.category.DEFAULT”的 Activity 匹配。

Intent 定义了类别常量，如表 3.2 所示。

表 3.2　常用类别常量

常　　量	含　　义
CATEGORY_DEFAULT	默认的 Category
CATEGORY _BROWSABLE	指定该 Activity 能被浏览器安全调用
CATEGORY _HOME	设置该 Activity 随系统启动而运行
CATEGORY _LAUNCHER	该 Activity 列在应用程序启动器顶层，应用程序启动时首先被显示
CATEGORY _PREFERENCE	该 Activity 是参数面板
CATEGORY _INFO	用于提供包信息
CATEGORY _TEST	该 Activity 是一个测试

通过 addCategory()方法添加一个 Category，通过 emoveCategory()方法删除一个 Category，通过 getCategories()可以获取当前对象的所有 Category。

5. Extra

Extra(附加信息)属性用于在多个 Action 之间进行数据交换。

Extra 可以被当作一个 Bundle 对象，存入多组 key-value 对(键-值对)，这就可以通过 Intent 在不同 Action 之间进行数据交换了。

Intent 通过调用 putExtras()方法来添加一个新的键-值对，而在目标 Activity 中调用 getExtras()方法来获取 Extra 属性值。

6. Flag

Flag(标志)属性是一些有关系统如何启动组件的标志。指导 Android 系统启动一个 Activity 以及 Activity 启动后对其进行处理。

3.5.2 启动 Activity

启动 Activity 分为显式启动和隐式启动两种。显式启动，必须在 Intent 中指明启动的 Activity 对应的类。隐式启动不指明启动的 Activity 对应的类，系统会根据 Intent 指定的规则去启动符合条件的 Activity。

1. 显式启动

使用 Intent 显式启动 Activity，在创建一个 Intent 后，指定当前的应用程序上下文以及要启动的 Activity，把创建好的这个 Intent 作为参数传递给 startActivity()方法。

【例 3.3】 显式启动 Activity 示例。

【解题思路】

在应用项目 ExplicitStart 中，包含两个 Activity，一个是 ExplicitStartActivity，另一个是 SecondActivity。

程序默认启动的 Activity 是 ExplicitStartActivity，进入 ExplicitStartActivity 界面，当用户单击“启动 Activity”按钮后，程序使用 Intent 显式启动的 Activity 是 SecondActivity，显式启动 Activity 的代码如下：

```
Intent intent = new Intent(ExplicitStartActivity.this, SecondActivity.class);
startActivity(intent);
```

【开发步骤和程序分析】

(1) 在 Eclipse 中创建一个 ExplicitStart 应用项目，包名为 com.application.explicitstart。

(2) 在 src/com.application.explicitstart 包下的 ExplicitStartActivity.java 文件中，使用 Intent 显式启动 SecondActivity。

在该文件中编辑代码如下：

```
1  package com.application.explicitstart;
2
3  import com.application.explicitstart.R;
4  import android.app.Activity;
5  import android.content.Intent;
6  import android.os.Bundle;
7  import android.view.View;
8  import android.view.View.OnClickListener;
9  import android.widget.Button;
10
11 public class ExplicitStartActivity extends Activity {
12
13     @Override
14     public void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.main);
17         Button button = (Button)findViewById(R.id.btn);
18         button.setOnClickListener(new OnClickListener(){
```

```
19              public void onClick(View view){
20                  Intent intent = new Intent(ExplicitStartActivity.this, SecondActivity.class);
21                   startActivity(intent);
22              }
23          });
24      }
25  }
```

第 19 行至第 21 行(加黑部分),在单击事件 onClick(View view)方法中,首先,使用 Intent 构造方法创建一个实例 intent,其中的第一个参数是应用程序上下文 ExplicitStartActivity,第 2 个参数是接收 Intent 的目标组件 SecondActivity,这里使用显式启动方式,直接指明了需要启动的 Activity,然后,使用 startActivity(intent)方法显式启动 SecondActivity。

【运行结果】

应用项目 ExplicitStart 运行结果如图 3.8 和图 3.9 所示。

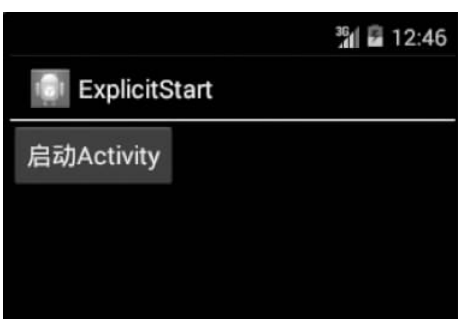

图 3.8　默认启动 ExplicitStartActivity

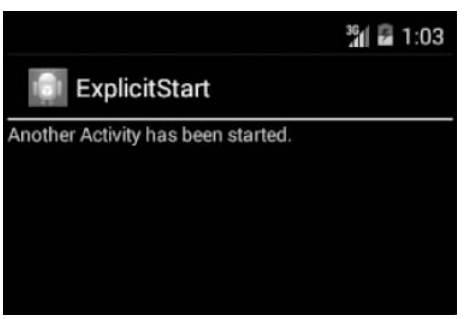

图 3.9　显式启动 SecondActivity

2. 隐式启动

隐式启动不指明启动的 Activity 对应的类,Android 系统会根据 Intent 指定的属性:Action、Data、Category 去启动符合条件的 Activity。

隐式启动的优点是不必指明需要启动哪一个 Activity,而由系统来决定,这样有利于降低组件之间的耦合度,提高 Android 组件的可复用性。

【例 3.4】 隐式启动 Activity 示例。

【解题思路】

在应用项目 ImplicitStart 中,需要启动网页 http://www.baidu.com。

在隐式启动 Activity 中,Intent 的动作是 Intent.ACTION_VIEW,数据是 Web 地址,使用 Uri.parse(urlString)方法。Android 系统在匹配 Intent 时,根据动作 Intent.ACTION_VIEW 和数据提供的是 Web 地址 http://www.baidu.com,判定 Intent 需要启动具有网页浏览功能的 Activity。

隐式启动 Activity 的代码如下:

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(urlString));
```

```
startActivity(intent);
```

【开发步骤和程序分析】

（1）在 Eclipse 中创建一个 ImplicitStart 应用项目，包名为 com. application . implicitstart。

（2）在 src/com. application. implicitstart 包下的 ImplicitStart. java 文件中，使用 Intent 隐式启动 Activity，提示用户在 http://后输入 Web 地址，以启动具有网页浏览功能的 Activity。

在该文件中编辑代码如下：

```
1  package com.application.implicitstart;
2
3  import com.application.implicitstart.R;
4  import android.app.Activity;
5  import android.content.Intent;
6  import android.net.Uri;
7  import android.os.Bundle;
8  import android.view.View;
9  import android.view.View.OnClickListener;
10 import android.widget.Button;
11 import android.widget.EditText;
12
13 public class ImplicitStart extends Activity {
14
15     @Override
16     public void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.main);
19         final EditText editText = (EditText)findViewById(R.id.edit_url);
20         final Button button = (Button)findViewById(R.id.btn);
21         button.setOnClickListener(new OnClickListener(){
22             public void onClick(View view){
23                 String urlString = editText.getText().toString();
24                 Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(urlString));
25                 startActivity(intent);
26             }
27         });
28     }
29 }
```

第 22 行至第 25 行（加黑部分），在单击事件 onClick（View view）方法中，首先，使用 Intent 构造方法创建一个实例 intent，其中第一个参数的动作是显示数据 Intent. ACTION_VIEW，第 2 个参数的数据是 Web 地址，使用 Uri. parse（urlString）方法，这里使用隐式启动方式，提示用户在 http://后输入 Web 地址，当用户输入完成 Web 地址 http://www. baidu. com 并单击“浏览网页”按钮后，启动具有网页浏览功能的 Activity，显示百度页面。

【运行结果】

应用项目 ExplicitStart 运行结果如图 3.10 和图 3.11 所示。

图 3.10　提示用户在 http://后输入 Web 地址

图 3.11　隐式启动 Web 页面

3.5.3　Intent 过滤器

Intent 过滤器(Intent Filter)是一个包含 Intent 对象的 action、data、category 属性限制条件的集合，Intent Filter 要检测隐式 Intent 的 action、data、category 这三个属性，其中任何一项失败，Android 系统都不会传递 Intent 给此组件。

一个组件可以有多个 Intent Filter，Intent 只要通过其中的某个 Intent Filter 检测，就可以调用此组件。

Intent 过滤器在 AndroidManifest. xml 文件中进行声明，Intent 过滤器使用< intent-filter >子标签来进行声明。

Intent 解析机制主要是通过查找已注册在 AndroidManifest. xml 中的所有 Intent Filter 及其中定义的 Intent 属性，最终找到匹配的 Intent。

(1) Action 检查：一个 Intent 只能设置一种 Action，而一个 Intent Filter 可以设置多个 Action。如果 Intent 指明定了 action，则目标组件的 Intent Filter 的 action 列表中就必须包含有这个 action，否则不能匹配；如果 Intent 没有指定 action，将自动通过检查。

(2) Category 检查：在一个 Intent Filter 中，可以设置多个 Category。如果 Intent 指定了一个或多个 category，这些类别必须全部出现在组件的 category 列表中。

(3) Data 检查：对数据的检查有两部分，一是对数据 URI 进行检查，一是对数据类型

进行检查，对数据 URI 的检查包括 schema、authority 和 path。

【例 3.5】 Intent 过滤器示例。

【解题思路】

在应用项目 IntentFilterExample 中，在 AndroidManifest.xml 文件<intent-filter>节点的<action>标签、<category>标签和<data>标签，分别定义 Intent 过滤器的“动作”“类别”和“数据”，以进行相关检查，最终找到匹配的 Intent。

【开发步骤和程序分析】

(1) 在 Eclipse 中创建一个 IntentFilterExample 应用项目，包名为 com.application.IntentFilterExample。

(2) 在 AndroidManifest.xml 文件中，分别定义了 UserActivity1 和 UserActivity2 的 Intent 过滤器，包括动作、类别和数据等。

在该文件中编辑代码如下：

```
1  <?xml version = "1.0" encoding = "utf - 8"?>
2  < manifest xmlns:android = "http://schemas.android.com/apk/res/android"
3      package = "com.application.IntentFilterExample"
4      android:versionCode = "1"
5      android:versionName = "1.0" >
6      < uses - sdk android:minSdkVersion = "14" />
7      < application
8          android:icon = "@drawable/ic_launcher"
9          android:label = "@string/app_name" >
10         < activity
11             android:label = "@string/app_name"
12             android:name = "com.application.IntentFilterExample.UserActivity1" >
13             < intent - filter >
14                 < action android:name = "android.intent.action.MAIN" />
15                 < category android:name = "android.intent.category.LAUNCHER" />
16             </intent - filter >
17         </activity>
18         < activity android:name = "com.application.IntentFilterExample.UserActivity2"
19                    android:label = "@string/app_name">
20             < intent - filter >
21                 < action android:name = "android.intent.action.VIEW" />
22                 < category android:name = "android.intent.category.DEFAULT" />
23                 < data android:scheme = "schemodemo" android:host = "com.application" />
24             </intent - filter >
25         </activity>
26     </application>
```

① 在第 10 行到第 17 行，定义了第 1 个 Activity 及其 Intent 过滤器，第 1 个 Activity 名为 UserActivity1，第 13 行到第 16 行是第 1 个 Activity 的 Intent 过滤器(加黑部分)，动作为 android.intent.action.MAIN，类别为 android.intent.category.LAUNCHER，由此得

出，第 1 个 Activity 是应用程序启动后显示的默认用户界面。

② 在第 18 行到第 25 行，定义了 2 个 Activity 及其 Intent 过滤器，第 2 个 Activity 名为 UserActivity2，第 20 行到第 24 行是第 2 个 Activity 的 Intent 过滤器(加黑部分)，过滤器的动作是 android. intent. action. VIEW，表示根据 Uri 协议，以浏览的方式启动相应的 Activity；类别是 android. intent. category. DEFAULT，表示数据的默认动作；数据的协议部分是 android: scheme="schemodemo"，数据的主机名称部分是 android:host="com. application"。

(3) 在 src/com. application. IntentFilterExample 包下的 UserActivity1. java 文件中，定义的 Intent 动作和数据分别与 UserActivity2 的 Intent 过滤器定义的动作、数据要求相匹配，该 Intent 用于启动 UserActivity2。

在该文件中编辑代码如下：

```
1  package com.application.IntentFilterExample;
2
3  import com.application.IntentFilterExample.R;
4  import android.app.Activity;
5  import android.content.Intent;
6  import android.net.Uri;
7  import android.os.Bundle;
8  import android.view.View;
9  import android.view.View.OnClickListener;
10 import android.widget.Button;
11
12 public class UserActivity1 extends Activity {
13
14     @Override
15     public void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.main);
18         Button button = (Button)findViewById(R.id.btn);
19         button.setOnClickListener(new OnClickListener(){
20             public void onClick(View view){
21                 Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("schemodemo://
                   com.application/path"));
22                 startActivity(intent);
23             }
24         });
25     }
26 }
```

① 第 21 行至第 22 行，定义了一个 Intent 用来启动另一个 Activity，这个 Intent 与 Activity 设置的 Intent 过滤器是完全匹配的。

② 在第 21 行定义的 Intent(加黑部分)，动作为 Intent. ACTION_VIEW，Uri 是 "schemodemo://edu. hrbeu/path"，其中的协议部分为"schemodemo"，主机名部分为"edu.

hrbeu",分别与 Intent 过滤器定义的动作、数据要求完全匹配。因此,当代码第 18 行定义的 Intent,在 Android 系统与 Intent 过滤器列表进行匹配时,会与 AndroidManifest. xml 文件中 UserActivity2 定义的 Intent 过滤器完全匹配。

【运行结果】

应用项目 IntentFilterExample 运行结果如图 3.12 和图 3.13 所示。

图 3.12　进入 UserActivity1 界面

图 3.13　进入 UserActivity2 界面

3.5.4　Activity 组件之间通过 Intent 通信

下面通过一个例题说明 Activity 组件之间通过 Intent 通信。

【例 3.6】 Activity 组件之间通过 Intent 通信举例。

两个 Activity：FirstActivity 和 SecondActivity，界面上首次进入的 Activity 为 FirstActivity,通过单击按钮来实现 FirstActivity 和 SecondActivity 的相互跳转,使用 Intent 对象实现两个 Activity 之间的通信。

【解题思路】

在应用项目 ActivityIntentExample 中，FirstActivity 的布局文件为 first_main. xml，SecondActivity 的布局文件为 second_main. xml，FirstActivity 的 Java 代码文件为 FirstActivity. java,SecondActivity 的 Java 代码文件为 SecondActivity. java。

在 FirstActivity 和 SecondActivity 组件中,使用了按钮控件,因此在相应的布局文件中,需要声明按钮控件,其标签为< Button >。

在 Java 代码文件中,对按钮控件设置监听,使用方法 setOnClickListener(),如果监听到按钮被单击,则执行 onClick()事件方法定义的操作。

在两个 Activity 调用中,使用显式启动,两个 Activity 调用需要返回信息,使用 startActivityForResult()方法发送 Intent 对象,startActivityForResult()方法的格式如下：

```
startActivityForResult(Intent intent, int requestCode)
```

如果是从 A 发送 B,然后从 B 返回到 A,并且需要传递信息,则在 A 代码中使用 startActivityForResult()方法发送 Intent 到 B,并且重写 onActivityResult()方法用于处理返回的数据；在 B 代码中使用 setResult()方法准备好回传的数据,并且使用 finish()方法将打包好的数据发回给 A,并运行 A 中的 onActivityResult()部分代码。

【开发步骤和程序分析】

(1) 在 Eclipse 中创建一个 ActivityIntentExample 应用项目,包名为 com. application . activityintentexample,有两个 Activity: FirstActivity 和 SecondActivity。

(2) 设计布局。

布局文件为 first_main. xml 和 second_main. xml。

在 res/layout 目录中,编写 FirstActivity 的布局文件 first_main. xml,定义了一个"进入 SecondActivity"按钮。

在该文件中编辑代码如下:

```
1  <?xml version = "1.0" encoding = "utf - 8"?>
2  < LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
3       android:orientation = "vertical"
4       android:layout_width = "fill_parent"
5       android:layout_height = "fill_parent"
6       >
7      < Button android:id = "@ + id/button1"
8            android:layout_width = "wrap_content"
9            android:layout_height = "wrap_content"
10           android:text = "进入 SecondActivity" />
11 </LinearLayout >
```

第 7 行至第 10 行,定义了一个按钮,其中,第 7 行定义该按钮的 id 变量名为 button1,并添加到 R. java 文件中,为 Java 代码提供调用,第 10 行定义该按钮显示文本内容为"进入 SecondActivity"。

在 res/layout 目录中,编写 SecondActivity 的布局文件 second_main. xml,定义了一个"返回 FirstActivity"按钮。

在该文件中编辑代码如下:

```
1  <?xml version = "1.0" encoding = "utf - 8"?>
2  < LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
3       android:orientation = "vertical" android:layout_width = "fill_parent"
4       android:layout_height = "fill_parent">
5       < Button android:id = "@ + id/button2"
6            android:layout_width = "wrap_content"
7            android:layout_height = "wrap_content"
8            android:text = "返回 FirstActivity" />
9 </LinearLayout >
```

第 5 行至第 8 行,定义一个按钮,其中,第 5 行定义该按钮的 id 变量名为 button2,并添加到 R. java 文件中,第 8 行定义该按钮显示文本内容为"返回 FirstActivity"。

(3) 在包 com. application. activityintentexample 下的 FirstActivity. java 文件中,加载 first _ main. xml 布局文件,使用 startActivityForResult ( ) 方法发送 Intent1 到 SecondActivity,并重写 onActivityResult()方法用于处理返回的数据。

在该文件中编辑代码：

```
1  package com.application.activityintentexample;
2
3  import android.app.Activity;
4  import android.content.Intent;
5  import android.os.Bundle;
6  import android.view.View;
7  import android.view.View.OnClickListener;
8  import android.widget.Button;
9  public class FirstActivity extends Activity {
10     OnClickListener listener1 = null;
11     Button button1;
12     static final int REQUEST_CODE = 1;
13
14     @Override
15     public void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         listener1 = new OnClickListener() {
18             public void onClick(View v) {
19                 //创建一个 Intent 对象,对象名为 intent1
20                 Intent intent1 = new Intent(FirstActivity.this, SecondActivity.class);
21                 //向 intent1 中添加附加信息,一组键 - 值对的名为"firstactivity",
22                 //值为"从 FirstActivity 进入 -- "
23                 intent1.putExtra("firstactivity", "从 FirstActivity 进入 -- ");
24                 //使用 startActivityForResult()方法发送 intent1 对象,同时发送一个请求码
25                 startActivityForResult(intent1, REQUEST_CODE);
26             }
27         };
28         setContentView(R.layout.first_main);
29         button1 = (Button) findViewById(R.id.button1);
30         button1.setOnClickListener(listener1);
31         setTitle("查看信息内容页面          -- 首次进入 FirstActivity -- ");
32   }
33     //重写 onActivityResult()方法,通过判断请求码值和返回码值,来确定是否正确获得
34     //回传数据,如果是正确的,则取出回传数据并显示在标题栏中
35     @Override
36     protected void onActivityResult(int requestCode, int resultCode, Intent data) {
37         if (requestCode == REQUEST_CODE) {
38             if (resultCode == RESULT_CANCELED)
39                 setTitle("取消");
40             else if (resultCode == RESULT_OK) {
41                 String temp = null;
42                  Bundle extras = data.getExtras();
43                     if (extras != null) {
```

```
44                      temp = extras.getString("store");
45                  }
46                  setTitle("查看信息内容页面          --" + temp);
47              }
48          }
49      }
50  }
```

① 第 17 行至第 27 行(加黑部分),创建一个监听 listener1,同时定义一个 onClick 事件,在事件中定义了当监听到按钮被单击,则进行相应的操作。其中:

- 第 20 行创建一个 Intent 对象,对象名为 intent1,其动作值为 FirstActivity. this,数据值为 SecondActivity. class,这是使用 Intent 显式启动 Activity。
- 第 23 行向 intent1 中添加附加信息:一组键-值对的 Bundle 信息,其名为“firstactivity”,值为“从 FirstActivity 进入--”。
- 第 25 行使用 startActivityForResult()方法发送 intent1 对象,同时发送一个请求码 REQUEST_CODE 给 SecondActivity。

② 第 35 行至第 49 行(加黑部分)重写 onActivityResult()方法,通过判断请求码值和返回码值,来确定是否正确获得回传数据,如果是正确的,则取出回传数据并显示在标题栏中。

(4) 在包 com. application. activityintentexample 下的 SecondActivity. java 文件中,加载 second_main. xml 布局文件,使用 setResult()方法准备好回传的数据,并且使用 finish()方法将打包好的数据发回给 FirstActivity。

在该文件中编辑代码:

```
1  package com.application.activityintentexample;
2
3  import android.app.Activity;
4  import android.content.Intent;
5  import android.os.Bundle;
6  import android.view.View;
7  import android.view.View.OnClickListener;
8  import android.widget.Button;
9
10 public class SecondActivity extends Activity {
11     OnClickListener listener1 = null;
12     Button button1;
13
14     @Override
15     public void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.second_main);
18         listener1 = new OnClickListener() {
19             public void onClick(View v) {
20                 //创建一个 Bundle 对象,对象名为 bundle
```

```
21                  Bundle bundle = new Bundle();
22                  //将一组键-值对保存到 bundle,其名为 "store",
23                  //其值为"自 SecondActivity 返回 --"
24                  bundle.putString("store", "自 SecondActivity 返回 --");
25                  //创建一个 Intent 对象,对象名为 intent2
26                  Intent intent2 = new Intent();
27                  //向 intent2 中添加已保存在 bundle 中的键-值对信息
28                  intent2.putExtras(bundle);
29                  //打包回传的数据,包括返回码值 RESULT_OK 和 intent2 对象
30                  setResult(RESULT_OK, intent2);
31                  //将打包好的数据发回给 FirstActivity,并且运行
32                  //FirstActivity.java 中 onActivityResult()里面的代码
33                  finish();
34              }
35          };
36      button1 = (Button) findViewById(R.id.button2);
37      button1.setOnClickListener(listener1);
38      //从 FirstActivity 的 intent1 对象中取出附加信息赋值给 extras,如果其键-值对非空,
39      //则取出名为"firstactivity"的对应值给字符串变量 data,并将 data 的值显示在标题栏中
40      String data = null;
41       Bundle extras = getIntent().getExtras();
42          if (extras != null) {
43               data = extras.getString("firstactivity");
44          }
45      setTitle("显示信息内容页面          --" + data);
46  }
47 }
```

① 第 18 行至第 35 行(加黑部分),创建一个监听 listener1,同时定义一个 onClick()方法,在方法中定义了当监听到按钮被单击,则进行相应的操作。其中:

- 第 21 行和第 24 行创建一个 Bundle 对象,对象名为 bundle,并将一组键-值对保存到其中,其名为“store”,其值为“自 SecondActivity 返回--”。
- 第 26 行创建一个 Intent 对象,对象名为 intent2。
- 第 28 行向 intent2 中添加附加信息,此附加信息是已保存在 bundle 中的键-值对信息。
- 第 30 行是打包回传的数据,包括返回码值 RESULT_OK 和 intent2 对象。
- 第 33 行将打包好的数据发回给 FirstActivity,并且运行 FirstActivity.java 中 onActivityResult()里面的代码。

② 第 40 行至第 45 行(加黑部分)从 FirstActivity 的 intent1 对象中取出附加信息赋值给 extras,如果其键-值对非空,则取出名为“firstactivity”的对应值给字符串变量 data,并将 data 的值显示在标题栏中。

(5) 编写根目录下的 AndroidManifest.xml 文件的代码,增加 Activity 组件 SecondActivity 的声明。

```
<?xml version = "1.0" encoding = "utf - 8"?>
< manifest xmlns:android = "http://schemas.android.com/apk/res/android"
    package = "com.application.activityintentexample"
    android:versionCode = "1"
    android:versionName = "1.0" >
    < uses - sdk android:minSdkVersion = "10" />
    < application
        android:icon = "@drawable/ic_launcher"
        android:label = "@string/app_name" >
         < activity
            android:name = ".FirstActivity"
            android:label = "@string/app_name" >
            < intent - filter >
                < action android:name = "android.intent.action.MAIN" />
                < category android:name = "android.intent.category.LAUNCHER" />
            </intent - filter >
        </activity >
        //增加 Activity 组件 SecondActivity 的声明
        < activity android:name = ".SecondActivity" ></activity >
    </application >
</manifest >
```

【运行结果】

应用项目 IntentFilterExample 运行结果如图 3.14～图 3.16 所示。

图 3.14　首次进入“查看信息内容页面”

图 3.15　单击“进入”按钮后进入“显示信息内容页面”

图 3.16　单击“返回”按钮后返回“查看信息内容页面”

3.6 小　　结

本章主要介绍了以下内容：

(1) Android 应用程序生命周期指从启动到终止的全过程，应用程序的生命周期是由 Android 系统进行调度和控制，而不是由应用程序直接控制的。Android 应用程序组件有其生命周期，指从创建到销毁的全过程，Activity 组件是 Android 应用生命周期的重要部分之一。

(2) 进程(Process)是程序的一次执行，进程由程序、数据和进程控制块构成，进程是一个可拥有资源的独立实体，又是一个可以独立调度的基本单位。在 Android 操作系统中，进程是应用程序的具体实现。组件运行的进程由 Androidmanifest 文件控制。

线程(Thread)是进程中的一个实体，是被系统独立调度的基本单位。线程基本上不拥有系统资源，只有一些在运行中必不可少的资源(如程序计数器、一组寄存器和栈)，但它可共享所属进程的全部资源。

(3) Android 应用中常用的基本组件有 Activity(活动)、Service(服务)、BroadcastReceiver(广播接收器)、ContentProvider(数据提供器)、Intent(意图)等。

Activity 用于提供可视化用户界面，它是最常用的组件；Service 是一个常用组件，一般用于没有用户界面，又需要长时间在后台运行的应用；BroadcastReceiver 是另一个常用组件，用来接收广播消息，不包含任何用户界面，其监听的事件源是其他组件；ContentProvider 组件是 Android 系统提供的一种标准的共享数据的机制，用来管理和共享应用程序的数据存储；Intent 是不同组件间通信的载体，是连接各个组件的桥梁。

(4) Activity 的生命周期中存在五种状态：启动状态，运行状态，暂停状态，停止状态，销毁状态。Activity 的回调方法有：onCreate()、onStart()、onResume()、onPause()、onStop()、onRestart()、onDestroy()、onSaveInstanceState()、onRestoreInstanceState()等。Activity 的生命周期可分为完全生命周期、可视生命周期和活动生命周期，每种生命周期中包含不同的回调方法。

(5) Fragment 有自己的生命周期，但它的生命周期受其所在的 Activity 生命周期控制，Fragment 不能独立存在，它必须嵌入到 Activity 中。当 Activity 暂停时，它拥有的所有的 Fragment 都暂停了；当 Activity 销毁时，它拥有的所有 Fragment 都被销毁；当 Activity 处于活动状态时(在 onResume()之后，onPause()之前)，用户可以通过方法操作每个 Fragment。

(6) 无论是启动 Activity、启动 Service 或者启动 BroadcastReceiver 等某个组件，Android 使用统一的 Intent 来封装对某个组件的“启动意图”，以利于高层次的解耦。此外，Intent 还是组件之间通信的重要媒介。Intent 对象包含 Component、Action、Data、Category、Extra 及 Flag 等 6 种属性。启动 Activity 分为显式启动和隐式启动两种。Intent 过滤器(Intent Filter)是一个包含 Intent 对象的 action、data、category 属性限制条件的集合，Intent Filter 要检测隐式 Intent 的 action、data、category 这三个属性，其中任何一项失

败，Android 系统都不会传递 Intent 给此组件。

习　题　3

一、选择题

3.1　在 Android 系统的进程优先级中，高优先级的进程是_____。

A. 服务进程　　B. 可见进程　　C. 后台进程　　D. 前台进程

3.2　用于提供可视化用户界面的组件是_____。

A. Service　　B. Activity

C. ContentProvider　　D. BroadcastReceiver

3.3　没有用户界面、长时间在后台运行的组件是_____。

A. Service　　B. Activity

C. ContentProvider　　D. BroadcastReceiver

3.4　_____组件用来封装对某个组件的“启动意图”。

A. Service　　B. Activity

C. ContentProvider　　D. Intent

3.5　Activity 可见，获得焦点，可与用户进行交互的状态是_____。

A. 启动状态　　B. 停止状态　　C. 暂停状态　　D. 运行状态

3.6　Activity 失去焦点，但仍可见，依然保持活力的状态是_____。

A. 启动状态　　B. 停止状态　　C. 暂停状态　　D. 运行状态

3.7　创建 Activity 时被回调，且只会被调用一次的方法是_____。

A. onPause()　　B. onResume()　　C. onCreate()　　D. onStart()

3.8　恢复 Activity 时被回调，当 Activity 可以开始与用户进行交互之前被调用的方法是_____。

A. onPause()　　B. onResume()　　C. onCreate()　　D. onStart()

3.9　显示指定数据的动作常量是_____。

A. ACTION_MAIN　　B. ACTION_VIEW

C. ACTION_EDIT　　D. ACTION_CALL

3.10　应用程序入口的动作常量是_____。

A. ACTION_MAIN　　B. ACTION_VIEW

C. ACTION_EDIT　　D. ACTION_CALL

二、填空题

3.11　Android 应用中常用的基本组件有 Activity、Service、BroadcastReceiver、ContentProvider 和_____等。

3.12　Activity 的生命周期中存在五种状态：启动状态，暂停状态，停止状态，销毁状态和_____。

3.13　Activity 的回调方法有：onCreate()、onResume()、onPause()、onStop()、onRestart()、onDestroy()和_____。

3.14　Intent 对象包含 Component、Action、Data、Extra、Flag 和_____ 6 种属性。

3.15　Intent 过滤器(Intent Filter)是一个包含 Intent 对象的 action、category 和_____属性限制条件的集合。

三、问答题

3.16　简述 Activity 的生命周期中存在的五种状态和状态之间的转换关系。

3.17　简述 Android 应用中常用的基本组件及其用途。

3.18　简述 Activity 的回调方法和调用顺序。

3.19　简述 Android 系统的前台进程、可见进程、服务进程、后台进程、空进程以及前台进程的特点和优先级。

3.20　什么是 Fragment? Fragment 有哪些特点?

3.21　简述 Fragment 的生命周期。

3.22　Intent 对象包含哪几种属性? 各有何作用?

四、编程题

3.23　编写程序,使浏览器显示本校的主页。

3.24　编写使用 Intent 拨打电话的程序,并给你的同学打一个电话。

第 4 章 Android 基本控件

本章要点

- Android 应用开发的一个重要内容是用户界面开发，View 和 ViewGroup 是 Android 平台上最基本的两个用户界面表达单元。
- View(视图)是所有可视化窗体控件的基类，ViewGroup (视图组)可以充当 View 的容器。
- 常用的布局有线性布局(LinearLayout)、表格布局(TableLayout)、帧布局(FrameLayout)、网格布局(GridLayout)、相对布局(RelativeLayout)、绝对布局(AbsoluteLayout)。
- 常用的基本控件有 TextView、EditText、Button、ImageButton、ImageView、Checkbox、RadioButton、AnalogClock、DigitalClock、DatePicker、TimePicker 等。

用户界面(User Interface，UI)是系统和用户之间进行信息交换的媒介，控件是 Android 用户界面中的组成元素。本章介绍 Android 用户界面设计、常用布局、常用基本控件等内容。

4.1 用户界面设计

Android 应用开发的一个重要内容是用户界面开发，提供友好的用户界面是设计人员的主要工作。

1. 用户界面设计

用户界面(User Interface，UI)是系统和用户之间进行信息交换的媒介，实现信息的内部形式与人类可以接受形式之间的转换。当前流行的图形用户界面(Graphical User Interface，GUI)是采用图形方式与用户进行交互的界面。

Android 应用开发的一个重要内容是用户界面开发，一个应用项目如果没有提供友好的图形用户界面，再优秀的应用项目也很难吸引用户。与此相反，如果应用项目提供了友好的图形用户界面，往往会受到用户的欢迎。Android 提供了大量功能丰富的 UI 控件，界面设计人员按一定的规则采用“搭积木”的方式，并具备一定的设计经验后，就可设计出优秀的图形用户界面。

在 Android 手机上进行用户界面设计具有创造性和挑战性，由于界面设计与程序逻辑完全分离，手机界面设计人员与程序开发人员是独立并行工作的；而不同型号手机的尺寸、长宽比例和屏幕分辨率的不同，程序界面需要根据屏幕信息进行自动调整；并且需要设计人员合理利用有限的显示空间设计出优秀的手机界面。