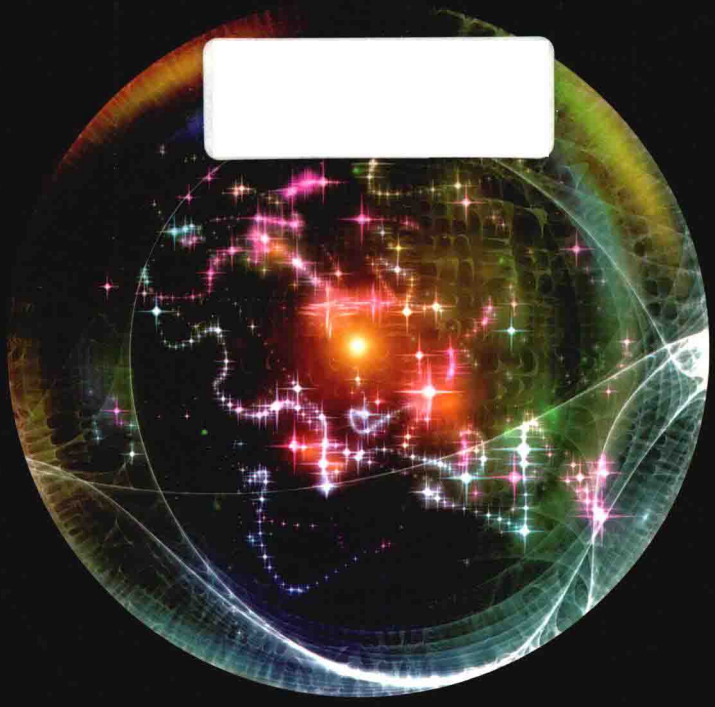
异步图书
www.epubit.com.cn

- 全面剖析进程 / 线程、内存管理、Binder 机制、GUI 显示系统、多媒体管理、输入系统、Android Dalvik/Art 虚拟机、Android 的安全机制、Gradle 自动化构建工具等核心知识在 Android 系统中的设计思想
- 通过大量图片与实例来引导读者学习，以求尽量在源码分析外，为读者提供更易于理解的思维路径
- 由浅入深，由总体框架再到细节实现，帮助读者彻底理解 Android 内核的实现原理

深入理解 Android 内核设计思想

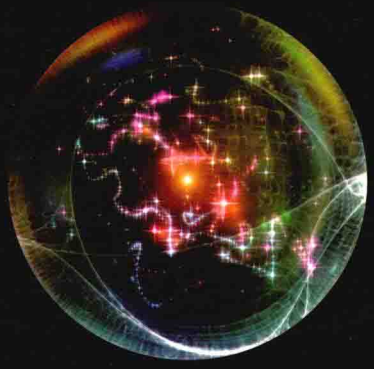
第 2 版 | 下册

林学森 ◆ 著



中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

深入理解
Android内核设计思想 第2版



本书学习路线图

Android 系统框架、建立编译环境、Android 系统映像文件、第三方ROM 的移植、Android编译系统、Android 系统GDB 调试、Android进程/线程和内存优化、 Android 应用程序如何利用CPU 的多核处理能力、Android 中的内存泄露与内存监测、进程间通信、进程间的数据传递载体、Android启动过程、系统关键服务的启动简析、多用户管理、管理 Activity和组件运行状态的系统进程、GUI 系统之Surfaceflinger、“黄油计划”（Project Butter）、GUI 系统之“窗口管理员”（WMS）、应用程序中的View 框架、“作画“工具集（Canvas）、 View 动画、事件的分类、事件注入、音频系统、音频采样、音频系统的核心（Audio Flinger）、音频数据流、Android安全机制透析、Android 系统安全保护的三重利剑、 APK 的加固保护分析Intent的匹配规则、Android虚拟机、垃圾回收、Art 虚拟机整体框架、Android 虚拟机的典型启动流程、堆管理器和堆空间释义、Art 虚拟机的“中枢系统”、APK 应用程序的资源适配、Android字符编码格式、Android和OpenGL ES、“系统的 UI”、Android 系统工具、IDE 和Gradle、Gradle 的核心要点、Android常用的工具“小插件”（Widget机制）、Android应用程序的编译和打包、软件版本管理、系统调试辅助工具等。

异步社区
人民邮电出版社
www.epubit.com.cn

异步社区 www.epubit.com.cn
新浪微博 @人邮异步社区
投稿/反馈邮箱 contact@epubit.com.cn

封面设计：董志桢

分类建议：计算机／程序设计／移动开发
人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-45263-4

ISBN 978-7-115-45263-4
定价：158.00 元（上、下册）

深入理解 Android内核设计思想

第2版 | 下册

林学森 ◆ 著

人民邮电出版社
北京

图书在版编目（CIP）数据

深入理解Android内核设计思想 : 全2册 / 林学森著
. -- 2版. -- 北京 : 人民邮电出版社, 2017.7
ISBN 978-7-115-45263-4

Ⅰ. ①深… Ⅱ. ①林… Ⅲ. ①移动终端－应用程序－
程序设计 Ⅳ. ①TN929.53

中国版本图书馆CIP数据核字(2017)第105893号

内 容 提 要

全书从操作系统的基础知识入手，全面剖析进程/线程、内存管理、Binder 机制、GUI 显示系统、多媒体管理、输入系统、虚拟机等核心技术在 Android 中的实现原理。书中讲述的知识点大部分来源于工程项目研发，因而具有较强的实用性，希望可以让读者“知其然，更知其所以然”。本书分为编译篇、系统原理篇、应用原理篇、系统工具篇，共 4 篇 25 章，基本涵盖了参与 Android 开发所需具备的知识，并通过大量图片与实例来引导读者学习，以求尽量在源码分析外为读者提供更易于理解的思维方式。

本书既适合 Android 系统工程师，也适合于应用开发工程师来阅读，从而提升 Android 开发能力。读者可以在本书潜移默化的学习过程中更深刻地理解 Android 系统，并将所学知识自然地应用到实际开发难题的解决中。

◆ 著　　林学森
责任编辑　张　涛
责任印制　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：787×1092　1/16
印张：63.75
字数：1681 千字　　2017 年 7 月第 2 版
印数：11 001 – 14 000 册　　2017 年 7 月河北第 1 次印刷

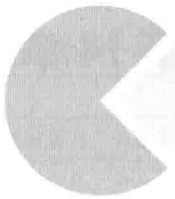
定价：158.00 元（上、下册）

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

目　录

第 3 篇　应用原理篇

第 4 篇　Android 系统工具

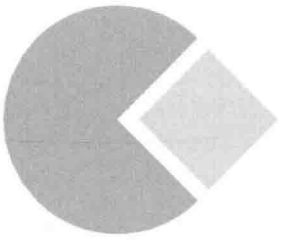


第 3 篇

应用原理篇

操作系统对于用户来说最大的价值就体现在应用程序中——无论是 Windows、iOS 还是 Android，生态系统都无一不是建立在相当规模数量的应用程序上。因而如何为 APK 应用提供高效的开发、调试与发布机制；并针对研发中的常见问题提供统一的解决方案，是保证 Android 系统“长盛不衰”的法宝。

阅读本篇内容，希望读者已经清楚 Android 应用程序的开发流程以及常用控件、布局的使用。做到这一点并不难——只要依照 Android 的 SDK 向导尝试建立一个“HelloWorld”入门级应用，然后加入一个想实现的功能，如使用 MediaPlayer 类来完成简易的音乐播放器即可。



第 14 章　Intent 的匹配规则

写过 Android 应用程序的开发者对 Intent 一定不陌生。从字面来看，Intent 是“意愿”“意图”——这个名称很好地表达了 Intent 设计者的“初衷”。如果说 Binder 是程序间的“沟通媒介”，那么 Intent 就是系统判断需要在哪两个进程间建立 Binder 关联的重要考量因素。

打个比方，“婚介所”（Android）在匹配一对新人（进程）时，假设女方的要求（Intent）是“年龄在 30 岁以下，有房，有车”，那么“媒人”就需要从一堆资料中找到年纪不超过 30 并且有房有车的男士。如果恰好有多个目标对象同时满足要求怎么办呢？有两种选择：

- 由系统自动按某种优先级进行比较，如有房有车的，且又长得帅的，那么就可以择优录取；
- 或者把这些资料都列出来，由女生自己来选择。

后一种方式比较民主，因为“萝卜青菜，各有所爱”，我们确实没有办法准确知道每个人的喜好——这也是 Android 系统采用的解决方法。当系统匹配出有多个目标进程符合 Intent 的“意愿”时，它会弹出提示框让用户自己选择。

Intent 并不是某个组件（如 Activity）专用的，它所承载的“意愿”可以来源于多种发起者。也就是说，上面的“婚介所”既可以为女生提供寻找优秀男士的服务，也可以为男生找寻合适的另一半“牵线搭桥”。不管发起方是谁，目标对象又是谁，他们所采用的“意愿”格式都不会发生本质的改变，只不过在条件的填写上有所不同罢了。

14.1 Intent 属性

在 Android 系统的设计中，Intent 可以被应用于除 ContentProvider 外的其他 3 种组件（即 Activity、Service 和 BroadcastReceiver）。因此，Intent 的属性定义一定要有共通性。下面我们来具体看看它有哪些可选属性。

- Component Name

Intent 可以被分为两类，即显性（Explicit Intents）和隐性（Implicit Intents）。如果我们在 Intent 中特别指定了目标方的“Component Name”，比如：“com.example.project.HelloActivity”；同时指定它所在的 PackageName，如“com.example.project”，那么系统就会直接将此 Intent 发往这个特定的应用，而不需要做额外的匹配工作。举个例子，如果女方的择偶要求规定男方的名字就得叫作“某某人”，且带有详细的居住地址（这样才能唯一确定一个人，因为同名的人肯定是存在的），那么婚介所的工作就简单了，直接约见住在这一地址的叫作“某某人”的男生便可。当然这种情况比较少见，因为它降低了女方的可选择性，而且前提是女生事先就已经认识这位男士。如果一个应用程序通过这种方式显性调用另一个应用进程，多半是这两个应用程序同属于一家研发公司，或者是同一进程中的两个组件。

- Category

Category 是“种类”的意思，它将 Intent 从大的方向上进行了区分和归类。如果说上面的 Component Name 是某人的名字，那么 Category 就好比国籍。Android 系统中已经预设了一些“国家”，

我们摘录其中的部分核心元素。另外，由于 Intent 的所有属性值实际上都只是一串字符，因而是可以自定义的。我们从它的 addCategory(String)方法的入参也可以推测到这点，如表 14-1 所示。

表 14-1 系统预设的 Category（节选）

Category Name	Description
CATEGORY_BROWSABLE	目标方能解析并正确显示网页链接所指向的内容，如图片、E-mail 等
CATEGORY_SAMPLE_CODE	这个应用程序是一个 Sample
CATEGORY_TEST	这个应用程序将被当成测试使用
CATEGORY_HOME	这个应用程序是系统启动后的第一个应用，即 Launcher
CATEGORY_LAUNCHER	这个应用程序可以通过点击 Launcher 中的程序图标来启动
CATEGORY_PREFERENCE	这个应用程序是 preference panel

- Action

动作。表明要做什么，或者什么事件发生了（常用于广播的情况。比如设备开机时会有系统广播发出，如果应用程序希望实现开机自启动，就可以监听这个广播）。和 Category 一样，用户也可以自定义一项唯一的 Action，如“com.ThinkingInAndroid.action.example”。表 14-2 是 Android 系统中预定义的部分常见 Action，读者可以参考一下。

表 14-2 系统预设的 Action（节选）

Action Name	Description
ACTION_CALL	希望启动一个可以拨打电话的 Activity
ACTION_EDIT	希望启动一个提供编辑功能的 Activity，通常是和其他属性配合使用
ACTION_MAIN	启动的 Activity 是其应用程序的初始界面
ACTION_SYNC	启动的 Activity 可以与服务器进行数据同步
ACTION_BATTERY_LOW	从这一项开始到表格末尾的 ACTION 都属于“事件”通知，它们面向 Broadcast 组件。比如 ACTION_BATTERY_LOW 是当电池电量低时发出的
ACTION_HEADSET_PLUG	当耳机插入或者拔出时发出
ACTION_TIMEZONE_CHANGED	时区变更时发出
ACTION_SCREEN_ON	屏幕开启时发出

- Data

打个比方，如果上面的 Action 中表明了某人去公安局出入境处“办理签证”的“动作”，那么这里的 Data 就作为“签证”业务的补充材料——比如这个人的名字、身份证件等。所以，Action 理论上是围绕 Data 提供的数据来开展业务的。当然也有不需要 Data 补充信息的情况，如在 ACTION_CALL 的情况下，电话号码是必须作为 Data 来传递的；而针对 Broadcast（如 ACTION_SCREEN_ON）组件的 Action，它们本身就蕴含了足够的信息，因而不需要 Data 的支持。

不同的 Action，其对应的 Data 格式会有所差异，我们在下一小节的匹配过程中再进一步讲解。

- Extras

Extras 可以理解为 Extra Data，它是对上面 Data 属性的补充。不过两者在数据的格式上有明显区别。Data 采用了类似 scheme://uri 的表达方式；而 Extras 则是一种键值对实现。它们在表达不同场景的数据时有各自的优势，使用者应该“具体问题具体分析”。发送方通过一系列 putXXX()方法将键值对存入 Intent 中，然后接收方就可以用相对应的 getXXX()来获取到这些 Extra 数据。这些方法的内部会维护一个 Bundle 对象来保证进程间数据的准确传输。

- Flags

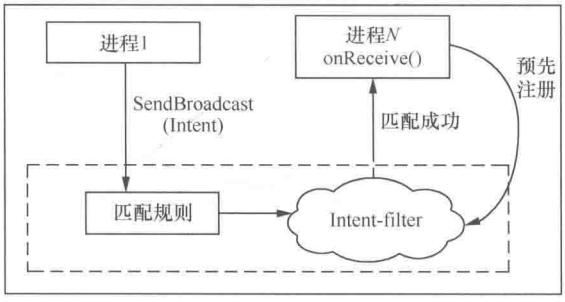
Flags 和 Activity 中的 LaunchMode 功能基本相同，它规定了系统如何去启动一个 Activity（比如指定即将启动的 Activity 应该属于哪一个 Task）。举个例子，如果一开始 Task 栈从下往上的顺序是 A-B-C，随后 C 通过带有 Flag 为 FLAG_ACTIVITY_CLEAR_TOP 的 Intent 来启动另一个 Activity。如果该 Activity 在栈中已经存在，比如说是 B，那么启动后的 Task 栈就变成了 A-B；否则该 Activity 直接入栈，而不会先清理栈顶。

14.2 Intent 的匹配规则

Intent 是和 Intent-filter 配套使用的。具体而言，Intent-filter 是每个组件的属性标签，它们在 AndroidManifest.xml 声明时就已经“贴上”了。而 Intent 则是程序运行过程中产生的实时“需求”。系统接收到这些请求后与现有的 Intent-filter 进行匹配，然后选择最合适的组件元素以响应。

还以前述的“婚介所”为例子，Intent 代表了女生的择偶意愿，而 Intent-filter 则是众男士的属性描述——年龄、长相、收入等。

图 14-1 描述了广播 BroadcastReceiver 的匹配流程，其他组件也类似。



▲图 14-1　Broadcast 匹配流程图

从图中可以看到，Intent 的典型匹配过程包括如下几个步骤。

Intent 匹配流程步骤 1：组件注册

当应用程序安装到系统中时，Android 会通过扫描它的 AndroidManifest 文件来得到一系列信息——这其中就包括了它的<intent-filter>；而且系统还会根据组件的不同进行相应的细化归类。比如 Activity 和 Service，它们响应 Intent 的场合是不同的（前者是 startActivity，后者则是 startService）。换句话说，当某人调用了 startActivity 后，显然系统只需在所有 Activity 中做匹配即可，而不应该再去考虑 Service 中的 Intent-filter。

除了在 AndroidManifest 中静态注册外，BroadcastReceiver 还可以在程序运行过程中进行动态注册。

这两种方式的区别如下：

- 静态注册

所谓静态，即不是在运行过程中执行的。应用程序事先将 Intent-filter 书写到 AndroidManifest 文件中。

范例如下：

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
<activity android:name=".MainActivity"android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
```

```
</intent-filter>
</activity>
<receiver android:name="Receiver1">#receiver 组件
     <intent-filter> #用 intent-filter 来匹配一个感兴趣的广播
          <action android:name="android.intent.action.BOOT_COMPLETED"/>
     </intent-filter>
</receiver>
<receiver android:name="Receiver2"> #系统对应用进程注册的 filter 个数没有限制
     <intent-filter>
          <action android:name="android.intent.action. BATTERY_LOW"/>
     </intent-filter>
</receiver>
</application>
```

- 动态注册

动态是指注册操作发生在程序运行过程中。这种注册方式的典型范例如下所示：

```
IntentFilter filter = new IntentFilter("android.intent.action.BOOT_COMPLETED");//生成
IntentFliter 对象
DynamicBroadcastReceiver br = new DynamicBroadcastReceiver();
registerReceiver(new DynamicBroadcastReceiver(), filter);//动态注册
…
class DynamicBroadcastReceiver extends BroadcastReceiver//扩展 BroadcastReceiver 类
{
    public void onReceive(Context context, Intent intent) //当匹配成功时的回调函数
    {
        //Intent 是通过 sendBroadcast 发送出来的
        if(android.intent.action.BOOT_COMPLETED.equals(intent.getAction)
        {…////相应处理
        }
        else
        {…////其他处理
        }
    }
}
```

Intent 匹配流程步骤 2：发起方主动向系统提供 Intent

这一步是在程序运行过程中发生的，此时系统已经掌握了所有组件的<intent-filter>信息。发起方根据自己的需求填写 Intent，并按照目标方是 Activity，Service 还是 BroadcastReceiver 来调用对应的函数。如下所示：

```
Activity→对应 startActivity();
Service →对应 startService();
BroadcastReceiver→对应 sendBroadcast();
```

应该清楚的是，系统会严格区分对待这些组件分类。所以利用 startActivity()是绝对不可能启动 Service 的——即便 Intent 和 intent-filter 能成功匹配。

Intent 匹配流程步骤 3：系统将 Intent 和对应组件类型（Activity，Service 等）里所有的 intent-filter 进行匹配，以寻找最佳的结果。

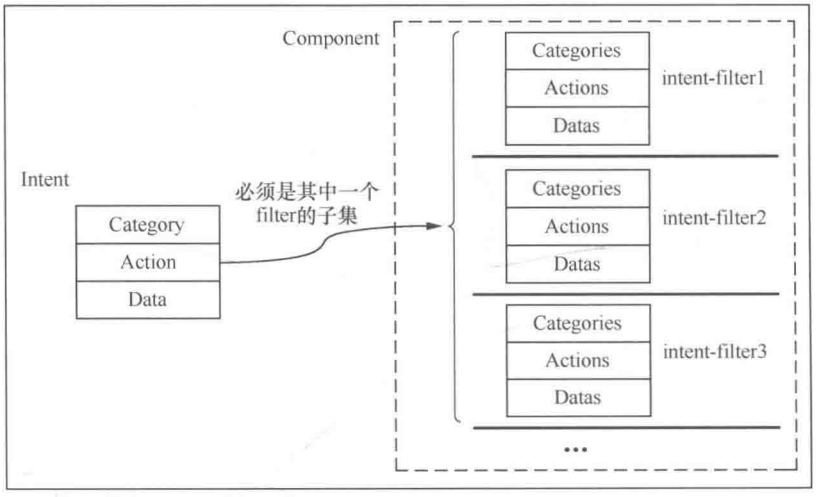
这是整个匹配流程中的核心。前面说过，Intent 分为显性和隐性两种。如果是显性的情况就不需要做匹配了，直接根据 Intent 中的 Component Name 来启动目标组件（即便这个组件没有声明任何 intent-filter）。所以，下面我们以隐性 Intent 为例来讲解匹配的规则。

读者在学习的过程中也可以和下一章节的“资源最佳匹配过程”进行比较，以加深理解。总的来说，Intent 的最终匹配结果可以是多个，而资源匹配则只会有一个胜出者。

影响 Intent 匹配规则的只有 3 个关键因素，即：

- Category；
- Action；
- Data（URI 和数据类型都要同时匹配）。

而其余两个属性 Extras 和 Flags 则只有在选中的组件运行后才能起作用。和资源匹配不同的是，一个组件可以同时声明多个 intent-filter。而在匹配过程中，只要它包含的任何一个 filter 通过了测试，它就会被选中。如果以“身份”来比喻一个组件，这相当于它可以同时拥有多个国籍（Category），或者承担多份工作（Action，如在中国是教师，在美国是牧师，这是有可能的），以及每份工作所包含的数据（Data）不同。在匹配测试中，系统遵循“子集”的概念。换句话说，Intent 中的 3 个关键因素至少要是该组件所包含的其中一个 intent-filter 的子集（Data 有点特殊，下面有详细解释），才能通过验证，如图 14-2 所示。



▲图 14-2　Intent 和 Intent-filters

图中有几个地方需要向读者解释一下。

- 每个 Component（Activity、Service、BroadcastReceiver）都可以有若干个 intent-filter。这和资源匹配不同，因为每种资源只能有一种标签。
- 影响匹配的只有 Category、Action 和 Data，不包括 Extras 和 Flags。
- 每个 filter 里的上述 3 种属性都可以不是唯一的（因此我们在图中标注的是复数形式），可以参照下面的例子。
- 匹配时，Intent 中的 3 种属性都需要通过测试。接下来，我们逐个分析各关键因素在匹配时采用的详细规则。

Category 测试：

Intent 中的 Category 需要在 filter 中找到完全匹配的项，才能通过测试。有的读者可能会问：那如果 Intent 中不指定任何 Category，是不是一定能通过检查？理论上确实是这样的。不过 Android 系统已经预料到这种情况，因而这里有个特殊的地方需要注意——如果 Intent 中不指定任何 Category，系统会自动为其添加一个“android.intent.category.DEFAULT”。这也就是想接收隐性 Intent 的 filter 里一般都要添加一个 DEFAULT 类型 category 的原因。当然，如果 Filter 里已经带有“android.intent.action.MAIN”和“android.intent.category.LAUNCHER”，就可以不用再另外书写 DEFAULT 了，这是一个例外。

Action 测试：

Action 的匹配相对简单，概括起来只有 3 点。

- 如果 filter 没有任何 action，那么所有 intent 都无法通过测试。
- 如果 filter 中的 action 不为空，那么 intent 中不带 action 的 filter 可以通过这项属性测试。

- 如果 filter 中的 action 不为空，且 intent 中的 action 也不为空，那么后者必须是前者的子集才能通过测试。读者可以参见下面的 Note Pad 例子以帮助理解。

Data 测试：

3 项属性测试中最复杂的是 Data 检验，Data 的内容通常包括两部分。

- MIME Type

Multipurpose Internet Mail Extensions（MIME）最早是电子邮件协议 SMTP（RFC 2046）中所规定的 Internet Email 的文件格式类型，后来逐渐被运用于其他多种互联网协议（比如 HTTP、RTP 等）中。它由两部分组成，即主类型（type）和子类型（subtype），中间以“/”分隔。

MIMEType 的目的很简单，就是指明某段数据是什么格式类型，以保证程序能正确解析处理。比如电子邮件中的附件，如果不特别说明，接收方客户端就没办法知道它们是图片、文本还是应用程序。这样的结果就是用户找不到合适的途径来对附件进行解析。国际标准中已经预设了很多常见的文件类型，举例如下。

➢ Application

```
application/javascript   ##Javascript 或者 ECMAScript 类型
application/pdf          ##Portable Document Format
application/zip          ##ZIP 文档
```

➢ Audio

```
audio/mp4                ##mp4 音频文件
audio/mpeg               ##MP3 或者其他 MPEG 音频
audio/ogg                ##Ogg Vorbis，Speex 等
```

➢ Video

```
video/mp4                ##MP4 视频文件
video/quicktime          ##QuickTime 视频
video/ogg                ##Ogg Theora 等视频
```

➢ Text

```
text/html                ##Html 文件
text/xml                 ##Xml 文件
text/plain               ##文本文件
```

➢ Image

```
image/gif               ##Gif 文件
image/jpeg              ##JPEG 文件
image/png               ##PNG 文件
```

另外，type 和 subtype 中以 x-开头的属于非标准格式（未向 IANA 注册）；而 subtype 中以 vnd 开头的则是厂商自定义的（vendor-specific）。

比如：

```
application/vnd.oasis.opendocument.text  ##OpenDocument 文本
application/vnd.ms-excel                 ##Microsoft Excel 文件
application/x-latex                      ##LaTeX 文件
audio/x-caf                              ##Apple 公司的 CAF 音频文件
```

Android 系统中也有不少自定义的类型，在后面的例子中会看到。

- URI

URI（Uniform resource identifier）是某种资源的全球唯一定位标志，因而通常被应用于网络

环境中，格式如图 14-3 所示。

▲图 14-3　URI 的格式

其中，scheme 不仅包括了传统的“http”等网络协议，还有“content”来表示本地 ContentProvider 所提供的数据。如图所示，“host”是主机的名称，“port”指明通信的端口，它们统称为“Authority”；而且如果 host 不存在，后面的端口号也会被忽略。最后一部分是文件的路径，它是该资源在 host 中的具体位置。

虽然我们允许 URI 中有部分信息缺失的情况，但要特别注意它们之间并不是完全独立的——如果 scheme 不指定，则 authority 就没有意义；同样，如果 scheme 和 authority 不指定，那么 path 也没有意义。所有元素组合后构成了一条完整的 URI，比如：

```
content://com.google.provider.NotePad/notes
content://com.example.project:200/folder/subfolder/etc
```

了解了 Data 中的组成元素后，我们再来具体看看它的匹配流程。Data 和上面的 Category 和 Action 有本质区别。如果说前面的 Category 和 Action 遵循的是“intent 中属性必须是 filter 子集”的原则，那么 Data 则大相径庭——只有 filter 中存在的那部分属性（比如某 filter 中只指定了 mimeType，那么只匹配 mimeType），才需要进行匹配。

根据概率组合，具体有如下 4 种可能。

（1）Intent 中既没有指定数据类型，也没有填写 URI。

在这种情况下，只有 filter 中也同样没有指定数据和 URI，才可能通过测试。

（2）Intent 中没有指定类型（且无法从 URI 中推断出），但有 URI。

值得一提的是，数据类型有可能从 URI 中推断出来——如果是就属于第四种情况。前面已经说过，“只有 filter 中存在的那部分属性，才需要进行匹配”，因而这种情况下 filter 必须没有指定类型，且 Intent 中的 URI 也符合要求，才可能通过测试。

（3）Intent 中只指定了类型，没有 URI。

这和上面的情况类似。此时如果 filter 中也没有 URI，且 Intent 中指定的类型符合要求，才能通过测试。

（4）Intent 中同时指定了类型（或可以从 URI 推断出）和 URI。

这时类型和 URI 都必须通过测试。具体来说，以 filter 中的 data 为主导，将其中存在的属性逐一与 Intent 中的 data 进行比较。不过有一种特殊情况，即 filter 默认就支持 content:和 file:的 scheme。换句话说，即使它只填写了类型部分，URI 中也会有这两个 scheme 存在。

接下来举一个 Android 官方文档中的例子以帮助大家加深理解。这个 NotePad 范例在 SDK 中可以找到源码，它的 intent-filter 定义如下所示：

```
/*<sdk>/samples/NotePad/ , AndroidManifest.xml*/
/*为了讲解方便，下面我们给每个 filter 都标注了序号*/
<manifest xmlns:android=http://schemas.android.com/apk/res/androidpackage="com.examp
le.android.notepad">
<application android:icon="@drawable/app_notes"android:label="@string/app_name" >
<provider android:name="NotePadProvider"android:authorities="com.google.provider.NotePad" />
<activity android:name="NotesList" android:label="@string/title_notes_list">
 <intent-filter> //NotesList-filter-1
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
 <intent-filter>//NotesList-filter-2
  <action android:name="android.intent.action.VIEW" />
  <action android:name="android.intent.action.EDIT" />
  <action android:name="android.intent.action.PICK" />
  <category android:name="android.intent.category.DEFAULT" />
  <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
```

```
  </intent-filter>
  <intent-filter>//NotesList-filter-3
   <action android:name="android.intent.action.GET_CONTENT" />
   <category android:name="android.intent.category.DEFAULT" />
   <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
  </intent-filter>
 </activity>

 <activity android:name="NoteEditor"android:theme="@android:style/Theme.Light"
                  android:label="@string/title_note" >
  <intent-filter android:label="@string/resolve_edit"> //NoteEditor -filter-1
   <action android:name="android.intent.action.VIEW" />
   <action android:name="android.intent.action.EDIT" />
   <action android:name="com.android.notepad.action.EDIT_NOTE" />
   <category android:name="android.intent.category.DEFAULT" />
   <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
  </intent-filter>
  <intent-filter> //NoteEditor -filter-2
   <action android:name="android.intent.action.INSERT" />
   <category android:name="android.intent.category.DEFAULT" />
   <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
  </intent-filter>
 </activity>

 <activity android:name="TitleEditor" android:label="@string/title_edit_title"
                  android:theme="@android:style/Theme.Dialog">
  <intent-filter android:label="@string/resolve_title"> //TitleEditor -filter-1
   <action android:name="com.android.notepad.action.EDIT_TITLE" />
   <category android:name="android.intent.category.DEFAULT" />
   <category android:name="android.intent.category.ALTERNATIVE" />
   <category android:name="android.intent.category.SELECTED_ALTERNATIVE" />
   <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
  </intent-filter>
 </activity>
 </application>
 </manifest>
```

根据上面的 AndroidManifest 文件，可以看到 NotePad 共包含了 3 种 Activity，即 NotesList（用于显示已经保存的文章），NoteEditor（用于编辑文章）和 TitleEditor（用于编辑文章的标题）。

下面我们通过列举几个不同的 Intent 来分析系统的匹配情况。

Intent 1：

```
action: android.intent.action.MAIN
category: android.intent.category.LAUNCHER
```

Category 测试：根据子集原则，只有 NotesList-filter-1 通过了检查。

Action 测试：依据上面所分析的 Action 测试第 3 点，NotesList-filter-1 通过了检查。

Data 测试：属于第一种情况，即 intent 里既没有指定类型，也无 URI。而另外，filter 也是同样的情况，因此根据“只有 filter 里有的那部分属性才需要检查”的原则，NotesList-filter-1 最终通过了匹配。这个 Intent1 将对应 NoteList。

Intent2：

```
action: android.intent.action.VIEW
data: content://com.google.provider.NotePad/notes
```

Category 测试：Intent 中没有指定 Category，系统会自动为其加上 DEFAULT 值。因为所有 filter 都写上了这个默认值，因而全部通过测试。

Action 测试：只有 NotesList-filter-2 和 NoteEditor-filter-1 通过测试。

Data 测试：表面上属于第二种情况，即只指定了 URI 而没有类型。但实际上从这个例子中的 content 可以推断出 type，因而属于第四种情况。推断出的类型为"vnd.android.cursor.dir/vnd.google.note"（可以参见下一小节对推断过程的源码解析），因而最终通过测试的是 NotesList-filter-2。

Intent3：

```
action: android.intent.action.GET_CONTENT
data type: vnd.android.cursor.item/vnd.google.note
```

Category 测试：同 Intent2。

Action 测试：只有 NotesList-filter-3 通过测试。

Data 测试：只有 Type，而没有 URI，属于第三种情况。因为 NotesList-filter-3 也是同样的情况，而且它们的类型也是匹配的，所以最终通过测试。

Intent4：

```
action: android.intent.action.INSERT
data: content://com.google.provider.NotePad/notes
```

Category 测试：同 Intent2，由此可见 intent-filter 中加上 DEFAULT 还是非常必要的。

Action 测试：只有 NotesEditor-filter-2 符合要求。

Data 测试：同 Intent2。

Intent5：

```
action: com.android.notepad.action.EDIT_TITLE
data: content://com.google.provider.NotePad/notes/ID
```

Category 测试：同 Intent2。

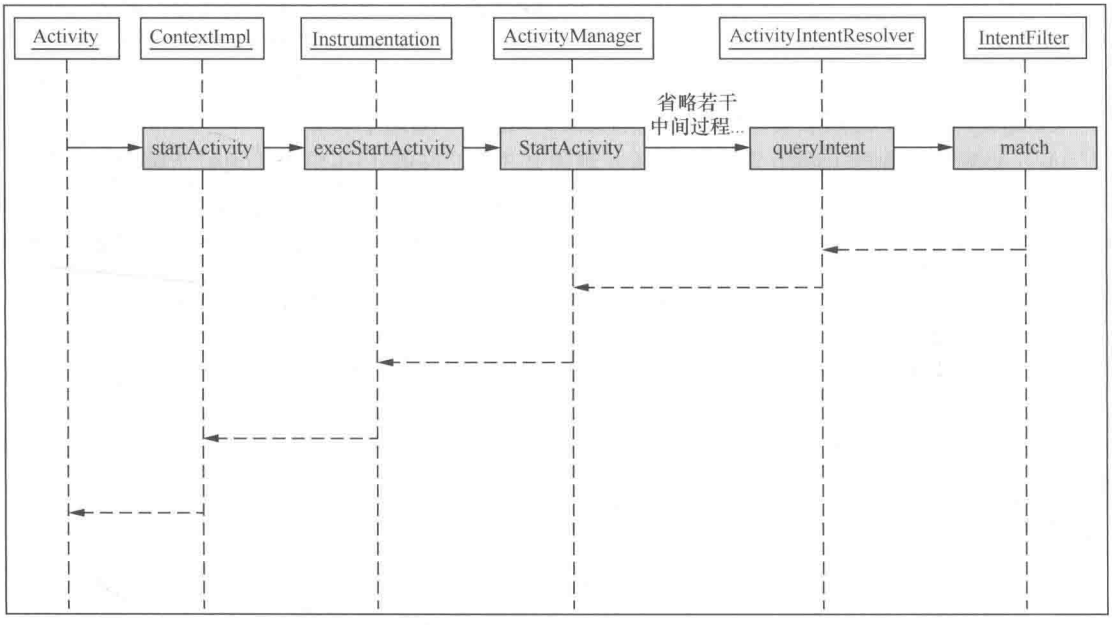
Action 测试：只有 TitleEditor-filter-1 符合要求。

Data 测试：同样可以推断出类型，即"vnd.android.cursor.item/vnd.google.note"，因而上面的 TitleEditor-filter-1 最终通过测试。

14.3 Intent 匹配源码简析

本节将对 Intent 匹配过程所涉及的核心源码进行简单的分析。

我们以 startActivity 为例，程序调用流程如图 14-4 所示。



▲图 14-4　从 startActivity 中看 Intent 匹配过程