第3章 Android应用程序

学习目标

本章主要介绍 Android 项目中的目录结构、AndroidManifest.xml 文件等项目构成文件，同时，对 Android 应用程序组件 Activity、Service、Intent 和 IntentFilter、BroadcastReceiver、ContentProvider 进行介绍，并就 Android 程序生命周期和组件生命周期详细讲解。通过本章的内容，读者能够学习以下知识要点。

（1）Android 项目文件及应用程序。

（2）Android 系统的进程优先级的变化方式。

（3）Android 系统的基本组件。

（4）Android 程序生命周期及 Activity 的生命周期中各状态的变化关系。

（5）Activity 事件回调函数的作用和调用顺序。

（6）Android 应用程序的调试方法和工具。

在 Android 的应用过程中，对于初学者而言，通常会混淆 Android 项目和 Android 应用程序的概念。它们之间既有区别又有联系，要创建 Android 应用，必须先创建 Android 项目，然后才能在项目中创建 Android 应用程序，下面就 Android 项目的构成和 Android 应用程序的组成进行介绍。

3.1 Android 项目的构成

3.1.1 目录结构

在建立新项目的过程中，ADT 会自动建立一些目录和文件，这些目录和文件有其固定的作用，有的允许修改，有的不能修改。Android Studio 环境下和 Eclipse 环境下创建 Android 项目的目录结构不同，本书针对 Android

Studio 环境下新建项目 Hello 工程进行介绍，项目工程结构包含.gradle、.idea、app、gradle、.gitignore、build.gradle、gradle.properties、gradlew、gradlew.bat、local.properties、Hello.iml、setting.gradle，如图 3-1 所示，下面逐一介绍。

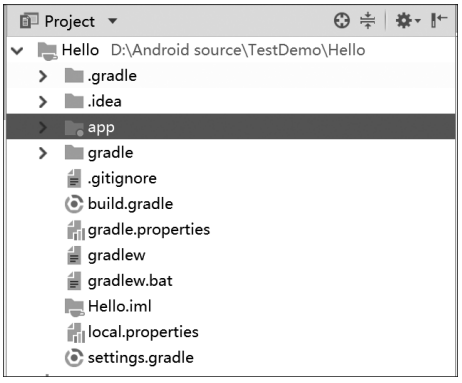

图 3-1 Android Studio 环境下 Android 工程目录结构

.gradle 目录和.idea 目录：分别为 Android Studio 集成环境下自动化构建工具 gradle（基于 JVM 的构建工具）产生的文件和开发工具 idea（其全称是 IntelliJ IDEA，开发集成工具）产生的文件，存放项目的配置信息，包括历史记录，版本控制信息等。

iml 是 intellij idea 自动创建的模块文件，存储模块开发相关的信息、依赖信息等。

app 目录：包含 build 目录、libs 目录、src 目录、.gitignore 文件、build.gradle 文件、app.iml 文件、proguard-rules.pro 文件。

- build 目录包含了编译时自动生成的文件。
- libs 目录存放项目需要的第三方 JAR 包。
- src 目录是源代码目录，所有允许用户修改的 Java 文件和用户自己添加的 Java 文件都保存在这个目录中的 main 目录下。
- .gitignore 是 git 源码管理文件 ignore，作用是将 app 模块内指定的目录或文件排除在代码版本控制之外。
- build.gradle 是 app 模块中的 gradle 的配置文件，另外一个 build.gradle 文件位于工程根目录下。
- app.iml 是 idea 创建的 android 模块文件，保存模块路径、依赖关系及其他配置信息。
- proguard-rules.pro 是代码混淆配置模板。

gradle.properties：运行环境配置文件，在它里面可以定义全局的属性，然后在各个模块的 build.gradle 中直接引用。

gradlew：是 Linux 下的 shell 脚本，Linux 用户可以通过它来执行 gradle 任务。

gradlew.bat：是 Windows 下的可执行脚本，用户可以通过它执行 gradle 任务，与 gradle 文件夹配合使用。

local.properties：是项目工程的本地配置。该文件是在工程编译时自动生成的，用于描述开发者本机的环境配置，如 SDK 的本地路径、NDK 的本地路径等。

setting.gradle：配置同时编译的模块。初始内容为 include': app'，表示只编译 app

模块。

Hello.iml：工程的配置文件。

3.1.2 AndroidManifest.xml 文件简介

AndroidManifest.xml 是 XML 格式的 Android 程序声明文件，是全局描述文件，包含了 Android 系统运行 Android 程序前所必须掌握的重要信息，这些信息包含应用程序名称、图标、包名称、模块组成、授权和 SDK 最低版本等。创建的每个 Android 项目应用程序必须在根目录下包含一个 AndroidManifest.xml 工程文件。

AndroidManifest.xml 文件的代码：

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.        package="com.hisoft"
4.      >
5.        <application
6.          android:allowBackup="true"
7.          android:icon="@mipmap/ic_launcher"
8.          android:label="@string/app_name"
9.          android:roundIcon="@mipmap/ic_launcher_round"
10.         android:supportsRtl="true"
11.         android:theme="@style/AppTheme">
12.           <activity android:name=".HelloWorldActivity">
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.       </application>
19. </manifest>
```

AndroidManifest.xml 文件的根元素是 manifest，包含了 xmlns：android、package、android：versionCode 和 android：versionName 共 4 个属性。

xmlns：android 定义了 Android 的命名空间，值为 http://schemas.android.com/apk/res/android。

package 定义了应用程序的包名称。

manifest 元素仅能包含一个 application 元素，application 元素中能够声明 Android 程序中最重要的 4 个组成部分，包括 Activity、Service、BroadcastReceiver 和 ContentProvider，所定义的属性将影响所有组成部分。

第 6 行属性 android：allowBackup 定义是否允许备份应用的数据，默认是 true，表示用户可通过 adb backup 和 adb restore 来对应用数据进行备份和恢复。实际中为安全起见，建议开发者将 allowBackup 标志值设置为 false 来关闭应用程序的备份和恢复功能，以免造成信息泄露。

第 7 行属性 android：icon 定义了 Android 应用程序的图标，其中@mipmap/ic_launcher 是一种资源引用方式，表示资源类型是图像，资源名称为 ic_launcher，对应的资源文件为 res/ mipmap 目录下的 ic_launcher.png。

第 8 行属性 android：label 则定义了 Android 应用程序的标签名称。@string/app_name 同样属于资源引用，表示资源类型是字符串，资源名称为 app_name，资源保存在 res/values 目录下的 strings.xml 文件中。

第 9 行属性 android：roundIcon 与第 7 行属性类似，不同的是其引用了圆形图片。

第 10 行属性 android：supportsRtl 表示 application 是否支持从右到左（RTL，right-to-left）的布局。如果设置为 true，targetSdkVersion 设置应为 17（Android4.2 以后）或更高，各种 RTL 的 API 将被激活，应用程序可以显示 RTL 布局。如果 targetSdkVersion 设置为 16 或更低，RTL 的设置将被忽略。

第 11 行属性 android：theme 定义主题风格，其中的@style/AppTheme 是引用的 res/values/styles.xml 中的主题样式。

第 12 行属性 android：name 定义了实现 Activity 类的名称，可以是完整的类名称，也可以是简化后的类名称。

activity 元素是对 Activity 子类的声明，必须在 AndroidManifest.xml 文件中声明的 Activity 才能在用户界面中显示。

intent-filter 中声明了两个子元素 action 和 category，intent-filter 使 HelloAndroid 程序在启动时，将. HelloWorldActivity 这个 Activity 作为默认启动模块。

3.1.3 build 目录

在上文的目录结构中已讲述，在 app 目录下 build/generated/not_namespaced_r_class_sources/debug/processDebugResources/r/包名的目录下只存放一个由 ADT 自动生成，并不需要人工修改的 R.java 文件。

R.java 文件包含对 drawable、layout 和 values 目录内的资源的引用指针，Android 程序能够直接通过 R 类引用目录中的资源。

Android 系统中资源引用有两种方式：一种是在代码中引用资源；另一种是在资源中引用资源。

代码中引用资源需要使用资源的 ID，可以通过[R.resource_type.resource_name]或[android.R.resource_type.resource_name]获取。

resource_type 代表资源类型，也就是 R 类中的内部类名称。

resource_name 代表资源名称，对应资源的文件名或在 XML 文件中定义的资源名称属性。

资源中引用资源，引用格式：@[package：]type：name。

- @表示对资源的引用。
- package 是包名称，如果在相同的包，package 则可以省略。

R.java 文件不能手工修改，如果向资源目录中增加或删除资源文件，则需要在工程名称上右击，选择 Refresh 来更新 R.java 文件中的代码。

R 类包含的几个内部类，分别与资源类型相对应，资源 ID 便保存在这些内部类中，例

如子类 drawable 表示图像资源,内部的静态变量 icon 表示资源名称,其资源 ID 为 0x7f020000。一般情况下,资源名称与资源文件名相同。

3.1.4 res 目录

在 Android studio 环境下,res 目录位于 app/src/main/下,res 目录中包含了 9 个子目录,介绍如下。

drawable 目录:存放背景文件 ic_launcher_background.xml 文件,使用 SVG 格式绘制出带纹理的底图。

drawable-v24 目录:存放前景文件 ic_launcher_foreground.xml 文件,使用 SVG 格式绘制出一个带有投影效果的 Android 机器人 Logo。

layout 目录:用来保存与用户界面相关的布局文件,这些布局文件都是 XML 文件,默认存放的是 activity_main.xml 文件。

mipmap-anydpi-v26 目录:该文件夹用来存放自适应图标,用 ic_launcher.xml 和 ic_launcher_round.xml 两个文件表示,分别资源引用 drawable 目录中的。

mipmap-hdpi 目录:里面主要放高分辨率的图片,如 WVGA(480×800)、FWVGA(480×854),默认存放的是 ic_launcher.png 图片。

mipmap-mdpi 目录:里面主要放中等分辨率的图片,如 HVGA(320×480)。

mipmap-xhdpi 目录:里面主要放超高分辨率的图片,如 QVGA(720×1280)。

mipmap-xxhdpi 目录:里面主要放超超高分辨率的图片,如 QVGA(1080×1920)。

mipmap-xxxhdpi 目录:里面主要放超高分辨率的图片,如 2K 屏幕。

上述 mipmap 各个文件夹代表像素密度(dpi),mipmap 仅用于应用启动图标,可以根据不同分辨率进行优化。其他的图标资源,要放到 drawable 文件夹中。系统会根据机器的分辨率来分别到这几个文件夹里面去找对应的图片。

valuse 目录:保存文件颜色、风格、主题和字符串等,默认存放的是 strings.xml 文件。

上述 activity_main.xml 文件,是界面布局文件,利用 XML 语言描述的用户界面界面布局的相关内容将在后续章节用户界面设计中进行详细介绍。

(1) activity_main.xml 文件代码:

```
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.  <TextView
8.      android:layout_width="fill_parent"
9.      android:layout_height="wrap_content"
10.     android:text="@string/hello"
11.     />
12. </LinearLayout>
```

第 7 行的代码说明在界面中使用 TextView 控件，TextView 控件主要用来显示字符串文本。

第 10 行代码说明 TextView 控件需要显示的字符串，非常明显，@string/hello 是对资源的引用。

(2) strings.xml 文件代码：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, HelloWorldActivity!</string>
    <string name="app_name">HelloWorld</string>
</resources>
```

通过 strings.xml 文件的第 3 行代码分析，在 TextView 控件中显示的字符串应是 Hello World, HelloAndroidActivity!

如果修改 strings.xml 文件的第 3 行代码的内容，重新编译、运行后，模拟器中显示的结果也应该随之更改。

3.2 Android 应用程序组成

3.2.1 Android 应用程序概述

Android 应用程序是在 Android 应用框架之上，由一些系统自带和用户创建的应用程序组成。组件是可以调用的基本功能模块，Android 应用程序就是由组件组成的。一个 Android 的应用程序通常包含 4 个核心组件和一个 Intent，4 个核心组件分别是：Activity、Service、BroadcaseReceiver 和 ContentProvider。Intent 是组件之间进行通信的载体，它不仅可以在同一个应用中起传递信息的作用，还可以在不同的应用间传递信息，如图 3-2 所示。

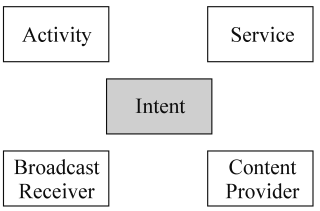

图 3-2 Android 应用程序组件

3.2.2 Activity 组件

Activity 是 Android 程序的呈现层，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 Android 应用程序可以包含一个或多个 Activity，其中一个作为 main Activity 用于启动显示，一般在程序启动后会呈现一个 Activity，用于提示用户程序已经正常启动。

Activity 通过 View 管理用户界面 UI。View 绘制用户界面 UI 与处理用户界面事件(UI event)，View 可通过 XML 描述定义，也可在代码中生成。一般情况下，Android 建议将 UI 设计和逻辑分离，Android UI 设计类似 swing，通过布局(layout)组织 UI 组件。

在应用程序中，每一个 Activity 都是一个单独的类，继承实现了 Activity 基础父类，这个类通过它的方法设置并显示由 Views 组成的用户界面 UI，并接受、响应与用户交互产生的界面事件。Activity 通过 startActivity 或 startActivityForResult 启动另外的 Activity。

在应用程序中一个 Activity 在界面上的表现形式通常有：全屏窗体、非全屏悬浮窗体、对话框等。

3.2.3 Service 组件

Service 常用于没有用户界面,但需要长时间在后台运行的应用。与应用程序的其他模块(例如 activity)一同运行于主线程中。一般通过 startService 或 bindService 方法创建 Service,通过 stopService 或 stopSelf 方法终止 Service。通常情况下,都在 Activity 中启动和终止 Service。

在 Android 应用中,Service 的典型应用是音乐播放器。在一个媒体播放器程序中,大概要有一个或多个活动(Activity)来供用户选择歌曲并播放它。然而,音乐的回放就不能使用活动(Activity)了,因为用户希望能够切换到其他界面时音乐继续播放。这种情况下,媒体播放器活动(Activity)要用 Context.startService()启动一个服务来在后台运行保持音乐的播放。系统将保持这个音乐回放服务的运行直到它结束。需要注意,你要用 Context.bindService()方法连接服务(如果它没有运行,要先启动它)。当连接到服务后,你可以通过服务暴露的一个接口和它通信。对于音乐服务,它支持暂停、倒带、重放等功能。

3.2.4 Intent 和 IntentFilter 组件

1. Intent

Android 中提供了 Intent 机制来协助应用间的交互与通信,Intent 负责对应用中一次操作的动作、动作涉及数据、附加数据进行描述。Android 则根据此 Intent 的描述,负责找到对应的组件,将 Intent 传递给调用的组件,并完成组件的调用。Intent 不仅可用于应用程序之间,也可用于应用程序内部的 Activity/Service 之间的交互。因此,Intent 在这里起着一个媒体中介的作用,类似于消息、事件通知,它充当 Activity、Service、broadcastReceiver 之间联系的桥梁,专门提供组件互相调用的相关信息,实现调用者与被调用者之间的解耦。具体详述见 8.1.2 节。

通常 Intent 分为显式和隐式两类。显式的 Intent,就是指定了组件名字,由程序指定具体的目标组件来处理。即在构造 Intent 对象时就指定接收者,指定了一个明确的组件(setComponent 或 setClass)来使用处理 Intent。

```
Intent intent = new Intent(
    getApplicationContext(),
    Test.class
);
startActivity(intent);
```

特别注意：被启动的 Activity 需要在 AndroidManifest.xml 中进行定义。

隐式的 Intent 就是没有指定 Intent 的组件名字,没有指定明确的组件来处理该 Intent。使用这种方式时,需要让 Intent 与应用中的 IntentFilter 描述表相匹配。需要 Android 根据 Intent 中的 Action、data、Category 等来解析匹配。由系统接收调用并决定如何处理,即 Intent 的发送者在构造 Intent 对象时,并不知道也不关心接收者是谁,有利于

降低发送者和接收者之间的耦合。如：startActivity(new Intent(Intent.ACTION_DIAL));。

```
Intent intent = new Intent();
intent.setAction("test.intent.IntentTest");
startActivity(intent);
```

目标组件(Activity、Service、Broadcast Receiver)是通过设置它们的 Intent Filter 来界定其处理的 Intent。如果一个组件没有定义 Intent Filter,那么它只能接受处理显式的 Intent,只有定义了 Intent Filter 的组件才能同时处理隐式和显式的 Intent。

一个 Intent 对象包含了很多数据的信息,由 6 部分组成:

- Action——要执行的动作。
- Data——执行动作要操作的数据。
- Category——被执行动作的附加信息。
- Extras——其他所有附加信息的集合。
- Type——显式指定 Intent 的数据类型(MIME)。
- Component——指定 Intent 的目标组件的类名称比如要执行的动作、类别、数据、附加信息等。

下面就一个 Intent 中包含的信息进行简要介绍。

1) Action

一个 Intent 的 Action 在很大程度上说明这个 Intent 要做什么,例如查看(View)、删除(Delete)、编辑(Edit)等。Action 是一个字符串命名的动作,Android 中预定义了很多 Action,可以参考 Intent 类查看,表 3-1 是 Android 文档中的几个动作。

表 3-1 Android 动作

Constant(全局常量)	Target component (目标组件)	Action (动作)
ACTION_CALL	Activity	调用拨打电话界面
ACTION_EDIT	Activity	提供显式的可使用户编辑访问给定的数据
ACTION_MAIN	Activity	发起一个任务或程序的起始 Activity,无数据输入和返回信息
ACTION_SYNC	Activity	用于移动设备上带有数据的服务的数据同步
ACTION_BATTERY_LOW	broadcast Receiver	调用电池电量低的系统警告对话框框
ACTION_HEADSET_PLUG	broadcast Receiver	无线耳机插入和拔出状态检测
ACTION_SCREEN_ON	broadcast Receiver	屏幕亮屏监听
ACTION_TIMEZONE_CHANGED	broadcast Receiver	时间时区设置更改

此外,用户也可以自定义 Action,比如 ACTION_CALL_BUTTON:拨打按钮,调用拨号程序。定义的 Action 最好能表明其所表示的意义,要做什么,这样 Intent 中的数据才

好填充。Intent 对象的 getAction()可以获取动作,使用 setAction()可以设置动作。

2) Data

其实就是一个 URI,用于执行一个 Action 时所用到的数据的 URI 和 MIME。不同的 Action 有不同的数据规格,比如 ACTION_EDIT 动作,数据就可以包含一个用于编辑文档的 URI。如果是一个 ACTION_CALL 动作,那么数据就是一个包含了 tel: 6546541 的数据字段,所以上面提到的自定义 Action 时要规范命名。数据的 URI 和类型对于 Intent 的匹配是很重要的,Android 往往根据数据的 URI 和 MIME 找到能处理该 Intent 的最佳目标组件。

3) component(组件)

指定 Intent 的目标组件的类名称。通常 Android 会根据 Intent 中包含的其他属性的信息,比如 Action、Data/Type、Category 进行查找,最终找到一个与之匹配的目标组件。

如果设置了 Intent 目标组件的名字,那么这个 Intent 就会被传递给特定的组件,而不再执行上述查找过程,指定了这个属性以后,Intent 的其他所有属性都是可选的。也就是我们说的显式 Intent。如果不设置,则是隐式的 Intent,Android 系统将根据 Intent Filter 中的信息进行匹配。

4) Category

Category 指定了用于处理 Intent 的组件的类型信息,一个 Intent 可以添加多个 Category,使用 addCategory()方法即可,使用 removeCategory()删除一个已经添加的类别。Android 的 Intent 类里定义了很多常用的类别,可以参考使用。

5) Extras

有些用于处理 Intent 的目标组件需要一些额外的信息,那么就可以通过 Intent 的 put..()方法把额外的信息塞入到 Intent 对象中,用于目标组件的使用,一个附件信息就是一个 key-value 的键值对。Intent 有一系列的 put 和 get 方法用于处理附加信息的塞入和取出。

2. IntentFilter

应用程序的组件为了告诉 Android 自己能响应、处理哪些隐式 Intent 请求,可以声明一个甚至多个 Intent Filter。每个 Intent Filter 描述该组件所能响应 Intent 请求的能力——组件希望接收什么类型的请求行为,什么类型的请求数据。比如在请求网页浏览器这个例子中,网页浏览器程序的 Intent Filter 就应该声明它所希望接收的 Intent Action 是 WEB_SEARCH_ACTION,以及与之相关的请求数据是网页地址 URI 格式。如何为组件声明自己的 Intent Filter? 常见的方法是在 AndroidManifest.xml 文件中用属性<Intent-Filter>描述组件的 Intent Filter。

Intent 解析机制主要是通过查找已注册在 AndroidManifest.xml 中的所有 IntentFilter 及其中定义的 Intent,最终找到匹配的 Intent。在这个解析过程中,Android 是通过 Intent 的 Action、Type、Category 这 3 个属性来进行判断的,判断方法如下。

- 如果 Intent 指明定了 Action,则目标组件的 IntentFilter 的 Action 列表中就必须包含有这个 Action,否则不能匹配。
- 如果 Intent 没有提供 Type,系统将从 Data 中得到数据类型。和 Action 一样,目标

组件的数据类型列表中必须包含 Intent 的数据类型，否则不能匹配。

- 如果 Intent 中的数据不是 content 类型的 URI，而且 Intent 也没有明确指定它的 type，将根据 Intent 中数据的 scheme（比如 http：或者 mailto：）进行匹配。同上，Intent 的 scheme 必须出现在目标组件的 scheme 列表中。
- 如果 Intent 指定了一个或多个 category，这些类别必须全部出现在组建的类别列表中。比如 Intent 中包含了两个类别：LAUNCHER _ CATEGORY 和 ALTERNATIVE_CATEGORY，解析得到的目标组件必须至少包含这两个类别。

一个 Intent 对象只能指定一个 action，而一个 IntentFilter 可以指定多个 action。action 的列表不能为空，否则它将组织所有的 intent。

一个 Intent 对象的 Action 必须和 IntentFilter 中的某一个 Action 匹配，才能通过测试。如果 IntentFilter 的 Action 列表为空，则不通过。如果 Intent 对象不指定 Action，并且 IntentFilter 的 Action 列表不为空，则通过测试。

下面针对 Intent 和 IntentFilter 中包含的子元素 Action（动作）、Data（数据）以及 Category（类别）进行比较并对具体规则进行详细介绍。

1）动作测试

＜intent-filter＞元素中可以包括子元素＜action＞，比如：

```
<intent-filter>
<action android:name="com.example.project.SHOW_CURRENT" />
<action android:name="com.example.project.SHOW_RECENT" />
<action android:name="com.example.project.SHOW_PENDING" />
</intent-filter>
```

一条＜intent-filter＞元素至少应该包含一个＜action＞，否则任何 Intent 请求都不能和该＜intent-filter＞匹配。如果 Intent 请求的 Action 和＜intent-filter＞中某一条＜action＞匹配，那么该 Intent 就通过了这条＜intent-filter＞的动作测试。如果 Intent 请求或＜intent-filter＞中没有说明具体的 Action 类型，那么会出现下面两种情况。

如果＜intent-filter＞中没有包含任何 Action 类型，那么无论什么 Intent 请求都无法和这条＜intent-filter＞匹配；

反之，如果 Intent 请求中没有设定 Action 类型，那么只要＜intent-filter＞中包含有 Action 类型，这个 Intent 请求就将顺利地通过＜intent-filter＞的行为测试。

2）类别测试

＜intent-filter＞元素可以包含＜category＞子元素，比如：

```
<intent-filter ...>
<category android:name="android.Intent.Category.DEFAULT" />
<category android:name="android.Intent.Category.BROWSABLE" />
</intent-filter>
```

只有当 Intent 请求中所有的 Category 与组件中某一个 IntentFilter 的＜category＞完全匹配时，才会让该 Intent 请求通过测试，IntentFilter 中多余的＜category＞声明并不会导致匹配失败。一个没有指定任何类别测试的 IntentFilter 仅仅只会匹配没有设置类别的 Intent 请求。

3) 数据测试

数据在<intent-filter>中的描述如下:

```
<intent-filter ... >
<data android:type="video/mpeg" android:scheme="http" ... />
<data android:type="audio/mpeg" android:scheme="http" ... />
</intent-filter>
```

<data>元素指定了希望接收的 Intent 请求的数据 URI 和数据类型,URI 被分成 3 部分来进行匹配: scheme、authority 和 path。其中,用 setData()设定的 Intent 请求的 URI 数据类型和 scheme 必须与 IntentFilter 中所指定的一致。若 IntentFilter 中还指定了 authority 或 path,它们也需要相匹配才会通过测试。

3.2.5 BroadcastReceiver 组件

在 Android 中,Broadcast 是一种广泛运用在应用程序之间传输信息的组件。而 BroadcastReceiver 是接收并响应广播消息的组件,对发送出来的 Broadcast 进行过滤接收并响应,它不包含任何用户界面,可以通过启动 Activity 或者 Notification 通知用户接收到重要信息,在 Notification 中有多种方法提示用户,如闪动背景灯、振动设备、发出声音或在状态栏上放置一个持久的图标。

BroadcastReceiver 过滤接收的过程如下。

在需要发送信息时,把要发送的信息和用于过滤的信息(如 Action、Category)装入一个 Intent 对象,然后通过调用 Context. sendBroadcast()、sendOrderBroadcast() 或 sendStickyBroadcast()方法,把 Intent 对象以广播的方式发送出去。

当 Intent 发送后,所有已经注册的 BroadcastReceiver 会检查注册时的 IntentFilter 是否与发送的 Intent 相匹配,若匹配则调用 BroadcastReceiver 的 onReceive()方法。因此在我们定义一个 BroadcastReceiver 时,通常都需要实现 onReceive()方法。

BroadcastReceiver 注册有两种方式:

一种方式是静态地在 AndroidManifest.xml 中用<receiver>标签声明注册,并在标签内用<intent-filter>标签设置过滤器;

另一种方式是动态地在代码中先定义并设置好一个 IntentFilter 对象,然后在需要注册的地方调用 Context.registerReceiver()方法,如果取消时就调用 Context.unregisterReceiver()方法。

不管是用 XML 注册的还是用代码注册的,在程序退出时,一般都需要注销,否则下次启动程序可能会有多个 BroadcastReceiver。另外,若在使用 sendBroadcast()的方法时指定了接收权限,则只有在 AndroidManifest.xml 中用<uses-permission>标签声明了拥有此权限的 BroascastReceiver 才会有可能接收到发送来的 Broadcast。

同样,若在注册 BroadcastReceiver 时指定了可接收的 Broadcast 的权限,则只有在包内的 AndroidManifest.xml 中用<uses-permission>标签声明了拥有此权限的 Context 对象所发送的 Broadcast 才能被这个 BroadcastReceiver 所接收。

3.2.6 ContentProvider 组件

ContentProvider 是 Android 系统提供的一种标准的共享数据的机制,在 Android 中

每一个应用程序的资源都为私有，应用程序可以通过 ContentProvider 组件访问其他应用程序的私有数据（私有数据可以是存储在文件系统中的文件，或者是存放在 SQLite 中的数据），如图 3-3 所示。

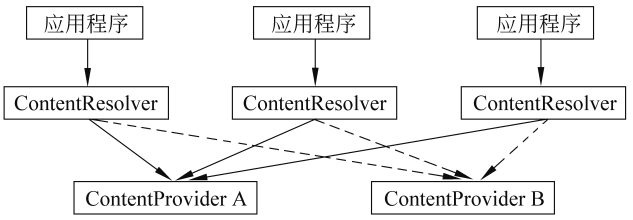

图 3-3　应用程序、ContentResolver 与 ContentProvider

对 ContentProvider 的使用，有两种方式：

- ContentResolver 访问。
- Context.getContentResolver()。

Android 系统内部也提供一些内置的 ContentProvider，能够为应用程序提供重要的数据信息。使用 ContentProvider 对外共享数据的好处是统一了数据的访问方式。

3.3　Android 生命周期

3.3.1　程序生命周期

程序的生命周期是指在 Android 系统中进程从启动到终止的所有阶段，也就是 Android 程序启动到停止的全过程。程序的生命周期是由 Android 系统进行调度和控制的。

Android 系统中的进程分为：前台进程、可见进程、服务进程、后台进程、空进程。

Android 系统中的进程优先级由高到低，如图 3-4 所示。

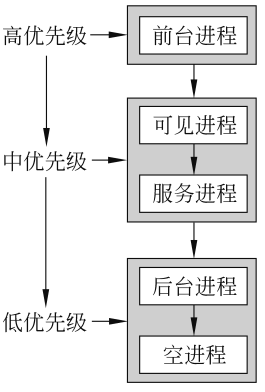

图 3-4　Android 系统的进程及优先级

1. 前台进程

前台进程是 Android 系统中最重要的进程，是指与用户正在交互的进程，包含以下 4 种情况：

- 进程中的 Activity 正在与用户进行交互。
- 进程服务被 Activity 调用，而且这个 Activity 正在与用户进行交互。
- 进程服务正在执行声明周期中的回调方法，如 onCreate()、onStart()或 onDestroy()。
- 进程的 BroadcastReceiver 正在执行 onReceive()方法。

Android 系统有多个前台进程同时运行时，可能会出现资源不足的情况，此时会清除部分前台进程，保证主要的用户界面能够及时响应。

2. 可见进程

可见进程指部分程序界面能够被用户看见,但不在前台与用户交互,不响应界面事件的进程。如果一个进程包含服务,且这个服务正在被用户可见的 Activity 调用,此进程同样被视为可见进程。

Android 系统一般存在少量的可见进程,只有在特殊的情况下,Android 系统才会为保证前台进程的资源而清除可见进程。

3. 服务进程

服务进程是指包含已启动服务的进程,通常特点:

- 没有用户界面。
- 在后台长期运行。

Android 系统在不能保证前台进程或可视进程所必要的资源时,才会强行清除服务进程。

4. 后台进程

后台进程是指不包含任何已经启动的服务,而且没有任何用户可见的 Activity 的进程。

Android 系统中一般存在数量较多的后台进程,在系统资源紧张时,系统将优先清除用户较长时间没有见到的后台进程。

5. 空进程

空进程是指不包含任何活跃组件的进程,空进程在系统资源紧张时会被首先清除。但为了提高 Android 系统应用程序的启动速度,Android 系统会将空进程保存在系统内存中,在用户重新启动该程序时,空进程会被重新使用。

除了以上的优先级外,以下两方面也决定它们的优先级:

- 进程的优先级取决于所有组件中的优先级最高的部分。
- 进程的优先级会根据与其他进程的依赖关系而变化。

3.3.2 组件生命周期

所有 Android 组件都具有自己的生命周期,是指从组件的建立到组件的销毁整个过程。在生命周期中,组件会在可见、不可见、活动、非活动等状态中不断变化。下面就各个组件的生命周期逐一进行讲述。

1. Service 生命周期

Service 组件通常没有用户界面 UI,其启动后一直运行于后台;它与应用程序的其他模块(如 activity)一同运行于程序的主线程中。

一个 Service 的生命周期通常包含:创建、启动、销毁这几个过程。

Service 只继承了 onCreate()、onStart()、onDestroy()3 个方法,当第一次启动 Service 时,先后调用了 onCreate()、onStart()这两个方法;当停止 Service 时,则执行 onDestroy()方法。需要注意的是,如果 Service 已经启动了,当再次启动 Service 时,不会再执行

onCreate()方法，而是直接执行 onStart()方法。

创建 Service 的方式有两种，一种是通过 startService 创建，另外一种是通过 bindService 创建 Service。两种创建方式的区别在于，startService 是创建并启动 Service，而 bindService 只是创建了一个 Service 实例并取得了一个与该 Service 关联的 binder 对象，但没有启动它，如图 3-5 所示。

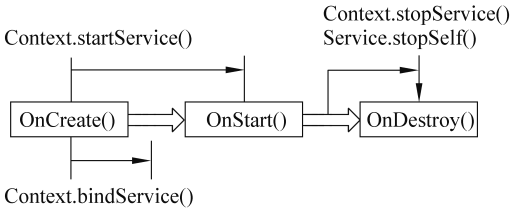

图 3-5 Service 的生命周期

如果没有程序停止它或者它没有自己停止，Service 将一直运行。在这种模式下，Service 开始于调用 Context.startService()，停止于 Context.stopService()。Service 可以通过调用 stopService() 或 Service.stopSelfResult() 停止自己。不管调用多少次 startService()，只需要调用一次 stopService()就可以停止 Service。一般在 Activity 中启动和终止 Service。它可以通过接口被外部程序调用。外部程序建立到 Service 的连接，通过连接来操作 Service。建立连接开始于 Context.bindService()，结束于 Context.unbindService()。多个客户端可以绑定到同一个 Service，如果 Service 没有启动，bindService()可以选择启动它。

上述两种方式不是完全分离的。通过 startService()启动的服务。如一个 Intent 想要播放音乐，通过 startService() 方法启动后台播放音乐的 Service。然后，用户想要操作播放器或者获取当前正在播放的乐曲的信息，一个 Activity 就会通过 bindService()建立一个到这个 Service 的连接。这种情况下 stopService() 在全部的连接关闭后才会真正停止 Service。

像 Activity 一样，Service 也有可以通过监视状态实现的生命周期。但是比 Activity 要少，通常只有 3 种方法，并且是 public 而不是 protected 的属性的，具体如下：

```
1.  void onCreate()
2.  void onStart(Intent intent)
3.  void onDestroy()
```

通过实现上述 3 种方法，可以监视 Service 生命周期的两个嵌套循环：

整个生命周期从 onCreate()开始，到 onDestroy()结束，像 Activity 一样，一个 Android Service 的生命周期在 onCreate()中执行初始化操作，在 onDestroy()中释放所有用到的资源。如：后台播放音乐的 Service 可能在 onCreate()创建一个播放音乐的线程，在 onDestroy()中销毁这个线程。

活动生命周期开始于 onStart()。这个方法处理传入到 startService()方法的 Intent。音乐服务会打开 Intent 查看要播放哪首歌曲，并开始播放。当服务停止的时候，没有方法检测到(没有 onStop()方法)、onCreate()和 onDestroy()用于所有通过 Context.startService() or

Context.bindService()启动的 Service。onStart()只用于通过 startService()开始的 Service。

如果一个 Android Service 生命周期是可以从外部绑定的,它就可以触发以下的方法:

```
1.  IBinder onBind(Intent intent)
2.  boolean onUnbind(Intent intent)
3.  void onRebind(Intent intent)
```

onBind()回调被传递给调用 bindService 的 Intent、onUnbind()被 unbindService()中的 intent 处理。如果服务允许被绑定,那么 onBind()方法返回客户端和 Service 的沟通通道。如果一个新的客户端连接到服务,onUnbind()会触发 onRebind()调用。

后续案例将会讲解说明 Service 的回调方法。将通过 startService 和 bindService()启动的 Service 分开了,但是要注意不管它们是怎么启动的,都有可能被客户端连接,因此都有可能触发到 onBind()和 onUnbind()方法。

2. Service 生命周期应用案例

具体实现步骤如下。

(1) 在 Android Studio 中选择 File→New→New Project,创建一个新的 Android 工程,项目名为 ServiceTestDemo,目标 API 选择 28(即 Android 9.0 版本),应用程序名为 ServiceTestDemo,包名为 com.hisoft.activity,创建的 Activity 的名字为 ServiceTestDemoActivity,最小 SDK 版本根据选择的目标 API 会自动添加为 15。

(2) 修改 res 目录下 layout 文件夹中 main.xml 代码,添加 4 个 Button 控件,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.      <TextView
8.          android:id="@+id/text"
9.          android:layout_width="fill_parent"
10.         android:layout_height="wrap_content"
11.         android:text="@string/hello"
12.         />
13.     <Button
14.         android:id="@+id/startservice"
15.         android:layout_width="fill_parent"
16.         android:layout_height="wrap_content"
17.         android:text="启动 Service"
18.     />
19.     <Button
```

```
        android:id="@+id/stopservice"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="停止 Service"
    />
    <Button
        android:id="@+id/bindservice"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="绑定 Service"
    />
    <Button
        android:id="@+id/unbindservice"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="解除 Service"
    />
</LinearLayout>
```

(3) 在上述包下,新建一个 Service,命名为 MyService.java,代码如下:

```
package com.hisoft.service;
import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.text.format.Time;
import android.util.Log;
public class MyService extends Service {
//定义一个 Tag 标签
private static final String TAG ="TestService";
//这里定义把一个 Binder 类用在 onBind()方法里,这样 Activity 可以获取到
private MyBinder mBinder =new MyBinder();
@Override
public IBinder onBind(Intent intent) {
Log.e(TAG, " -----start IBinder-----~~~");
return mBinder;
}
@Override
public void onCreate() {
Log.e(TAG, "-----start onCreate-----");
super.onCreate();
}

@Override
```

```
25. public void onStart(Intent intent, int startId) {
26. Log.e(TAG, "-----start onStart-----");
27. super.onStart(intent, startId);
28. }
29.
30. @Override
31. public void onDestroy() {
32. Log.e(TAG, "-----start onDestroy-----");
33. super.onDestroy();
34. }
35.
36.
37. @Override
38. public boolean onUnbind(Intent intent) {
39. Log.e(TAG, "-----start onUnbind-----");
40. return super.onUnbind(intent);
41. }
42.
43. //这里是一个获取当前时间的函数,不过没有格式化
44. public String getSystemTime(){
45.
46. Time t =new Time();
47. t.setToNow();
48. return t.toString();
49. }
50.
51. public class MyBinder extends Binder{
52. MyService getService()
53. {
54. return MyService.this;
55. }
56. }
57. }
```

(4) 修改 com.hisoft.activity 包下的 ServiceTestDemoActivity.java 文件,代码如下:

```
1.  package com.hisoft.activity;
2.  import android.app.Activity;
3.  import android.content.ComponentName;
4.  import android.content.Context;
5.  import android.content.Intent;
6.  import android.content.ServiceConnection;
7.  import android.os.Bundle;
8.  import android.os.IBinder;
9.  import android.view.View;
```

```
10. import android.view.View.OnClickListener;
11. import android.widget.Button;
12. import android.widget.TextView;
13. public class ServiceTestDemoActivity extends Activity implements
        OnClickListener{
14.     private MyService  mMyService;
15.     private TextView mTextView;
16.     private Button startServiceButton;
17.     private Button stopServiceButton;
18.     private Button bindServiceButton;
19.     private Button unbindServiceButton;
20.     private Context mContext;
21.     //这里需要用到 ServiceConnection,在 Context.bindService()和 Context.
        //unBindService()里用到
22.     private ServiceConnection mServiceConnection = new ServiceConnection() {
23.     //当单击 bindService 按钮时,让 TextView 显示 MyService 里 getSystemTime()
        //方法的返回值
24.         public void onServiceConnected(ComponentName name, IBinder service) {
25.             //TODO Auto-generated method stub
26.             //mMyService = ((MyService.MyBinder)service).getService();
27.         }
28.
29.         public void onServiceDisconnected(ComponentName name) {
30.             //TODO Auto-generated method stub
31.
32.         }
33.     };
34.     public void onCreate(Bundle savedInstanceState) {
35.         super.onCreate(savedInstanceState);
36.         setContentView(R.layout.main);
37.         setupViews();
38.     }
39.
40.     public void setupViews(){
41.
42.         mContext = ServiceDemo.this;
43.         mTextView = (TextView)findViewById(R.id.text);
44.
45.         startServiceButton = (Button)findViewById(R.id.startservice);
46.         stopServiceButton = (Button)findViewById(R.id.stopservice);
47.         bindServiceButton = (Button)findViewById(R.id.bindservice);
48.         unbindServiceButton = (Button)findViewById(R.id.unbindservice);
49.
```

```
        startServiceButton.setOnClickListener(this);
        stopServiceButton.setOnClickListener(this);
        bindServiceButton.setOnClickListener(this);
        unbindServiceButton.setOnClickListener(this);
    }

    public void onClick(View v) {
        //TODO Auto-generated method stub
        if(v == startServiceButton){
            Intent i  = new Intent();
            i.setClass(ServiceTestDemoActivity.this, MyService.class);
            mContext.startService(i);
        }else if(v == stopServiceButton){
            Intent i  = new Intent();
            i.setClass(ServiceTestDemoActivity.this, MyService.class);
            mContext.stopService(i);
        }else if(v == bindServiceButton){
            Intent i  = new Intent();
            i.setClass(ServiceTestDemoActivity.this, MyService.class);
            mContext.bindService(i, mServiceConnection, BIND_AUTO_CREATE);
        }else{
            mContext.unbindService(mServiceConnection);
        }
    }

}
```

(5) 修改 AndroidManifest.xml 代码,在<application>标签下的根目录下,添加注册新创建的 MyService,如下代码第 14 行:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.hisoft.activity "
android:versionCode="1"
android:versionName="1.0">
<application android:icon="@drawable/icon" android:label=
"@string/app_name">
<activity android:name=". ServiceTestDemoActivity"
android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
```

```
14. <service android:name=".MyService" android:exported="true"></service>
15. </application>
16. </manifest>
```

(6) 部署工程,并执行上述工程,运行结果如图 3-6 所示。

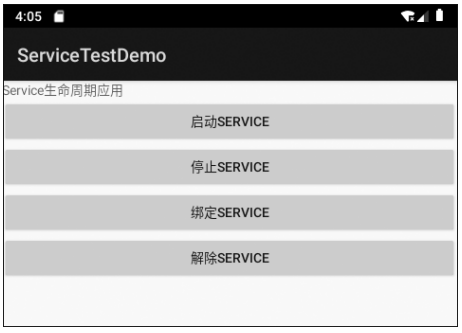

图 3-6 Service 生命周期运行界面

① 单击"启动 SERVICE"按钮时,程序先后执行了 Service 中 onCreate()、onStart()这两个方法,打开日志界面 Logcat 视窗,显示如图 3-7 所示内容。

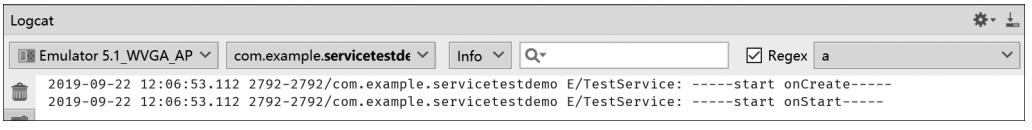

图 3-7 启动 Service 调用顺序

② 然后单击 Home 键进入 Settings(设置)→Applications(应用)→Running Services(正在运行的服务)查看刚才新启动的服务,如图 3-8 所示。

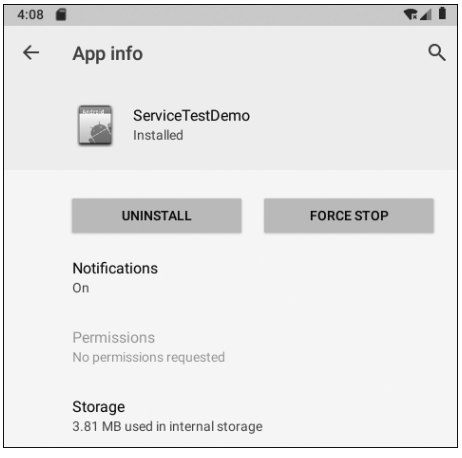

图 3-8 新启动的 Service 服务

③ 单击"停止 SERVICE"按钮时,Service 则执行了 onDestroy()方法,如图 3-9 所示。

④ 再次单击"启动 SERVICE"按钮,然后再单击 bindService 按钮(通常 bindService 都