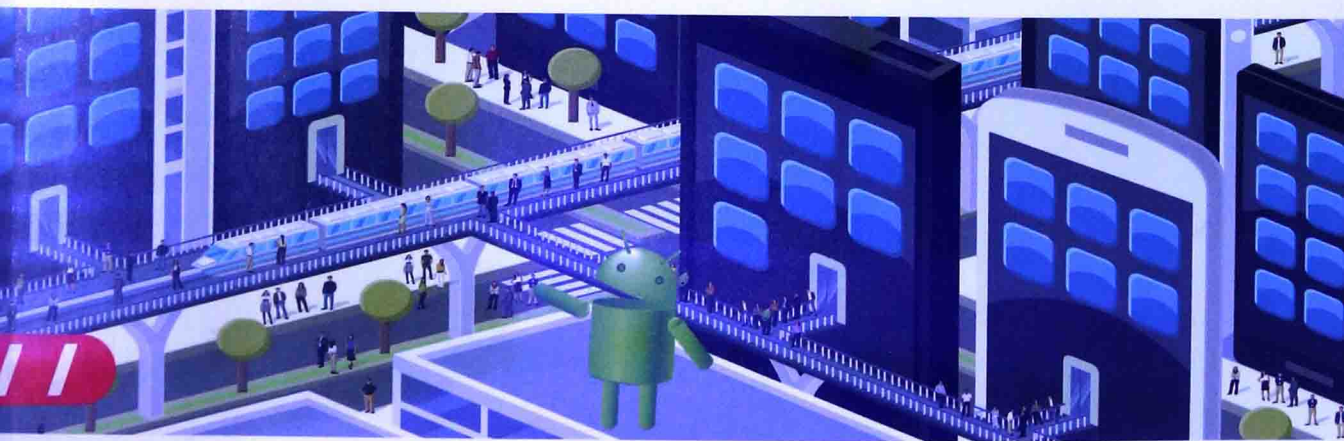


通过众多案例深入解读Android UI设计的方法和技巧，从实用角度出发，诠释以用户为中心的设计方法

以Google设计语言Material Design指导UI设计模式，轻松自信地设计和交付精美的移动App

# Android UI设计

## Android User Interface Design



李维勇 主编

杜亚杰 张以利 陈宇 参编



机械工业出版社  
China Machine Press

# Android UI设计

Android User Interface Design

作为一款开源智能手机操作系统，Android在当今移动市场上风头正劲。许多开发人员需要一本Android UI设计入门级教程，其能够同时针对移动UI的设计模式和碎片化解决方案进行深入分析。

本书面向创建移动应用的产品经理、设计师和开发者，系统讲解了从事Android UI设计必须要掌握的Android平台的主要技术和特性，全面总结了Android UI的设计原理、设计理念和设计模式，并通过一个综合的案例项目阐述Android UI设计的方法和技巧。

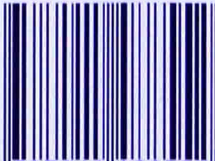
## 本书特色：

- **深度讲解** 从最基础的技术概念开始，系统阐述基于Android平台UI设计的基本理论，知识结构清晰，语言简洁。
- **项目驱动** 紧密结合初学者的学习习惯和认知规律，采用了大量简单而又实用的设计案例分析UI设计的基本理念。
- **强化技能** 以移动UI设计师的核心岗位能力统筹全书的编写，代码设计突出项目开发的实战性与健壮性。
- **遵守规范** 以Google最新推出的设计语言Material Design指导UI设计模式，并遵循移动UI设计领域最流行的扁平化风格和响应式交互设计。



上架指导：计算机/移动开发

ISBN 978-7-111-48855-2



9 787111 488552 >

定价：59.00元

投稿热线：(010) 88379604

客服热线：(010) 88378991 88361066

购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com

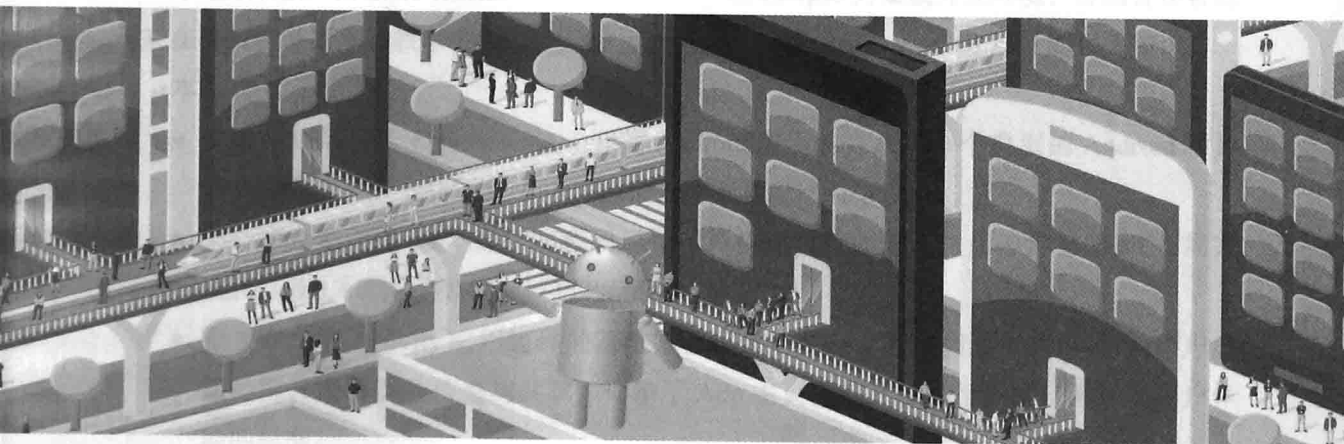
网上购书：www.china-pub.com

数字阅读：www.hzmedia.com.cn



# Android UI设计

Android User Interface Design



李维勇 主编

杜亚杰 张以利 陈宇 参编



机械工业出版社  
China Machine Press

## 图书在版编目 (CIP) 数据

Android UI 设计 / 李维勇主编. —北京: 机械工业出版社, 2015.2  
(UI / UE 系列丛书)

ISBN 978-7-111-48855-2

I. A… II. 李… III. 移动终端—应用程序—程序设计 IV. TN929.53

中国版本图书馆 CIP 数据核字 (2014) 第 295631 号

本书是基于 Android KitKat 平台进行移动应用开发的入门级教程, 通过众多开源案例项目全面系统地介绍 Android UI 设计的方法、技巧和模式。

全书共 12 章, 从 Android 应用设计者的角度系统讲解了从事 Android UI 设计必须要掌握的 Android 平台的所有技术和特性, 主要内容包括可视化的 UI 设计与管理、常见 UI 控件设计与事件处理、UI 容器与导航设计、菜单与对话框设计、自定义控件设计、桌面 UI 设计、平板 UI 设计, 以及主题样式和动画设计等, 全面总结了 Android UI 的设计原理、设计理念和设计模式, 最后通过一个综合的案例项目阐述 Android UI 设计的方法和技巧。

本书以案例贯穿全程, 知识结构清晰, 语言简洁, 易于学习和提高, 非常适合初学 Android UI 设计的在校大学生和希望系统掌握 Android UI 编程的开发人员阅读。

## Android UI 设计

出版发行: 机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码: 100037)

责任编辑: 李 燕

责任校对: 董纪丽

印 刷: 北京市荣盛彩色印刷有限公司

版 次: 2015 年 3 月第 1 版第 1 次印刷

开 本: 186mm×240mm 1/16

印 张: 17.75

书 号: ISBN 978-7-111-48855-2

定 价: 59.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88378991 88361066

投稿热线: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzjsj@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问: 北京大成律师事务所 韩光 / 邹晓东

Android 是 Google 推出的一款广受移动应用软件开发开发者追捧的开源操作系统，近年来，Android 手机的市场占有率一直排名第一。

本书以 Android SDK KitKat 4.4 为开发平台，以 Eclipse 为集成开发环境，并结合作者近年来在手机软件研发和教学中积累的经验，详细介绍了 Android 平台 UI 设计的相关知识。

本书共 12 章。

- 第 1 章介绍基于 Eclipse + ADT 开发 Android 应用的方法及一个典型的 Android 项目的架构组成，并分析了移动 App 的设计原则和设计风格。
- 第 2 章介绍通过 ADT 插件实现图形化用户界面设计的方法、几种常见的 UI 布局方式，以及 UI 布局的原则、技巧和优化方法。
- 第 3 章介绍 Activity 应用组件的基础知识，包括创建、管理和退出 Activity，用户界面的跳转及数据的传递与共享，列举了 App 主页面的几种常见模式，阐述了用户体验的标准。
- 第 4 章主要介绍 Android 平台 Service、BroadcastReceiver 和 ContentProvider 等应用组件的核心知识，使用 Intent 在组件之间传递消息的机制，以及基于 Mashup 模式的应用模型。
- 第 5 章介绍常用表单控件的设计、适配器控件的设计，以及用户界面常见事件的触发与响应方法，分析了移动 App 表单 UI 的设计、大数据的加载模式以及提高搜索用户体验的方法。
- 第 6 章介绍 Toast、Notification 和 AlertDialog 这 3 种用户信息提示的方法，选项菜单和内容菜单的设计方法，以及动作栏和用户界面导航的设计，分析了用户通知设计的策略和原则。
- 第 7 章介绍常见容器 UI 的设计，包括导航类容器设计、特定容器设计，以及广泛使



用的第三方容器控件的设计,分析了用户引导页的设计技巧。

- 第8章介绍自定义控件设计的知识,包括定制一个基于 View 的控件、重构一个 View 子类,以及使用 Skia 绘制用户界面,并列举了几种常见的开源 UI 工具。
- 第9章介绍主题和样式的设计方法、系统主题资源的应用,以及设计帧动画、补间动画和属性动画的方法,分析了应用风格设计的8个技巧,阐述了用户界面动态设计的原则和技巧。
- 第10章介绍桌面 UI 设计方法,包括桌面组件的布局与属性描述、桌面组件的广播响应,以及基于集合的桌面应用组件的设计,并介绍了桌面组件的设计规范。
- 第11章介绍平板 UI 设计的知识,包括使用 Fragment 灵活构建 UI 界面的方法、管理 Fragment 之间的通信,以及设计平板设置界面的方法,分析了 Google 关于平板设计的原则和几种常见的平板布局模式。
- 第12章通过 Apollo 音乐播放器案例的用户界面设计,系统阐述了移动 App 开发中 UI 设计的知识、技巧和模式应用。

本书紧密结合初学者的学习习惯和认知规律,采用了大量简单而又实用的设计案例,使得读者在阅读时不会有障碍,并可通过简单的代码移植生成新的应用。书中采用的开源案例项目把与 Android 开发相关的技术和设计完美结合,别具一格,弥补了 Android 设计人员知识的不足。

本书由李维勇担任主编,杜亚杰、张以利、陈宇参与编写。南京信息职业技术学院软件学院移动互联网应用技术教研室全体同仁共同参与了本书的校对和文稿的审核。本书的编写得到了南京信息职业技术学院、南京工业职业技术学院、南京审计学院金审学院等兄弟院校的大力支持和帮助,上海尚强信息科技有限公司对教材案例项目的策划、开发和测试提供了大量信息,机械工业出版社的编辑为本书的策划和出版提供了宝贵的经验和支持,在此表示衷心感谢。同时,本书在编写过程中参考了大量的相关资料,吸取了许多同仁的宝贵经验,在此一并致谢。

由于作者水平有限,难免存在疏漏,恳请广大读者批评指正,并欢迎提出宝贵意见和建议。另本书的配套课件、习题答案及源代码均可从华章公司网站([www.hzbook.com](http://www.hzbook.com))下载。

作者

2014年12月

## Contents 目 录

### 前 言

## 第 1 章 Android 开发基础 ..... 1

### 1.1 Eclipse 中的 Android 开发 ..... 1

#### 1.1.1 创建项目 ..... 1

#### 1.1.2 创建 AVD ..... 2

#### 1.1.3 运行项目 ..... 3

### 1.2 Android 项目架构 ..... 5

#### 1.2.1 Java 代码解析 ..... 5

#### 1.2.2 项目资源解析 ..... 8

#### 1.2.3 AndroidManifest.xml 解析 ..... 10

### 1.3 Eclipse 中的常用窗口 ..... 12

#### 1.3.1 Console 窗口 ..... 13

#### 1.3.2 LogCat 窗口 ..... 13

#### 1.3.3 DDMS 窗口 ..... 14

### 1.4 移动 App 的设计原则 ..... 16

### 1.5 移动 App 的设计风格 ..... 18

#### 1.5.1 扁平化设计 ..... 19

#### 1.5.2 卡片式设计 ..... 21

## 第 2 章 ADT 中的 UI 设计 ..... 23

### 2.1 图形布局编辑器 ..... 23

### 2.2 几种常见的布局方式 ..... 25

#### 2.2.1 创建布局 ..... 26

#### 2.2.2 相对布局 ..... 27

#### 2.2.3 线性布局 ..... 29

#### 2.2.4 帧布局 ..... 31

### 2.3 优化布局 ..... 32

#### 2.3.1 复用布局 ..... 32

#### 2.3.2 多设备支持 ..... 33

#### 2.3.3 使用 Hierachy Viewer 调试 用户界面 ..... 34

### 2.4 界面布局技巧 ..... 38

#### 2.4.1 布局设计原则 ..... 38

#### 2.4.2 布局设计技巧 ..... 39

### 2.5 习题 ..... 42

## 第 3 章 Activity 与 UI 管理 ..... 43

### 3.1 Activity 基础 ..... 43

#### 3.1.1 创建 Activity ..... 43

#### 3.1.2 Activity 的生命周期 ..... 45

#### 3.1.3 退出 Activity ..... 47

### 3.2 Activity 之间的调用 ..... 47

#### 3.2.1 调用其他 Activity ..... 48

3.2.2 Activity 的回调	48	4.4.2 Intent 对象	82
3.3 Activity 之间的数据传递	50	4.4.3 Intent 的解析	84
3.3.1 使用 Intent 传递数据	50	4.5 基于组件的应用模型	86
3.3.2 使用 Bundle 传递数据	50	4.6 习题	88
3.3.3 使用 Application 共享数据	52		
3.4 Activity 栈与任务	53	<b>第 5 章 Widgets 设计与事件处理</b>	89
3.4.1 Activity 栈	53	5.1 表单控件设计	89
3.4.2 任务管理	55	5.1.1 文本控件	89
3.4.3 Activity 的加载模式	55	5.1.2 按钮控件	92
3.4.4 保存 Activity 的状态	58	5.1.3 单选 / 复选按钮控件	94
3.5 应用主页设计技巧	61	5.1.4 进度条控件	95
3.6 用户体验设计	63	5.2 适配器控件设计	96
3.7 习题	65	5.2.1 适配器概述	96
		5.2.2 Gallery	96
<b>第 4 章 Android 组件编程</b>	66	5.2.3 Spinner	97
4.1 Service 与后台服务	66	5.2.4 ListView	98
4.1.1 创建 Service	66	5.2.5 GridView	100
4.1.2 Service 的生命周期	67	5.2.6 适配器控件的大数据加载	100
4.1.3 Started Service	68	5.3 Widgets 事件处理	102
4.1.4 Bound Service	69	5.3.1 按键事件处理	102
4.2 ContentProvider 与数据共享	71	5.3.2 触屏事件处理	103
4.2.1 系统中的 ContentProvider	72	5.3.3 手势事件处理	105
4.2.2 通用资源标志符	73	5.3.4 感应器事件处理	108
4.2.3 使用 ContentProvider	75	5.4 Widgets 设计技巧	109
4.3 BroadcastReceiver 与广播意图	77	5.4.1 官方设计指引	110
4.3.1 BroadcastReceiver 的工作机制	77	5.4.2 表单控件设计技巧	112
4.3.2 广播的类型	78	5.4.3 数据加载模式设计	115
4.3.3 接收广播	80	5.4.4 搜索设计技巧	118
4.3.4 注册广播	80	5.5 习题	120
4.4 Intent 与组件通信	81		
4.4.1 Intent 处理机制	81	<b>第 6 章 对话框、菜单与导航</b>	121
		6.1 对话框设计	121



6.1.1	Toast 通知 .....	121
6.1.2	Notification 提示 .....	121
6.1.3	AlertDialog 对话框 .....	124
6.1.4	对话框的托管 .....	126
6.2	菜单设计 .....	127
6.2.1	Options Menu .....	127
6.2.2	Context Menu .....	129
6.3	动作栏与导航设计 .....	130
6.3.1	动作栏设计 .....	130
6.3.2	ActionMode 设计 .....	131
6.3.3	导航设计 .....	133
6.3.4	导航设计技巧 .....	137
6.4	用户通知设计技巧 .....	140
6.4.1	Android 中的消息提示 .....	140
6.4.2	通知设计策略 .....	141
6.4.3	通知设计原则 .....	142
6.4.4	通知的导航机制 .....	143
6.4.5	声音提醒 .....	145
6.5	习题 .....	147

## 第7章 容器 UI 设计 .....

7.1	导航类容器设计 .....	148
7.1.1	使用 ViewPager 设计导航页 .....	148
7.1.2	使用 ViewFlipper 设计滑屏 窗口 .....	151
7.1.3	使用 TabHost 设计标签页 .....	152
7.2	特定容器设计 .....	154
7.2.1	使用 WebView 显示网页 .....	154
7.2.2	使用 MapView 显示地图 .....	156
7.2.3	使用 VideoView 播放视频 .....	158
7.3	第三方容器控件设计 .....	159

7.3.1	使用 SlidingMenu 设计菜单 容器 .....	159
7.3.2	使用 TimesSquare 设计日期 .....	162
7.4	引导页设计技巧 .....	162
7.5	习题 .....	164

## 第8章 自定义控件设计 .....

8.1	概述 .....	165
8.2	定制控件 .....	165
8.3	重载控件 .....	170
8.3.1	重构 AdapterView .....	170
8.3.2	应用控件 .....	174
8.4	绘制 UI .....	175
8.5	开源 UI 工具 .....	176
8.6	习题 .....	177

## 第9章 样式、主题与动画设计 .....

9.1	样式与主题 .....	178
9.1.1	Style .....	178
9.1.2	Theme .....	181
9.2	动画设计 .....	184
9.2.1	帧动画 .....	185
9.2.2	补间动画 .....	186
9.2.3	属性动画 .....	190
9.3	应用风格设计 .....	193
9.4	动态效果设计 .....	198
9.4.1	动态设计原则 .....	198
9.4.2	动态设计技巧 .....	199
9.5	习题 .....	203

## 第10章 桌面 UI 设计 .....

10.1	设计简单的桌面组件 .....	204
------	-----------------	-----

10.1.1 RemoteViews .....	205	11.2.4 使用 Support Library .....	234
10.1.2 AppWidgetProviderInfo .....	206	11.3 管理 Fragment .....	235
10.1.3 AppWidgetProvider .....	207	11.3.1 Fragment 的生命周期 .....	235
10.1.4 声明 App Widgets .....	210	11.3.2 使用 FragmentManager 处理 事务 .....	238
10.2 配置和管理桌面组件 .....	211	11.3.3 Fragment 之间的通信 .....	240
10.2.1 Configuration Activity .....	211	11.4 PreferenceFragment .....	243
10.2.2 AppWidgetManager .....	213	11.5 平板 UI 设计技巧 .....	246
10.3 设计集合桌面组件 .....	214	11.5.1 Google 的准则 .....	246
10.3.1 Collection Views .....	214	11.5.2 横竖屏布局设计 .....	249
10.3.2 RemoteViewsService .....	217	11.5.3 常见平板布局 .....	252
10.3.3 RemoteViewsFactory .....	218	11.6 习题 .....	254
10.3.4 子视图事件 .....	220		
10.4 桌面组件设计规范 .....	223	<b>第 12 章 Android UI 综合应用</b> .....	255
10.4.1 桌面组件的种类 .....	223	12.1 项目概述 .....	255
10.4.2 桌面组件的尺寸 .....	224	12.2 用户界面设计 .....	256
10.4.3 桌面组件设计技巧 .....	225	12.2.1 结构设计 .....	256
10.5 习题 .....	226	12.2.2 交互设计 .....	259
		12.2.3 视觉设计 .....	260
<b>第 11 章 平板 UI 设计</b> .....	227	12.3 用户界面功能实现 .....	261
11.1 Fragment 概述 .....	227	12.3.1 主界面设计 .....	261
11.1.1 Fragment 布局特性 .....	227	12.3.2 歌曲列表界面设计 .....	265
11.1.2 Fragment 与 Activity .....	228	12.3.3 系统设置界面设计 .....	267
11.2 创建 Fragment .....	229	12.3.4 桌面应用组件设计 .....	269
11.2.1 创建 ListFragment .....	230	12.4 UI 测试 .....	271
11.2.2 创建 Fragment .....	232		
11.2.3 添加 Fragment 到 Activity .....	233	<b>参考文献</b> .....	274

# Android 开发基础

## 1.1 Eclipse 中的 Android 开发

Eclipse 是著名的跨平台开源集成开发环境，对开发 Android 应用提供了良好的支持。

### 1.1.1 创建项目

在 Eclipse 中创建 Android 项目的步骤如下：

①启动 Eclipse 集成开发环境<sup>①</sup>。

②运行 File → New → Android Application Project 菜单命令，打开 New Android Application 向导，显示如图 1-1 所示界面。

在 New Android Application 向导中输入如下信息：

- Application Name: HelloWorld
- Project Name: HelloWorld
- Package Name : com.liweiyong.helloworld (包的名称必须和所有安装在 Android 系统中的应用程序的包名不相同)

其他默认选择如下：

- Minimum Required SDK: API 14
- Target SDK: API 18

① 如果 Eclipse 没有安装 ADT 插件，请参阅官方指导文档 <http://developer.android.com/sdk/installing/installing-adt.html>。



- Compile With: API 19
- Theme: Holo Light with Dark Action Bar

③单击 Next 按钮，默认 Configure Project 设置和 Configure the attribute of the icon set 界面设置，选择 BlankActivity，单击 Finish 按钮，完成 Hello World 项目的创建。

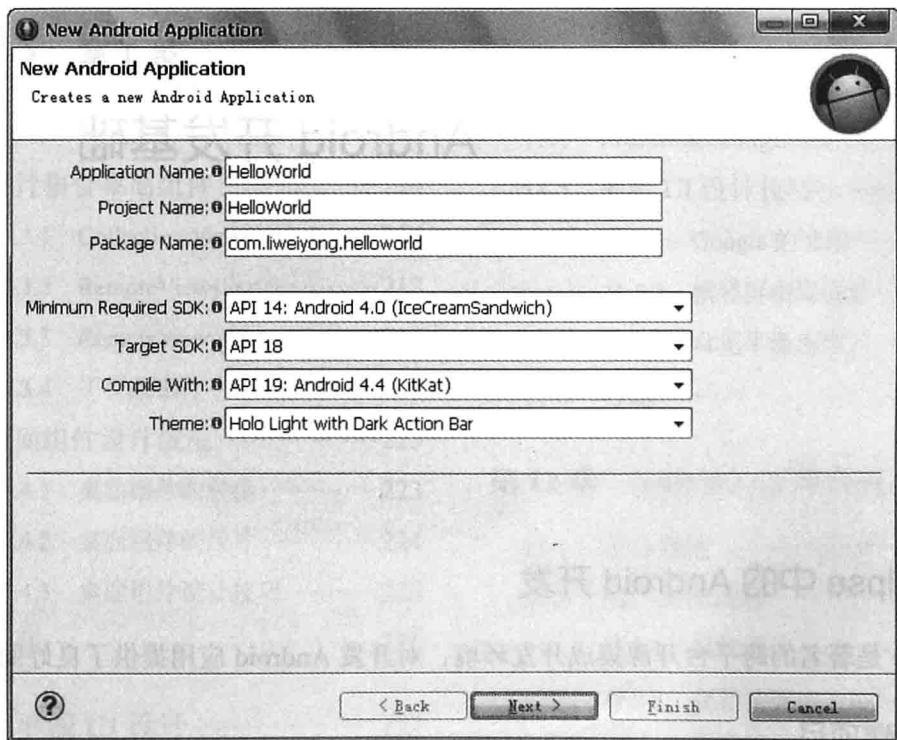


图 1-1 New Android Application 向导

## 1.1.2 创建 AVD

Android Virtual Device (简称 AVD) 是运行 Android 项目的虚拟设备。AVD 通过对硬件和软件的配置进行定义来模拟一个实际的设备。

### 1. 创建 SD Card 映像文件

Android 模拟器自身已经具备了一个持久化存储空间，但这并不够大，有时需要为应用程序和文件提供更大的存储空间。为了在模拟器上开发使用扩展存储空间的程序，需要在 PC 上模拟一个 SD Card (Secure Digital Memory Card，一种基于半导体快闪记忆器的新一代记忆设备) 的虚拟文件，然后加载到模拟器中。

创建 SD Card 虚拟文件的步骤如下：

①在 Windows 中，运行“开始”→“运行”菜单命令，在打开的“运行”窗口中输入“cmd”并单击“确定”按钮，打开命令行窗口。

②在窗口中输入如下命令：

```
mkshdcard -l mycard 500M C:\mysdcard.img
```

该命令的含义是在本地磁盘 C 盘创建一个 500 MB 大小的映像文件 mysdcard.img。此时，查看 C 盘可以看到一个名称为 mysdcard.img 的文件。

## 2. 创建 AVD 并关联 SD Card

在 Eclipse 中创建 AVD 并关联 SD Card 的步骤如下：

①在 Eclipse 中，运行 Window → Android Virtual Device Manager 菜单命令，打开 Android Virtual Device Manager 对话框。单击对话框中的 New 按钮，弹出如图 1-2 所示的创建 AVD 对话框。

- 在 AVD Name 文本框中输入 AVD 的名称（可以自定义）。
- Device 用于设置模拟器的尺寸和分辨率。SDK 提供的常见分辨率（sdk\platforms\android-#\skins\）包括：HVGA（320×480）、QVGA（320×240）、WQVGA400（400×240）、WQVGA432（432×240）、WVGA800（800×480）、WVGA854（854×480）和 WXGA（1280×800）等。
- 在 Target 中选择需要的 SDK 版本（平板电脑开发的最低版本是 Android API 11）。
- SD Card 的大小可以自定义输入数值；也可以选择 File 单选按钮后，单击 Browse 按钮，在打开的对话框中选择前面创建的 SD Card 映像文件。
- 其他默认设置即可。

②单击 OK 按钮，完成 AVD 的创建。

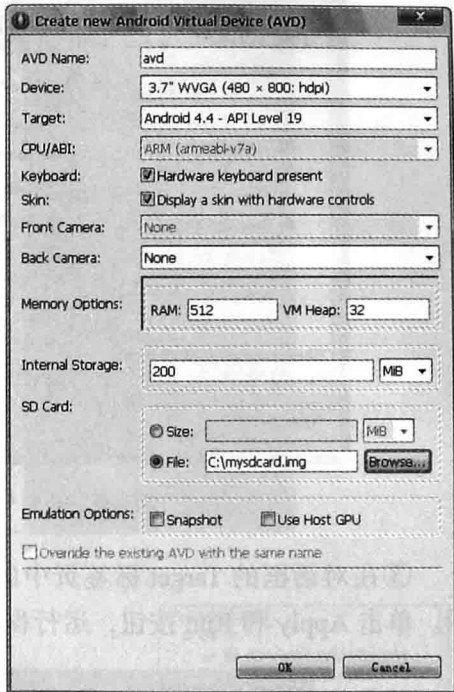


图 1-2 创建 AVD 对话框

### 1.1.3 运行项目

下面介绍在 Eclipse 中运行 Android 项目的方法，步骤如下：

①在 Eclipse 的项目窗口中，右击项目节点名称 HelloWorld，运行 Run as → Run Configurations 菜单命令，打开 Run Configurations 对话框。

②在对话框的左侧选择 Android Application，并单击上方的 New launch configuration 按钮，在右侧的 Android 标签页中单击 Browse 按钮，打开 Project Selection 对话框，选择 HelloWorld 项目，如图 1-3 所示。

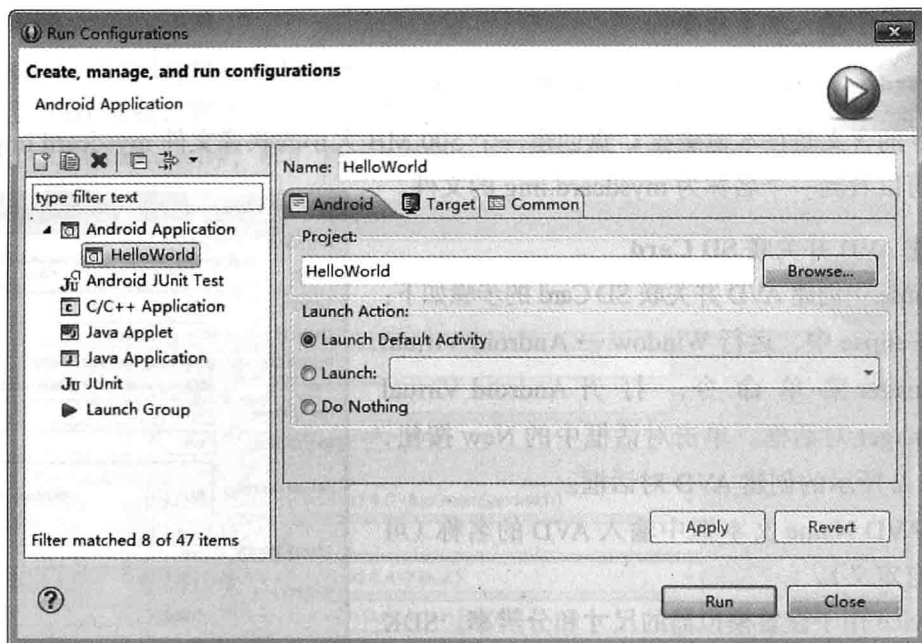


图 1-3 Run Configurations 对话框——选择项目

③在对话框的 Target 标签页中的 AVD 列表中勾选合适的 Android 模拟器，如图 1-4 所示。单击 Apply 和 Run 按钮，运行程序<sup>④</sup>。

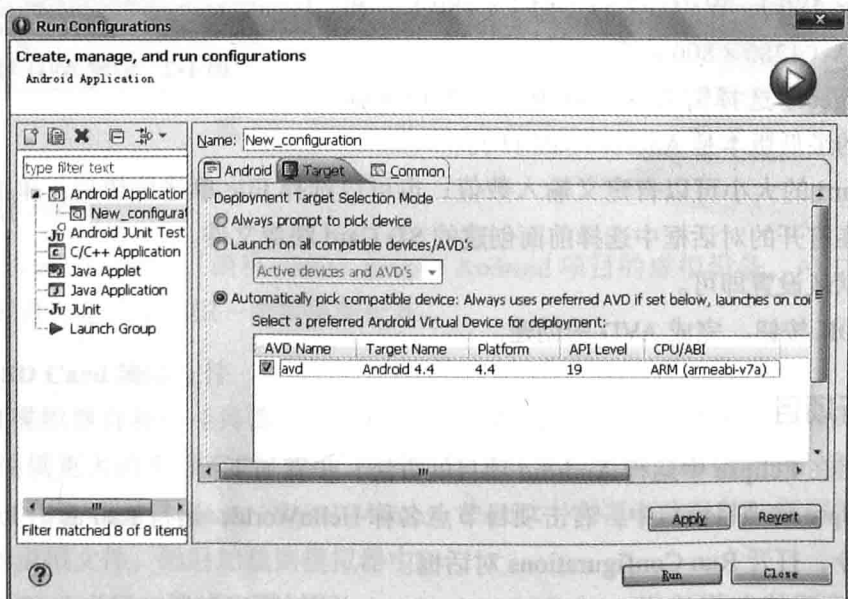


图 1-4 Run Configurations 对话框

④ 如果需要对 AVD 进行缩放，特别是使用平板模拟器时，可以在 Run Configurations 对话框的 Additional Emulator Command Line Options 中输入类似“-scale 0.8”这样的命令，这样模拟器的显示尺寸将缩小到 80%。



图 1-5 显示了项目的运行结果。

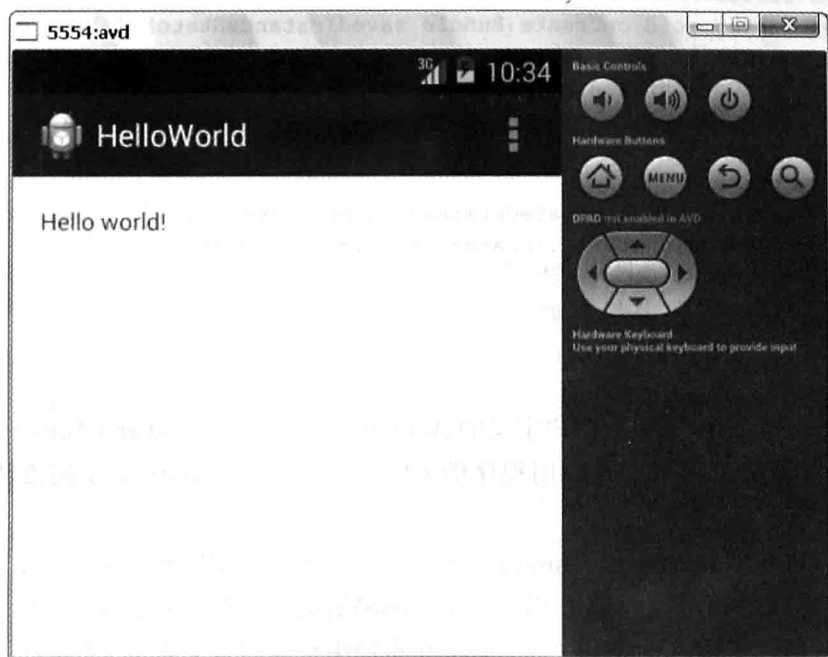


图 1-5 项目运行结果

## 1.2 Android 项目架构

在 Eclipse 的 Package Explorer 窗口中展开 HelloWorld 项目，项目架构如图 1-6 所示。

一个 Android 工程包含了组成 Android 应用的所有源代码文件。Android 工程主要由 src、gen、assets、res 文件夹和 AndroidManifest.xml 等文件组成，下面分别对其进行解析。

### 1.2.1 Java 代码解析

Android 项目的 Java 代码主要存放在 src 文件夹和 gen 文件夹的包文件夹下。

#### 1. MainActivity 解析

src 文件夹用来存放项目的源代码。

在 Package Explorer 窗口中，展开项目的 src 文件夹，打开通过向导生成的 MainActivity.java，核心代码如下：

```
01 public class MainActivity extends Activity {
```

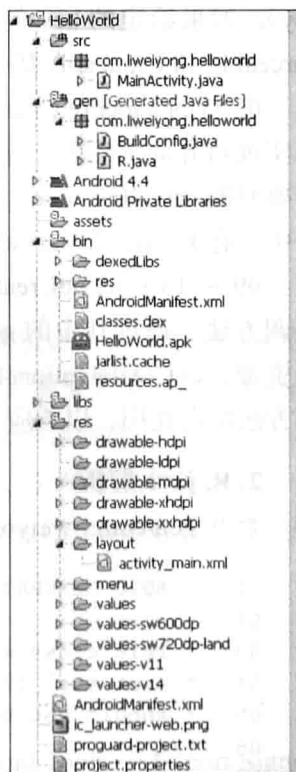


图 1-6 HelloWorld 项目架构

```

02
03     @Override
04     protected void onCreate(Bundle savedInstanceState) {
05         super.onCreate(savedInstanceState);
06         setContentView(R.layout.activity_main);
07     }
08
09     @Override
10     public boolean onCreateOptionsMenu(Menu menu) {
11         getMenuInflater().inflate(R.menu.main, menu);
12         return true;
13     }
14
15 }

```

01 行的 MainActivity 是一个用户定义的 Activity，继承自 android.app.Activity 类。Activity 是应用程序的表示层，用于构建应用程序的 UI 界面。关于 Activity 的使用方法将在第 3 章介绍。

03 ~ 07 行重载的 onCreate (Bundle) 方法是 Activity 生命周期的组成部分，用于初始化 Activity。例如界面的显示内容通过调用 setContentView() 方法来指定显示布局，如 activity\_main.xml。然后通过 findViewById (int) 方法在布局中检索需要交互的 UI 控件。

04 行的 Bundle 类用于 Activity 之间传递数据。该类提供了公有方法 containKey (String key)，如果给定的 key 包含在 Bundle 的映射中，则返回 true，否则返回 false。该类实现了 Parcelable 和 Cloneable 接口，所以它具有这两者的特性。

05 行的 super.onCreate ( savedInstanceState) 方法的作用是调用父类中的 onCreate() 方法来实现对界面的绘制工作。注意，从 savedInstanceState 中读取保存到存储设备中的数据时，需要判断 savedInstanceState 是否为 null，因为 Activity 第一次启动时并没有数据被存储在设备中。有关 savedInstanceState 的介绍参见 3.4.4 节。

09 ~ 13 行的 onCreateOptionsMenu (Menu) 方法是 Activity 中提供的用于创建菜单项的回调方法。通过其中的 getMenuInflater().inflate (int, Menu) 方法加载 res/menu 中定义的菜单资源。onCreateOptionsMenu (Menu) 方法通常和 onOptionsItemSelected (MenuItem) 回调方法配合使用，以响应菜单的选择事件。

## 2. R.java 解析

打开 gen/com.liweiyong.helloworld 中的 R.java 文件，部分代码如下：

```

01  /* AUTO-GENERATED FILE.  DO NOT MODIFY.
02  *
03  * This class was automatically generated by the
04  * aapt tool from the resource data it found.  It
05  * should not be modified by hand.
06  */
07
08  public final class R {

```

```

09     public static final class array {
10         public static final int action_file=0x7f040005;
11         ....
12     }
13     public static final class layout {
14         public static final int activity_main=0x7f030000;
15     }
16     ...
17 }

```

程序的第 01 ~ 06 行为注释说明, R.java 文件由 aapt 工具根据 res 中的资源自动生成, 不要手动修改该文件。R.java 由 ADT 根据 res 中的资源自动生成 drawable、layout、string 等静态匿名内部类。不同的静态内部类分别根据其 res 中的资源定义一系列资源标识符, 如 “public static final int activity\_main=0x7f030000;” 对应的是 layout 目录下的 activity\_main.xml 文件。

每当 res 中的资源发生变化, aapt 工具都会自动在 R.java 对应的内部类中生成一个静态 int 类型的常量, 以对新添加的资源进行索引。如果从 res 中删除一个资源, R.java 中对应的索引也会自动删除。

通过 R.java 可以很快地查找需要的资源, 另外编译器也会检查 R.java 列表中的资源是否被使用, 没有被使用到的资源不会编译进 apk 中, 这样可以减少应用在手机中占用的空间。

### 3. BuildConfig.java 解析

打开 gen/com.liweiyong.helloworld 中的 BuildConfig.java 文件, 代码如下:

```

01 /** Automatically generated file. DO NOT MODIFY */
02 package cn.liweiyong.helloworld;
03
04 public final class BuildConfig {
05     public final static boolean DEBUG = true;
06 }

```

ADT 允许开发者只在调试模式下运行某些代码。BuildConfig 类包含一个 DEBUG 常量, 该常量会根据 Build 类型自动设置值。可以通过 BuildConfig.DEBUG 常量来编写只在 Debug 模式下运行的代码。如果有些代码不想在应用发布后执行, 也可以使用该功能。

例如, 在调试日志时, 不想在软件发布后被其他开发者看到, 过去的方式是设置一个全局变量, 标记软件为调试模式还是发布模式。现在, 有了 BuildConfig.DEBUG 之后, 在代码中可以直接写入:

```

01 if (BuildConfig.DEBUG) {
02     Log.d(TAG, "output something");
03 }

```

在发布前, BuildConfig.DEBUG 的值自动为 true。当通过 Android Tools → Export Signed Application Package 命令发布包时, BuildConfig.DEBUG 的值自动变为 false。

## 1.2.2 项目资源解析

Android 项目的资源组织在 `res` 文件夹中，资源包括 `drawable`、`layout`、`values`、`anim`、`xml`、`raw`、`color` 以及 `menu` 等。下面分别对最常用的 `drawable`、`layout`、`values` 资源的定义和使用进行解析。

### 1. drawable 解析

`drawable` 用于存放图片文件资源，或者能被编译为 `drawable` 类型的 XML 文件。包括 `drawable-hdpi`、`drawable-mdpi`、`drawable-ldpi` 和 `drawable-xhdpi` 等多个存放图片的文件夹。这些文件夹的作用如下：

- `drawable-hdpi` 里面存放高分辨率<sup>①</sup>的图片，如 WVGA(480×800)、FWVGA(480×854)。
- `drawable-mdpi` 里面存放中等分辨率的图片，如 HVGA(320×480)。
- `drawable-ldpi` 里面存放低分辨率的图片，如 QVGA(240×320)。
- `drawable-xhdpi` 里面存放至少 960×720 分辨率的图片。

在设计应用的图标时，对于五种主流的像素密度（MDPI、HDPI、XHDPI、XXHDPI 和 XXXHDPI）应按照 2:3:4:6:8 的比例进行缩放。例如，一个启动图标的尺寸为 48×48dp<sup>②</sup>，这表示在 MDPI 的屏幕上其实际尺寸应为 48×48px<sup>③</sup>，在 HDPI 的屏幕上其实际

① 可以通过如下的方法获取屏幕分辨率信息：

```
getWindowManager().getDefaultDisplay().getMetrics(new DisplayMetrics());
```

② dp（即 dip，设备独立像素）是 Android 布局时最常用的尺寸单位，它与像素密度（dpi）密切相关。简单地说，dp 就是一种基本上和设备无关的单位，它可以保证一套 UI 在不同机器上面的适配，而显示效果不会出现很大的偏差。例如，可以使得同一个图片在不同分辨率下的屏幕上保持基本相同的物理大小。dp 与 px 之间的关系可以通过下面的工具类转换：

```
public class DensityUtil {
```

```
    /**
```

```
     * 根据手机的分辨率从 dp 的单位转成为 px(像素)
```

```
     */
```

```
    public static int dip2px(Context context, float dpValue) {
```

```
        final float scale = context.getResources().getDisplayMetrics().density;
```

```
        return (int) (dpValue * scale + 0.5f);
```

```
    }
```

```
    /**
```

```
     * 根据手机的分辨率从 px(像素) 的单位转成为 dp
```

```
     */
```

```
    public static int px2dip(Context context, float pxValue) {
```

```
        final float scale = context.getResources().getDisplayMetrics().density;
```

```
        return (int) (pxValue / scale + 0.5f);
```

```
    }
```

```
}
```

③ px 即像素，1px 代表屏幕上一个物理的像素点。px 单位不被建议使用，因为同样 px 的图片，在不同手机上显示的实际大小可能不同。

大小是 MDPI 的 1.5 倍，即  $72 \times 72\text{px}$ ，在 XDPI 的屏幕上其实际大小是 MDPI 的 2 倍，即  $96 \times 96\text{px}$ ，依此类推。

对 drawable 中的图片资源的引用主要有 3 种形式：

①在 res 的 XML 文件中使用 @drawable/image\_name。

②在 Java 代码中引用的方法是 R.drawable.image\_name。

③在 Java 代码中也可以通过 getResources().getDrawable(R.drawable.image\_name) 方法返回 R.drawable.image\_name 所代表的 drawable 对象，或通过 getResources().openRawResource(R.drawable.image\_name) 方法获取 drawable 的输入流。

## 2. layout 解析

layout 中存放用于编译成屏幕布局的 XML 文件。这些文件可以使用可视化编辑器生成，也可以通过编写 XML 生成。

下面的代码是本项目自动生成的 activity\_main.xml 文件信息：

```
01 <RelativeLayout
02     xmlns:android="http://schemas.android.com/apk/res/android"
03     xmlns:tools="http://schemas.android.com/tools"
04     android:layout_width="match_parent"
05     android:layout_height="match_parent"
06     android:paddingBottom="@dimen/activity_vertical_margin"
07     android:paddingLeft="@dimen/activity_horizontal_margin"
08     android:paddingRight="@dimen/activity_horizontal_margin"
09     android:paddingTop="@dimen/activity_vertical_margin"
10     tools:context=".MainActivity" >
11
12     <TextView
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:text="@string/hello_world" />
16
17 </RelativeLayout>
```

activity\_main.xml 中的第 01 行根节点 RelativeLayout 表明该界面是一个相对布局。xmlns:android 是一个 XML 命名空间，表示 Android 开发工具准备使用 Android 命名空间里的一些通用属性。xmlns:tools 可以给 ADT 传递一些信息，例如，tools:ignore="HardcodedText"，就是让 ADT 忽略硬编码文字的检查。下面的 tools:context=".MainActivity" 的含义是告诉 ADT，布局编辑器在当前的布局文件里面设置对应的渲染上下文，说明当前的布局所在的渲染上下文是 MainActivity，如果这个 Activity 在 AndroidManifest 文件中设置了主题，那么 ADT 的布局编辑器会根据这个主题来渲染当前的布局。

布局 activity\_main.xml 中包括一个文本显示子控件 TextView，并通过 android:layout\_width、android:layout\_height 和 android:text 等属性为其赋予显示属性。

对 layout 中的布局资源的引用主要有两种形式：

①在 Java 代码中引用的方法是 `R.layout.filename`。

②在 `res` 的 XML 文件中使用 `@ [package:]layout/filename`。有关 `package` 的使用将在 8.2 节进行介绍。

除了默认的 `layout` 目录，还可以在 `res` 中创建 `layout-land` 和 `layout-port` 目录，分别用于存放横屏和竖屏的指定布局文件，也可以创建 `layout-1024×720`、`layout-1024×552` 等目录，用于存放指定分辨率下的布局文件。

### 3. values 解析

`values` 中主要包含如下名称类型的 XML 文件：

- `strings.xml` 用于定义字符串和数值。
- `array.xml` 用于定义数组信息。
- `colors.xml` 用于定义颜色。
- `dimens.xml` 用于定义尺寸数据。
- `styles.xml` 用于定义样式。

下面的代码是本项目自动生成的 `strings.xml` 文件信息：

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <resources>
03     <string name="app_name">HelloWorld</string>
04     <string name="action_settings">Settings</string>
05     <string name="hello_world">Hello world!</string>
06 </resources>
```

对 `strings.xml` 中的字符串资源的引用主要有两种形式：

①在 `res` 的 XML 文件中使用 `@ string/string_name`。

②在 Java 代码中引用的方法是 `R.string.string_name`，也可以通过 `getResources().getString (string_name)` 或 `getResources().getText (string_name)` 取得字符串资源。

除了默认的 `values` 目录，有时在创建项目时还自动生成 `values-v11` 和 `values-v14` 目录，分别用于在 API 11+ 的设备和在 API 14+ 的设备上使用，特别是其中的 `styles.xml` 资源。

除了本项目中用到的这些资源类型外，还有如下资源：

- `anim` 编译成动画对象的 XML 文件。
- `menu` 定义应用程序菜单的 XML 文件。
- `raw` 任意原始的资源文件。
- `xml` 配置应用程序组件的其他 XML 文件。
- `libs` 包含私有的方法库。

#### 1.2.3 AndroidManifest.xml 解析

`AndroidManifest.xml` 是一个用来描述 Android 应用程序整体信息的结构化 XML 配

置文件。该文件描述了应用程序的环境及其支持的 Activity、Service、ContentProvider 和 BroadcastReceiver 组件，及其能够处理的 Intent 信息，描述了项目的权限、外部库和设备特性等信息。在 Android 启动一个组件之前，它必须能够了解到这个组件的存在。因此，应用程序将它们的组件定义在 Android 包中的 AndroidManifest.xml 文件中。在所有的应用项目中该文件都以 AndroidManifest.xml 命名，并保存在项目的根目录下。

下面的代码是本项目自动生成的 AndroidManifest.xml 文件信息：

```

01 <?xml version="1.0" encoding="utf-8"?>
02 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
03     package="com.liweiyong.helloworld"
04     android:versionCode="1"
05     android:versionName="1.0" >
06
07     <uses-sdk
08         android:minSdkVersion="14"
09         android:targetSdkVersion="18" />
10
11     <application
12         android:allowBackup="true"
13         android:icon="@drawable/ic_launcher"
14         android:label="@string/app_name"
15         android:theme="@style/AppTheme" >
16         <activity
17             android:name="com.liweiyong.helloworld.MainActivity"
18             android:label="@string/app_name" >
19             <intent-filter>
20                 <action android:name="android.intent.action.MAIN" />
21
22                 <category android:name="android.intent.category.LAUNCHER" />
23             </intent-filter>
24         </activity>
25     </application>
26
27 </manifest>

```

04 ~ 05 行是应用的版本信息。07 ~ 09 行声明应用程序所要求的 Android API 的最低版本级别信息。11 ~ 25 行声明所有的 application 级别组件和属性，其中 19 ~ 23 行的 <intent-filter> 组件声明：

```

<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />

```

表明这个 Activity 是应用程序的入口，是程序运行后见到的第一个 Activity，将在应用程序加载器中显示。Android 的一个核心特性就是一个应用程序可以使用其他应用程序的组件来完成既定的任务，而不用自己再开发一个程序实现这个任务。为达到这个目的，系统必须能够在一个应用程序的任何一部分被需要时启动一个此应用程序的进程，并将这个部分的 Java 对



象实例化。因此，不像其他大多数系统上的应用程序，Android 应用程序并没有为应用程序提供一个单独的入口点（例如没有 `main()` 方法），而是为系统提供了可以实例化和运行所需的必备组件。

在所有的元素中只有 `<manifest>` 和 `<application>` 是必需的，且只能出现一次。如果一个元素包含其他子元素，必须通过子元素的属性来设置其值。处于同一层次的元素的说明是没有顺序的。

通过 `AndroidManifest.xml` 文件可以提供如下功能：

- 命名应用程序的 Java 数据包，这是应用程序的唯一标识。
- 描述应用程序的组件——Activity、Service、BroadcastReceiver 和 ContentProvider。对实现每个组件和公布其功能的类进行声明。这些声明使得 Android 系统了解这些组件以及它们在什么条件下可以被启动。
- 决定应用程序组件运行在哪个进程里面。
- 声明应用程序所必须具备的权限，用以访问受保护的部分 API，以及和其他应用程序的交互。
- 声明应用程序其他的必备权限，用于组件之间的交互。
- 列举测试设备 Instrumentation 类，用来提供应用程序运行时所需的环境配置及其他信息，这些声明只在程序开发和测试阶段存在，发布前将被删除。
- 声明应用程序所要求的 Android API 的最低版本级别。
- 列举 Application 所需要链接的库<sup>①</sup>。

在一个 Android 项目的结构中，除了上面介绍的内容外，还有两个重要的文件：`proguard-project.txt` 和 `project.properties`，它们是混淆器文件，如果需要对项目进行全局混码，只需要将 `project.properties` 中 “`#proguard.config=${sdk.dir}/tools/proguard/proguard-android.txt: proguard-project.txt`” 前面的 “#” 去掉即可。

### 1.3 Eclipse 中的常用窗口

在 Eclipse 中，经常使用如下两种方式来打开相关子窗口。

**方式 1：**执行 `Window → Show View → others` 菜单命令，在打开的 Show View 对话框中展开 Android 节点，如图 1-7 所示。

**方式 2：**执行 `Window → Open Perspective → others` 菜单命令，打开 Open Perspective 对话框，如图 1-8 所示。

下面列举几个在开发 Android 应用程序的过程中经常使用的 Eclipse 子窗口。

---

① 关于 `AndroidManifest.xml` 的详细属性说明，请参阅作者博客：<http://blog.csdn.net/njcit/article/details/9790841>。



图 1-7 Show View 对话框

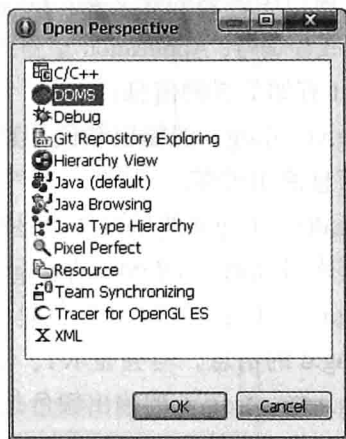


图 1-8 Open Perspective 对话框

### 1.3.1 Console 窗口

Console 窗口输出一些程序运行过程中的相关信息。例如，Console 窗口中显示类似如下信息时，表示该应用程序已经在 AVD 中安装并启动完成。

```
01 Uploading NotesList.apk onto device 'emulator-5554'
02 Installing NotesList.apk...
03 Success!
04 Starting activity com.example.android.notepad.NotesList on device
05     emulator-5554
06 ActivityManager: Starting: Intent {act=android.intent.action.MAIN
07     cat=[android.intent.category.LAUNCHER]
08     cmp= com.example.android/.NotesList }
```

### 1.3.2 LogCat 窗口

LogCat 窗口（如图 1-9 所示）常用于显示如下两种情况产生的信息：

- ①程序强制关闭或者异常退出的情况，即 Force Closed。
- ②程序无响应的情况，即 Application No Response（界面操作过程中，线程响应时间超过 5 秒，或者是 HandleMessage 回调方法执行过程中超过 10 秒）。

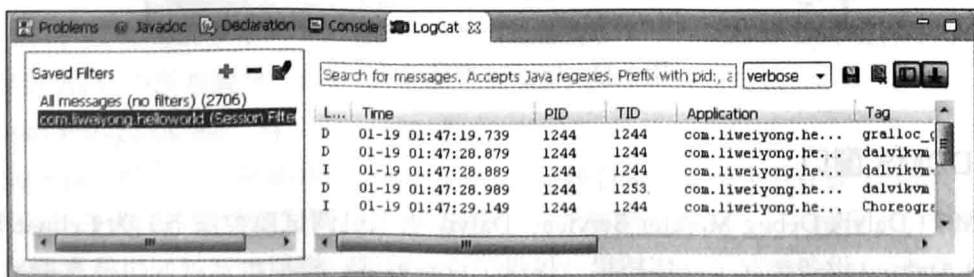


图 1-9 LogCat 信息输出窗口

图 1-9 窗口中各列的含义是：Level 表示 LogCat 的等级；Time 表示具体日志的记录时间；PID 是所在线程编号；Application 是测试对象；Tag 为 LogCat 记录的标题；Text 则是相应的信息。

LogCat 有如下 5 种信息：

① Log.v()：Log.v 的输出颜色是黑色，任何消息都会输出，这里的 v 代表 verbose(啰嗦)，这种输出信息使用较多。

② Log.d()：Log.d 的输出颜色是蓝色，仅输出调试 (debug) 信息，但它会输出上层的信息，过滤起来可以通过 DDMS 的 LogCat 标签来选择。

③ Log.i()：Log.i 的输出颜色是绿色，一般提示性的消息 (information)，它不会输出 Log.v 和 Log.d 的信息，但会显示 i、w 和 e 的信息。

④ Log.w()：Log.w 的输出颜色是橙色，可以看作为警告 (warning)，一般需要用户注意优化 Android 代码，同时选择它后还会输出 Log.e 的信息。

⑤ Log.e()：Log.e 的输出颜色是红色，可以看作为错误 (error)，这里仅显示红色的错误信息，对这些错误需要进行认真分析。

产生的 LogCat 信息存储在手机目录 data/log 中，但是 AVD 不会产生 LogCat 文件。

LogCat 窗口具有良好的过滤器功能，并可以定制过滤，另外，还提供了保存 LogCat 到本地磁盘的方法。图 1-10 所示为 LogCat 的工具栏。

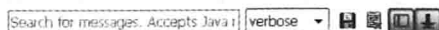


图 1-10 LogCat 工具栏

单击 LogCat 窗口左侧的  按钮，打开如图 1-11 所示的对话框，用于设置 LogCat 消息的过滤。

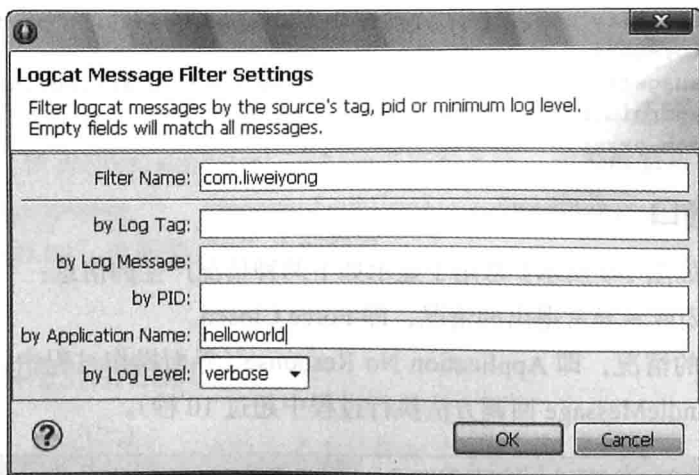


图 1-11 LogCat 消息过滤设置

### 1.3.3 DDMS 窗口

DDMS (Dalvik Debug Monitor Service, Dalvik 虚拟机调试监控服务) 为 Eclipse 和 AVD 及真正的 Android 设备架起了一座桥梁，提供了设备截屏、模拟电话呼叫以及虚拟地理坐标等服务，并能够针对特定的进程查看正在运行的进程及堆栈信息，DDMS 窗口如图 1-12 所示。

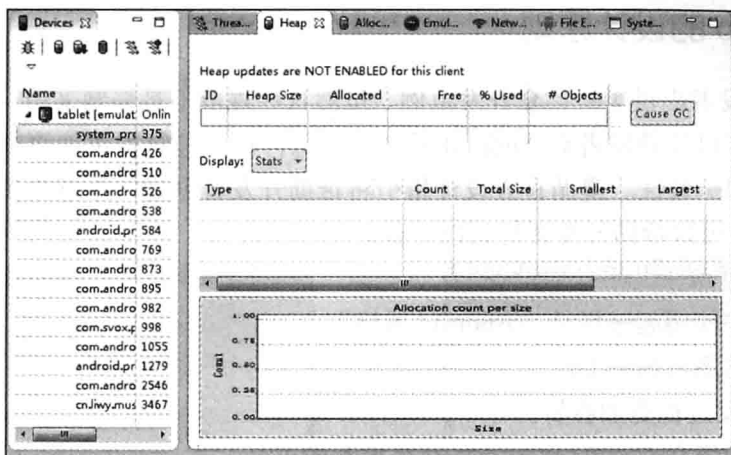


图 1-12 DDMS 管理器

下面详细阐述其中 File Explorer 窗口的使用方法。

File Explorer 窗口显示 Android 的文件系统，如图 1-13 所示。

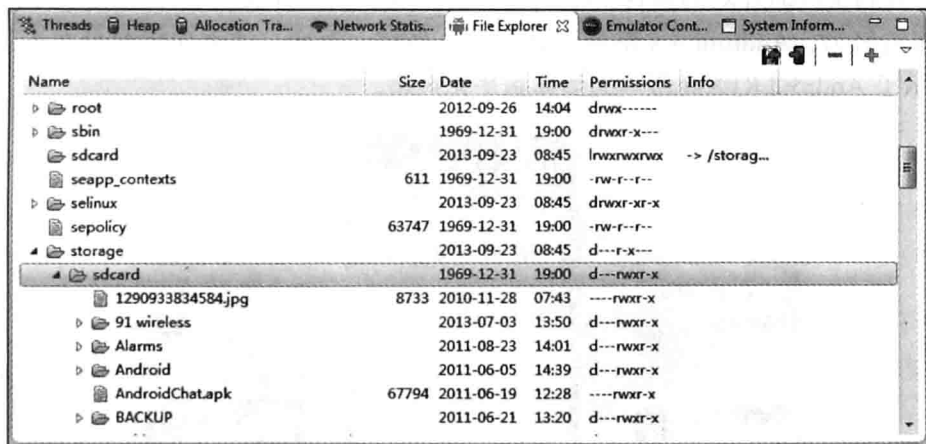


图 1-13 File Explorer 窗口

一般情况下，File Explorer 会有如下几个目录：

① data 对应手机的 RAM，会存放 Android 运行时的 Cache 等临时数据（/data/dalvik-cache 目录）；没有 root 权限时 apk 程序安装在 /data/app 中（只是存放 .apk 文件本身）；/data/data 中存放 AVD 中所有程序（系统 apk+ 第三方 apk）的详细目录信息。

② mnt/storage/sdcard 对应 microSD 卡。

③ system 对应手机的 ROM，操作系统以及系统自带 apk 程序等存放在这里。

选项卡右上角有“上传”、“下载”、“删除”和“新建文件夹”4 个按钮，分别用于实现把文件上传到 AVD、从 AVD 上下载文件到本地磁盘、删除文件及创建文件夹操作。可以直接将本地的文件向 File Explorer 窗口拖送，但是不支持从 AVD 向本地拖送。

## 1.4 移动 App 的设计原则

Google 为开发者提供了一整套开发范例。虽然这些规范不是强制采用，但按照手册来进行开发设计无疑可以让开发者对 Android 系统如何运行程序理解得更加清晰，用户的使用体验也能保证完好的一致性。遵循官方设计指引将帮助开发者创建一个感觉更加像“原装”的应用，设计指引还可以帮助开发者得到以下获益：

- 便于 App 适应于几乎任何设备。
- 使应用程序使用起来更具“Android 风格”。
- 提供用户熟悉习惯的 UI。
- 使 App 开发过程更加容易。
- 增加应用程序在 Google Play 市场获得推荐的机会。

以下是根据 Android 的官方设计指导文档<sup>①</sup>和项目开发实践整理而来。

### 1. 理解 Android 的外观和感觉

手机软件运行于手机操作系统的软件环境，界面的设计应该是基于这个应用平台的整体风格，这样有利于产品外观的整合。Google 在为其所有的产品提供一个一致的视觉体验上投入了非常多的努力，Android 4.x 整体风格简单、平面、干净，注重功能本身而不是形式感。图 1-14 演示了 Android KitKat 版本的整体扁平化风格。



图 1-14 Android KitKat 的扁平化风格

### 2. 基于心理模型的界面设计

用户界面应是基于用户的心理模型，而不是基于工程实现模型，它是把后台很复杂的事情设计成符合用户日常生活中常用的浏览方式或操作方式。这是设计师把生活中的细节和应用结合的凝聚点，用户的心理模型抓得越准，界面就会越优秀。在图 1-15 所示的“大众点

① 网址 <http://developer.android.com/design/patterns/index.html>。

评”的高级搜索界面中，价格搜索体现了本原则。

### 3. 以用户使用情景的思维方式做界面设计

例如微信应用，对于一个社交即时通讯产品，添加好友的功能是好友汇聚的来源。所以，将这么重要的功能放置在应用程序的哪个位置是非常重要的。图 1-16 中的几种添加好友的方式体现了本原则。



图 1-15 大众点评高级搜索界面



图 1-16 微信添加好友界面

### 4. 为不同的设备设计

当设计移动 App 的时候，首先要确保它能够在大部分的设备上正常运行。不仅要适用于不同的屏幕尺寸和屏幕方向，也要注意适用于低亮度的屏幕或者是对比度较差的屏幕，以及速度慢的低配置机器。同时需要注意：

- 使用对比较强的文字和元素颜色，例如在重要的元素上避免使用白色或者浅灰色，因为可能在较差的屏幕上看不见。
- 在不同亮度的环境下，不同的屏幕亮度（低亮度、高亮度、自动亮度）设置下检查设计效果。
- 即使在使用标准尺寸时，也要确认文字和 UI 元素在小屏幕或低分辨率屏幕上的显示能够足够大；也可以单独为这些屏幕设置特殊的文字或视觉元素显示尺寸。

### 5. 使用 Density-independent 定义布局

确保 UI 元素在不同分辨率的 Android 设备上看起来差不多大小是提供一致性体验里很重要的一部分。这看起来是一件非常费力的任务，但其实大可不必通过复杂的像素运算来得出每个按钮或字体在每个特定的屏幕上如何显示，可以让设备帮助开发者处理这个问题。通过 Density-independent pixels 的方式定义尺寸，需要确保在每个屏幕上显示的视觉元素的物

理尺寸一致。图 1-17 是实际使用中证明很好用的一个各视觉元素的尺寸设置建议。

## 6. 为不同的分辨率设计

为了在几乎所有的 Android 设备上清晰地显示应用界面，需要满足四种分辨率：低分辨率（LDPI）、中分辨率（MDPI）、高分辨率（HDPI）和超高分辨率（XXHDPI）。一般从 640×960 屏幕分辨率开始，然后缩小以适应其他分辨率屏幕。同时，一个叫做 XXHDPI 的标准已经被添加到下一代移动设备中作为支持，这些设备将有大约 480 DPI 的屏幕。所以需要在设计各视觉元素时提前做好准备将 HDPI 拓展至 200% 的 XXHDPI。

## 7. 考虑不同的系统版本

许多 Android 设备将不能够升级到最新的操作系统，并且新的系统往往也需要很长的时间才能全面占领市场。随着设备的更新换代，用户会逐渐不满足于过时的应用图标和控件样式。因此，需要尽可能提供最新的体验，如果打算支持应用程序运行在旧的平台，可以为这些设备创建一个单独的版本。

## 8. 为应用提供拓展组件和壁纸

善于利用 Android 的一些特殊优势，例如桌面组件、壁纸和消息通知。桌面组件可以让用户在不运行应用时接收更新，消息推送则可以帮助提升版本更新的安装量。Google 为设计师和开发者提供了各种方便通知用户的方式。Android 用户非常喜欢定制他们的设备，使其看起来富有个性，所以这些组件或墙纸就会给用户很大的弹性空间来做这些事情。

## 9. 为平板设备设计

如果开发者的目标就是做一个真正跨多终端的 Android 应用，那就必须同时考虑 Android 平板。官方设计指引为平板的 UI 和界面提供了多窗格布局的设计指引，以满足碎片化设备的统一化体验。平板和手机一样使用同样的图形库，但需要特别考虑平板使用的情景。例如说，相比于手机，人们通常使平板离自己的眼睛更远，并且输入没有那么精确。所以平板的 UI 需要更大的字体、更大的按钮和更多的留白。

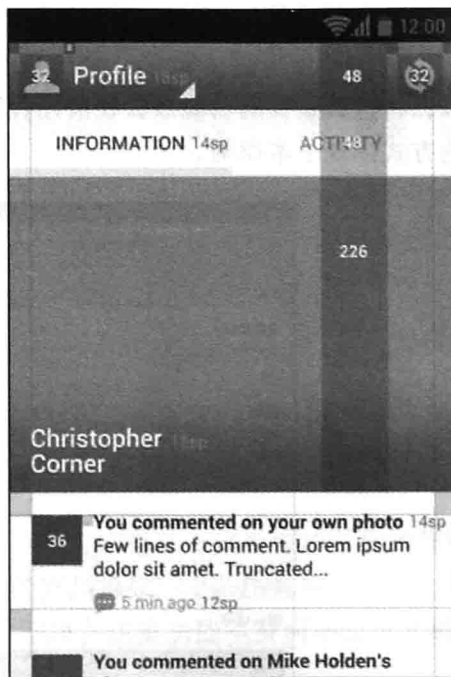


图 1-17 Density-independent 设置建议

# 1.5 移动 App 的设计风格

本节主要介绍现在最为流行的移动 App 的设计风格。



## 1.5.1 扁平化设计

扁平化设计 (Flat Design) 是一种设计风格术语, 它抛弃任何使得作品突显 3D 效果的特性。通俗地说即设计中不使用透视、纹理、阴影等效果。

设计师们如今似乎正朝扁平化风格倾斜, 因为这种风格干净利索, 现代气息浓郁, 而且能让设计师专注最重要的内容和信息要素。这样设计出来的作品不容易过时, 而且看起来简洁、高效。

Designmodo 设计师 Carrie Cousins 在网站上介绍了扁平化的五大特点, 以及“准”扁平化设计的优缺点, 具体如下。

### 1. 拒绝特效

扁平化设计仅仅采用二维元素。所有元素都不加修饰——阴影、斜面、突起、渐变, 会带来深度变化的设计都是不应该的。从图片框到按钮, 再到导航栏都干脆有力, 极力避免羽化、阴影这样的特效。

因为这种设计有着鲜明的视觉效果, 所以它所使用的元素之间有着清晰的层次和布局, 这使得用户能直观地了解每个元素的作用以及交互方式。在手机上, 因为屏幕的限制, 使得这一风格在用户体验上更有优势, 更少的按钮和选项使得界面干净整齐, 使用起来格外简单。图 1-18 显示了扁平化的界面设计风格。

### 2. 仅使用简单的元素

扁平化设计通常采用许多简单的用户界面元素, 诸如按钮或者图标之类。设计师们通常坚持使用简单的外形 (矩形或者圆形), 并且尽量突出外形, 这些元素一律为直角 (极少的一些为圆角)。这些用户界面元素可以方便用户点击, 极大地减少用户学习新交互方式的成本, 用户凭借以往经验就能大概知道每个按钮的作用。

UI 元素应该在保持高可用性的前提下尽可能的简单, 保证应用直观、易用, 无需引导。为了同时达到简单但直观的效果, 可以尝试为按钮填充深色, 以鼓励用户点击。

### 3. 注重排版

因为扁平化设计要求元素更简单, 排版的重要性就更为突出了, 排版好坏直接影响视觉效果, 甚至可能间接影响用户体验。

字体是排版中很重要的一部分, 字体的大小应该匹配整体设计, 高度美化的字体将与极简设计原则相冲突。字形上应该使用粗体, 文案要求精简、干练, 最终保证产品在视觉上和措辞上的一致性。字体选择上可以使用简单的无衬线字体, 通过字体大小和比重来区分元素。

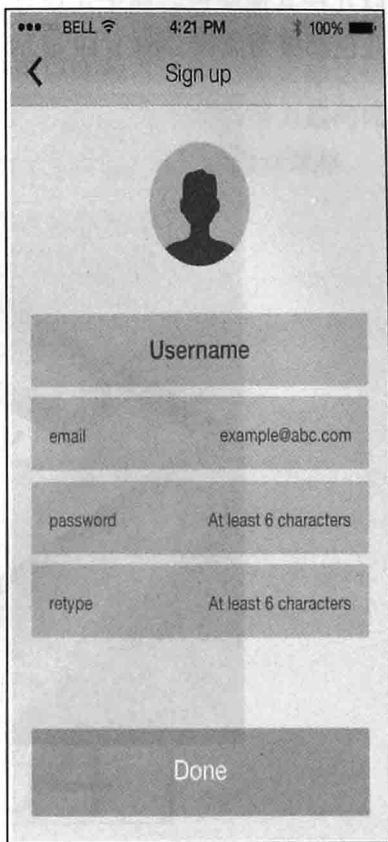


图 1-18 扁平化的 UI 设计风格

#### 4. 关注配色

色彩的使用对于扁平化设计来说也非常重要。扁平化设计通常采用比其他风格更明亮、更炫丽的颜色。同时，扁平化设计中的配色还意味着更多的色调。例如，其他设计最多只包含两三种主要颜色，但是扁平化设计中会平均使用 6 ~ 8 种颜色。

扁平化设计中往往倾向于使用单色调，尤其是纯色，并且不做任何淡化或柔化处理（最受欢迎的颜色是纯色和二次色）。其主要、次要颜色通常都是非常大众化的颜色，然后再配以几种其他颜色。扁平化设计的另一个趋势在于复古颜色的使用——浅橙色、紫色、绿色、蓝色都极为流行。图 1-19 显示了扁平化的界面色彩搭配风格。

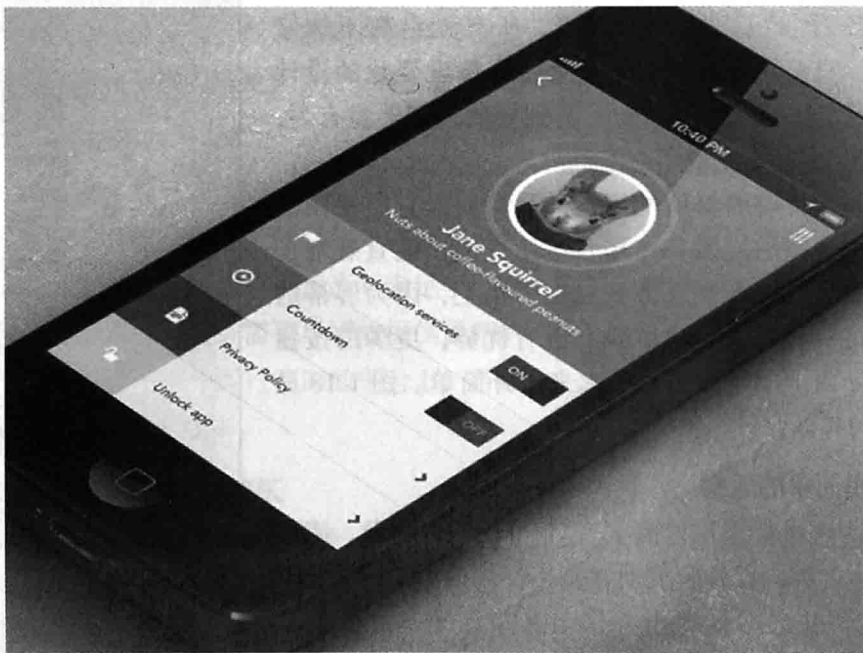


图 1-19 扁平化的色彩搭配风格

#### 5. 简约方案

扁平化设计在整体上趋近极简主义设计理念。设计师要尽量简化自己的设计方案，避免不必要的元素出现在设计中。简单的颜色和字体就足够了，如果还想添加则尽量选择简单的图案。

移动 App 设计中一个比较重要的设计因素是效率的最大化。前面介绍的界面扁平注重通过弱化视觉效果来强化功能主义。但界面风格的扁平化还远远不够，还应该从信息架构的角度进一步应用扁平化的设计理念，这就是信息架构的扁平化趋势。信息框架扁平的目的是减少信息层级，追求信息到达用户的最短距离。通常，移动 App 的层级关系不比网页直观，不存在类似站点地图的功能，允许用户任意跳转。纵深的返回机制会增加用户操作成本，扁平的信息框架会从根本上解决上述问题。

## 1.5.2 卡片式设计

卡片是交互信息的承载体，通常以矩形的方式呈现，就像信用卡。所有卡片的主要特点就是交互性。它们不仅仅提供了信息，而且用另外一种委婉的方式要求一次互动。卡片通常包括与用户交互的方法。

卡片式设计是在栅格的基础上更进一步将整个页面的内容切割为  $N$  个区域，不仅能给人很好的视觉一致性，而且更易于设计上的迭代。这样的设计在处理 PC 和移动多平台页面一致性时有很好的效果。现在卡片式设计的应用场景非常广泛，最为常见的就是浏览器上的选项卡了。浏览器把要浏览的网页做成一个一个卡片的样子，用户可以来抽取、移动和删除这些卡片，若干个选项卡还可以独立成浏览堆栈，显得非常方便。Google 全线产品都努力趋向简洁，其中，卡片式设计成为 Google 最重视的元素。图 1-20 演示了 Google 卡片式设计风格。

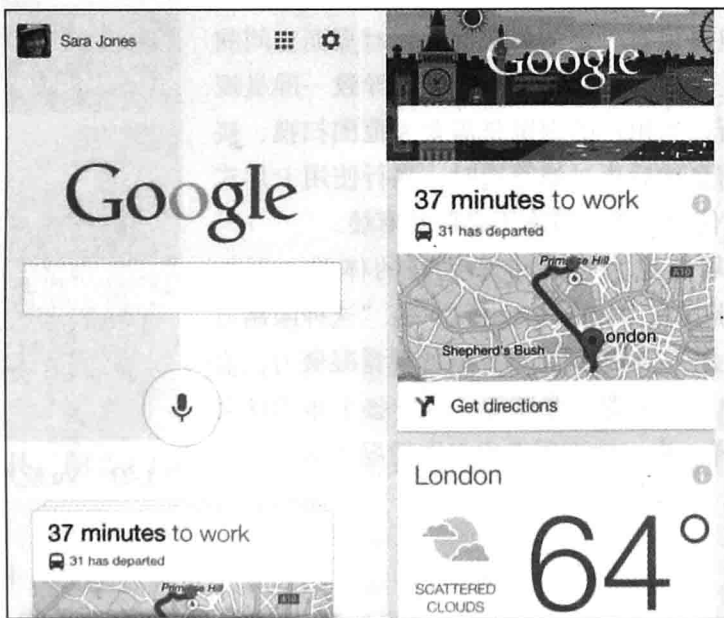


图 1-20 Google 卡片式设计风格

Google 的这种卡片式设计一改之前信息显示的杂乱无章，在某种意义上甚至重新定义了信息显示。首先，它直接排除了多余的信息，尤其是广告；其次，它淘汰了超链接，能让用户专注于信息本身而不是寻找信息。另外值得一提的是，Google 卡片能在各种尺寸的屏幕上实现无差别体验，甚至是在 Glass 上也一样。

对于移动设备碎片化的屏幕来说，卡片是完美的设计形式。在手机上，卡片通常以垂直方式展现（如信息流）。在平板上，卡片也可以以水平方式呈现（当平板横置时，可添加一栏新内容），卡片的高度可根据内容进行调整。总结起来，卡片式设计具有如下优点：

- 卡片能够抓住眼球。相对于过度的无理由的文本，卡片是更好的选择。
- 卡片是响应式的。卡片可以很好地帮助手机浏览器实现响应式设计。

- 卡片易读。因为空间有限，卡片不能说得太多，但是这是一件好事！读者如果想要知道更多的话就会点击它。
  - 卡片是可共享的。卡片使读者能够快速并轻松地通过社交、手机和邮件平台分享内容。
- 目前，卡片式设计主要用来解决三类问题：

1) 信息分类。例如 Google +，把 feed 信息做成卡片样式易于浏览，常见的瀑布流式布局的信息展示其实也是一种卡片布局。

2) 导航。例如 Evernote 或 Vu 应用（如图 1-21 所示）里的卡片，类似传统 Tab 栏的功能来区分不同的内容和功能，在移动端使用居多，与手势操作结合，易于理解和操作。

3) 任务管理。卡片式设计运用在移动端可实现多任务管理。

卡片式设计自身也有一定的局限性，它对页面空间的消耗非常大，并且将内容元之间强行割裂，导致一屏呈现的信息量很小。所以当用户的浏览是需要大范围扫视、接收大量相关性的信息然后再过滤筛选时，强行使用卡片式设计会降低用户的使用效率，带来不必要的麻烦。

但是这种局限性也为设计带来更好的体验，正如 Android 的用户体验总监 Matias Duarte 所说：“这种限制迫使我们要做到‘专注’，要特别注意界面的视觉凝聚力，迫使我们在一张卡片上只放置一张图像，这样整个布局就会显得很整洁。”这样一来，就不用考虑多图情况下的排版问题了。

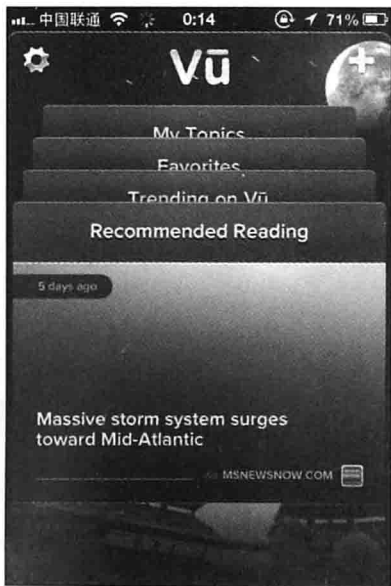


图 1-21 Vu 应用的卡片式设计风格

## ADT 中的 UI 设计

### 2.1 图形布局编辑器

ADT (Android Development Tools, 是一款用于开发 Android 应用程序的 Eclipse 插件) 为 Android UI 提供了一种图形布局编辑器, 当打开 layout 下的 xml 布局文件时, 系统自动打开这个 UI 设计工具。图 2-1 是打开上一章 HelloWorld 项目的 activity\_main.xml 布局文件时的显示效果。

#### 1. Palette 窗口

Palette 窗口提供了构建 Android UI 的基本控件, 可以直接拖动 Palette 窗口中的控件到图 2-1 中部的布局容器中。Palette 窗口如图 2-2 所示。

Palette 窗口提供了控件的不同预览效果, 包括仅显示图标、显示图标和文本等。

- Form Widgets 包括 TextView、Button、CheckBox、RadioButton、Spinner、ProgressBar 和 SeekBar 等。
- Text Fields 包括各种定制的 TextField 控件。
- Layouts 包括 GridLayout、LinearLayout、RelativeLayout、FrameLayout、Include Other Layout、Fragment 和 TableLayout 等。
- Composite 包括 ListView、GridView、ScrollView、SearchView、SlidingDrawer、TabHost 和 WebView 等。
- Images & Media 包括 ImageView、ImageButton、Gallery、MediaController 和 VideoView。

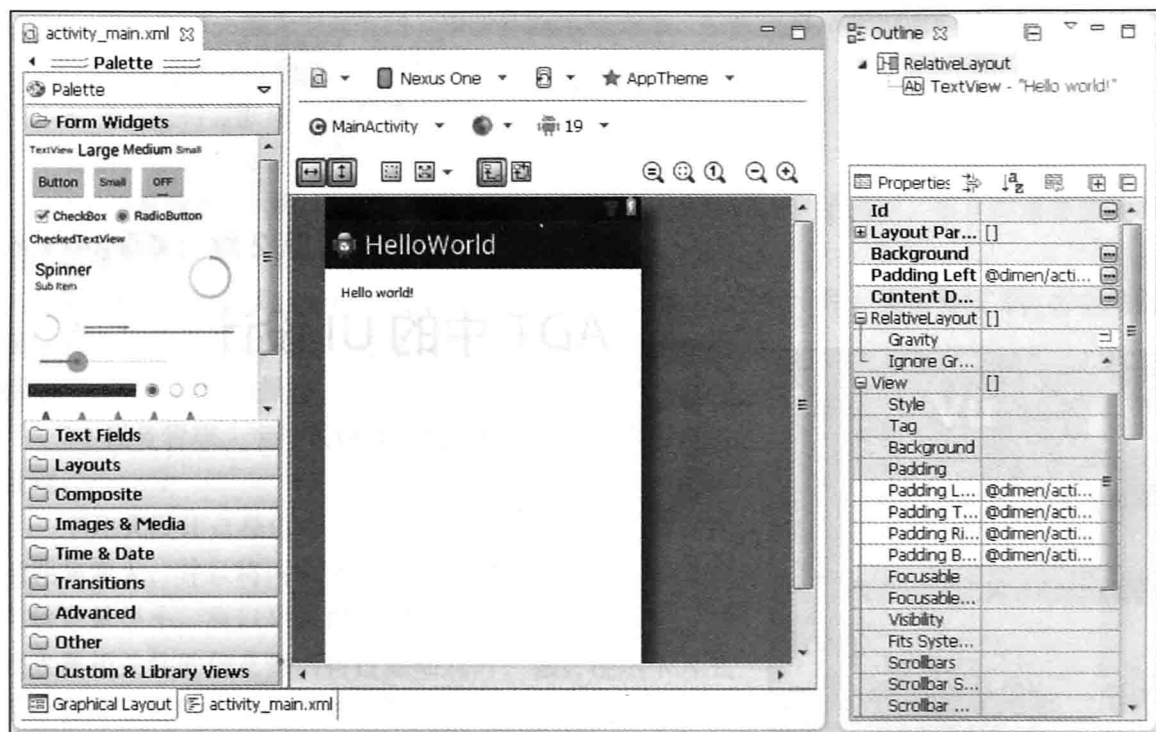


图 2-1 图形布局编辑器窗口

- Time & Date 包括 TimePicker、DatePicker、CalendarView、Chronometer 和 AnalogClock。
- Transitions 包括 ImageSwitcher、AdapterViewFlipper、StackView、TextSwitcher、ViewAnimator、ViewFlipper 和 ViewSwitcher。
- Advanced 包括 requestFocus、View、ViewStub、GestureOverlayView、SurfaceView、ZoomControls 和 NumberPicker 等。
- Other 提供了一些特有 API 版本包含的控件，如 TextClock 等。
- Custom & Library Views 包含了用户自定义控件，以及第三方 API 提供的控件。

## 2. Configuration 面板

Configuration 面板可以根据项目的不同应用环境设置不同的参数。Configuration 面板如图 2-3 所示。

Configuration 面板提供的功能包括：

- 不同屏幕尺寸的预览效果。
- 横屏、竖屏预览效果。

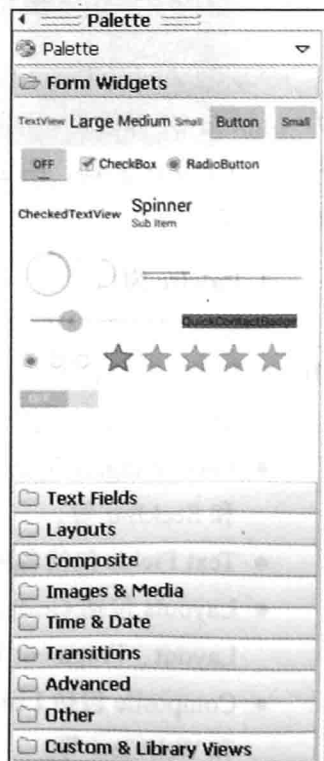


图 2-2 Palette 窗口

- 不同的显示主题预览效果。
- 不同的语言环境预览效果。
- 不同的 SDK 版本预览效果。
- 预览缩放模式控制等。



图 2-3 Configuration 面板

### 3. 布局容器窗口

布局容器窗口就是中部的图形化编辑区域，该区域展示了在当前硬件环境配置下的显示效果。可以直接从 Palette 窗口拖动控件到布局容器窗口中，如图 2-4 所示。

在布局容器窗口中右击容器中的控件，在打开的菜单中可以执行查看控件的属性、创建动画及动画预览、展开内嵌的布局等命令。

### 4. Outline 窗口

Outline 窗口包括两个部分（如图 2-1 右侧所示）：上方的窗口显示了当前布局的层次结构，也可以直接从 Palette 窗口拖动控件到 Outline 窗口中。下方的窗口是当前选中控件的属性设置窗口，在属性设置窗口中，可以设置控件的布局参数、显示样式等信息。

属性窗口有如下特点：

- 重要的属性使用粗体显示。
- 当前控件编辑过的属性值使用蓝色显示。
- 显示部分默认属性：例如“Text Color Link”属性没有定义，图中显示的是系统的默认值（@ android: color/holo\_blue\_light），并且还会显示颜色值的预览效果。
- 属性可以嵌套，例如布局属性 width、hight 等。
- 属性可以按照字符顺序排列，方便查找。
- 高级属性默认隐藏，可以单击显示隐藏属性的按钮来显示。

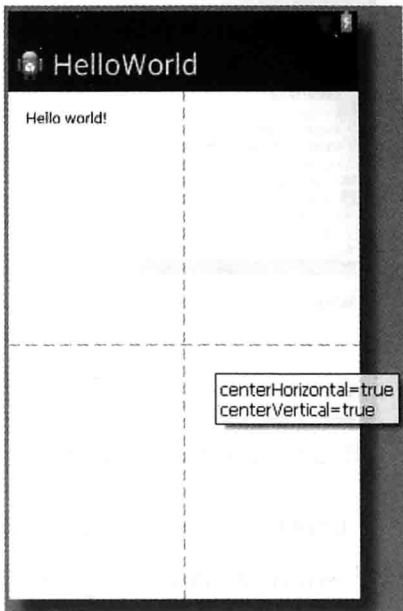


图 2-4 布局容器窗口

## 2.2 几种常见的布局方式

布局指的是 Activity 中 UI 的结构。Android 为构建用户界面提供了灵活的布局，这些布局包括线性布局、表格布局、相对布局、帧布局和绝对布局等。布局里面还可以套用其他布局，这样就可以实现界面的多样化以及设计的灵活性。



## 2.2.1 创建布局

创建布局文件的步骤如下：

①在 Android 项目的 res/layout 节点上右击，执行 New → Android XML File 命令，打开 New Android XML File 对话框，如图 2-5 所示。

②在 File 文本框中输入布局文件名，在 Root Element 列表中选择布局的根节点，单击 Next 按钮，打开 Choose Configuration Folder 对话框，如图 2-6 所示。

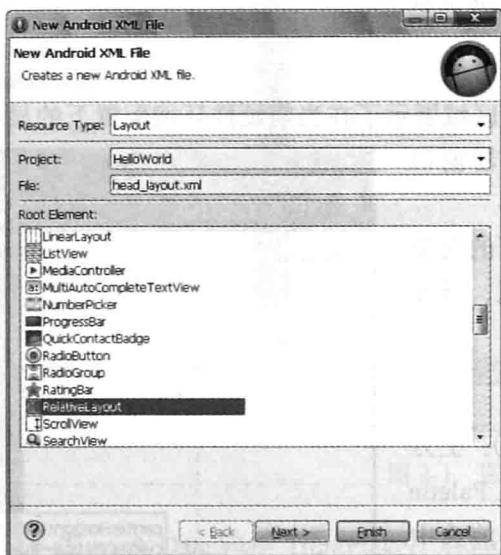


图 2-5 New Android XML File 对话框

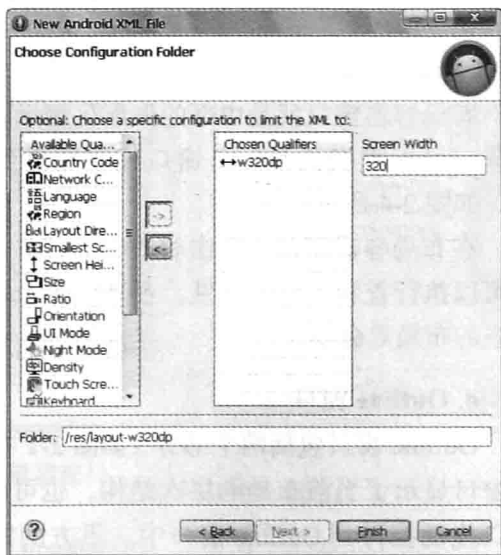



图 2-6 Choose Configuration Folder 对话框

Choose Configuration Folder 对话框用于设置对 XML 文件的特殊限制。例如在左侧列表选择 Screen Width，单击  按钮，将其加入到右侧的 Chosen Qualifiers 列表中，并在 Screen Width 框中输入值，则新的布局文件将被放置在特定的文件路径中。

③单击 Finish 按钮完成布局文件的创建。

View 对象是 Android 平台中用户界面体现的基础单位，组成 Android 界面的基本 UI 元素由 android.view.View 提供实现，该类是 Android 中用于绘制块状视图的基类，并负责在它所辖的这个矩形区域之中所有测量、布局、焦点转换、滚动以及按键/触屏手势的处理。作为一个用户界面对象，View 同时也担任着用户交互关键点以及交互事件接受者的角色。

需要注意的是，每个布局文件必须包含一个根元素，该元素必须是一个 View 或者 ViewGroup 对象。一旦定义了根元素，就可以加入额外的布局对象或者 Widget 控件作为其子元素，逐渐构成完整布局。

在编译工程时，每个 XML 布局文件都被编译成一个 View 资源。在 Activity.onCreate() 回调方法的实现中，调用 setContentView() 方法并将布局资源的引用 R.layout.layout\_file\_name 作为该方法的参数。

## 2.2.2 相对布局

相对布局 (RelativeLayout) 方式是将 ViewGroup 以相对位置显示它的子视图元素, 一个视图可以指定相对于它的兄弟视图的位置 (例如在给定视图的左边或者下面) 或相对于 RelativeLayout 的特定区域的位置 (例如底部对齐或中间偏左)。

下面是一个相对布局的示例 (LayoutProject/activity\_relative\_layout.xml<sup>①</sup>):

```

01 <RelativeLayout
02     xmlns:android="http://schemas.android.com/apk/res/android"
03     xmlns:tools="http://schemas.android.com/tools"
04     android:layout_width="match_parent"
05     android:layout_height="match_parent"
06     tools:context=".RelativeLayoutActivity" >
07
08     <TextView
09         android:id="@+id/textView1"
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:text=" 请输入信息: "
13         android:textSize="20sp" />
14
15     <Button
16         android:id="@+id/button1"
17         android:layout_width="wrap_content"
18         android:layout_height="wrap_content"
19         android:layout_alignParentBottom="true"
20         android:layout_alignParentRight="true"
21         android:text=" 保存 " />
22
23     <Button
24         android:id="@+id/button2"
25         android:layout_width="wrap_content"
26         android:layout_height="wrap_content"
27         android:layout_alignBaseline="@+id/button1"
28         android:layout_alignBottom="@+id/button1"
29         android:layout_marginRight="5dp"
30         android:layout_toLeftOf="@+id/button1"
31         android:text=" 清空 " />
32
33     <EditText
34         android:id="@+id/editText1"
35         android:layout_width="match_parent"
36         android:layout_height="match_parent"
37         android:layout_above="@+id/button1"
38         android:layout_alignLeft="@+id/textView1"
39         android:layout_alignParentRight="true"
40         android:layout_below="@+id/textView1"

```

① 项目代码参见华章网站 ([www.hzbook.com](http://www.hzbook.com)) 中本教材的教辅资源。

```

41         android:ems="10"
42         android:inputType="textMultiLine" >
43
44         <requestFocus />
45     </EditText>
46
47 </RelativeLayout>

```

在使用图形布局编辑器时，先拖动一个 TextView 控件到容器的左上角，再拖动一个 Button 到屏幕的右下角，再拖动一个 Button 到上一个 Button 的左侧，最后拖动一个多行的 EditText 到 TextView 的下方，并拖动高度到 Button 的上方。图 2-7 所示演示了程序的运行效果。

activity\_relative\_layout.xml 的 04 ~ 05 行的 android:layout\_width 和 android:layout\_height 是每个 View Group 都必须包含的属性，用于设置 View 的宽度和高度。可以使用精确值来指定宽度和高度。更常见的是把 View 对象的大小设为和它的内容相合适，或者尽可能大的填满其父对象（分别对应 wrap\_content：表示大小刚好足够显示当前控件里的内容；match\_parent：填满父控件的空白）。也可以使用 px、in、mm、pt、dp/dip 和 sp 等单位。推荐使用 dp。

09 行的 android:id 属性用于设置 View 对象的 ID，用来在布局树中唯一标识它。当应用程序被编译时，这个 ID 作为一个整数引用。XML 中的 ID 语法如下：

```
android:id="@+id/id_index"
```

字符串前的 @ 符号表示 XML 解析器应该解析和扩展剩下的 ID 字符串，并将其作为 ID 资源。“+”符号表示这是一个新的资源名字，它必须被创建且加入到 R.java 文件中。Android 框架提供一些其他的 ID 资源。当引用一个 Android 资源 ID 时，不需要“+”符号，但是必须添加 Android 包名字空间，例如：android:id="@android:id/empty"。

下面介绍相对布局时常用的属性。

(1) 设置控件相对于另一个视图的位置

- ① android:layout\_below 在某元素的下方。
- ② android:layout\_above 在某元素的上方。
- ③ android:layout\_toLeftOf 在某元素的左边。
- ④ android:layout\_toRightOf 在某元素的右边。

(2) 设置控件相对于另一个控件的对齐方式

- ① android:layout\_alignBaseline 本元素的基线和某元素的基线对齐。
- ② android:layout\_alignTop 本元素的上边缘和某元素的上边缘对齐。
- ③ android:layout\_alignLeft 本元素的左边缘和某元素的左边缘对齐。

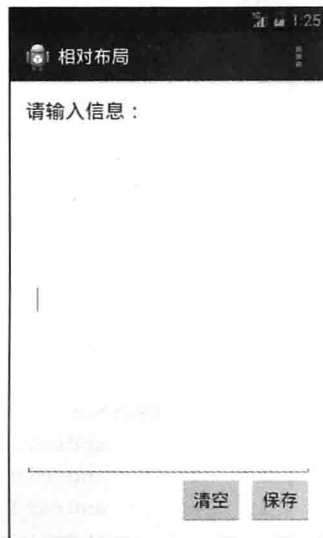


图 2-7 相对布局示例

④ `android:layout_alignBottom` 本元素的下边缘和某元素的下边缘对齐。

⑤ `android:layout_alignRight` 本元素的右边缘和某元素的右边缘对齐。

(3) 设置控件相对于父控件的对齐方式

① `android:layout_alignParentBottom` 贴紧父元素的下边缘。

② `android:layout_alignParentLeft` 贴紧父元素的左边缘。

③ `android:layout_alignParentRight` 贴紧父元素的右边缘。

④ `android:layout_alignParentTop` 贴紧父元素的上边缘。

(4) 设置控件的方向

① `android:layout_centerHorizontal` 在父元素中水平居中。

② `android:layout_centerVertical` 在父元素中垂直居中。

③ `android:layout_centerInparent` 相对于父元素完全居中。

以下 4 个属性设置控件的留白

① `android:layout_marginLeft` 当前控件左侧的留白。

② `android:layout_marginRight` 当前控件右侧的留白。

③ `android:layout_marginTop` 当前控件上方的留白。

④ `android:layout_marginBottom` 当前控件下方的留白。

**【`android:padding` 与 `android:layout_margin` 的区别】**

`android:padding` 指该控件内部内容, 如文本距离该控件的边距。`android:layout_margin` 指该控件距离父控件的边距。两者的区别如图 2-8 所示。

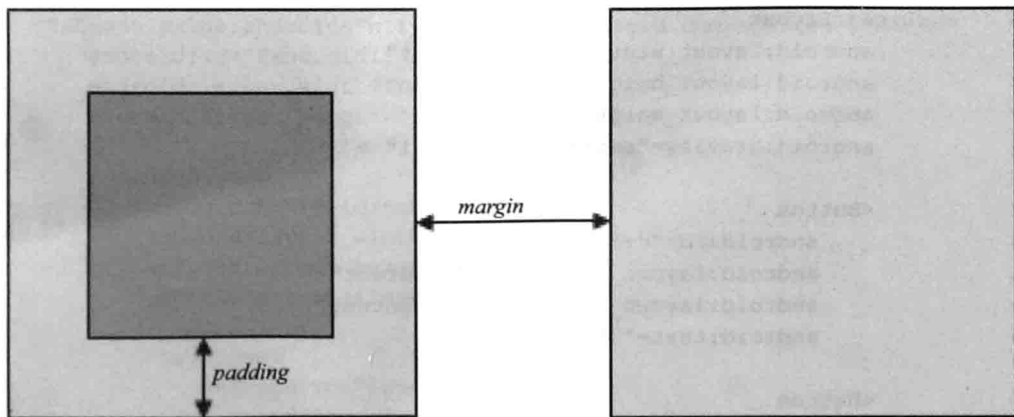


图 2-8 `padding` 与 `layout_margin` 的区别

### 2.2.3 线性布局

线性布局 (LinearLayout) 方式是将 ViewGroup 以线性方向显示它的子视图元素, 即后一个元素垂直或水平显示在上一个子元素之后。

下面是一个用线性布局实现上一节 UI 设计效果的示例 (LayoutProject/activity\_linear\_layout.xml):

```
01 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
02     xmlns:tools="http://schemas.android.com/tools"
03     android:layout_width="match_parent"
04     android:layout_height="match_parent"
05     android:orientation="vertical"
06     tools:context=".RelativeLayoutActivity" >
07
08     <TextView
09         android:id="@+id/textView1"
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:text=" 请输入信息: "
13         android:textSize="20sp" />
14
15     <EditText
16         android:id="@+id/editText1"
17         android:layout_width="match_parent"
18         android:layout_height="match_parent"
19         android:layout_weight="1"
20         android:ems="10"
21         android:inputType="textMultiLine" >
22
23         <requestFocus />
24     </EditText>
25
26     <LinearLayout
27         android:layout_width="match_parent"
28         android:layout_height="match_parent"
29         android:layout_weight="6"
30         android:gravity="center_horizontal" >
31
32         <Button
33             android:id="@+id/button2"
34             android:layout_width="wrap_content"
35             android:layout_height="wrap_content"
36             android:text=" 清空 " />
37
38         <Button
39             android:id="@+id/button1"
40             android:layout_width="wrap_content"
41             android:layout_height="wrap_content"
42             android:text=" 保存 " />
43     </LinearLayout>
44
45 </LinearLayout>
```

线性布局中有两个重要的属性:

### (1) android: orientation

该属性用于设置布局中子视图的显示方式。如果 android: orientation 取值为 “horizontal”，则子视图水平显示，这也是线性布局默认的显示方式。如果 android : orientation 取值为 “vertical”，则子视图垂直显示。

### (2) android: layout\_weight

该属性用于给一个线性布局中的诸多子视图的重要程度赋值。所有的视图都有一个 layout\_weight 值，默认为 0。若赋一个高于 0 的值，则将父视图中的可用空间分割，分割大小具体取决于每一个视图的 layout\_weight 值以及该值在当前屏幕布局的整体 layout\_weight 值和在其他视图屏幕布局的 layout\_weight 值中所占的比率。

### 【 android: gravity 与 android: layout\_gravity 的区别 】

android : gravity 属性用于控制布局中控件的对齐方式。如果是没有子控件的控件设置此属性，则表示其内容的对齐方式，例如 TextView 里面文字的对齐方式；若有子控件的控件设置此属性，则表示其子控件的对齐方式。gravity 的常用值有 top、bottom、left、right、center 等，可以使用 “|” 将多个值连接。android: layout\_gravity 用于设置控件在父布局容器中的对齐方式。

## 2.2.4 帧布局

帧布局 (FrameLayout) 方式是将 ViewGroup 中的视图自动地按照层次堆放在左上角，后加进来的控件覆盖前面的控件。定义任何空间位置相关的属性都毫无意义。

TabHost 是一个典型的帧布局使用示例 (LayoutProject/activity\_frame\_layout.xml):

```

01 <?xml version="1.0" encoding="utf-8"?>
02 <TabHost xmlns:android="http://schemas.android.com/apk/res/android"
03     android:id="@android:id/tabhost"
04     android:layout_width="fill_parent"
05     android:layout_height="fill_parent" >
06
07     <LinearLayout
08         android:id="@+id/tablayout"
09         android:layout_width="fill_parent"
10         android:layout_height="fill_parent"
11         android:orientation="vertical" >
12
13         <TabWidget
14             android:id="@android:id/tabs"
15             android:layout_width="fill_parent"
16             android:layout_height="wrap_content" />
17
18         <FrameLayout
19             android:id="@android:id/tabcontent"
20             android:layout_width="fill_parent"
21             android:layout_height="fill_parent" />
22     </LinearLayout>
23
24 </TabHost>

```

图 2-9 所示演示了程序的运行效果。

在 Android 中,除了上面介绍的布局方式外,还有绝对布局 (AbsoluteLayout)、栅格布局 (GridLayout)、表格布局 (TableLayout),以及一些用于特定场景的集成布局,如 ListView、GridView、MapView、WebView、VideoView 和 TabHost 等。

## 2.3 优化布局

本节主要介绍布局的复用、布局的多设备支持和使用 Hierarchy Viewer 调试布局性能等知识。

### 2.3.1 复用布局

在 Android 中,通常使用两种方式来复用布局,一种是通过 <include> 标签来嵌入布局,一种是通过 Fragment 来复用布局。本节主要介绍 <include> 的用法,Fragment 将在第 11 章介绍。

将 2.2.2 节的 TextView 和 EditText 控件放置在 head.xml 布局文件中,然后使用 <include> 将 head.xml 包含在布局容器中。示例 (LayoutProject/activity\_include\_layout.xml) 如下:

```

01 <?xml version="1.0" encoding="utf-8"?>
02 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
03     android:layout_width="fill_parent"
04     android:layout_height="fill_parent"
05     android:orientation="vertical" >
06
07     <include
08         android:layout_width="fill_parent"
09         android:layout_height="fill_parent"
10         android:layout_weight="1"
11         layout="layout/head" />
12
13     <LinearLayout
14         android:layout_width="match_parent"
15         android:layout_height="match_parent"
16         android:layout_weight="4"
17         android:gravity="center_horizontal" >
18
19         <Button ...
20             android:text=" 清空 " />
21
22         <Button ...

```

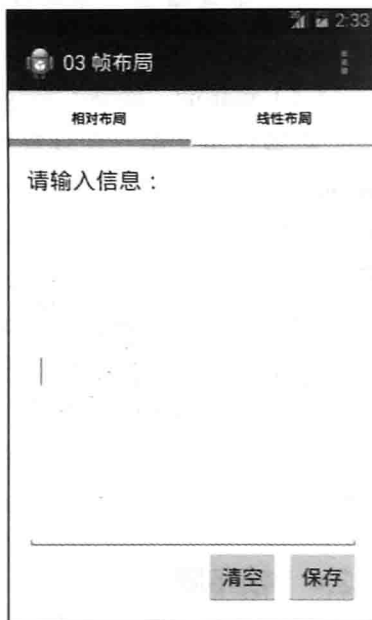


图 2-9 帧布局示例



```

23         android:text="保存" />
24     </LinearLayout>
25
26 </LinearLayout>

```

使用 `<include>` 时需要注意:

- 如果 `<include>` 指定了 id, 就不能把它里面的控件当成主布局中的控件来直接获得, 必须先获得这个 XML 布局文件, 再通过这个布局文件的 `findViewById()` 方法来获得其子控件。

例如, 如果需要在相对布局中将一个控件放置在 `<include>` 包含的布局的下方, 这时候就需要设置 `<include>` 的 id。如果此时需要设置 `<include>` 包含的布局中的控件, 则会抛出 `java.lang.NullPointerException` 异常。

- 如果在一个布局文件里使用 `<include>` 多次导入相同的布局文件, 则获取 `<include>` 中控件有直接和间接两种方式, 且只有间接方式能获取相同 id 的控件。

例如, `<include>` 的布局中有一个 id 为 `text` 的 `TextView` 控件, 则使用:

```

01 textView1 = (TextView) findViewById(R.id.text);
02 textView2 = (TextView) findViewById(R.id.text);

```

此时, 只能获得 `textView1` 的引用。

如果使用:

```

01 textView1 = (TextView) findViewById(R.id.include1).findViewById(R.id.text);
02 textView2 = (TextView) findViewById(R.id.include2).findViewById(R.id.text);

```

此时, 两个 `TextView` 实例都能获得。

- 使用 `<include>` 和 `<merge>` 的组合可以优化 UI 层次结构。

例如, 在一个 `FrameLayout` 中包含一个 `ImageView` 和一个 `TextView`。此时, 使用 `hierarchyviewer.bat` (关于 `Hierarchy Viewer` 的使用将在 2.3.3 节介绍) 工具查看当前 UI 结构, 可以看到有两个重复的 `FrameLayout` 节点, 这就造成布局层次的浪费。此时, 使用 `<merge>` 标签代替 `<FrameLayout>` 便可解决 `FrameLayout` 节点重复造成的浪费。

注意, `<merge>` 一般用于 XML 布局文件的根元素, 如果在代码中使用 `LayoutInflater.inflate()` 加载一个以 `merge` 为根元素的布局文件时, 需要使用 `View` 的 `inflate(int resource, ViewGroup root, boolean attachToRoot)` 方法指定一个 `ViewGroup` 作为其容器, 并且要设置 `attachToRoot` 的值为 `true`。

## 2.3.2 多设备支持

Android 应用的多设备支持包括语言、布局、平台版本等多种因素, 本节主要介绍布局的多设备支持技术。

Android 用两种常规属性来分类设备屏幕: 尺寸和像素密度。尺寸通常分为 4 类: `small`、

normal、large 和 xlarge。像素密度通常也分为 4 种：low (ldpi)、medium (mdpi)、high (hdpi) 和 extra high (xhdpi)。

为了声明用于不同屏幕的不同布局文件和图像资源文件，必须把这些可选的资源文件分别放在不同的目录。

### 1. 创建不同屏幕尺寸的布局文件

为了提高在不同设备屏幕上的用户体验，应当为每一种想要支持的屏幕尺寸创建一个独有的 XML 布局文件。每一布局文件应当存放到恰当的资源目录下，该目录以屏幕尺寸作为后缀。例如，一个用于大屏幕的布局文件应当放在 `res/layout-large/` 目录下。例如，下面的项目包含了一个默认的 layout 和一个可选的用于大屏幕的 layout 目录：

```
MyProject/
  res/
    layout/
      main.xml
    layout-large/
      main.xml
```

目录中的布局文件名必须保持一致，但是它们的内容可以不同，以便提供优化的 UI 来支持相对应的屏幕尺寸。

### 2. 创建横竖屏的布局文件

屏幕的方向 (landscape 或 portrait) 也被认为是一种屏幕尺寸的变化，所以大多数应用都应该修改 layout 布局来提高用户在不同屏幕方向上的体验。

下面是另外一个案例，在这个项目中使用了可选的横向布局来支持横向屏幕：

```
MyProject/
  res/
    layout/
      main.xml
    layout-land/
      main.xml
```

## 2.3.3 使用 Hierachy Viewer 调试用户界面

在 Android 的 SDK 工具包中，有一款叫 Hierachy Viewer 的可视化调试工具，可以很方便地在开发者设计、调试和调整界面时，提高开发效率。Hierachy Viewer 的主要功能有：

- 从可视化的角度直观地获得 UI 布局设计结构和各种属性的信息，帮助开发者优化布局设计。
- 结合调试帮助观察特定的 UI 对象进行 invalidate 和 requestLayout 操作的过程。

为了说明 Hirerachy Viewer 的使用，这里以一个浏览图片的 GalleryProject 为例。用户界面代码如下：

```

01 <?xml version="1.0" encoding="utf-8"?>
02 <RelativeLayout
03     xmlns:android="http://schemas.android.com/apk/res/android"
04     android:layout_width="match_parent"
05     android:layout_height="match_parent">
06     <ImageSwitcher android:id="@+id/switcher"
07         android:layout_width="match_parent"
08         android:layout_height="match_parent"
09         android:layout_alignParentTop="true"
10         android:layout_alignParentLeft="true" />
11     <Gallery android:id="@+id/gallery"
12         android:background="#55000000"
13         android:layout_width="match_parent"
14         android:layout_height="60dp"
15         android:layout_alignParentBottom="true"
16         android:layout_alignParentLeft="true"
17         android:gravity="center_vertical"
18         android:spacing="16dp" />
19 </RelativeLayout>

```

## 1. 启动 Hierachy Viewer

模拟器启动 GalleryProject 程序后，双击 SDK/tools 目录下的 Hirerachyview.bat 程序，打开 Hirerachy Viewer 浏览器，如图 2-10 所示。

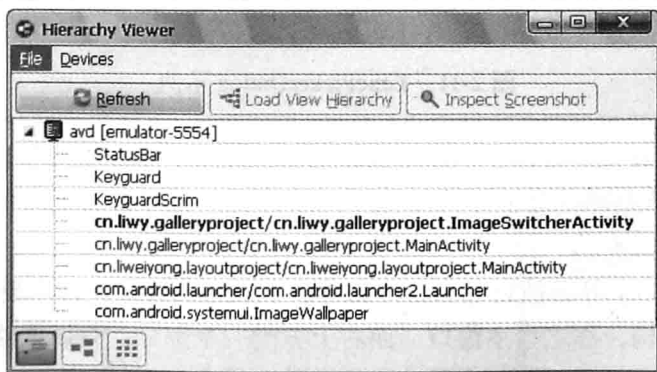



图 2-10 Hirerachy Viewer 浏览器

也可以在 Eclipse 中，执行 Windows → Open Perspective → Hierarchy View 命令打开 Hirerachy View 窗口。

可以看到，工具栏上除了刷新按钮外，还有两个按钮，其功能如下：

- Load View Hierarchy 查看界面的控件层次。
- Inspect Screenshot 进入界面精确查看模式。

## 2. 查看界面的控件层次

单击设备列表下的 cn.liwy.galleryproject/cn.liwy.galleryproject.ImageSwitcherActivity，再单击工具栏上的 ，显示如图 2-11 所示界面。

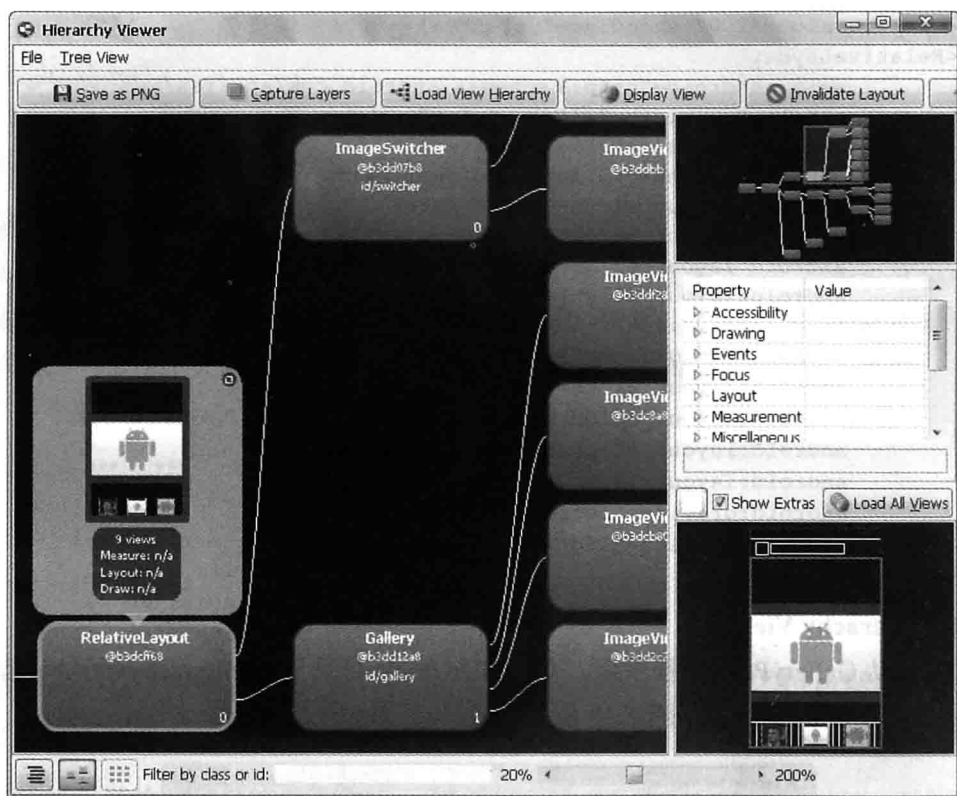


图 2-11 TimePickerDialog 示例

在屏幕的左下方有三个按钮，当点击最左边的按钮时，返回到模拟器的列表界面（也就是刚进入 Hierarchy Viewer 的界面），中间按钮则是 Load View Hierarchy 的主界面，用户可以在这两种状态中来回切换。


Load View Hierarchy 界面被划分为四个部分，分别是最左边（面积最大一块），该部分显示界面控件的层次结构，称之为主窗口。而右上方的一个部分，是以缩略图的方式显示整个应用中的各控件的层次关系，当一个界面中的控件比较多时，可以通过鼠标在该显示区域进行移动，则左边的主窗口中会相应显示相关的控件信息。右边区域的中间部分，显示当前控件的具体属性，是控件的属性面板。而右下角部分的区域，则是当用户点击界面中的某个控件时，会在该部分显示用户所点击的控件在界面中的具体位置。并使用红色部分标出，方便用户辨识。



**注意** Layout View 列出的 View 结构是从视图的根节点开始的，例如针对 GalleryActivity 使用的 Layout，它的底层基础布局 RelativeLayout 实际上是放在一个 FrameLayout 里的，该 FrameLayout 又是被 PhoneWindow 的 DecorView 管理的。

当在主窗口中单击一个控件时，将可以看到很多关于这个控件的详细信息，此时会在该

控件的上方弹出一个窗口，其中会显示该控件的实际效果图，通过 View 的数目显示该控件及其子控件的数目、该控件的节点测量、布局以及绘制视图的时间。控件框如果是绿色的，表示该控件在该阶段比其他 50% 的控件的速度要快；如果为黄色，则表示比其他 50% 的控件的速度要慢；如果为红色，则表示该控件在该阶段的处理速度是最慢的，如图 2-12 所示。

选中一个 View 的图示，单击工具栏的 Display View 按钮 ，就可以看到这个 View 的实际显示效果，如图 2-13 所示。

可以勾选 Show Extras，这个功能也比较实用，可以显示出该 View 中不同元素显示的边界，帮助检查是否正确。

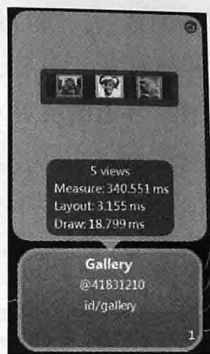


图 2-12 控件的详细信息

### 3. 界面精确查看模式


单击主界面 Inspect Screenshot 按钮，或单击界面左下角类似九宫格的按钮 ，就进入了 Android 称之为 Pixel Perfect View 的界面，即界面精确查看模式，可从细节上观察 UI 的设计效果，如图 2-14 所示。




图 2-13 Display View 示例

图 2-14 的左侧是浏览视图，右侧是全局的视图，中间是当前关注区域的细节放大，是像素级别的，对于观察细节非常有用。窗口下方控件的含义是：



图 2-14 Pixel Perfect View 界面示例

- Overlay 主要用来测试在当前视图上加载新的图片后的效果，单击 Load Overlay (  ) 选择图片后，可以通过滑动条控制在当前界面上显示的透明度 (0% ~ 100% 之间)。如果选择了工具栏上的 Show in Loupe 复选框，右侧的放大视图也会将加载的图片的细节结合透明度显示出来。
- Refresh Rate 用来控制 View 从模拟器或者真机上更新一次视图数据的时间。
- Zoom 是放大局部细节用的，细节显示在最右边的视图上。

## 2.4 界面布局技巧

本节主要介绍布局设计的基本原则和技巧。

### 2.4.1 布局设计原则

#### 1. 布局设计整体原则

- 为高分辨率的屏幕创建资源；
- 需要点击的元素要够大；
- 图标设计遵循 Android 的准则；
- 使用适当的间距，图 2-15 演示了良好的水平间距和垂直间距设计 (①为 24dp，②为 56dp，③为 8dp，④为 72dp)；
- 支持 D-pad 和 trackball 导航；
- 正确管理活动堆栈；
- 正确处理屏幕方向变化；
- 使用主题样式、尺寸和颜色资源来减少多余的值；
- 和视觉交互设计师合作。

#### 2. 布局设计需要关注的因素

- 屏幕的物理尺寸；
- 屏幕密度；
- 屏幕的方向 (竖向和横向)；
- 主要的 UI 交互方式；
- 软键盘还是物理键盘；
- 了解不同设备之间的相异之处；
- 了解设备可能差异的地方；
- 了解屏幕尺寸和密度分类。

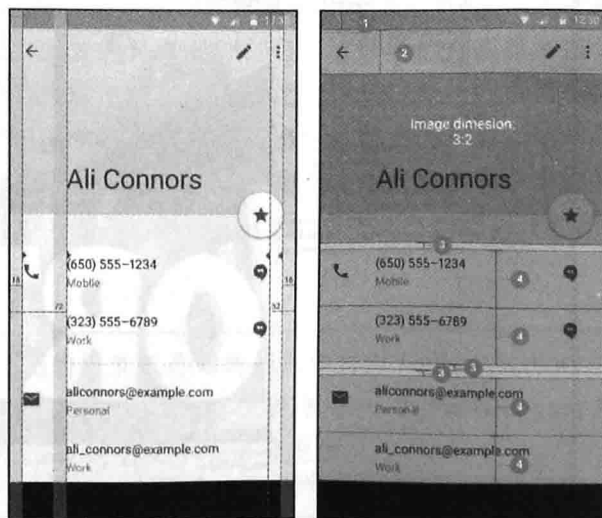


图 2-15 水平间距与垂直间距示例

## 2.4.2 布局设计技巧

### 1. 合理利用 UI 空间

手机屏幕空间非常有限，每一块小小的区域都可以完成大作用，通过共享空间的利用可以完成多个界面的跳转及不同状态之间的切换。标签、滚动还有各种翻转切换都是通过空间共享的方式来承载不同的信息。图 2-16 所示为豆瓣应用中通过中部的 SlideView 来滚动显示音乐信息。

移动设计的挤压案例也很多。Android 的 ExpandableListView 展示方式就是一种自上而下的挤压处理。图 2-17 所示的 91 助手应用的升级界面演示了挤压式界面布局的优势。



图 2-16 豆瓣应用界面示例

### 2. 注意拇指死角

触屏手机交互研究与 PC 交互研究存在着本质的变化。输入设备从鼠标键盘变成手指，一方面操作的精准度上受到了挑战，在密集的信息处理上，用户常常会出现许多误操作；另一方面，手指操作所特有的“死角和热区”问题也是 PC 界面设计中不会面临的。

实验表明，在拇指自然弯曲且处在指肚触及的状态下，手持机的稳定性最佳，手指移动的灵活性也最佳。模拟这种状态下手指触及的区域，大致呈现出一个“扇环”区域（探索热区），而其他区域为“非探索热区”，如图 2-18 所示。

“探索热区”与“非探索热区”存在正确率上的显著差异，并且这种差异在不同刺激参数值下都成立。例如，将屏幕划分为 16 个区域，通过比较 16 个区域的拇指操作正确率均值，发现图 2-19 的 A4、C4、D1、D4 四个区域触击正确率显著低于整体均值。死角区误操作比率高，在控件布局时应考虑死角问题，特别是在处理高频操作位置时。

### 3. 横屏模式设计

由于横屏模式下纵向空间变得格外宝贵，导航栏、标签栏、键盘都需要被压扁。横屏模式一定要考虑的是简单拉伸适配还是重新设计，如果应用不适合在横屏模式下使用，就屏蔽横屏，如果应用需要支持横屏模式（甚至是带侧滑键盘的横屏机器），就需要提供设计方案。

可以发现，用户在不同的使用情景、不同的应用类型下，对横屏的预期还是有所不同的。



图 2-17 91 助手应用升级界面示例



### ● 游戏类

如果是横屏模式下用户的游戏体验最好，不妨在游戏启动时，就直接切换到横屏。在强制横屏时，不需要提醒用户，只要用横向的启动画面引导，当用户看到闪屏是横向时，自然会知道要把屏幕翻转了。

### ● 视频类

用户在播放视频之后，可以以一个合适的引导动画效果切换到横屏模式（如果用户已经锁定为不要旋转屏幕了，则尽量不要强制横屏）。在横屏模式下，如果是为了帮助用户关注到内容本身的应用，则可以把导航栏和工具栏设置为透明的，或者让导航栏和工具栏可以自动隐藏。当然，如果用户需要时，单击一下空白处，也可以唤起操作栏。如图 2-20 所示的视频播放横屏模式。

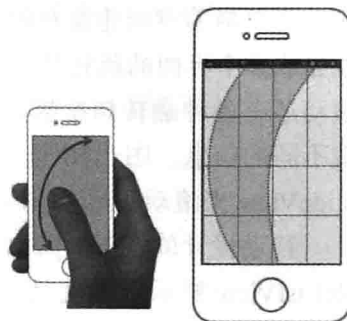


图 2-18 右手拇指操作热区

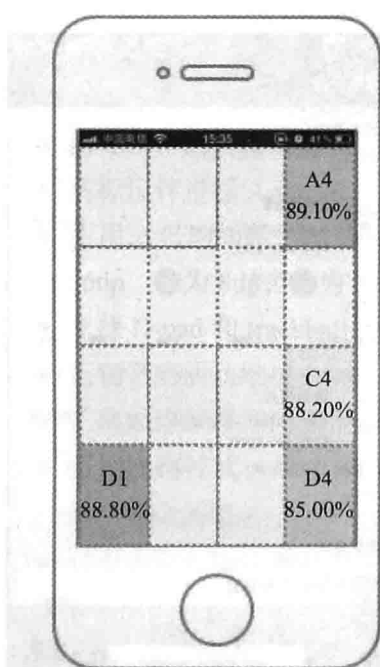
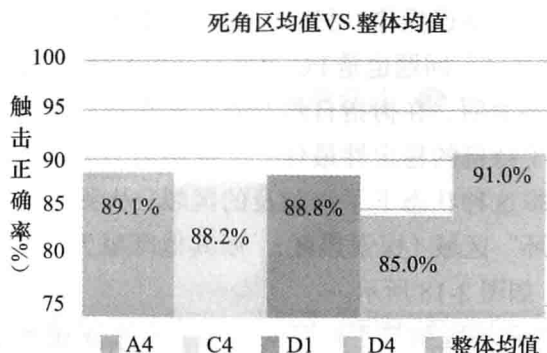


图 2-19 右手拇指死角区域



另外，图片浏览类、阅读类、工具栏等应用的横屏模式都有自身的特点和需求。总的来说，横屏模式下（特别是平板应用）应当遵循：

- 要在所有方向下都能运行，尽力满足用户需求；
- 考虑改变展示辅助信息和功能的方式；
- 避免随意改变布局；

- 如果可能的话，应该尽量避免重组信息，重排文字；
- 为每一种方向提供独特的启动图片。



图 2-20 视频播放应用的横屏模式

#### 4. 触摸设计

设计触屏手机产品时，要注意所有的可点击元素都有足够的点击区域，但是这并不是说要把所有的按钮图标链接都设计得足够大，手机上的视觉焦点和操作焦点是不一样的，操作焦点是可被放大或移动的点击区域。

##### (1) 扩大操作焦点

Android 4.0 规定的有效可触摸的 UI 元素标准是 48dp。一般来说，48dp 转化为一个物理尺寸约为 9 毫米。建议的目标大小为 7 ~ 10 毫米，这一点与 iPhone 一致，这是一个用户手指能准确并且舒适触摸的区域。图 2-21 演示了这种设计规范。

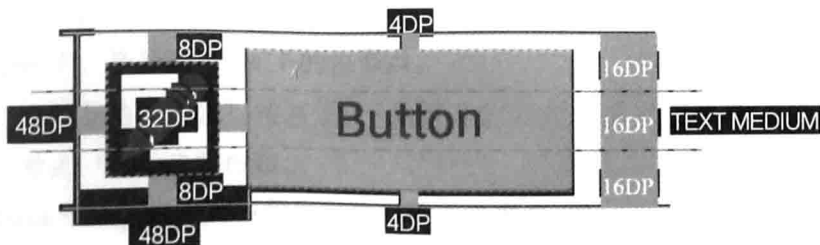


图 2-21 按钮的 UI 设计

图 2-21 中，虽然图标仅有 32dp，按钮仅有 40dp，但是它们的实际可操作焦点都达到了 48dp 的大小，提高了操作的准确率。

### (2) 下移操作焦点

由于用户在进行各种操作时，需要确保自己准确地点击到相应的元素，所以一般情况下，右手持机的用户，落点会偏右且偏下。如果应用是操作密集型，则可以考虑调整有效点击区域，使其整体向右向下偏移几像素，这样准确率会有所提升。

## 2.5 习题

设计一个如下图所示的闪屏界面的布局文件。



## Activity 与 UI 管理

### 3.1 Activity 基础


Activity 是一种应用程序组件，是应用程序的表示层，用于构建应用程序的 UI 界面，使得用户可以通过交互来完成指定的任务。每一个 Activity 被给予一个窗口，窗口通常充满屏幕（但也可以应用小于屏幕的主题而浮于其他窗口之上）。

#### 3.1.1 创建 Activity

一个 Android 应用程序可以由一个或者多个 Activity 组成，这些 Activity 之间通过相互调用来切换屏幕，并且还可以调用其他应用程序中的 Activity。

创建 Activity 的一般步骤如下：

①在 Eclipse 中，选择项目 src 下的包节点。

②单击工具栏上的“新建”按钮，在打开的新建对话框中选择 Android → Android Activity 命令，单击“下一步”按钮。

③选择 BlankActivity（也可以根据具体的需求选择 FullScreen Activity、Login Activity 等类别），单击“下一步”按钮，打开如图 3-1 所示的 New Activity 对话框，并输入 Activity 的相关信息。

④单击 Finish 按钮，完成 Activity 的创建。

自动生成的 Activity 的核心代码如下：

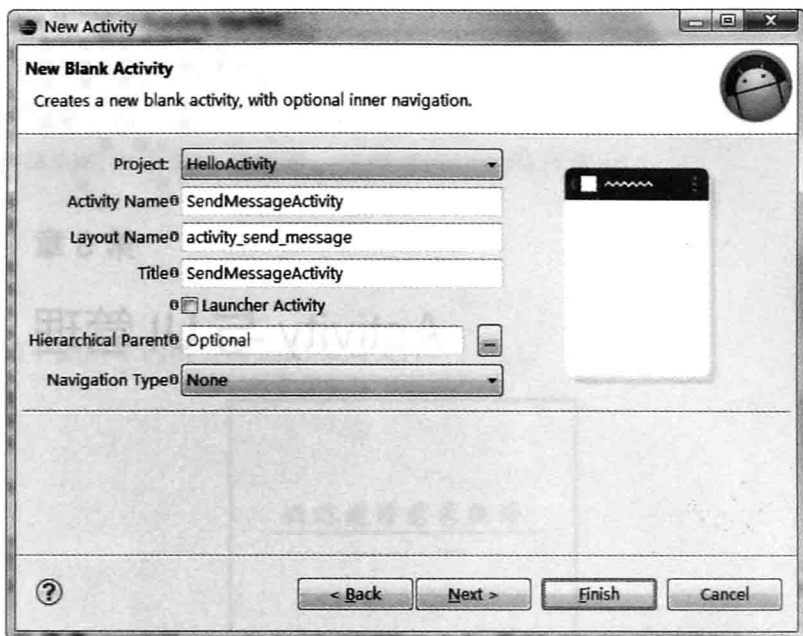


图 3-1 New Activity 对话框

```

01 public class SendMessageActivity extends Activity {
02
03     @Override
04     protected void onCreate(Bundle savedInstanceState) {
05         super.onCreate(savedInstanceState);
06         setContentView(R.layout.activity_send_message);
07     }
08
09     @Override
10     public boolean onCreateOptionsMenu(Menu menu) {
11         // Inflate the menu; this adds items to the action bar if it is present.
12         getMenuInflater().inflate(R.menu.activity_send_message, menu);
13         return true;
14     }
15
16 }

```

通过这种方式创建的 Activity，不但创建好了为该 Activity 服务的 XML 布局文件，还自动地在 AndroidManifest.xml 中进行了如下注册：

```

01 <activity
02     android:name=".SendMessageActivity"
03     android:label="@string/title_activity_send_message" >
04 </activity>

```



**注意** Android 项目中的每个 Activity 实例都必须在 AndroidManifest.xml 中注册。



说明 如果声明的 Activity 前有“.”, 如“.SendMessageActivity”, 表明该 Activity 在项目包的根目录。

Activity 的主体包括两个主要部分: 一个是 Content (data); 另外一个是对应用户交互事件的行为。在 Activity 中通过 onCreate(Bundle) 方法来初始化 Activity, 通过 setContentView(int) 方法来设置 UI 布局所使用的 layout 资源, 通过 findViewById(int) 方法来获得对应的视图。

### 3.1.2 Activity 的生命周期

实施生命周期回调方法来管理 Activity 对开发强大和灵活的应用是很重要的。一个与其关联的其他 Activity 直接影响其 Activity 的生命周期、任务和返回堆栈。

#### 1. 用户界面与生命周期

当用户运行应用程序时, 应用程序的 Activity 实例会在不同的生命周期状态中变化。一般情况下, 总有一个 Activity 被标记为用户在应用程序启动时第一个看到的用户界面。在 Activity 显示在屏幕上获取到用户焦点的这个过程中, Android 系统调用 Activity 一系列的生命周期方法来建立用户界面和其他组件。

如果用户的操作启动了新的界面, 从而调用另外一个 Activity 或者启动了另外一个应用程序, 原来的 Activity 转到后台 (此时 Activity 不可见, 但其实例状态仍然保存), 系统将会调用另外一些 Activity 的生命周期方法。

当一个 Activity 的状态发生变化时, 可能会收到多个回调方法, onCreate()、onStop()、onResume()、onDestroy() 等这些回调方法在不同的状态实现不同的功能。例如, 当停止一个 Activity 时, 一般需要释放大对象, 如网络或数据库连接。恢复该 Activity 时, 需要重新获得必要的资源和恢复被中断的 Activity。这些状态转换是所有的 Activity 程序周期的一部分。

Activity 的生命周期如图 3-2 所示。

表 3-1 是 Activity 生命周期中各个方法的详细说明。

#### 2. Activity 的状态迁移

一个 Activity 在运行期间可以存在 3 种状态

##### (1) Resumed 状态

处于 Resumed 状态 (有时也简称为运行状态) 的 Activity 处于栈顶, 且是可见的、有焦点的、能够与用户互动的。当另一个 Activity 进入栈顶时, 当前的 Activity 将进入 Paused 状态。

##### (2) Paused 状态

另一个 Activity 在屏幕前, 取得焦点, 但原来的 Activity 仍然可见。暂停的 Activity 完全是存在的 (Activity 对象保留在内存中, 它维护所有状态和成员信息, 并保持窗口管理器的联系), 如果系统内存不足, 可以被系统终止。

##### (3) Stopped 状态

Activity 完全被另一个 Activity 遮住了（现在 Activity 是在后台）。此时，Activity 也仍然存在。然而，它已不再是对用户可见。如果系统内存不足，可以被系统终止。

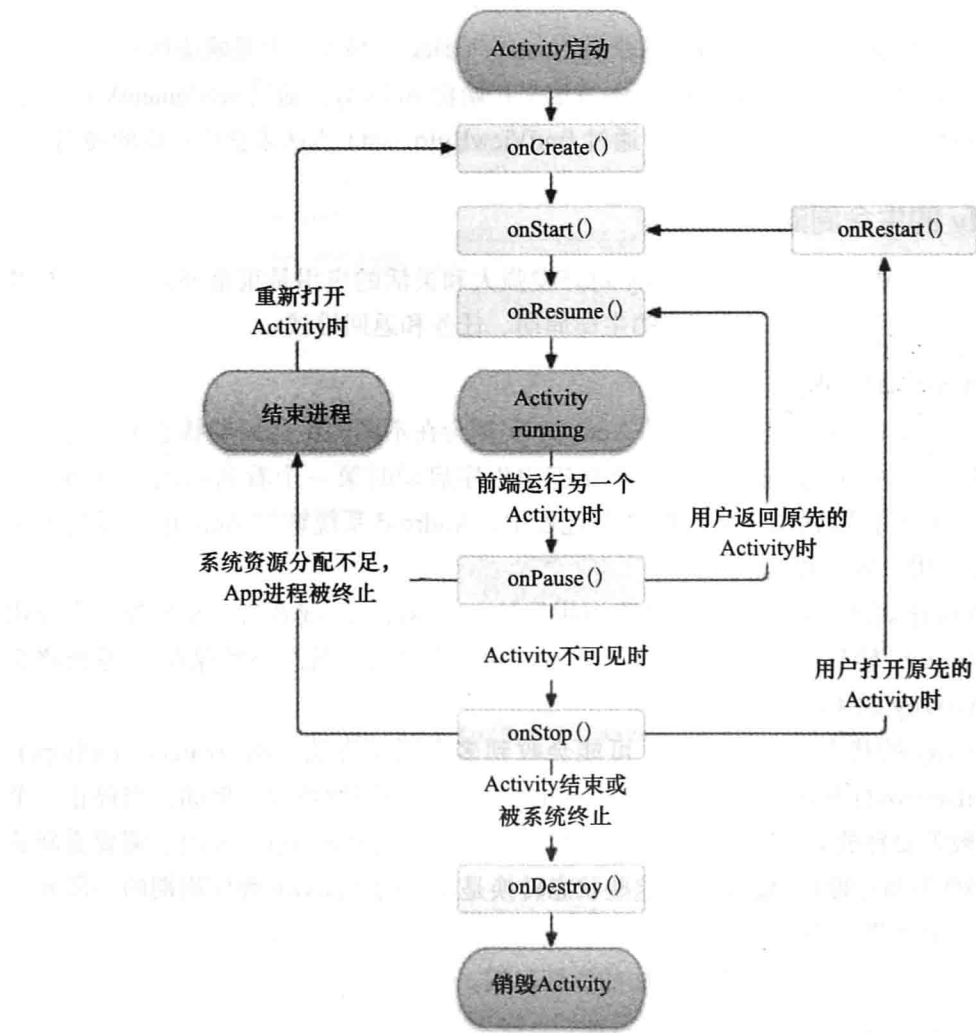


图 3-2 Activity 生命周期

表 3-1 Activity 的生命周期回调方法

方 法	说 明	调用后是否 可终止？	继续调用
onCreate()	当 Activity 第一次创建时被调用。方法中完成所有的正常静态设置，如创建一个视图、绑定列表的数据等。如果能捕获到 Activity 状态的话，这个方法传递进来的 Bundle 对象将存放了 Activity 当前的状态	否	onStart()
onRestart()	在 Activity 被停止后重新启动时会调用该方法	否	onStart()
onStart()	当 Activity 对于用户可见前调用该方法。如果 Activity 回到前台则接着调用 onResume()，如果 Activity 隐藏则调用 onStop()	否	onResume() 或 onStop()



(续)

方 法	说 明	调用后是否 可终止？	继续调用
onResume()	在 Activity 开始与用户交互前调用该方法。这时该 Activity 处于 Activity 栈的顶部，并且接受用户的输入	否	onPause()
onPause()	在系统准备恢复其他 Activity 时调用该方法。方法中通常用来提交一些还没保存的更改到持久数据中，停止一些动画或其他一些耗 CPU 的操作等。无论在该方法里面进行任何操作，都需要较快的速度完成，因为如果它不返回的话，下一个 Activity 将无法恢复。如果 Activity 返回到前台将会调用 onResume()，如果 Activity 变得对用户不可见了将会调用 onStop()	是	onResume() 或 onStop()
onDestroy()	在 Activity 对用户不可见时将调用该方法。可能会因为当前 Activity 正在被销毁，或另一个 Activity 已经恢复了正准备重载它而调用该方法。如果 Activity 正准备返回与用户交互时会调用 onRestart()，如果 Activity 正在被释放则会调用 onDestroy()	是	onRestart() 或 onDestroy()
onStop()	在 Activity 被销毁前会调用该方法。这是 Activity 能接收到的最后一个调用。可能会因为调用了 finish() 方法使得当前 Activity 正在关闭，或系统为了保护内存临时释放这个 Activity 的实例而调用该方法。可以用 isFinishing() 方法来区分这两种不同的情况	是	—

### 3.1.3 退出 Activity

一般调用 finish() 方法即可退出当前 Activity，也可以通过按返回键，让程序调用 onDestroy() 方法来关闭当前的显示界面。

但是，通过查看其进程可以发现调用了 onDestroy() 方法之后该 Activity 还在运行，甚至调用了 finish() 方法之后程序还能在进程中看到。如果需要在调用另一个 Activity 时，结束本 Activity（即使点击返回按钮也不显示），可以使用下面的方法：

```

01 Intent intent = new Intent();
02 Intent.setClass(A.this, B.class);
03 intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
04 intent.putExtra("flag", EXIT_APPLICATION);
05 context.startActivity(intent);

```

其中的 A 和 B 是 Activity 的实例。这是一种通过为 Intent 设置 flag 的方法，有关 Intent 的 Flag 的使用参见 3.4.3 节。

## 3.2 Activity 之间的调用

在 Android 中通过调用另外一个 Activity 来实现 UI 界面之间的切换。

### 3.2.1 调用其他 Activity

在 Android 中调用另一个 Activity 有两种方式。

#### 1. 显式调用

Activity 之间可以通过在 Intent 中设置明确的被调 Activity 来显式地相互调用，如 ActivityProject/SendMessageActivity 示例中的 sendMessage() 方法：

```
01 public void sendMessage(View view) {
02     Intent intent = new Intent(this, DisplayMessageActivity.class);
03     EditText editText = (EditText) findViewById(R.id.edit_message);
04     String message = editText.getText().toString();
05     intent.putExtra(EXTRA_MESSAGE, message);
06     startActivity(intent);
07 }
```

使用 Context.startActivity() 来启动的 Activity 必须在启动者应用包的 AndroidManifest.xml 文件中有对应的 <activity> 声明。

#### 2. 隐式调用

Activity 之间也可以通过在 Intent 中设置 Action 等信息来隐式地调用其他应用中的 Activity，如 ActivityProject/OpenGoogleActivity 示例的 openGoogle() 方法用来启动系统的浏览器 Activity：

```
01 private void openGoogle() {
02     Uri uri = Uri.parse("http://www.google.com");
03     Intent it = new Intent(Intent.ACTION_VIEW, uri);
04     startActivity(it);
05 }
```

由于隐式的 Intent 没有明确的目标组件，因此，当隐式的 Intent 被抛出后，系统在众多组件中根据 Intent 过滤器中的 action、datatype、uri 等信息来寻找与其匹配的处理方法。有关隐式调用中 Intent 的匹配规则见 4.4.3 节。

### 3.2.2 Activity 的回调

当一个 Activity 启动另一个 Activity 时，有时需要从被调的 Activity 回传数据，并根据回传参数做出相应的处理，这时需要用到 startActivityForResult() 方法。该方法的原型是：

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

ActivityProject/StartForResultActivity 示例在 goTarget 按钮的单击事件回调代码如下：

```
01 if (v == goTarget) {
02     Intent intent = new Intent();
03     intent.setClass(StartForResultActivity.this, TargetActivity.class);
04     Bundle mBundle = new Bundle();
```


```

05     mBundle.putString("msg", msgText.getText().toString().trim());
06     intent.putExtras(mBundle);
07     StartForResultActivity.this.startActivityForResult(intent,
08         nRequestCode_01);
09 }

```

其中第 08 行的 “nRequestCode\_01” 标示着本次 Activity 调用的请求码。

---

 **注意** 不能够在 Activity 调用了 `startActivityForResult()` 方法之后调用 `finish()` 方法，否则将无法接收到返回结果。

---

当 Activity 使用了 `startActivityForResult()` 方法后，需要重载 `onActivityResult()` 方法。该方法的原型为：

```
onActivityResult(int requestCode, int resultCode, Intent data)
```

其中的 `requestCode` 用于判断是哪个 `startActivityForResult()` 方法产生的回调，`resultCode` 是被调 Activity 返回的结果码，`data` 是回调返回的数据封装。例如：

```

01 @Override
02 protected void onActivityResult(int requestCode, int resultCode,
03     Intent data) {
04
05     if (nRequestCode_01 == requestCode) {
06         switch (resultCode) {
07             case Activity.RESULT_OK:
08                 // 获取 Target Activity 返回来的信息
09                 Bundle bundle = data.getExtras();
10                 String strReturn = bundle.getString("msg");
11                 Toast.makeText(getApplicationContext(), strReturn,
12                     Toast.LENGTH_LONG).show();
13                 break;
14             case Activity.RESULT_CANCELED:
15                 break;
16             default:
17                 break;
18         }
19     }
20
21     super.onActivityResult(requestCode, resultCode, data);
22 }

```

被调 Activity 一般通过 `setResult()` 方法返回主调 Activity，该方法的原型是：

```
setResult(int resultCode, Intent data)
```

或

```
setResult(int resultCode)
```

例如：

```
01 Intent intent = new Intent();
02 Bundle bundle = new Bundle();
03 bundle.putString("msg", "信息已收到");
04 intent.putExtras(bundle);
05
06 TargetActivity.this.setResult(Activity.RESULT_OK, intent);
07 TargetActivity.this.finish();
```

### 3.3 Activity 之间的数据传递

在 Android 中，Activity 之间传递数据通常有 3 种方法。

#### 3.3.1 使用 Intent 传递数据

通常使用 Intent 的 `putExtra(key, value)` 方法传递数据，如 3.2.1 节显式调用示例的 05 行。获取数据时，使用 Intent 的 `getExtra(key)` 方法。

Intent 提供了各种常用类型重载后的 `putExtra()` 方法，例如：`putExtra(String name, String value)`、`putExtra(String name, long value)` 等。在 `putExtra()` 方法内部会判断当前 Intent 对象内部是否已经存在一个 Bundle 对象，如果不存在就会新建 Bundle 对象，然后调用 `putExtra()` 方法将传入的值存放于该 Bundle 对象。下面是 Intent 的 `putExtra(String name, String value)` 方法的源码片段：

```
01 public class Intent implements Parcelable {
02     private Bundle mExtras;
03     public Intent putExtra(String name, String value) {
04         if (mExtras == null) {
05             mExtras = new Bundle();
06         }
07         mExtras.putString(name, value);
08         return this;
09     }
10 }
```

Intent 也会使用特定的 Key 来传递特殊的数据，下面的代码实现了传送邮件附件的功能：

```
01 Intent it = new Intent(Intent.ACTION_SEND);
02 it.putExtra(Intent.EXTRA_SUBJECT, "The email subject text");
03 it.putExtra(Intent.EXTRA_STREAM, "file:///sdcard/eoe.mp3");
04 sendIntent.setType("audio/mp3");
05 startActivity(Intent.createChooser(it, "Choose Email Client"));
```

#### 3.3.2 使用 Bundle 传递数据

Activity 之间的数据传递一般借助 Bundle 类来完成。Bundle 类是一个封装了

Map<String, Object> 的键值对, 可以从 String 类型的键中获得任意类型的对象。Bundle 类的构造方法如下:

```
01 public Bundle() {
02     mMap = new HashMap<String, Object>();
03     mClassLoader = getClass().getClassLoader();
04 }
```

Bundle 封装了一些常用的方法, 例如:

- clear() 清除此 Bundle 映射中所有保存的数据。
- clone() 克隆当前 Bundle。
- containsKey (String key) 返回指定 key 的值。
- hasFileDescriptors() 指示是否包含任何捆绑打包文件描述符。
- isEmpty() 如果这个捆绑映射为空, 则返回 true。
- getString (String key) 返回指定 key 的字符。
- putString (String key, String value) 插入一个给定 key 的字符串值。

Bundle 除了支持全部的基本数据类型 (如 byte、char、boolean、short、int、long、float、double 等) 外, 还支持数组、List 等。如 putStringArray (String key, String[] value)、putIntegerArrayList (String key, ArrayList<Integer> value) 等方法。

- readFromParcel (Parcel parcel) 读取这个 parcel 的内容。在 Android 中, Parcel 是一个存储基本数据类型和引用数据类型的容器, 通过 IBinder 来绑定数据在进程间传递数据。
- remove (String key) 移除指定 key 的值。
- writeToParcel (Parcel parcel, int flags) 写入这个 parcel 的内容。

使用 Bundle 传递数据一般遵循如下步骤 (参见示例 ActivityProject/TransmitDataActivity<sup>①</sup>):

在传送数据的 Activity 中:

- ①新建一个 Bundle 类;
- ②通过 Bundle.putString (key, value) 等方法将数据存入 Bundle;
- ③通过 Intent.putExtra (Bundle) 方法将数据附加到 Intent 对象上。

例如:

```
01 private void startTransmitData() {
02     Intent mIntent = new Intent();
03     mIntent.setClass(getApplicationContext(), ShowImageAcitivity.class);
04     Bundle mBundle = new Bundle();
05     mBundle.putInt("imageId", R.drawable.logo);
06     mBundle.putString("activityName", getClass().toString());
07     mIntent.putExtra(mBundle);
08 }
```

① 参见华章网站 (<http://www.hzbook.com>) 中本书的教辅资源。

```
08     startActivity(mIntent);
09 }
```

在接收数据的 Activity 中:

- ①新建一个 Bundle 类;
- ②通过 Intent.getExtras() 方法从 Intent 对象获取封装了的数据包;
- ③通过 Bundle.getString (key) 等方法从键中获取数据。

例如:

```
01 Intent mIntent = this.getIntent();
02 Bundle mBundle = mIntent.getExtras();
03 int drawableId = mBundle.getInt("imageId");
04 String msg = mBundle.getString("activityName");
```

虽然 Activity 之间传递数据可以直接使用 Intent.putExtra (key, value) 方法, 但是, 如果一个封装了的数据要依次从 Activity A 传给 Activity B, Activity B 再传给 Activity C, 则建议使用 Bundle 来进行传递。这样就可以通过 putExtra (String name, Bundle value) 和 getBundleExtra (String name) 方法来直接传递, 而不需要对数据重新进行封装。

### 3.3.3 使用 Application 共享数据

除了上面介绍的使用 Intent 和 Bundle 传递数据的方法外, 还有一种在多个 Activity 之间共享数据的方法, 就是使用 Application Context。

一般步骤如下:

- ①新建一个类, 继承自 Application, 例如:

```
01 public class MyApp extends Application {
02     private String myState;
03
04     public String getState() {
05         return myState;
06     }
07
08     public void setState(String s) {
09         myState = s;
10     }
11 }
```

- ②在 AndroidManifest.xml 的 application 标签中添加 name 属性, 例如:

```
01 <application android:name="cn.liweiyong.activityproject.bean.MyApp"
02     ...>
```

- ③在使用时使用类似如下 06 行的形式即可:

```
01 class ApplicationDataActivity extends Activity {
02     @Override
```

```

03     public void onCreate(Bundle b){
04         ...
05         TextView mTextView = (TextView) findViewById(R.id.textview);
06         MyApp appState = ((MyApp) getApplicationContext());
07         mTextView.setText(appState.getState());
08         ...
09     }
10 }

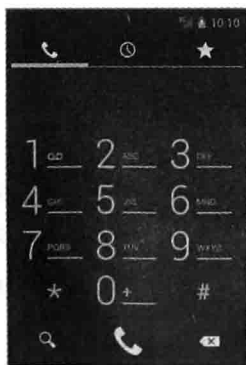
```

## 3.4 Activity 栈与任务

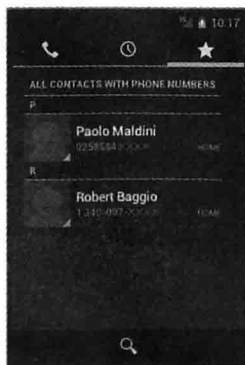
Activity 中的任务是与用户交互的一组 Activity 的集合, Activity 会被按启动顺序安排在一个堆栈里。

### 3.4.1 Activity 栈

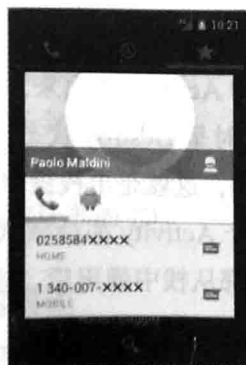
首先通过一个给朋友打电话的应用场景了解 Activity 栈和任务之间的关系。这个过程一般用到如下四个 Activity: 拨号、联系人列表、查看联系人、呼叫过程, 如图 3-3 所示。



a) 拨号界面



b) 联系人列表



c) 联系人详情



d) 呼叫联系人

图 3-3 拨号过程



在这个打电话的过程中，用户之所以能够从一个界面跳转到另一个界面，是因为 Android 系统针对 Activity 设计了一个线性的导航历史以供用户追溯访问，这就是 Activity 栈 (Stack Back)。当用户启动了一个新的 Activity，它就被添加进 Activity 栈，以便按返回键时能够返回到上一个 Activity。

首先来看一个 Android 官方文档给出的示例图，该图展示了多个 Activity 之间的切换，并以时间线为线索展示这种行为。当用户进行某项操作时，任务栈就收集相互交互的 Activity，Activity 会被安排在堆栈中，堆栈中的 Activity 会按顺序重新打开，如图 3-4 所示。

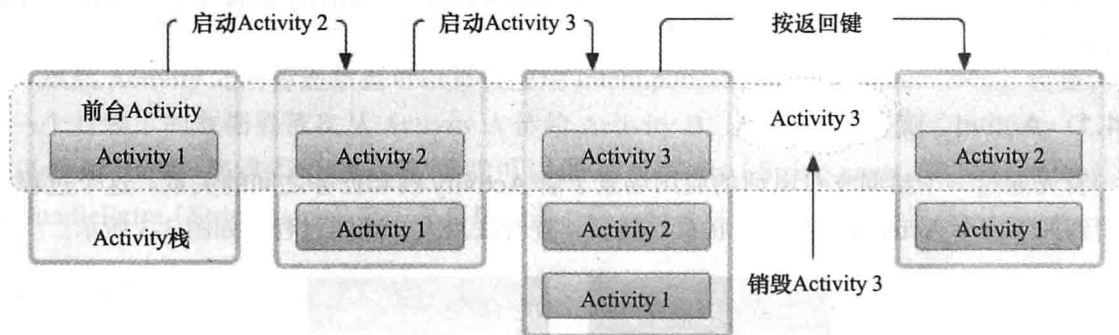


图 3-4 Activity 栈的后进先出示意图

一个设备的主屏幕是大多数任务栈的起点。当用户触屏图标（或者在屏幕上的快捷方式）时，该应用程序就会到达任务栈的最前面。如果该应用在任务栈中不存在，就会在任务栈中创建一个新的任务，并将该应用作为“主”Activity 放置在根任务栈中。

堆栈的特性是后进先出，两个主要操作是压栈和出栈。每当用户切换到一个新的 Activity 时，Android 会将其压入到 Stack Back，使其成为栈顶，也就是用户当前看到的 Activity；栈顶后面的 Activity 此时呈 Stop 状态，对应着 Activity 生命周期方法中的 onStop()，并且 Android 还会保留其状态，这就是压栈操作。

当用户按下返回键，Android 就会将栈顶部的 Activity 弹出来，紧随其后的 Activity 顶上去，就实现了回退的功能。被弹出的 Activity 此时呈 Destroy 状态，对应着 onDestroy()，新栈顶 Activity 呈 Resume 状态，对应着 onResume()，这就是出栈操作。

如果用户不停地按下返回键，那么栈中每个 Activity 都会依次弹出，并且显示前一个 Activity，直至用户回到主屏幕。当所有 Activity 都从栈中弹出后，栈就不再存在。



**注意** Activity 栈里面存放的只能是 Activity，Activity 中的视图、窗体、菜单和对话框信息则不能存放在栈中。

### 3.4.2 任务管理

用户为了完成某个功能而执行的一系列操作形成了一个 Activity 序列，例如上一节的打电话事件。这个序列在 Android 中称之为任务，它是从用户体验的角度出发，把一组相关的 Activity（这些 Activity 可以来自不同的应用程序）组织在一起而抽象出来的概念。因此，任务包含一个按照用户交互顺序排序的 Activity 集合，任务可以把每个 Activity 按照用户执行顺序放到后台，并且保持状态且不丢失工作。

当一个 Activity 启动时，任务也随之启动的话，那么这个 Activity 就是根 Activity。Android 系统内部一旦有任务，那么按返回键就可以回到上一个 Activity。Activity 栈可以是多个任务的组成部分。

任务中的所有 Activity 是作为一个整体进行移动的，这就是任务的聚合性。整个任务（即整个 Activity 堆栈）可以移到前台或退至后台。例如，当前任务在堆栈中存有四个 Activity——三个在当前 Activity 之下。当用户按下 Home 键的时候，回到了应用程序启动器，然后选择了一个新的应用程序（实际上就是一个新任务）。则当前任务遁入后台，而新任务的根 Activity 显示出来。然后，当用户再次回到了应用程序启动器而又选择了前一个应用程序（实际上就是前一个任务）时，那个任务便带着它堆栈中的四个 Activity 再一次回到前台。当用户按下返回键时，屏幕不会显示出用户刚才离开的 Activity（上一个任务的根 Activity）。相反，当前任务的堆栈中最上面的 Activity 被弹出，而同一任务中的前一个 Activity 显示了出来。如图 3-5 所示演示了任务的聚合性特点。

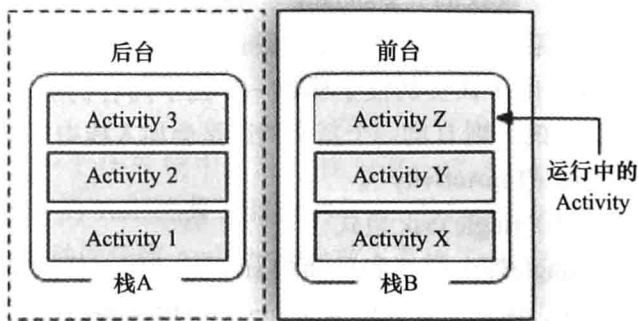


图 3-5 任务的聚合性示意图

因为 Activity 栈中的 Activity 顺序永远不会改变，所以，如果应用允许某个 Activity 可以让用户启动多次，则新的实例会压入栈顶而不是打开之前位于栈顶的 Activity。于是，一个 Activity 可能会初始化多次（甚至会位于不同的任务中），如图 3-6 所示。如果用户用返回键返回，则 Activity 的每个实例都会按照原来打开的顺序显示出来。可以通过 `getTaskId()` 方法获取当前任务的 ID。

### 3.4.3 Activity 的加载模式

加载模式定义了一个新的 Activity 实例与当前任务的关联方式。定义加载模式的方法有如下两种。



图 3-6 Activity 被多次实例化示意图

## 1. AndroidManifest 中的声明

在 AndroidManifest 文件中声明一个 Activity 时，通过对 Activity 元素的 android:launchMode 属性进行设置来指定它启动时与任务的关联方式。

Activity 有四种加载模式：standard、singleTop、singleTask 和 singleInstance。例如：

```
01 <activity android:name=".MainActivity"  
02           android:launchMode="singleTask" />
```

### (1) standard 模式

standard 模式是默认的模式。每次访问都实例化新的 Activity，即系统在启动 Activity 的任务中创建一个新的 Activity 实例，并且把 Intent 传送路径指向它。该 Activity 可以被实例化多次，各个实例可以属于不同的任务，一个任务中也可以存在多个实例。参见示例 StandardActivity<sup>①</sup>。

### (2) singleTop 模式

每次访问时，首先看栈顶元素的目标对象，如果是当前 Activity 实例，则返回，不再实例化该 Activity，否则，重新实例化新的 Activity。即如果 Activity 的一个实例已经存在于当前任务的栈顶，该系统就会使用 onNewIntent() 方法通过 Intent 传递给已有实例，而不是创建一个新的 Activity 实例。Activity 可以被实例化多次，各个实例可以属于不同的任务，一个任务中可以存在多个实例。例如，假设任务的 Activity 栈中包含了根 Activity A 和 Activities B、C、D（顺序是 A-B-C-D），D 在栈顶。这时候传过来的是启动 D 的 Intent，如果 D 的启动模式是默认的“standard”，则会启动一个新的实例，栈的内容就会变为 A-B-C-D-D。但是，如果 D 的启动模式是“singleTop”，则已有的 D 的实例会通过 onNewIntent() 接收这个 Intent，由于该实例位于栈顶——栈中内容仍然维持 A-B-C-D 不变。当然，如果 Intent 是要启动 B 的，则 B 的一个新实例还是会加入栈中，即使 B 的启动模式是“singleTop”。参见示例 SingleTopActivity<sup>②</sup>。

### (3) singleTask 模式

singleTask 模式和后面的 singleInstance 模式都是只创建一个实例，保证 Activity 只实例化一次，由此所开启的 Activity 和当前 Activity 位于同一任务中。系统会创建一个新的任务，并把 Activity 实例作为根放入其中。但是，如果 Activity 已经在其他任务中存在实例，则系统会通过调用其实例的 onNewIntent() 方法把 Intent 传给已有实例，而不是再创建一个新实例。参见示例 SingleTaskActivity<sup>③</sup>。

singleTask 模式启动 Activity 的特点：

① 设置了“singleTask”启动模式的 Activity 在启动时，首先在系统中查找属性值 affinity 等于它的任务是否存在，如果存在这样的任务，它就会在这个任务中启动，否则就会在新任务中启动。

① 参见华章公司网站中本书的教辅资源。

② 同上。

③ 同上。

②如果设置了“singleTask”启动模式的 Activity 不是在新的任务中启动时，它会在已有的任务中查看是否已经存在相应的 Activity 实例，如果存在，就会把位于这个 Activity 实例上面的 Activity 全部结束掉，即最终这个 Activity 实例会位于任务的堆栈顶端中。

在启动了一个启动模式设为 singleTask 的 Activity，且有一个后台任务中已存在实例时，这个后台任务就会整个转到前台。这时，当前的 Activity 栈就包含了这个转入前台的 Task 中所有的 Activity，位置是在栈顶。图 3-7 所描述的就是这一种场景。

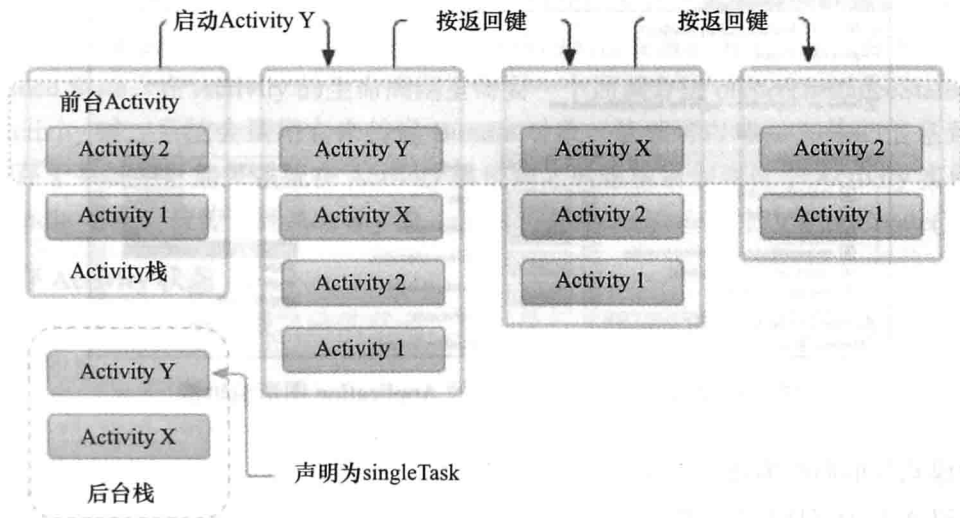


图 3-7 启动模式为“singleTask”的 Activity 加入 Activity 栈的示意

#### (4) singleInstance 模式

singleInstance 模式下的 Activity 单独在一个任务栈中，这个栈只有一个 Activity。除了系统不会把其他 Activity 放入当前实例所在的 Task 之外，其他均与 singleTask 模式相同，Activity 总是它所在 Task 的唯一成员；它所启动的任何 Activity 都会放入其他 Task 中。参见示例 SingleInstanceActivity<sup>①</sup>。

例如，Android 的浏览器应用就把 Web 浏览器 Activity 声明为总是在它自己独立的任务中打开——把 Activity 设为 singleTask 模式。这意味着，如果应用提交 Intent 来打开 Android 的浏览器，则其 Activity 不会被放入应用所在的任务中。取而代之的是，或是为浏览器启动一个新的任务，或是浏览器已经在后台运行，只要把任务重新调入前台来处理新 Intent 即可。也可以在 Eclipse 中以图形界面编辑，如图 3-8 所示。

#### 2. 使用 Intent 标志

在启动 Activity 时，可以通过 Intent.setFlags() 方法在传给 startActivity() 的 Intent 中包含相应标识，用于修改 Activity 与任务的默认关系。这个标识可以修改的默认模式如下所示。

① 参见华章公司网站中本书的教辅资源。

### (1) FLAG\_ACTIVITY\_NEW\_TASK

在默认情况下，一个新 Activity 被另外一个调用了 `startActivity()` 方法的 Activity 载入了任务之中，并压入了调用者所在的堆栈。然而，如果传递给 `startActivity()` 的 Intent 对象包含了 `FLAG_ACTIVITY_NEW_TASK` 标记，系统会为该新 Activity 寻找一个不同的任务来驻留。但是，如果已经存在了一个与新 Activity 有着同样 affinity 属性值的任务，则 Activity 会载入到那个任务之中。如果没有，则启用新任务。

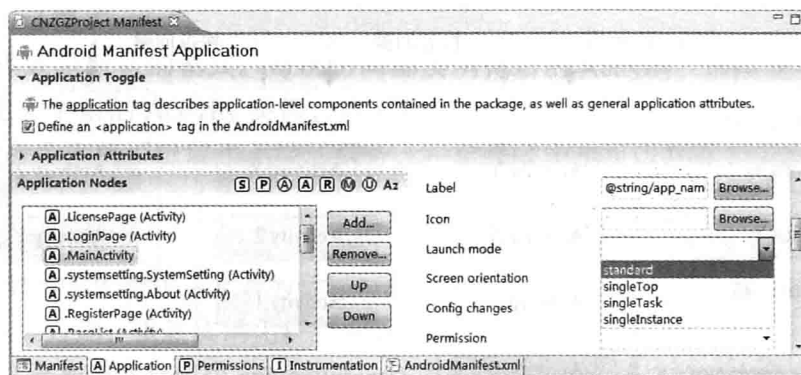


图 3-8 AndroidManifest.xml 的 Application 图形编辑器

这个模式与前面所描述的“singleTask”加载模式相同。

### (2) FLAG\_ACTIVITY\_SINGLE\_TOP

如果要启动的 Activity 就是当前 Activity（位于栈顶），则已存在的实例将接收到一个 `onNewIntent()` 调用，而不是创建一个 Activity 的新实例。

这个模式与前面所描述的“singleTop”加载模式相同。

### (3) FLAG\_ACTIVITY\_CLEAR\_TOP

如果目标任务的堆栈中已经存在一个能够响应此 Intent 的 Activity 类型的实例，则这个实例之上的所有 Activity 都将被清理，以使它位于堆栈的顶部来对那个 Intent 做出响应（通过 `onNewIntent()` 传入 Intent 并恢复 Activity）。如果此时指定的 Activity 的启动模式为“standard”，则它本身也会从堆栈中移除，并启动一个新的实例来处理到来的 Intent。

此种模式在 `launchMode` 中没有对应的属性值。

`FLAG_ACTIVITY_CLEAR_TOP` 经常与 `FLAG_ACTIVITY_NEW_TASK` 结合在一起使用。这些标识定位在其他任务中已存在的 Activity 中，并再将其放入可以响应 Intent 的位置上。

## 3.4.4 保存 Activity 的状态

一些设备配置在运行过程中可能会发生改变（例如屏幕横向布局、键盘可用性和语言），当这样的变化发生时，Android 会重新启动这个正在运行的 Activity（`onDestroy()` 方法会被调用，然后调用 `onCreate()` 方法）。这个重启的动作是为了通过自动往应用程序中载入可替代资

源，从而使应用适应新的配置。

## 1. 重建 Activity

如果因为系统资源紧张而导致 Activity 被销毁，用户在返回这个 Activity 时，系统会使用那些保存的记录数据来重新创建一个新的 Activity 实例。那些被系统用来恢复之前状态而保存的数据叫做“Instance State”，它是一些存放在 Bundle 对象中的键值对。

默认情况下，系统使用 Bundle 实例来保存每一个视图对象中的信息。因此，如果 Activity 被销毁并被重建，那么布局的状态信息会自动恢复到之前的状态。然而，Activity 可能存在更多需要恢复的状态信息，如记录用户进程的成员变量。为了可以保存额外更多的数据到 Instance State，在 Activity 的生命周期里需要一个回调方法 `onSaveInstanceState()`。当用户离开 Activity 时，系统会调用它来传递 Bundle 对象，从而可以增加额外的信息到 Bundle 中，并保存于系统中。如果系统在 Activity 被销毁之后想重新创建这个 Activity 实例，之前的那个 Bundle 对象会被传递到 Activity 的 `onRestoreInstanceState()` 方法与 `onCreate()` 方法中。

## 2. 保存 Activity 状态

为了获得 Activity 被销毁之前的状态，可以执行 Activity 的 `onSaveInstanceState()` 方法。该方法的原型是：

```
protected void onSaveInstanceState (Bundle outState)
```

Android 在 Activity 有可能被销毁之前调用此方法。它会将一个记录 Activity 动态状态的 Bundle 对象传递给该方法。当 Activity 再次启动时，这个 Bundle 会传递给 `onCreate()` 方法以及在 `onStart()` 方法之后调用的 `onRestoreInstanceState()` 方法。图 3-9 演示了 Activity 状态保存的处理机制。

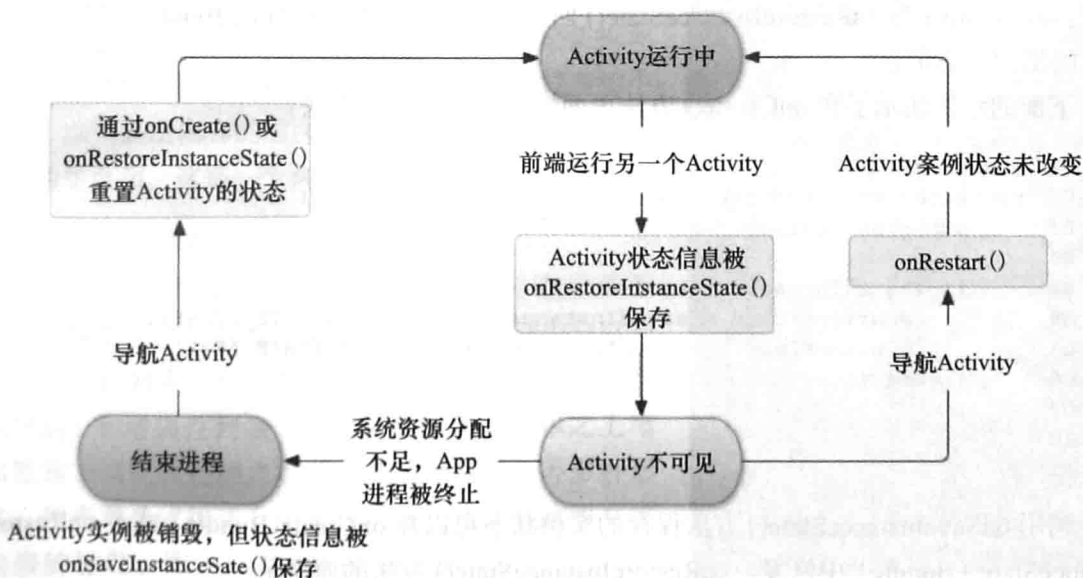


图 3-9 Activity 状态保存的处理机制

当 Activity 显示在当前栈的最上层时，其 `onSaveInstanceState()` 方法通常在如下几种情况下被执行：

- 当用户按下 Home 键时；
- 长按 Home 键，选择运行其他程序时；
- 按下电源按钮（关闭屏幕显示）时；
- 从当前 Activity 中启动一个新的 Activity 时；
- 屏幕方向切换时。

因为 `onSaveInstanceState()` 方法不总是被调用（如按下返回键而导致的销毁则不会调用），应该仅使用 `onSaveInstanceState()` 来记录 Activity 的临时状态，而不是持久的数据（一般使用 `onPause()` 来存储持久数据）。

总之，`onSaveInstanceState()` 的调用遵循一个重要原则，即当系统“未经许可”时销毁了 Activity，则 `onSaveInstanceState()` 会被系统调用，这是系统的责任，因为它必须要提供一个机会来保存数据。

下面的代码演示了 `onSaveInstanceState()` 方法的使用：

```
01 @Override
02 public void onSaveInstanceState(Bundle savedInstanceState) {
03     savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
04     savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);
05     super.onSaveInstanceState(savedInstanceState);
06 }
```

### 3. 恢复 Activity 状态

当 Activity 在被销毁后进行重建时，可以从系统传递给 Activity 的 Bundle 中恢复保存的状态。`onCreate()` 与 `onRestoreInstanceState()` 回调方法都接收到了同样的 Bundle，里面包含了同样的实例状态信息。

下面的示例演示了在 `onCreate()` 方法里面恢复一些数据的方法：

```
01 @Override
02 protected void onCreate(Bundle savedInstanceState) {
03     super.onCreate(savedInstanceState);
04
05     if (savedInstanceState != null) {
06         mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
07         mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
08     } else {
09     }
10     ...
11 }
```

调用 `onSaveInstanceState()` 方法保存的实例状态可以在 `onCreate(Bundle)` 或者 `onRestoreInstanceState(Bundle)` 中恢复。`onRestoreInstanceState()` 方法的原型是：

```
protected void onRestoreInstanceState (Bundle savedInstanceState)
```



下面的代码演示了 `onRestoreInstanceState()` 方法的使用：

```
01 public void onRestoreInstanceState(Bundle savedInstanceState) {
02     super.onRestoreInstanceState(savedInstanceState);
03     mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
04     mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
05 }
```

注意，`onSaveInstanceState()` 方法和 `onRestoreInstanceState()` 方法不一定是成对地被调用，`onRestoreInstanceState()` 方法被调用的前提是 Activity 确实被系统销毁了，而如果仅仅是停留在有这种可能性的情况下，则该方法不会被调用。例如，当正在显示 Activity 时，用户按下 Home 键回到主屏幕，然后用户紧接着又返回到该 Activity，这种情况下 Activity 一般不会因为内存的原因被系统销毁，故 Activity 的 `onRestoreInstanceState()` 方法不会被执行。

### 3.5 应用主页设计技巧

主页面是应用核心功能的集中显示界面，设计良好的主页面能让用户根据直觉使用应用程序，也能让用户非常容易地完成所有任务。主页面的常见设计模式包括链接列表式、列表菜单式、选项卡菜单式、陈列馆式、仪表式、隐喻式等。

#### 1. 链接列表式

链接列表式也称为“快速启动板”(Launchpad)，该模式主页面的特点是，主页面中的菜单选项就是进入各个应用的起点。该方式对操作系统没有特殊要求，在各种设备上都有良好表现。常见的链接列表式布局包括  $3 \times 3$ 、 $2 \times 3$ 、 $2 \times 2$  等网格形式。但也不一定要拘泥于网格，也可以按照不同的优先级顺序，放大或者缩小网格，以突显出内容的层次感。图 3-10 是“美图秀秀”应用的主页面，这是一个典型的链接列表式设计模式。

#### 2. 列表菜单式

列表菜单式与链接列表式一样，每个菜单项都是进入各个应用的起点。列表菜单式又可分为分组列表、个性化列表、行内扩展式列表（这个一般用做次级导航）和增强性列表（是在简单的列表菜单之上增加搜索、浏览或过滤之类的功能后形成的）等多种形式。列表菜单式很适合用来显示较长或拥有次级文字内容的标题。图 3-11 是 Photo Sticker 应用的主页面，这是一个典型的列表菜单式设计模式。



图 3-10 链接列表式设计模式

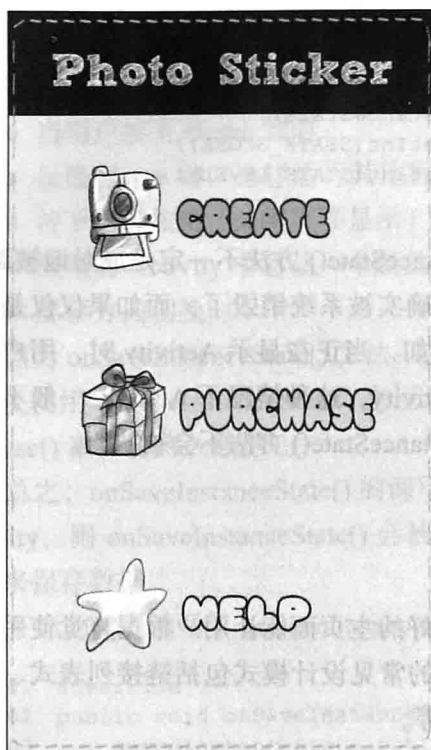


图 3-11 列表菜单式设计模式



图 3-12 选项卡式设计模式

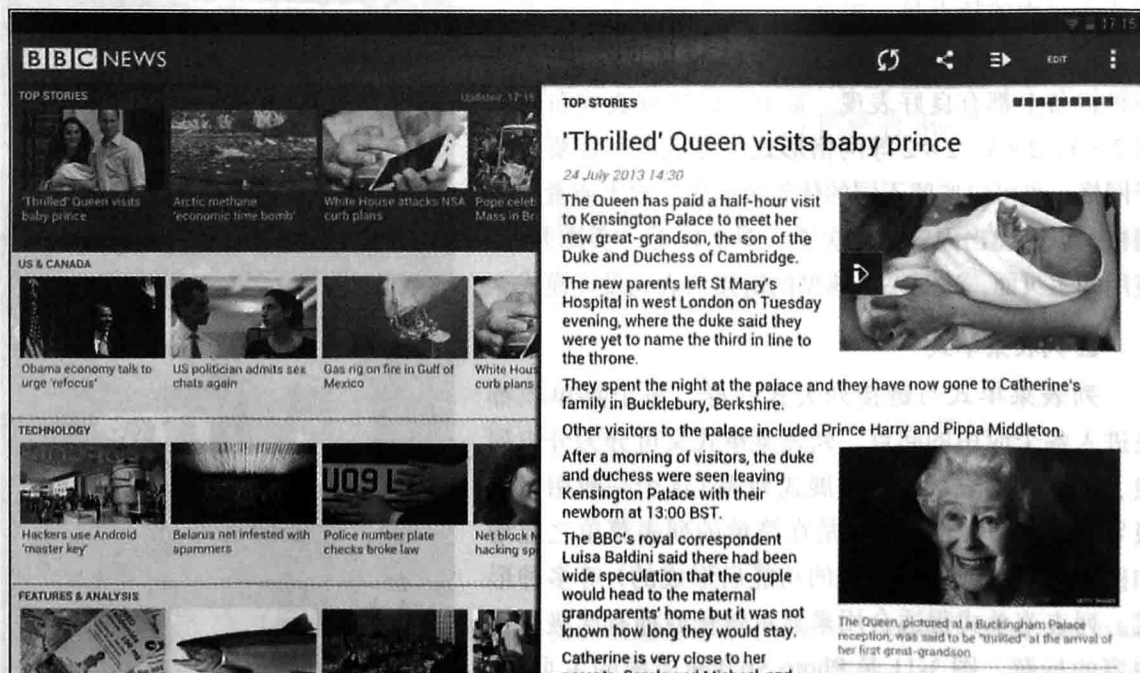


图 3-13 陈列馆式设计模式

### 3. 选项卡式

选项卡式类似标准网站的导航模式，不同的选项卡显示不同的应用。Android 中的选项卡位于屏幕的顶端，可以水平滚动选项卡，选项卡数量一般不超过 7 个。图 3-12 是 DailyCandy 应用的主页面，这是一个典型的选项卡式设计模式。

### 4. 陈列馆式

陈列馆式的主页面通过在平面上显示各个内容项来实现导航，可以将陈列馆布局成轮盘、网格或幻灯片演示。这种模式主要用来显示一些文章、菜谱、照片、产品等信息。图 3-13 是 BBC 新闻平板应用的主页面，这是一个典型的陈列馆式设计模式。

### 5. 隐喻式

隐喻式主页面的特点是使用页面显示一些隐喻的对象，用户看到这些对象就明白如何操作应用。这种导航主要用于游戏应用，但在帮助人们组织事物（如日记、书籍等），并对其进行分类的应用中也能看到。图 3-14 是悦读圈应用的主页面，这是一个典型的隐喻式设计模式。



图 3-14 隐喻式设计模式

## 3.6 用户体验设计

用户体验 (User Experience, 简称 UE) 是用户使用产品过程中建立起来的一种纯主观的感受。但是对于一个界定明确的用户群体来讲，其用户体验的共性是能够经由良好设计实验来认识到。在移动互联网技术飞速发展的今天，技术创新形态正在发生转变，“以用户为中心、以人为本”越来越得到重视，用户体验也因此被称作创新 2.0 模式的精髓。本节主要探讨在移动 APP 开发中，一些提高用户体验的做法。

### 1. 产品架构是否清晰

无论是九宫格式的架构还是标签页式的架构，都需要让用户一进入产品就可以一目了然地知道产品是干什么的、有几个功能模块、模块之间怎么切换。

同时，产品层级较深的设计师要清楚地了解产品有多少个二级页面、多少个三级页面。这些二级页面和三级页面的架构是复用一级页面的，还是有自己的架构。

有如下几个评估标准：

- 产品结构清晰，没有不必要的层级；
- 能快速了解产品有几个主要页面；
- 所有主要部分都能够通过首页访问；

- 清晰地指示了当前的位置。

## 2. 产品流程是否清晰

要想表现层越简单，背后的逻辑层可能就越复杂。那么评估流程时，不是以背后的逻辑层复杂度来评估，而是以表现层的简洁度来评估的。例如一个发布帖子的流程，总共需要几个步骤，涉及几个层级（一级页面到二级页面到三级页面……）。当然，不是说步骤越少、层级越浅就是好的设计，而是要简单、明确、清晰。没有不相关的干扰分支，没有经常会出现的误操作，没有停顿思考的空间，没有操作之后无反应的疑惑。

有如下几个评估标准：

- 明确产品有几个主要的任务流程；
- 每个任务流程清晰，没有太多分支；
- 任务流程符合用户操作流程；
- 用户可以取消正在执行的操作；
- 操作成功或失败都有明确的反馈；
- 在每个层级，都可以找到回到上一级的方法；
- 预防出错，如果出错要帮助用户从错误中恢复过来；
- 复杂的交互是否有很好的引导和帮助。

## 3. 控件使用是否准确

手机产品涉及很多的控件。一级标签栏、二级标签栏、列表、按钮、对话框、提示框、发布框等，这些控件的使用是否到位是衡量一个设计师细节设计能力的重要标准。例如有两个二级页面都需要二级标签栏，如果设计出来两个样式的话，那么说明没有用控件的思想来做设计。不仅设计师要设计两套二级控件，程序员要重复劳动，用户也会疑惑这两个控件是不是同一个含义、同一种操作方式。再例如，这个二级标签栏，顶层标签栏的从属内容是否在样式上与他有从属关系。如果不把握好这一细节，用户很可能将该二级标签栏看成了按钮。

有如下几个评估标准：

- 控件使用准确性（例如是否混淆了单选框和复选框，对话框层次过多等）；
- 控件的复用（例如两个页面都用到标签页，不用设计两个）；
- 控件的状态（例如不可点状态、可点状态、按下状态、长按状态）；
- 链接色的准确使用；
- 焦点状态的准确使用。

## 4. 信息传达是否到位

信息传达包含产品文案引导、按钮文案设计、列表文字布局、内容页排版、提醒文案设计等。有些公司的文案梳理工作是专门由内容编辑来做的，但是大部分公司都是由产品经理或者交互设计师直接完成的。那么文案是否准确，是否能有效地传达意思，也是衡量交互设

计的一个重要标准。文字长度限制、特殊情况处理是否考虑到位，也是衡量设计师工作的基准。而列表文字的布局、内容页的排版则是信息布局的重中之重。

有如下几个评估标准：

- 布局清晰；
- 文案简洁；
- 没有术语（例如“拉取失败”这种文案）；
- 合理排版（标题、作者、时间的字号、字色，页边距的运用）；
- 标签和内容的从属关系（能否看出当前标签页和当前标签页的从属内容）。

### 3.7 习题

设计一个下图所示的界面，单击 Login 按钮后，在新的界面显示登录信息。



## Android 组件编程

### 4.1 Service 与后台服务

Android 中的 Service 是一种应用组件，它可以长时间地在后台运行，并且不提供任何用户界面。Service 可以由其他应用组件启动并在后台运行。即使用户切换到其他应用程序，该 Service 也会一直运行。另外，其他组件可以绑定到 Service 上与之交互，甚至能够进行进程间通信。

#### 4.1.1 创建 Service

为了创建一个 Service，必须创建一个继承自 `android.app.Service` 或者它的子类的类，并重载一些回调方法来处理 Service 生命周期的关键环节，同时为组件提供一种合适的机制来绑定该 Service。下面是 Service 实现中重要的生命周期方法。

##### 1. `onStartCommand()`

当其他组件调用 `startService()` 来启动一个 Service 时，系统将调用 `onStartCommand()` 方法，这个 Service 被启动并可以无限期地运行在后台。如果实现了该方法，那么必须在 Service 完成时通过调用 `stopSelf()` 或 `stopService()` 停止它。如果 Service 只是提供 Bound Service，那么就不需要实现该方法。

##### 2. `onBind()`

当其他组件通过调用 `bindService()` 与 Service 绑定时，系统调用 `onBind()` 方法。在对这

个方法的实现中，必须通过返回一个 IBinder 对象提供一个供客户端和 Service 通信的接口。如果不允许服务被绑定，那么这里返回 null，否则必须实现该接口。

### 3. onCreate()

系统在 Service 第一次被创建时调用 onCreate() 方法，执行一次性的配置过程，例如音乐播放器中调用 initMediaPlayer() 方法初始化 MediaPlayer 对象。它是在 onStartCommand() 和 onBind() 方法之前被调用的。如果这个 Service 已经在运行了，那么这个方法不会被调用。

### 4. onDestroy()

系统在一个 Service 不再被使用，并且准备销毁时调用 onDestroy() 方法，它是 Service 收到的最后一个被调用的方法。在这里可以进行一些清除工作，例如线程、注册的监听以及接收器等。

如同 Activity 一样，必须在应用程序清单文件中声明所有 Service。为了声明 Service，需要在 AndroidManifest.xml 中添加 <service> 元素作为 <application> 元素的子元素。例如：

```
01 <application ... >
02   <service android:name=".MusicService" />
03   ...
04 </application>
```

其中，android:name 属性是唯一必要的属性，用于指定 Service 的类名。<service> 元素中还可以包含诸如启动 Service 所需的权限以及 Service 应该运行在的进程等属性。

## 4.1.2 Service 的生命周期

Service 的生命周期比 Activity 要简单得多。开发人员应更多地关注 Service 是如何被创建和销毁的，因为 Service 可以运行在后台而不被用户意识。图 4-1 演示了两种 Service 的生命周期方法。

通过图 4-1 可以看出，Service 的整个生命期发生在 onCreate() 被调用与 onDestroy() 返回之间，在 onCreate() 中完成它的初始化配置，而在 onDestroy() 中释放所有剩余资源。对于所有 Service 来说，onCreate() 和 onDestroy() 方法都会被调用，不管它是被 startService() 启动还是被 bindService() 绑定。

通过 startService() 方法启动的 Service，虽然调用 stopSelf() 和 stopService() 可以停止已启动的 Service，但对于 Service 来说没有相应的回调（没有 onStop() 回调）。所以，除非 Service 被绑定到一个客户端，否则系统在 Service 被停止时销毁 onDestroy() 是唯一接收到的回调。

通过 bindService() 方法启动的 Service 的激活生命期始于 onBind() 的调用，在 onUnbind() 返回时结束。如果单纯是一个 Bound Service，就不必人为地去管理 Service 的生命周期，Android 系统会根据 Service 是否被客户端绑定而去自动解绑定。





图 4-1 两种 Service 的生命周期

#### 4.1.3 Started Service

当一个应用程序组件（如 Activity）通过调用 `startService()` 方法启动一个 Service 时，该 Service 处于“被启动”状态，称为 Started Service。Service 被启动后，`onStartCommand()` 生命周期方法被调用，并接收 `startService()` 方法中传递来的 Intent。被启动的 Service 拥有独立于启动它的组件的生命周期，而且 Service 可以无限期地运行在后台中，即便启动它的组件已经销毁。因此，一个 Service 应该在任务完成后调用 `stopSelf()` 方法来自行停止，或者其他组件调用 `stopService()` 方法来停止它。通常，一个 Started Service 执行一个单一操作，并且不会返回结果给调用者。

下面的代码是一个音乐播放器后台的服务代码，在该方法中通过 `startService(intent)` 语句激活 MusicService 的 `onStartCommand()` 方法。`onStartCommand()` 方法代码如下：

```

01 @Override
02 public int onStartCommand(Intent intent, int flags, int startId) {
03     if (intent == null) {
04         return super.onStartCommand(intent, flags, startId);
05     }

```



```

06
07     Bundle bundle = intent.getExtras();
08     if (bundle == null) {
09         return super.onStartCommand(intent, flags, startId);
10     }
11
12     String path = bundle.getString("mediaPath");
13     position = bundle.getInt("index");
14     if (path != null) {
15         currentMusicPath = path;
16         playMusic(null);
17     }
18
19     return super.onStartCommand(intent, flags, startId);
20 }

```

该方法的主要作用是获取要播放音乐的路径和索引。

onStartCommand() 方法必须返回一个整型。这个整型告诉系统在系统杀死它的事件中应该如何继续执行 Service。onStartCommand() 返回值必须是以下常数之一：

- **START\_NOT\_STICKY** 如果系统在 onStartCommand() 返回后杀死 Service，那么不会重新创建 Service，除非有等待的 Intent 要传递。
- **START\_STICKY** 如果系统在 onStartCommand() 返回后杀死 Service，重启 Service，并且重新调用 onStartCommand()，但不重新传递最新的 Intent。
- **START\_REDELIVER\_INTENT** 如果系统在 onStartCommand() 返回后杀死 Service，那么重新创建 Service，并用最近传给 Service 的 Intent 调用 onStartCommand()。

#### 4.1.4 Bound Service

当一个应用程序组件通过调用 bindService() 方法绑定到一个 Service 时，该 Service 处于“被绑定”状态，称为 Bound Service。这是 Service 类的一个实现，它允许其他应用程序绑定到它并与之通信，以创建一个长期存在的连接。

##### 1. 创建 Bound Service

在创建提供绑定的 Service 时，必须提供一个 IBinder 接口，它为客户端提供可以用来与 Service 交互的编程接口。如果 Service 仅被本地应用程序使用，而不需要跨进程工作，那么就可以通过这种继承、扩展 Binder 类来实现，它使客户端直接访问 Service 中 Public 的方法。

创建 Bound Service 的一般步骤如下：

①在 Service 中创建一个 Binder 的接口，并实现：

- 包含客户端可以调用的 Public 方法。
- 或返回当前 Service 的实例——也包含客户端可以调用的 Public 方法。
- 或返回 Service 持有的其他类的实例——也包含客户端可以调用的 Public 方法。

例如：

```

01 public class MyBinder extends Binder {
02     public MusicService getService() {
03         return MusicService.this;
04     }
05 }

```

②在 `onBind()` 中返回 `Binder` 实例。例如：

```

01 @Override
02 public IBinder onBind(Intent intent) {
03     return new MyBinder();
04 }

```

③在客户端中，从 `onServiceConnected()` 回调方法中接收这个 `Binder`，并且使用 `Binder` 包含的 `Service` 提供的方法。例如：

```

01 private ServiceConnection connection = new ServiceConnection() {
02     @Override
03     public void onServiceConnected(ComponentName name, IBinder service) {
04         musicService = ((MusicService.MyBinder) service).getService();
05         updateProgress();
06         updatePlayPos.sendMessage(PROGRESS_CHANGED);
07     }
08
09     @Override
10     public void onServiceDisconnected(ComponentName name) {
11     }
12 };

```

所有客户端应该在合适的时机解除绑定，例如：

```

01 @Override
02 protected void onPause() {
03     if (isBound) {
04         unbindService(connection);
05         isBound = false;
06     }
07     super.onPause();
08 }

```

## 2. 绑定到 Service

应用程序组件通过调用 `bindService()` 绑定到一个 `Service`。然后 Android 系统调用 `Service` 的 `onBind()` 方法来返回一个用来与 `Service` 交互的 `IBinder` 对象。为了接收 `IBinder`，客户端必须创建一个 `ServiceConnection` 实例并将其传递给 `bindService()`。`ServiceConnection` 包含系统为了传递 `IBinder` 而调用的回调方法。



**注意** 只有 `Activity`、`Service` 和 `ContentProvider` 可以绑定到 `Service`，不能从一个 `BroadcastReceiver` 绑定到 `Service`。

从客户端绑定到 Service 的步骤如下：

①实现 ServiceConnection，同时必须重载如下两个回调方法：

- onServiceConnected() 系统通过调用该方法来传递 IBinder。
- onServiceDisconnected() 当客户端和 Service 之间的连接意外丢失时，系统调用该方法。客户端解绑定时，该方法不会被调用。

例如：

```
01 private ServiceConnection mConnection = new ServiceConnection() {
02     @Override
03     public void onServiceConnected(ComponentName className,
04         IBinder service) {
05         LocalBinder binder = (LocalBinder) service;
06         mService = binder.getService();
07         mBound = true;
08     }
09
10     @Override
11     public void onServiceDisconnected(ComponentName arg0) {
12         mBound = false;
13     }
14 };
```

②调用 bindService()，并传递 ServiceConnection 的实例，例如：

```
01 Intent intent = new Intent(this, LocalService.class);
02 bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
```

③当系统调用 onServiceConnected() 回调方法时，可以使用接口定义的方法开始调用 Service。

④为了断开与 Service 的连接，调用 unbindService()。

以下是关于 Service 绑定的注意事项：

- 必须处理 DeadObjectException 异常。它在连接断开时被抛出。这是远程方法抛出的唯一异常。
- Object 是跨进程的引用计数。
- 在客户端生命周期中启动、停止 Service 时，必须成对地使用绑定和解绑定。如果只是在 Activity 可见时才和 Service 交互，那么应该在 onStart() 时绑定，在 onStop() 时解绑定。如果想要 Activity 即使在后台运行停止时也要和 Service 交互，那么就应该在 onCreate() 时绑定，在 onDestroy() 时解绑定。

## 4.2 ContentProvider 与数据共享

ContentProvider 主要用来存储和查询数据，并且使这些数据对于所有的应用程序都可用。在 Android 中这是唯一一种在不同应用中共享数据的方法。

## 4.2.1 系统中的 ContentProvider

Android 系统为一些常用的信息（如音乐、视频、图像、联系人等）内置了一系列的 ContentProvider，这些信息都位于 android.provider 包中。只要在 AndroidManifest.xml 中声明相应的许可，就可以在自己开发的应用程序中访问它们。

例如，下面的 getList() 方法就是从系统 ContentProvider 获取音乐数据信息的常见处理方法，代码如下：

```

01 @Override
02 public List<?> getList() {
03     List<AudioBean> list = null;
04     if (context != null) {
05         Cursor cursor = context.getContentResolver().query(
06             MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, null, null,
07             null, null);
08
09         if (cursor != null) {
10             list = new ArrayList<AudioBean>();
11             while (cursor.moveToNext()) {
12                 int id = cursor.getInt(cursor
13                     .getColumnIndexOrThrow(MediaStore.Audio.Media._ID));
14                 String title = cursor
15                     .getString(cursor.getColumnIndexOrThrow(
16                         MediaStore.Audio.Media.TITLE));
17                 String album = cursor
18                     .getString(cursor.getColumnIndexOrThrow(
19                         MediaStore.Audio.Media.ALBUM));
20                 String artist = cursor
21                     .getString(cursor.getColumnIndexOrThrow(
22                         MediaStore.Audio.Media.ARTIST));
23                 String path = cursor
24                     .getString(cursor.getColumnIndexOrThrow(
25                         MediaStore.Audio.Media.DATA));
26                 String displayName = cursor
27                     .getString(cursor.getColumnIndexOrThrow(
28                         MediaStore.Audio.Media.DISPLAY_NAME));
29                 String mimeType = cursor
30                     .getString(cursor.getColumnIndexOrThrow(
31                         MediaStore.Audio.Media.MIME_TYPE));
32                 long duration = cursor
33                     .getInt(cursor.getColumnIndexOrThrow(
34                         MediaStore.Audio.Media.DURATION));
35                 long size = cursor
36                     .getLong(cursor.getColumnIndexOrThrow(
37                         MediaStore.Audio.Media.SIZE));
38                 if (size == 0) // 去除空文件
39                     continue;
40
41                 AudioBean audioBean = new AudioBean(id, title, album,

```

```

42         artist, path, displayName, mimeType, duration, size);
43         list.add(audioBean);
44     }
45
46     cursor.close();
47 }
48 }
49
50 return list;
51 }

```

05 ~ 07 行是查询 `ContentProvider` 数据的方法。`ContentProvider` 虽然提供了一组标准的接口可以使其他应用程序存取由它控制的数据，但应用程序并不会直接调用 `ContentProvider` 中的方法，而是通过类 `ContentResolver`。`ContentResolver` 能够与任何一个 `ContentProvider` 通信，它与 `ContentProvider` 合作管理进程间的通信。多数情况下是在一个 `Activity` 或其他组件内部通过调用 `getContentResolver()` 方法来获取 `ContentResolver` 对象，然后使用 `ContentResolver` 类提供的成员方法对 `ContentProvider` 中的数据进行查询、插入、修改和删除等操作。

有时，使用系统 `ContentProvider` 提供的数据需要相关的权限，例如，读取 `MediaStore` 库就需要在项目的 `AndroidManifest.xml` 中添加如下权限：

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

## 4.2.2 通用资源标志符

`ContentProvider` 通过暴露一个公共的通用资源标志符（Universal Resource Identifier，简称 `Uri`）来标识唯一的数据集。例如，4.2.1 节中 `getList()` 方法的 06 行的 `MediaStore.Audio.Media.EXTERNAL_CONTENT_URI` 就是一个 `Uri`。

`ContentProvider` 可以控制多个数据集，并为每一个数据集提供一个 `Uri`。所有的 `Uri` 开头都是一个“`content://`”的字符串。“`content://`”表示该数据是由 `ContentProvider` 来控制管理的。如果定义了一个 `ContentProvider`，就需要同时为它的 `Uri` 定义一个常量。

图 4-2 是一个 `Uri` 的典型构成。

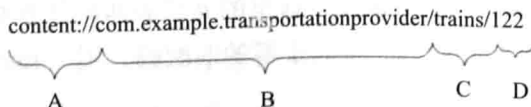


图 4-2 Uri 构成示例

其中的 A、B、C、D 四个部分的说明如下：

- A：标准前缀，用来说明这是一个 `ContentProvider` 所控制的数据集。

- B: Uri 的权限, 它定义了是哪个 ContentProvider 提供这些数据。对于第三方应用程序, 为了保证 Uri 标识的唯一性, 它必须是一个完整的、小写的字符串。该权限在 AndroidManifest.xml 的 <provider> 元素的 authorities 属性中声明。例如:

```
01 <provider android:name=".TransportationProvider"
02     android:authorities="com.example.transportationprovider"
03     ... >
```

- C: 路径, ContentProvider 使用这些路径来确定当前需要什么类型的数据, Uri 中可能不包括路径, 也可能包括多个。如果 ContentProvider 只展示一种数据类型 (例如 trains), 那可以缺省。如果 ContentProvider 展示的是一系列类型, 包括子类型, 例如: “land/bus”、“land/train”、“sea/ship”和“sea/submarine”四种可能性, 则后面需要添加片段。

- D: 如果 Uri 中包含该部分, 表示需要获取记录的 ID; 如果没有 ID, 就表示返回全部。

例如, content://com.example.transportationprovider/trains/122 表示操作 trains 表中\_ID 为 122 的记录; content://com.example.transportationprovider/trains/122/type 表示操作 trains 表中\_ID 为 122 的记录的 type 字段。

下面介绍两个和 Uri 相关的工具类的使用。

## 1. UriMatcher

UriMatcher 类主要用于匹配 Uri。其使用步骤一般是:

- ①注册需要匹配的 Uri 路径, 例如:

```
01 public static final UriMatcher sUriMatcher;
02 static {
03     sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
04     sUriMatcher.addURI(NotePad.AUTHORITY, "notes", NOTES); // NOTES=1
05     sUriMatcher.addURI(NotePad.AUTHORITY, "notes/#", NOTE_ID);
06     sUriMatcher.addURI(NotePad.AUTHORITY, "live_folders/notes",
07         LIVE_FOLDER_NOTES);
08 }
```

其中: 常量 UriMatcher.NO\_MATCH 表示不匹配任何路径, 返回码是 1。

addURI() 方法添加需要匹配的 Uri, 第一个参数传入标识 ContentProvider 的 Authority 字符串。第二个参数传入需要匹配的路径, 这里的 # 代表匹配任意数字, 另外还可以用 \* 来匹配任意文本。第三个参数必须传入一个大于零的匹配码, 用于 match() 方法对相匹配的 Uri 返回相对应的匹配码。

- ②使用 uriMatcher.match(uri) 方法对输入的 Uri 进行匹配, 如果匹配就返回匹配码 (大于 0)。
- 例如:

```
01 public int delete(Uri uri, String where, String[] whereArgs) {
02     SQLiteDatabase db = mOpenHelper.getWritableDatabase();
```

```

03     int count;
04     switch (sUriMatcher.match(uri)) {
05     case NOTES:
06         count = db.delete(NOTES_TABLE_NAME, where, whereArgs);
07         break;
08     case NOTE_ID:
09         String noteId = uri.getPathSegments().get(1);
10         count = db.delete(NOTES_TABLE_NAME, Notes._ID + "=" + noteId
11             + (!TextUtils.isEmpty(where)
12             ? " AND (" + where + ')' : ""), whereArgs);
13         break;
14     default:
15         throw new IllegalArgumentException("Unknown URI " + uri);
16     }
17     getContext().getContentResolver().notifyChange(uri, null);
18     return count;
19 }

```

若无法匹配传入的 Uri, 则抛出 `IllegalArgumentException` 异常。

## 2. ContentUris

`ContentUris` 用于获取 Uri 路径后面的 ID 部分, 它有两个比较实用的方法:

① `withAppendedId(uri, id)` 用于为路径加上 ID 部分。

如 `Uri noteUri = ContentUris.withAppendedId(NotePad.Notes.CONTENT_URI, rowId);`

② `parseId(uri)` 用于从路径中获取 ID 部分。

如 `Uri uri = Uri.parse("content://contacts/phones/10");`

`long phoneid = ContentUris.parseId(uri);` // 获取的结果为 10

### 4.2.3 使用 ContentProvider

若要查询一个 `ContentProvider`, 需要以下三方面的信息:

- 标识该 `ContentProvider` 的 Uri。
- 想要收到的数据字段的名称。
- 这些字段的数据类型。

如果想要查询某个特定记录, 还需要该条记录的 ID。

#### 1. 生成查询

通常使用 `ContentResolver.query()` 方法或 `Activity.managedQuery()` 方法来查询一个 `ContentProvider`, 返回同样的 `Cursor` 对象。其中, `managedQuery()` 方法的原型是:

```

01 public final Cursor managedQuery(Uri uri, String[] projection,
02     String selection, String[] selectionArgs, String sortOrder) {
03     Cursor c = getContentResolver().query(uri, projection, selection,
04         selectionArgs, sortOrder);
05     if (c != null) {

```

```
06         startManagingCursor(c);
07     }
08     return c;
09 }
```

`getContentResolver().query()` 方法中的参数说明如下:

- `uri` 用于查询 `ContentProvider` 的 `Uri`。
- `projection` 用于标识 `Uri` 中有哪些列需要包含在返回的 `Cursor` 对象中。例如: `String[] projection = { Contacts.PeopleColumns.NAME, Contacts.PeopleColumns.NOTES };`。
- `selection` 作为查询的过滤参数, 类似于 SQL 中 `Where` 语句之后的条件选择。例如: `String selection = Contacts.People.NAME + "=?"`。
- `selectionArgs` 查询条件参数, 配合 `selection` 参数使用。例如: `String[] selectionArgs = {"azul", "liweiyong"};`。
- `sortOrder` 查询结果的排序方式。例如: `String sortOrder = Contacts.PeopleColumns.NAME;`。

## 2. 读取查询结果

查询所返回的 `Cursor` 对象是一个允许访问的记录集。如果查询一个指定 ID 的记录, 则只会返回一个值。因为 `Cursor` 对象对每种类型的数据都使用单独的方法 (如 `getString()`、`getInt()`、`getFloat()`), 因此必须知道字段的数据类型, 才可以从指定的记录字段中读取数据。`Cursor` 允许通过该列的位置来获取某个列的名称, 或通过该列名称获得该列的位置。例如 4.2.1 节中 `getList()` 方法的 12 ~ 13 行。

## 3. 添加记录

首先创建 `ContentValues` (`ContentValues` 主要是存放表中的数据段, 以及其对应的值, 与 `Hashtable` 一样采用键值对的形式存储, 键是一个 `String` 类型, 值是基本数据类型) 对象, 每个 `key` 对应列的名称, `value` 就是所要插入到列的值。调用 `ContentResolver.insert()` 并传递这个 `ContentProvider` 所需要的 `Uri` 和 `ContentValues`。结果将返回一个新的记录, 它以 `Uri` 形式返回。例如:

```
01 ContentValues values = new ContentValues();
02 values.put(People.NAME, "Abraham Lincoln");
03 values.put(People.STARRED, 1);
04 Uri uri = getContentResolver().insert(People.CONTENT_URI, values);
```

## 4. 删除记录

删除单条记录, 可以调用 `ContentResolver.delete()` 传入指定行的 `Uri`。如果删除多条记录, 可以调用 `ContentResolver.delete()` 传入要删除记录的类型的 `Uri`, 例如:

```
01 String mSelectionClause = UserDictionary.Words.APP_ID + " LIKE ?";
02 String[] mSelectionArgs = {"user"};
```



```

03 int mRowsDeleted = 0;
04 ...
05 mRowsDeleted = getContentResolver().delete(
06     UserDictionary.Words.CONTENT_URI,           // the user dictionary content URI
07     mSelectionClause                             // the column to select on
08     mSelectionArgs                               // the value to compare to
09 );

```

## 4.3 BroadcastReceiver 与广播意图

在 Android 中，广播是一种广泛运用的在应用程序之间传输信息的机制，而 `BroadcastReceiver` 是对发送出来的广播进行过滤接受并响应的一类组件，它和事件处理机制类似，但是事件处理机制是程序组件级别的，而广播事件处理机制是系统级别的。

### 4.3.1 BroadcastReceiver 的工作机制

在程序中使用 `BroadcastReceiver` 的一般步骤如下：

①定义一个类继承 `BroadcastReceiver`，并且重载 `onReceiver()` 方法来响应事件。

例如，监听 SD 卡插拔的 `SDCardBroadCastReceiver` 的 `onReceiver()` 方法代码如下：

```

01 @Override
02 public void onReceive(Context context, Intent intent) {
03
04     String action = intent.getAction();
05
06     if (action.equals(Intent.ACTION_MEDIA_EJECT)) {
07         Toast.makeText(context, R.string.sdcard_removed, Toast.LENGTH_LONG)
08             .show();
09     } else if (action.equals(Intent.ACTION_MEDIA_MOUNTED)) {
10         Toast.makeText(context, R.string.sdcard_added, Toast.LENGTH_LONG)
11             .show();
12     }
13
14 }

```

②在程序中注册 `BroadcastReceiver`。

在 `Activity` 实例的 `onResume()` 生命周期方法中，调用如下语句启动监听服务：

```

01 Intent i = new Intent(MainActivity.this, ListenService.class);
02 i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
03 startService(i);

```

在执行监听服务的 `ListenService` 的 `onCreate()` 方法中注册监听广播，代码如下：

```

01 @Override
02 public void onCreate() {

```

```

03     super.onCreate();
04
05     IntentFilter filter = new IntentFilter();
06     filter.addAction(Intent.ACTION_MEDIA_EJECT);
07     filter.addAction(Intent.ACTION_MEDIA_MOUNTED);
08     filter.setPriority(1000);
09     filter.addDataScheme("file");
10     receiver = new SDCardBroadcastReceiver();
11     registerReceiver(receiver, filter);
12 }

```

在 `onDestroy()` 方法中取消注册监听广播，代码如下：

```

01 @Override
02 public void onDestroy() {
03     unregisterReceiver(receiver);
04     super.onDestroy();
05 }

```

这样，在应用退出时，关闭该监听（如果是静态注册监听，那么即使退出了应用程序，监听也会继续）。

③构建 `Intent` 对象，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，调用 `sendBroadcast()` 方法将广播发出。

④当 `Intent` 发送以后，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。

一个 `BroadcastReceiver` 对象只有在被调用 `onReceive (Context, Intent)` 时才有效，当从该方法返回后，该对象就无效了，其生命周期也结束了。

`BroadcastReceiver` 并没有提供可视化的界面来显示广播信息。一般都是使用 `Notification` 和 `NotificationManager` 来实现可视化的信息界面，用以显示广播信息的图标、内容及振动等提示信息。

### 4.3.2 广播的类型

有如下三种广播类型。

#### 1. 正常广播

正常广播通过 `Context.sendBroadcast()` 方法发送，是完全异步的。`BroadcastReceiver` 的 `onReceiver()` 方法不能包含所要使用的结果或中止广播的方法。

#### 2. 异步广播

异步广播通过 `Context.sendStickyBroadcast()` 方法发送，当处理完相应的 `Intent` 之后，`BroadcastReceiver` 依然存在，这时 `registerReceiver (BroadcastReceiver, IntentFilter)` 还能收到它的值，直到调用 `removeStickyBroadcast (intent)` 方法将其去掉为止。异步广播不能将处理结果传给下一个接收者，且无法终止广播。

异步广播的发送和移除都需要在 AndroidManifest.xml 里声明如下权限：

```
<uses-permission android:name="android.permission.BROADCAST_STICKY" />
```

另外，有如下方法：

```
sendStickyOrderedBroadcast(intent, resultReceiver, scheduler, initialCode,
initialData, initialExtras)
```

这个方法既有有序广播的特性也有异步广播的特性，发送该广播也需要上面的权限，否则将抛出 SecurityException 异常。同时，用这个方法发来的广播，在动态注册方式时需要注明优先级，如果都没有优先级，则代码注册的广播优先级最高。

### 3. 有序广播

有序广播通过 Context.sendOrderedBroadcast() 方法发送，每次被发送到一个 Receiver。所谓有序，就是每个 Receiver 执行后可以传播到下一个 Receiver，也可以完全中止传播（即不传播给其他 Receiver）。而 Receiver 运行的顺序可以通过比较 IntentFilter 里的 android:priority 来控制，当 priority 优先级相同时，Receiver 以任意的顺序运行。

发送有序广播的方法有：

```
sendOrderedBroadcast(intent, receiverPermission)
sendOrderedBroadcast(intent, receiverPermission, resultReceiver, scheduler, initialCode,
initialData, initialExtras)
```

参数说明：

- receiverPermission 是权限，如果为 null，表示不经许可的要求。
- resultReceiver 是自定义的 BroadcastReceiver 用来当作最后的广播接收器的。
- scheduler 用于调度自定义处理程序，用以安排 resultReceiver 回调。
- initialCode 是一种结果代码的初始值，通常为 Activity.RESULT\_OK。
- initialData 是一种结果数据的初始值，为 String 类型，通常情况下为空。
- initialExtras 是一种结果额外的初始值，是 Bundle 类型，通常情况下为空。

关于优先级的说明如下：

- 广播有级别之分，其优先级别在 <intent-filter> 的 priority 中声明，级别数值在 -1000 ~ 1000 之间，值越大，优先级越高。
- 同级别广播的接收是先后随机的，然后再到级别低的接收广播。
- 同级别接收时，如果先接收到的 Receiver 把广播终止了，那么同级别的其他 Receiver 是无法收到该广播的。
- 高级别的 Receiver 收到该广播后，可以决定是否将该广播截断。

注意，通过 sendBroadcast() 发出的 Intent 在 ReceiverActivity 不处于 onResume 状态时是无法接收到的，即使后面再次使其处于该状态也无法接收到。而 sendStickyBroadcast() 发出的 Intent 当 ReceiverActivity 重新处于 onResume 状态之后就能重新接收到，这就是广播滞留

的表现, 也就是说 `sendStickyBroadcast()` 发出的最后一个 `Intent` 会被保留, 当下次 `Receiver` 处于活跃状态时, 又会接收到它。

### 4.3.3 接收广播

接收广播通过定义一个继承 `BroadcastReceiver` 类来实现, 继承该类后重载其 `onReceive()` 方法, 并在该方法中响应事件。例如, 下面的代码演示了一个短信接收广播的工作方法:

```
01 public class SMSReceiver extends BroadcastReceiver {
02     @Override
03     public void onReceive(Context context, Intent intent) {
04         Bundle bundle = intent.getExtras();
05         if (bundle != null) {
06             Object[] objArray = (Object[]) bundle.get("pdus");
07             SmsMessage[] messages = new SmsMessage[objArray.length];
08             for (int i = 0; i < objArray.length; i++) {
09                 messages[i] = SmsMessage.createFromPdu((byte[]) objArray[i]);
10                 StringBuilder str = new StringBuilder("from: ");
11                 str.append(messages[i].getDisplayOriginatingAddress());
12                 str.append("\nmessage:\n");
13                 str.append(messages[i].getDisplayMessageBody());
14                 Toast.makeText(context, str.toString(), Toast.LENGTH_LONG)
15                     .show();
16             }
17         }
18     }
19 }
```

响应广播事件处理的 `Activity` 需要在 `onStart()` 中对相应的 `BroadcastReceiver` 进行注册, 例如:

```
01 protected void onStart() {
02     super.onStart();
03     smsReceiver = new SMSReceiver();
04     registerReceiver(callReceiver, new IntentFilter(
05         "android.provider.Telephony.SMS_RECEIVED"));
06 }
```

同时, 在 `onStop()` 中进行注销, 例如:

```
01 protected void onStop() {
02     unregisterReceiver(smsReceiver);
03     super.onStop();
04 }
```

### 4.3.4 注册广播

注册广播分为静态注册和动态注册两种。

## 1. 静态注册

在 AndroidManifest.xml 的 <application> 里定义 receiver 并设置要接收的 Action 和 IntentFilter, 例如:

```
<receiver android:name=".SMSReceiver">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
```

## 2. 动态注册

方法一: registerReceiver(receiver, filter)

第一个参数是要处理广播的 BroadcastReceiver (可以是系统的, 也可以是自定义的); 第二个参数是 Intent 过滤器。

方法二: registerReceiver(receiver, filter, broadcastPermission, scheduler)

第三个参数是广播权限; 第四个参数是 Handler。

注意, 如果在 AndroidManifest.xml 文件里已经声明了权限, 在 registerReceiver() 方法里再次声明权限, 则 Receiver 无法收到广播。如果在 AndroidManifest.xml 文件和 registerReceiver() 方法里都没有注册, Receiver 也无法收到广播。

一个 <receiver> 可以接收多个 action, 即可以有多个 <intent-filter>, 这需要在 onReceive() 里对 intent.getAction (actionName) 进行判断。

对于动态注册, 需要特别注意的是, 在退出程序前要记得调用 Context.unregisterReceiver() 方法对广播进行注销。如果在 Activity.onResume() 里面注册了, 就必须在 Activity.onPause() 中注销。

## 4.4 Intent 与组件通信

Android 中提供了 Intent 机制来协助应用之间的交互与通信, Intent 负责对应用中一次操作的动作、动作涉及的数据以及附加数据等信息进行描述。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。

### 4.4.1 Intent 处理机制

在 3.2.1 节中, 介绍了通过显式 Intent 和隐式 Intent 分别调用其他 Activity 的方法。下面以一个示例简要描述 Intent 的工作过程。

一个 Activity 的事件包括如下代码:

```
01 Intent intent = new Intent(AudioManager.ACTION_AUDIO_BECOMING_NOISY);
```

```
02 mContext.sendBroadcast(intent);
```

01 行构造一个 Intent，该 Intent 只传入了一个参数 Action，没有指定 Data 以及 Category。也就是说，如果某个 Receiver 在 AndroidManifest.xml 里面注册如下：

```
01 <receiver android:name="MediaButtonIntentReceiver">
02     <intent-filter>
03         <action android:name="android.media.AUDIO_BECOMING_NOISY" />
04     </intent-filter>
05 </receiver>
```

该 Receiver 的 onReceive() 方法将会被调用，其中 Receiver 表示处理该 Intent 消息的类，而 <intent-filter> 表示这个 Receiver 关心哪些 Intent，这里写明了该 Receiver 只关心 Action 为 android.media.AUDIO\_BECOMING\_NOISY 的 Intent。因此，当 02 行被执行时，该 Receiver 被激活。

通过如上示例可以看出，Intent 相当于应用程序之间的通信网络，是对执行某个操作的一个抽象描述。通过 Intent 去发布一个任务可以不用确切知道哪个应用程序组件将会去执行该任务，只要关心该任务是否被执行、是否按照要求完成就够了，剩下的细节由系统自动完成。Intent 不仅可用于应用程序之间，也可用于应用程序内部组件之间的交互。



**注意** Intent 在通信过程中需要检查相应的权限，如果 AndroidManifest.xml 中没有相应的权限说明，则无法激活组件。

## 4.4.2 Intent 对象

Intent 是一种运行时绑定 (Runtime Binding) 机制，它能在程序运行的过程中附加相应的组件名称、动作、数据、类别和附加信息。

Intent 声明的一般形式如下：

```
Intent <Intent_name> = new Intent ( <ACTION>, <Data> );
```

### 1. 组件

Intent 对象通过组件 (Component) 来显式设置 Intent 的访问对象。指定的方法有 setComponent (ComponentName)、setClass (Context, Class) 和 setClassName()，通过 getComponent() 读取组件。

例如，用下面的方法通过 setComponent() 启动系统短信发送程序：

```
01 private void startMMS() {
02     Intent intent = new Intent();
03     ComponentName comp = new ComponentName("com.android.mms",
04         "com.android.mms.ui.ConversationList");
05     intent.setComponent(comp);
06     intent.setAction("android.intent.action.VIEW");
```

```
07     startActivity(intent);
08 }
```

显式方式常用于自己应用内部的消息传递，例如应用中一个 Activity 启动一个相关的 Service 或者启动另一个 Activity。

## 2. 动作

动作 (Action) 是一个字符串，是对 Intent 执行动作的描述，该动作可以是系统预定义的，如 ACTION\_MAIN 声明的一般形式如下：

```
public static final String ACTION_MAIN = "android.intent.action.MAIN";
```

动作也可以是自己定义的字符串。自定义动作字符串应该将应用程序的包的名字作为前缀。如 ch04.intentproject.newAction。

一个 Intent 对象的动作通过 setAction() 方法设置，通过 getAction() 方法读取。动作很大程度上决定了剩下的 Intent 如何构建，特别是数据和附加字段。因此，应该尽可能明确指定动作，并紧密关联到其他 Intent 字段。即应该定义用户的组件能够处理的 Intent 对象的整个协议，而不仅仅是单独地定义一个动作。

## 3. 数据

数据 (Data) 是作用于 Intent 上的数据的 Uri 和数据的 MIME 类型。在 Android 系统中，传给 Intent 的数据用 Uri 格式表示，因此需要使用 Uri.parse() 方法将字符串格式化。不同的动作有不同的数据规格。例如，如果动作字段是 ACTION\_EDIT，数据字段将包含用于编辑文档的 Uri；如果动作是 ACTION\_CALL，数据字段将是一个 tel: URI 和将拨打的号码；如果动作是 ACTION\_VIEW，数据字段是一个 http: Uri，接收 Activity 将被调用去下载和显示 Uri 指向的数据。

例如，用下面的方法启动系统拨号程序：

```
01 private void startCALL(){
02     Intent intent = new Intent( );
03     intent.setAction(intent.ACTION_DIAL);
04     intent.setData(Uri.parse("tel://10086"));
05     startActivity(intent);
06 }
```

当为某个组件匹配一个可以处理数据的 Intent 时，通常除了要了解 Data 的 Uri 以外，重要的是要知道 Data 的类型 (MIME 类型)。在许多情况下，数据类型能够从 Uri 中推测，特别是 content: Uri，它表示位于设备上的数据且被 ContentProvider 控制。

除了使用 setData() 方法指定数据的 Uri 外，还可以使用 setType() 指定 MIME 类型，setDataAndType() 指定数据的 Uri 和 MIME 类型。通过 getData() 读取 Uri，getType() 读取类型。

## 4. 类别

类别 (Category) 是一个字符串，它描述了应该处理 Intent 的组件类型信息。可以在一个

Intent 对象中指定任意数量的类别描述。Intent 类定义了许多 Category 常数，如 CATEGORY\_HOME 声明如下：

```
public static final String CATEGORY_HOME = "android.intent.category.HOME";
```

在 AndroidManifest.xml 中，android.intent.action.MAIN 和 android.intent.category.LAUNCHER 分别表示 Activity 开始新的任务和转到启动列表界面。

向 Intent 对象添加一个类别使用 addCategory() 方法，删除一个之前添加的类别使用 removeCategory() 方法，获取 Intent 对象中的所有类别使用 getCategories() 方法。例如，用下面的方法返回到主界面：

```
01 private void back2Main(){
02     Intent intent = new Intent();
03     intent.setAction(Intent.ACTION_MAIN);
04     intent.addCategory(Intent.CATEGORY_HOME);
05     intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
06     startActivity(intent);
07 }
```

### 4.4.3 Intent 的解析

在 Android 平台上，用户的操作行为是由各种不同的事件组成的，系统会将每个事件都抽象为 Intent 对象，然后为这些 Intent 对象寻找解决需求相适应的具体组件和方法。

对于显式的 Intent，因为已经明确了目标组件，因此不需要系统去解析。由于隐式的 Intent 没有明确的目标组件，因此，当隐式的 Intent 被抛出后，系统在众多组件中根据 Intent 过滤器中的 action、datatype、Uri 来寻找与其匹配的处理方法。如果存在多个结果，用户可以根据需要选择合适的处理方法。

一个 Intent 过滤器是一个 IntentFilter 类的实例，它包括 3 个方面：Action、Data（包括 Uri 和 MIME）、Category。Intent 过滤器要检测隐式 Intent 的三个方面（Intent 对象的 Extra 和 Flag 在 Intent 过滤器方面并不发挥作用），其中任何一个匹配失败，Android 系统都不会传递 Intent 给该组件。然而，因为一个组件可以有多个 Intent 过滤器，即使一个 Intent 不能通过组件的某个过滤器检测，其他过滤器也仍可能通过检测。

Intent 过滤器通常不在 Java 代码中设置，而是在应用程序的 AndroidManifest.xml 中以 <intent-filter> 元素设置。例如：

```
01 <intent-filter>
02     <action android:name="android.intent.action.GET_CONTENT" />
03     <category android:name="android.intent.category.DEFAULT" />
04     <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
05 </intent-filter>
```

#### 1. Action 过滤规则

虽然一个 Intent 对象仅包含一个动作，但是一个 Intent 过滤器可以列出不止一个动作。



一个 Intent 过滤器必须至少包含一个 <action> 子元素, 否则它将阻塞所有的 Intent。例如:

```
01 <intent-filter>
02     <action android:name="android.intent.action.VIEW" />
03     <action android:name="android.intent.action.EDIT" />
04     <action android:name="android.intent.action.PICK" />
05     <!--下面的声明必须存在, 否则组件不被匹配 -->
06     <category android:name="android.intent.category.DEFAULT" />
07 </intent-filter>
```

对 Action 的过滤原则是: Intent 对象中指定的动作必须匹配 Intent 过滤器动作列表中的一个。如果 Intent 或 Intent 过滤器没有指定一个动作, 则: 如果 Intent 过滤器没有指定动作, 没有一个 Intent 被匹配, 所有的 Intent 都检测失败, 即没有 Intent 能够通过 Intent 过滤器; 如果 Intent 对象没有指定动作 (但必须指定其他属性), 将自动通过检查 (前提是 Intent 过滤器的动作列表不为空)。

## 2. Category 过滤规则

清单文件中的 <intent-filter> 元素以 <category> 子元素列出类别, 例如:

```
01 <intent-filter . . . >
02     <category android:name="android.intent.category.DEFAULT" />
03     <category android:name="android.intent.category.BROWSABLE" />
04     ...
05 </intent-filter>
```

对类别的过滤原则是: Intent 对象中的每个类别必须匹配 Intent 过滤器中的一个。即 Intent 过滤器能够列出额外的类别, 但是 Intent 对象中的类别都必须能够在 Intent 过滤器中找到, 只要有一个类别在 Intent 过滤器列表中没有, 就算类别检测失败。因此, 原则上如果一个 Intent 对象中没有类别 (即类别字段为空), 那么应该总是通过类别测试, 而不管 Intent 过滤器中有什么类别。但是有个例外, Android 对待所有传递给 Context.startActivity() 的隐式 Intent 至少包含 “android.intent.category.DEFAULT” 类别。

## 3. Data 过滤规则

当匹配一个 Intent 到一个能够处理数据的组件时, 通常需要知道数据的类型和它的 Uri。在许多情况下, 数据类型能够从 Uri 中推测。每个 <data> 元素指定一个 Uri 和数据类型。例如:

```
01 <intent-filter . . . >
02     <data android:mimeType="video/mpeg" android:scheme="http" ... />
03     <data android:mimeType="audio/mpeg" android:scheme="http" ... />
04     ...
05 </intent-filter>
```

Uri 有 scheme、host、port、path 四个属性, 有关 Uri 的详细介绍参见 4.2.2 节。

对数据的过滤原则是: 数据检测既要检测 Uri, 也要检测数据类型。规则如下:

①一个 Intent 对象既不包含 Uri, 也不包含数据类型: 仅当 Intent 过滤器也不指定任何

Uri 和数据类型时，才能通过检测，否则不能通过。

②一个 Intent 对象包含 Uri，但不包含数据类型：仅当 Intent 过滤器也不指定数据类型，同时它们的 Uri 匹配，才能通过检测。例如，mailto：和 tel：都不指定实际数据。

③一个 Intent 对象包含数据类型，但不包含 Uri：仅当过滤器也只包含数据类型，且与 Intent 相同时，才能通过检测。

④一个 Intent 对象既包含 Uri，也包含数据类型（或数据类型能够从 Uri 推断）：数据类型部分，只有与 Intent 过滤器中之一匹配才算通过；Uri 部分，它的 Uri 要出现在 Intent 过滤器中，或者它有 content：或 file：Uri，又或者 Intent 过滤器没有指定 Uri。

说明：当比较 Intent 对象和 Intent 过滤器的 Uri 时，仅仅比较 Intent 过滤器中出现的 Uri 属性。例如，如果一个 Intent 过滤器仅指定了 scheme，所有有此 scheme 的 Uri 都匹配 Intent 过滤器；如果一个 Intent 过滤器指定了 scheme 和 authority，但没有指定 path，所有匹配 scheme 和 authority 的 Uri 都通过检测，而不管它们的 path 是否匹配；如果四个属性都指定了，则要都匹配才能算是匹配。然而，Intent 过滤器中的 path 可以包含通配符来匹配 path 中的一部分。

#### 4. 通用匹配

<data> 元素的 type 属性指定数据的 MIME 类型。Intent 对象和 Intent 过滤器都可以用 “\*” 通配符匹配子类型字段，例如 “text/\*”，“audio/\*” 表示任何子类型。

通常，组件能够从文件或 ContentProvider 获取本地数据。因此，它们的 Intent 过滤器仅列出数据类型且不必明确指出 content：和 file：scheme 的名字。这是一种典型的情况，如一个 <data> 元素描述如下：

```
<data android:mimeType="image/*" />
```

这表明该组件能够从 ContentProvider 获取 image 数据并显示它。

因为大部分可用数据由 ContentProvider 分发，因此，Intent 过滤器指定一个数据类型但不指定 Uri 是一种非常通用的做法。

### 4.5 基于组件的应用模型

Android 应用模型的设计思想取自 Web 2.0 的 Mashup 概念，是基于组件的应用设计模式。在该模型下，每个应用都由一系列的组件搭建而成，组件通过应用的配置文件（AndroidManifest.xml）描述功能。

Mashup，在国内一般被译为“混搭”或“聚集”，是一种新型的基于 Web 的资源集成应用程序。来自 Wikipedia 的定义是：Mashup 指整合网络上两个及以上外部资料来源或功能，以创造新服务的网页或者应用。Mashup 是一种集成方案，与传统资源集成方案不同，Mashup 提供了一种基于 Web 的轻量级的内容集成方法，而且由于组成 Mashup 的服务和应用本来就是面向最终用户的，所以即使没有任何编程技能，用户也可以根据需要组装出自己

的 Mashup 应用程序。

Android 是真正意义上的面向组件的编程。这些组件包括 Activity、Service、ContentProvider 和 BroadcastReceiver。这四种组件是独立的，每个组件都可以作为应用程序的入口，Android 通过它们之间的互相调用（可以是应用内的调用，也可以调用其他应用中的组件），实现指定的功能。这些组件之间的通信是由 Intent 协助完成的。图 4-3 演示了 Android 中的 Mashup 模式的设计思想。

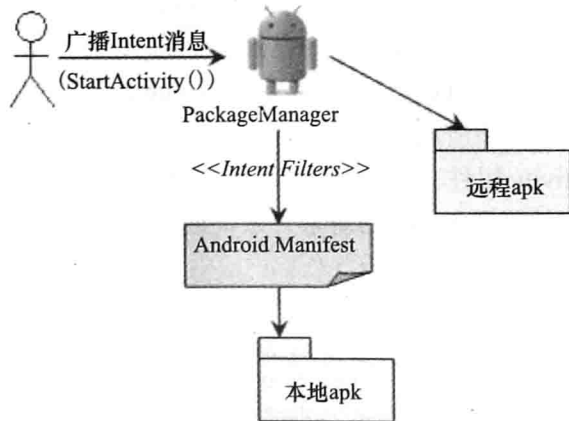


图 4-3 基于 Intent 的 Mashup 模式设计

从图 4-3 可以看出，Mashup 模式设计包括 3 个方面。

### 1. 组件

组件就是有特定功能和接口规范的实现单元。每类组件都有着不同的设计目标，或者负责界面展示（Activity），或者负责后台服务（Service），或者负责数据提供（ContentProvider），或者负责事件监听（BroadcastReceiver）。

每个 Android 组件都是一个黑盒模块，它们依照系统设计的接口和规范实现相关的功能。组件的构造、销毁等生命周期管理工作，都是由 Android 中的组件管理服务统一负责的。它了解所有组件的功能和特征，会选择合适的时机构造和销毁所需的组件。

### 2. 连接

连接是一个抽象的概念，指的是组件与组件之间的通信信道，是 Android 为不同类别的组件之间进行调用和通信预设的模式。它的实现方式根据连接两端组件类别的不同而有所变化。例如，与界面组件的通信连接，需要通过 Intent（android.content.Intent）对象来建立；而与数据源组件的通信，则通过 Uri 地址来定位并搭建连接通路。

连接的构造是由请求连接的组件、被连接的实现组件和组件管理服务共同维护的。请求连接的组件需要按照规范描述出所需组件的类型和特征；实现组件同样需要依照规范描述出它们的功能，并完成对应功能的实现；而组件管理服务会依照请求者的描述找到符合的实现

组件。

### 3. 配置

配置是用来描述组件的功能和实现特征的信息。在 Android 中，每个应用都有唯一的一个名为 `AndroidManifest.xml` 的配置文件，其中包含了该应用中所有组件的相关信息，包括组件的名称、类型、能够处理的数据格式、所需权限的信息等。

通过 Mashup 模式，可以开发出一个功能丰富的应用，而这些功能不需要用户自己来设计。例如，设计一个移动 Web UI 界面时，希望能够实现点击页面上的 E-mail 地址即可向该地址发送一封电子邮件。则在该链接地址处设计一个向 AMS (ActivityManagerService) 发送一个包含 Action 和 Data 的 Intent，系统将根据该 Intent 中包含的 Action 等信息，通过 `PackageManager` (封装了已安装应用程序的基本信息，可以通过 `getPackageManager()` 方法获得) 查询符合条件的 Activity 组件。组件可以是本应用程序包自身的，也可以来自其他 apk，并可能有多个 Activity 组件满足 Intent 的需求，系统给出用户选择处理的对话框。事件响应过程如图 4-4 所示。



图 4-4 E-mail 链接的处理模式

从上面的发送电子邮件的实例可以看出，Android 将一个应用的功能明确地封装成一个边界清晰的功能点，每一个功能点都像是一个黑盒，由预先定义的规则描述出其交互方式，实现了应用的模块化特征。同时，这些独立的模块能够在运行时，按照需求描述连接在一起，共同完成某项更大的功能，真正体现了动态性效果。

## 4.6 习题

设计一个音乐播放器应用项目。

## Widgets 设计与事件处理

### 5.1 表单控件设计

首先来看一下 QQ 登录界面的设计，显示效果如图 5-1 所示。

图 5-1 所示的 QQ 登录界面的布局（activity\_qqlogin.xml）包含 TextView、ImageView、EditText、Button、ImageButton 和 CheckBox 等基本表单控件。

#### 5.1.1 文本控件

##### 1. TextView

android.widget.TextView 是 android.view.View 类的直接子类，TextView 自身的直接子类包括 Button、EditText 等，间接子类包括 AutoCompleteTextView、CheckBox、RadioButton 等。因此，TextView 在 Android 中是一个相当重要的控件。

下面的代码是一个布局中的 TextView 的示例：

```
01 <TextView
02     android:id="@+id/TextView01"
03     android:layout_width="wrap_content"
04     android:layout_height="wrap_content"
05     android:text="@string/strAccInputLabel"
06     android:textColor="#ff3f3f3f"
07     android:textSize="16.0dip" />
```



图 5-1 QQ 登录界面

可以通过 `text`、`textColor` 和 `textSize` 等属性来设置文本及文本的颜色和字号等。如果文本过长，可以通过 `ellipsize` 属性设置文本的显示方式（如取值为 `marquee` 则以跑马灯的方式显示）。

`TextView` 最常见的方法就是通过 `getText()` 方法和 `setText()` 方法获取或设置 `TextView` 的文本。用户可以通过长按文本框选择文字，这一操作会进入文本选择模式，该模式提供对于选择的扩展以及对选中文字的操作。选择模式包括上下文操作栏和选择控制。

### （1）上下文操作栏

上下文操作栏展示了可以对选中文字进行的操作，包括剪切、复制和粘贴。如果需要的话，还可以增加更多命令。

### （2）选择控制

选择控制可以让用户调整文字选择，图 5-2 展示了这种选择。

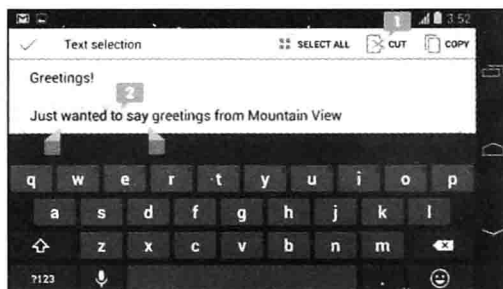


图 5-2 文本选择操作

## 2. EditText

`EditText` 是一个用于文本输入的控件，是 `TextView` 的直接子类。

常见 `EditText` 使用 `inputType` 属性为文本框指定输入类型，使用 `hint` 属性为文本框指定输入提示信息。

经常需要对 `EditText` 的输入进行监听，例如物理键盘回车的监听方法如下：

```
01 mEditText.setOnKeyListener(new EditText.OnKeyListener() {
02     public boolean onKey(View v, int keyCode, KeyEvent event) {
03         if (event.getAction() == KeyEvent.ACTION_DOWN
04             && keyCode == KeyEvent.KEYCODE_ENTER) {
05             Toast.makeText(getApplication(), "这是键盘触发的消息",
06                 Toast.LENGTH_SHORT).show();
07             return true;
08         }
09     }
10     return false;
11 }
12 });
```

软键盘回车的监听方法如下：

```
01 mEditText.setOnEditorActionListener(
02     new TextView.OnEditorActionListener() {
03         public boolean onEditorAction(TextView v, int actionId,
04             KeyEvent event) {
05             if (actionId == EditorInfo.IME_ACTION_DONE) {
06                 Toast.makeText(getApplication(), "这是软键盘触发的消息",
07                     Toast.LENGTH_SHORT).show();
08                 return true;
09             }
10             return false;
11         }
12     }
13 );
```



注意 键盘与软键盘不能同时监听。

除了上面介绍的 TextView 和 EditText, 有时还使用 TextSwitcher 控件为文本设计动态效果, 使用 AutoCompleteTextView 和 MultiAutoCompleteTextView 控件完成文本框的自动输入功能, 如图 5-3 所示。



图 5-3 自动完成输入的设计效果

### 3. 字体排版

自从 Ice Cream Sandwich 版本发布以来, Roboto 都被作为 Android 系统的默认字体集。在 Android L 版本中, 将 Roboto 做了进一步的全面优化, 以适配更广泛的平台。它变得稍宽了一点, 并进行了轻微圆化, 进一步提升了清晰度, 让阅读更加舒适, 如图 5-4 所示。

#### (1) 字体排版的缩放和基本样式

过多的字体尺寸和样式可以轻松毁掉任何一个布局。字体排版的缩放是包含了有限个字体尺寸的集合, 并且它们能够与布局结构和谐共存。最基本的样式集合就是基于 12、14、16、20 和 34 号的字体排版缩放。

这些尺寸和样式在传统应用场合中让内容密度和阅读舒适度取得平衡。字体尺寸是通过 SP (Scaleable Pixels, 可缩放像素数) 指定的, 这样可以使大尺寸字体获得更好的可接受度。图 5-5 展示了良好的字体排版和缩放效果。

#### (2) 基本色彩与色彩对比

最基本的常识是, 相同颜色的背景和文字是无法阅读的。同时, 带有过强对比度的文本也会让人炫目, 同样难以阅读。这一点在深色背景下尤其明显。

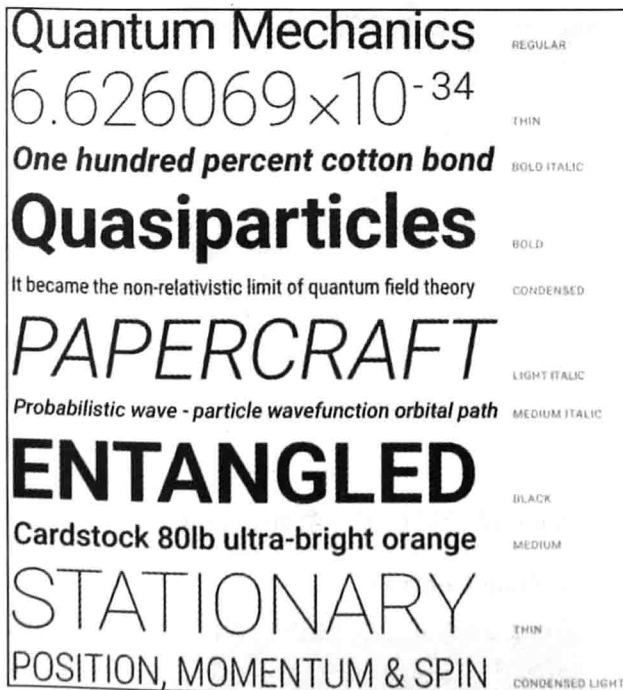


图 5-4 Roboto 字体

要获得良好的辨识度，文本应当满足最低的对比度，也就是 4.5 : 1 (依据明度计算)。7 : 1 的对比度是最适合阅读的。同样也要考虑某些非典型用户对于对比度的反应。

### (3) 每行长度包含的字符

要得到良好的阅读效果，应当在每行中包含 60 个左右的字符。每行所包含的字符数量是决定阅读舒适度的关键因素。如果每行文本过多，用户的眼睛将难以找到视觉文本的焦点。这是因为过长的文字导致用户难以判断一行的起始点。甚至，在大段文字中出现读错行的现象。如果每行文本过少，会导致眼睛来回扫视过于频繁，破坏阅读的节奏。过短的内容还会给人带来暗示，导致用户完成本行阅读前过早跳转到下一行阅读。

## 5.1.2 按钮控件

### 1. Button

Button 是一个按钮控件，用来响应用户的单击事件。对每个 Button 实例设置 `setOnClickListener()` 方法，然后使当前 Activity 实现 `OnClickListener` 接口并重载接口方法 `onClick()` 来处理按钮的响应事件。

例如，下面的代码常用于监听单个按钮的单击事件：

```
01 public class ButtonActivity extends Activity {
02     protected void onCreate(Bundle icicle) {
03         super.onCreate(icicle);
04         setContentView(R.layout.content_layout_id);
05         final Button button = (Button) findViewById(R.id.button_id);
06         button.setOnClickListener(new View.OnClickListener() {
07             public void onClick(View v) {
08                 // Perform action on click
09             }
10         });
11     }
12 }
```

Android 还提供了一个 `ImageButton`，其用法与 `Button` 类似，这里不再赘述。

### 2. ZoomControls

`ZoomControls` 是一组可缩放的控件，它包含两个按钮（“放大”按钮、“缩小”按钮），如图 5-6 所示。例如，下面的监听实现对文字字体的缩放：

```
01 zoomControls.setOnZoomInClickListener(new OnClickListener() {
02     @Override
03     public void onClick(View v) {
```

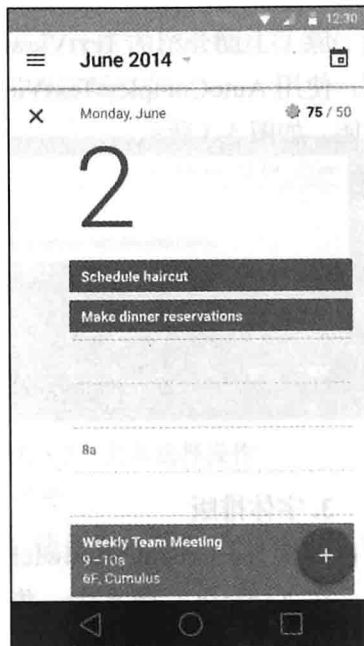


图 5-5 良好的字体排版和缩放效果



```

04         size = size + 2;
05         text.setTextSize(size);
06     }
07 });

```

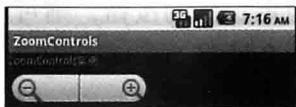


图 5-6 ZoomControls 设计效果

Android 中提供了一个缩放按钮控件 ZoomButton，用法与 Button 类似。

### 3. 按钮的设计风格

按钮由文字或者图案组成，文字或者图案必须能让人轻易地和点击后展示的内容联系起来，按钮的设计应当和应用的颜色主题保持一致。

Android 中主要有三种风格的按钮：

- 悬浮响应按钮：点击后会产生墨水扩散效果的圆形按钮，如图 5-7 所示。
- 浮动按钮：常见的方形纸片按钮，点击后会产生墨水扩散效果，如图 5-8 所示。
- 扁平按钮：点击后产生墨水扩散效果，和浮动按钮的区别是没有浮起的效果，如图 5-9 所示。

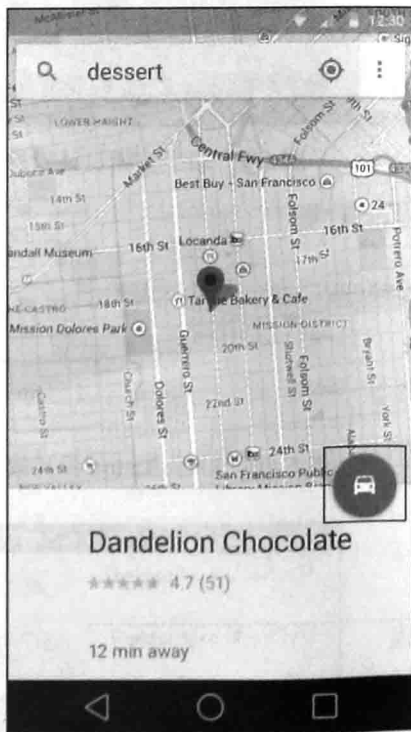


图 5-7 悬浮响应按钮设计效果

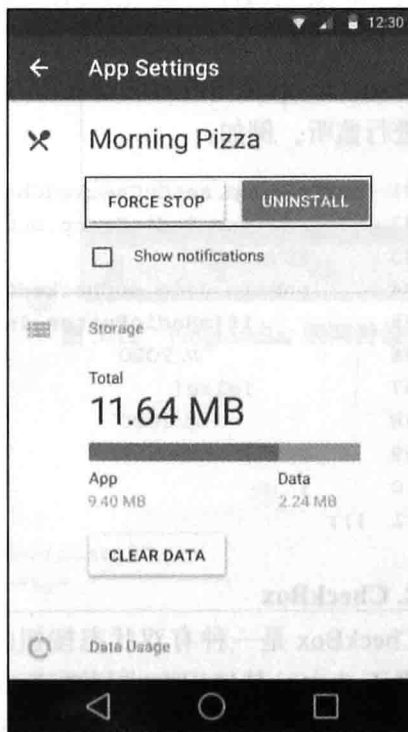


图 5-8 浮动按钮设计效果

按钮的设计原则包括：

- 对话框中使用扁平按钮作为主要按钮类型以避免过多的层次叠加。对于扁平按钮，应该在内部四周留出足够的空间以使按钮清晰可见。
- 如果需要一个对用户持续可见的功能按钮，应该首先考虑使用悬浮响应按钮。如果需要一个非主要，但是能快速定位到的按钮，则可以使用底部固定按钮。不可在底部固定按钮的区域内使用浮动按钮。

### 5.1.3 单选 / 复选按钮控件

#### 1. RadioGroup 与 RadioButton

RadioButton 是一种双状态的按钮，或处于选中状态，或处于未选中状态，如图 5-10 所示。

多个单选按钮通常与 RadioGroup 同时使用，在布局时被 <RadioGroup> 所嵌套。当一个 RadioGroup 包含几个单选按钮时，选中其中一个的同时将取消其他选中的单选按钮。

RadioButton 布局的常用属性是 checked，值为 true 表示默认被选中，值为 false 表示默认不选中。常用的方法是用 isChecked() 判断 RadioButton 是否被选中，用 setChecked() 设置 RadioButton 被选中。

RadioGroup 常用的方法就是对其内部 RadioButton 的选中进行监听，例如：

```
01 radioGroup.setOnCheckedChangeListener(
02     new RadioGroup.OnCheckedChangeListener() {
03         @Override
04         public void onCheckedChanged(RadioGroup group, int checkedId) {
05             if (mRadioButton.isChecked()) {
06                 // TODO
07             } else {
08                 // TODO
09             }
10         }
11     });
```

#### 2. CheckBox

CheckBox 是一种有双状态按钮的特殊类型，可以选中或者不选中。其使用方式与 RadioButton 类似，主要区别是可以在一组 CheckBox 中有多个被选中，图 5-11 是一个重构了的 CheckBox 示例<sup>①</sup>。



图 5-9 扁平按钮设计效果

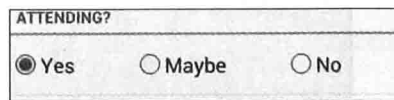


图 5-10 RadioButton 设计效果

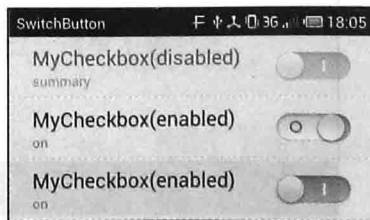


图 5-11 RadioButton 设计效果

① 项目主页 <https://github.com/Issacw0ng/SwitchButton>。

下面是 CheckBox 状态监听事件的示例：

```
01 @Override
02 public void onCheckedChanged(CompoundButton buttonView,
03     boolean isChecked) {
04     if (mCheckBox.isChecked()) {
05         ...
06     }
07 }
```

#### 5.1.4 进度条控件

ProgressBar 一般用在某项延续性工作的进展过程中，为了不让用户以为程序无响应，所以需要有一个活动的进度条，以此表示该过程正在进行中。或者是在某些操作的进度中的可视指示器，为用户呈现操作的进度。ProgressBar 有一个次要的进度条，用来显示中间进度，如流媒体播放的缓冲区的进度。一个进度条也可不确定其进度。在不确定模式下，进度条显示循环动画。这种模式常用于应用程序使用任务的长度是未知的情况下。

Android 中有两种样式进度条：长形进度条与圆形进度条（默认），如图 5-12 所示。

可以为 ProgressBar 设置如下样式：

- style="?android:attr/progressBarStyleLarge" 显示一个超大的圆。
- style="?android:attr/progressBarStyleSmall" 显示一个小型的圆。
- style="?android:attr/progressBarStyleSmallTitle" 显示一个标准型的圆。

也可以按照如下方式设置一个长型进度条：

```
01 <ProgressBar android:id="@+id/progressbar"
02     android:layout_width="wrap_content"
03     android:layout_height="wrap_content"
04     style="?android:attr/progressBarStyleHorizontal"
05     android:max="100" android:progress="50"
06     android:secondaryProgress="70" />
```

其中，android: max="100" 属性设置 ProgressBar 最大进度值为 100，android: progress="50" 属性设置 ProgressBar 初始化的进度值为 50，android: secondaryProgress="70" 属性设置 ProgressBar 初始化的底层第二个进度值为 70。显示效果如图 5-13 所示。

SeekBar 是 ProgressBar 的扩展，在其基础上增加了一个可拖动的滑块，通过拖动这个滑



图 5-12 ProgressBar 的两种设计效果

块改变 SeekBar 的数值。SeekBar 可以附加一个 SeekBar.OnSeekBarChangeListener 以获得用户操作的回调。

RatingBar 是 SeekBar 和 ProgressBar 的扩展，是 Android 平台的一种星型评级控件。



图 5-13 ProgressBar 的长型进度条设计效果

## 5.2 适配器控件设计

适配器控件是一种和适配器结合使用的控件，适配器控件通过适配器将数据集合显示在适配器控件中。常见的适配器控件包括 Gallery、Spinner、ListView、GridView 等。

### 5.2.1 适配器概述

适配器 (Adapter) 是 Android 中数据和适配器控件之间的桥梁，负责将每个数据项显示在子 View 上。Android 系统提供了如下几种适配器：ArrayAdapter<T>、BaseAdapter、CursorAdapter、HeaderViewListAdapter、ListAdapter、ResourceCursorAdapter、SimpleAdapter、SimpleCursorAdapter、SpinnerAdapter、WrapperListAdapter，比较常用的有 BaseAdapter、ArrayAdapter、SimpleCursorAdapter 等。

BaseAdapter 是 Android 中一个万能的适配器，使用 BaseAdapter 一般遵循以下步骤：

①定义 AdapterView 子 Item 的布局。

②获取数据集合。

③实现 BaseAdapter 实例，主要实现一个构造方法（主要用于获取数据集合），并重载 BaseAdapter 的如下 4 个方法：

- getCount() 方法 返回数据集合中的数据项个数；
- getItem (int arg0) 方法 返回数据集合中的一个数据项；
- getItemId (int arg0) 方法 返回数据项索引；
- getView (int position, View convertView, ViewGroup parent) 方法 返回数据项的显示视图。其中，往往通过 LayoutInflater 加载每个数据项的布局，然后将数据集合中的每个数据项的子数据元素与数据项布局中的每个控件进行绑定。

④应用适配器，一般格式为：

```
AdapterView.setAdapter (mBaseAdapter);
```

### 5.2.2 Gallery

Gallery 是一个锁定中心条目并且拥有水平滚动列表的视图，一般用于一组相同尺寸图片的显示，图 5-14 是扩展自 Gallery<sup>①</sup>的使用示例。



图 5-14 自定义 Gallery 设计效果

① 项目主页 <https://code.google.com/p/android-coverflow/>。

根据适配器控件的使用步骤，首先实现自定义的适配器，例如：

```

01 public class ImageAdapter extends BaseAdapter {
02     private Context mContext;
03     public ImageAdapter(Context context) {
04         mContext = context;
05     }
06
07     public int getCount() {
08         return imageUrl.length;
09     }
10
11     public Object getItem(int position) {
12         return imageUrl.getItem(position);
13     }
14
15     public long getItemId(int position) {
16         return position;
17     }
18
19     public View getView(int position, View convertView, ViewGroup parent) {
20         ImageView image = new ImageView(mContext);
21         image.setImageResource(imageList[position]);
22         image.setAdjustViewBounds(true);
23         image.setLayoutParams(new Gallery.LayoutParams(
24             LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
25         return image;
26     }
27 }

```

其中的 `imageList` 是包含了图像数据的集合。

然后在布局文件中定义 `Gallery`，并在 `Activity` 中通过 `findViewById()` 方法绑定该控件，例如：

```
01 Gallery gallery = (Gallery) findViewById(R.id.gallery);
```

最后为 `Gallery` 实例适配数据，并监听 `Gallery` 的事件（一般是单击事件），例如：

```

01 gallery.setAdapter(new ImageAdapter(this));
02
03 gallery.setOnItemClickListener(new OnItemClickListener() {
04     public void onItemClick(AdapterView parent, View v, int position,
05         long id) {
06         ...
07     }
08 });

```

### 5.2.3 Spinner

`Spinner` 是下拉列表框控件，提供了一种快速的选择方式。默认情况下，下拉菜单显示当

前选中的项。触摸后，显示其他可选项的下拉菜单，用户可以做出新的选择，如图 5-15 所示。

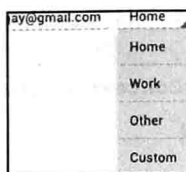


图 5-15 Spinner 设计效果

Spinner 的使用一般遵循以下过程：

①生成一个 `ArrayAdapter`，用于 Spinner 下拉列表的内容。例如：

```
01 String[] mItems = getResources().getStringArray(R.array.spinnerdata);
02 ArrayAdapter<String> adapter=new ArrayAdapter<String>(
03     this, android.R.layout.simple_spinner_item, mItems);
```

②通过 `adapter.setDropDownViewResource()` 方法设置下拉列表样式。例如：

```
01 adapter.setDropDownViewResource(
02     android.R.layout.simple_spinner_dropdown_item);
```

③使用 `Spinner.setAdapter(adapter)` 方法将数据源绑定。

④使用 `Spinner.setOnItemClickListener()` 方法响应下拉列表的选择。例如：

```
01 class SpinnerSelectedListener implements OnItemSelectedListener{
02     public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2,
03         long arg3) {
04         // TODO
05     }
06
07     public void onNothingSelected(AdapterView<?> arg0) {
08     }
09 }
```

## 5.2.4 ListView

`ListView` 是用于显示一组列表项的列表视图。`ListView` 中的列表项可以是一串文字，也可以是包含文字和图片的用户自定义的组合项。图 5-16 演示了带划屏刷新功能的 `ListView` <sup>②</sup>。

设计 `ListView` 的一般步骤如下：

①在 Activity 布局（如 `main.xml`）中添加 `ListView` 控件。

②定义列表项的布局 `list_item.xml`，用于在适配器中绑定数据项信息。

③在 Activity 中获取数据集合并。

④定义 `ListView` 的适配器。

② 项目主页 <https://github.com/johannilsson/android-pulltorefresh>。

⑤在 Activity 中绑定布局中的 ListView，并通过 setAdapter() 方法为 ListView 适配数据。

⑥为 ListView 定义 OnItemClickListener、OnItemLongClickListener 等事件。

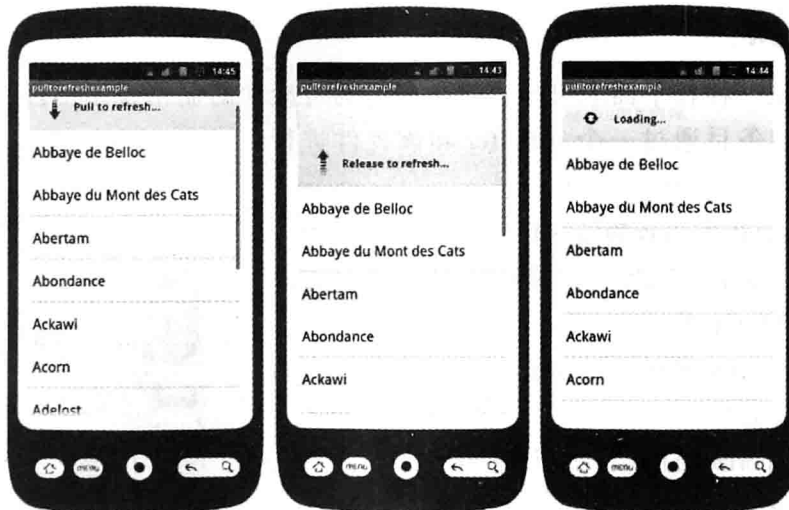


图 5-16 ListView 设计效果

ListActivity 是 Activity 的子类，用于显示一个绑定到数组或游标等数据源的列表，并且为列表的每一项提供一个单击事件的管理方法，当用户单击其中的列表项时就能进行相应的处理。

下面的代码演示了 ListView 的使用：

```
01 public class SimpleAdapterActivity extends ListActivity {
02     @Override
03     public void onCreate(Bundle savedInstanceState) {
04         super.onCreate(savedInstanceState);
05
06         SimpleAdapter adapter = new SimpleAdapter(this, getData(),
07             R.layout.main3, new String[] { "title", "info", "img" },
08             new int[] { R.id.title, R.id.info, R.id.img });
09         setListAdapter(adapter);
10     }
11
12     private List<Map<String, Object>> getData() {
13         List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
14
15         Map<String, Object> map = new HashMap<String, Object>();
16         map.put("title", "png1");
17         map.put("info", "google 1");
18         map.put("img", R.drawable.png1);
19         list.add(map);
20
21         ...
22     }
```

```

23         return list;
24     }
25 }

```

## 5.2.5 GridView

GridView 是一种在平面上可显示多个条目的可滚动的栅格视图控件，如图 5-17 所示。该控件中的条目通过一个 Adapter 和该控件进行关联。

GridView 控件的常用布局属性有：

① android:columnWidth 设置列的宽度。关联方法为 setColumnWidth (int)。

② android: numColumns 设置列数。关联方法为 setNumColumns (int)。

③ android: horizontalSpacing 设置两列之间的间距。关联方法为 setHorizontalSpacing (int)。

④ android: verticalSpacing 设置两行之间的间距。关联方法为 setVerticalSpacing (int)。

⑤ android: stretchMode 设置缩放模式。关联方法为 setStretchMode (int)。

下面的代码演示了 GridView 的使用：

```

01 public class AndroidGridLayoutActivity extends Activity {
02
03     @Override
04     public void onCreate(Bundle savedInstanceState) {
05         super.onCreate(savedInstanceState);
06         setContentView(R.layout.grid_layout);
07
08         GridView gridView = (GridView) findViewById(R.id.grid_view);
09
10         // Instance of ImageAdapter Class
11         gridView.setAdapter(new ImageAdapter(this));
12     }
13 }

```

其中 ImageAdapter 的实现过程类似于 5.2.2 节的 ImageAdapter。



图 5-17 GridView 设计效果

## 5.2.6 适配器控件的大数据加载

如果适配器控件的界面需要显示的数据项非常多，可以采用如下方式。

### 1. 分页显示

常见的分页显示模式的形式有：



- 水平分页显示，即在屏幕的底部给出分页指示器原点，通过手势水平滑动页面加载数据。
- “n/m 页”形式，如图 5-18 所示的窝窝团应用。

## 2. 拖动刷新

拖动刷新有如下两种形式：

- 通过手势下拉屏幕：在当前数据项的上方显示最新的数据，如图 5-19 所示的 Google News 应用。
- 通过手势上拉屏幕：在当前数据项的下方显示历史数据。

## 3. 垂直定位

垂直定位有如下两种常见形式：

- 在屏幕的右侧显示一个字母 A ~ Z 的索引列，通过单击索引中的字母，快速定位到目标数据项。如图 5-20 所示的联系人应用。
- 在屏幕右侧设计一个垂直滚动条，通过拖动滚动条及显示当前滚动位置数据的索引来快速定位到目标项。

## 4. 支持批量操作

常见的批量操作包括选择、添加 / 删除和重新排序。图 5-21 是文件管理器的批量操作模式。



图 5-18 窝窝团应用的分页显示



图 5-19 Google News 的下拉刷新



图 5-20 联系人应用的索引定位



图 5-21 文件管理器的批量操作模式

## 5.3 Widgets 事件处理

在用户与 UI 交互的过程中往往伴随着事件的发生。Android 平台提供了两种方式的事件处理：基于回调的事件处理和基于监听器的事件处理。基于监听器的事件处理，主要是为 Android 界面控件绑定特定的事件监听器；基于回调的事件处理，主要是重写 Android 控件特定的回调方法。5.1.2 节已经介绍了单击事件的处理方法，长按事件与单击事件类似，这里不再赘述。

### 5.3.1 按键事件处理

对按键事件的监听包括物理键盘的监听和软键盘的监听。

#### 1. 物理键盘监听

对物理键盘使用 `onKeyDown (int keyCode, KeyEvent event)`、`onKeyUp (int keyCode, KeyEvent event)` 方法来监听按键的按下和释放事件。

例如，下面的代码是对应用中“再按一次返回键退出”功能的实现：

```
01 @Override
02 public boolean onKeyDown(int keyCode, KeyEvent event) {
03     if (keyCode == KeyEvent.KEYCODE_BACK
04         && event.getAction() == KeyEvent.ACTION_DOWN) {
05         if (System.currentTimeMillis() - exitTime > 2000) {
06             Toast.makeText(this, "再按一次退出程序",
07                 Toast.LENGTH_SHORT).show();
08             exitTime = System.currentTimeMillis();
09         } else {
10             finish();
11             System.exit(0);
12         }
13         return true;
14     }
15
16     return super.onKeyDown(keyCode, event);
17 }
```

Android 系统对 Home 键的监听进行了屏蔽，有兴趣的读者可以参阅博客 <http://www.iteye.com/topic/1116709> 实现对 Home 键的监听。

#### 2. 软键盘监听

Android 是一个针对触摸屏专门设计的操作系统，单击编辑框，系统即自动为用户弹出软键盘，以使用户进行输入。

对软键盘的监听包括软键盘的显示 / 隐藏监听，以及软键盘按键的监听。

##### (1) 显示 / 隐藏监听

默认的，单击 `EditText` 时自动弹出软键盘。下面的代码可以在显示软键盘时关闭软键盘：

```

01 InputMethodManager imm = (InputMethodManager) getSystemService(
02     Context.INPUT_METHOD_SERVICE);
03 if (imm.isActive()) {
04     imm.toggleSoftInput(InputMethodManager.SHOW_IMPLICIT,
05         InputMethodManager.HIDE_NOT_ALWAYS);
06 }

```

## (2) 软键盘按键监听

使用 `dispatchKeyEvent (KeyEvent KEvent)` 方法对软键盘的按键进行监听。例如：

```

01 @Override
02 public boolean dispatchKeyEvent(KeyEvent KEvent) {
03     int keyaction = KEvent.getAction();
04
05     if (keyaction == KeyEvent.ACTION_DOWN) {
06         int keycode = KEvent.getKeyCode();
07         int keyunicode = KEvent.getUnicodeChar(KEvent.getMetaState());
08         char character = (char) keyunicode;
09
10         Toast.makeText(this,
11             "DEBUG MESSAGE KEY=" + character + " KEYCODE=" + keycode,
12             Toast.LENGTH_SHORT).show();
13     }
14
15     return super.dispatchKeyEvent(KEvent);
16 }

```

5.1.1 节介绍了对软键盘回车事件的监听。

## 5.3.2 触屏事件处理

Android 系统中，触屏事件通过 `onTouchEvent(MotionEvent event)` 方法进行监听。例如，如下的代码监听了单点触屏事件：

```

01 @Override
02 public boolean onTouchEvent(MotionEvent event) {
03
04     int action = event.getAction();
05     mPosX = (int) event.getX();
06     mPosY = (int) event.getY();
07     switch (action) {
08         case MotionEvent.ACTION_DOWN:
09             Log.i("test", "ACTION_DOWN");
10             break;
11         ...
12     }
13
14     return true;
15 }

```

多点触屏也是使用 `onTouchEvent (MotionEvent event)` 方法进行监听的，在监听时，

通过 Event 的 `getPointerCount()` 获取触屏点个数, 然后进行多点触屏事件处理 (参见示例 `EventListenerProject/MultiTouchEventActivity`)。图 5-22 演示了多点触屏的设计效果。



图 5-22 多点触屏设计效果

如果一个 Activity 调用一个 View, 那么首先执行的是 View 中的 `onTouchEvent` (`MotionEvent event`), 如果返回 `false`, 再执行 Activity 中的 `onTouchEvent` (`MotionEvent event`), 否则不执行 Activity 中的 `onTouchEvent` (`MotionEvent event`)。如果需要实现手势监听, 那么在触屏事件之后需要返回 `mGestureDetector.onTouchEvent(event)`。

5.3.3 手势事件处理

手势操作是触屏设备独特的交互模式，Android 平台提供了如表 5-1 所示的手势操作。

表 5-1 Android 中的手势

手势类别	手指操作	图示	实现功能
触摸	按下，抬起		触发项目的默认操作
长按	按住，等待，抬起		进入选择模式。使用户可以选择视图中的单个或者多个项目，并选择上下文操作栏的功能。取代了长按唤出上下文菜单的功能
滑动	按住，移动，抬起		<ul style="list-style-type: none"><li>• 横向滑动：实现平级切换、返回首页，拉出附属功能，开关、滑块、删除等</li><li>• 纵向滑动：实现下拉刷新，底部加载更多、全屏、上下篇切换、返回上一级，拉出附属功能等</li></ul>
拖动	长按，移动，抬起		重新排列视图中的数据或者将数据移动到容器（例如主屏幕上的目录）中
双击	快速连续两次触摸		放大内容。在文字选择中作为辅助手势
双击拖动	点击松开后立刻再次按住屏幕，上下拖动： <ul style="list-style-type: none"><li>• 向上滑动放大区域内容</li><li>• 向下滑动缩小区域内容</li></ul>		放大或者缩小点击区域的内容
放大	用两个手指按住，向相互远离的方向移动，抬起		放大内容
缩小	用两个手指按住，向相互接近的方向移动，抬起		缩小内容

1. GestureDetector

Android 系统提供了 GestureDetector 类，将基本的触屏事件转变成各种不同的手势动作。通过实现 GestureDetector.OnGestureListener 接口来构造一个 GestureDetector 的实例。如果仅仅只是想处理少部分的手势动作，可以选择继承 GestureDetector.SimpleOnGestureListener 而不用去实现 GestureDetector.OnGestureListener 接口。

GestureDetector 主要回调方法包括:

- onDown (MotionEvent e) 用户轻触单击触摸屏, 由一个 MotionEvent.ACTION\_DOWN 触发。
- onShowPress (MotionEvent e) 用户轻触单击触摸屏, 尚未松开或拖动, 由一个 MotionEvent ACTION\_DOWN 触发。注意和 onDown() 方法的区别, onShowPress 方法强调的是没有松开或者拖动的状态, 也就是说, 如果按下后立刻松开或者拖动, 是不会触发的。
- onSingleTapUp (MotionEvent e) 用户轻触单击触摸屏后松开, 由一个 MotionEvent.ACTION\_UP 触发。只有在单击后立刻松开才会触发, 长按滑动等操作后松开都不会触发。
- onFling (MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) 用户按下触摸屏、快速移动后松开触发, 由一个 MotionEvent ACTION\_DOWN、多个 MotionEvent .ACTION\_MOVE 和一个 ACTION\_UP 触发。e1 是第 1 个 MotionEvent.ACTION\_DOWN, e2 是最后一个 MotionEvent.ACTION\_MOVE, velocityX 是 X 轴上的移动速度 (像素 / 秒), velocityY 是 Y 轴上的移动速度, 触发条件是 X 轴的坐标位移大于 FLING\_MIN\_DISTANCE, 且移动速度大于 FLING\_MIN\_VELOCITY 个像素 / 秒。
- onLongPress (MotionEvent e) 用户长按触摸屏, 由多个 MotionEvent ACTION\_DOWN 触发。
- onScroll (MotionEvent e1, MotionEvent e2, float distanceX, float distanceY) 用户按下触摸屏, 拖动时触发, 由一个 MotionEvent ACTION\_DOWN、多个 ACTION\_MOVE 触发。

例如, 下面的代码通过滑动屏幕实现图片的分页浏览:

```

01 public class GestureEventActivity extends Activity
02     implements OnGestureListener {
03     private GestureDetector detector;
04     private ViewFlipper flipper;
05     ...
06     @Override
07     public void onCreate(Bundle savedInstanceState) {
08         super.onCreate(savedInstanceState);
09         ...
10         detector = new GestureDetector(this);
11     }
12
13     @Override
14     public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
15         float velocityY) {
16         if (e1.getX() - e2.getX() > 120) {
17             flipper.setInAnimation(AnimationUtils.loadAnimation(this,

```

```

18         R.anim.push_left_in));
19         flipper.setOutAnimation(AnimationUtils.loadAnimation(this,
20             R.anim.push_left_out));
21         flipper.showNext();
22     } else if (e2.getX() - e1.getX() > 120) {
23         flipper.setInAnimation(AnimationUtils.loadAnimation(this,
24             R.anim.push_right_in));
25         flipper.setOutAnimation(AnimationUtils.loadAnimation(this,
26             R.anim.push_right_out));
27         flipper.showPrevious();
28     }
29     return false;
30 }
31
32 }

```

## 2. 手势设计技巧

优秀的手势操作是那些可以让用户明确知晓如何使用，并且用户能知道它是否可用的手势。一个开发者不能在自己的小圈子里设计应用，只想着自家应用中的手势体验，还应该跳出自己的应用，去思考如何能做一个适配大多数应用的全局手势，这样设计出来的手势操作不仅可以在自家的应用中适配良好，而且还会引起其他应用的效仿，甚至能改变大多数应用的界面设计，减少手势碎片化。

下面介绍一些在手势设计时应该注意的事项。

### (1) 减少无用手势

手势操作似乎还没有像按钮操作或菜单操作那样被大大方方地搬上台面，手势有着不可见性、抽象性、需记忆性等特点。用户界面并不告诉用户可以对某个对象做什么，于是用户需要记住可以使用哪些手势。如果用户对其缺乏认知，这些手势操作就不易被发现，也便不会为人所用，甚至造成操作障碍。因此，在应用中不要滥用手势。

### (2) 善用区域性手势

如果一个手势不能在整个应用界面中使用，如新浪微博中的后退手势。那么就应该使用区域性手势，例如在导航栏区域内右滑动可以返回上个界面，左滑动回到上次浏览的界面。因为在微博、微信等大多数应用中都有导航栏，这样的区域性手势可以解决在长微博和评论界面中手势无法通用的问题，同时这个返回手势可以被很多应用借鉴使用，保证了用户在多应用中的统一体验。

### (3) 注重浅性引导

用户刚刚上手应用时是无法预知该应用的手势操作的，所以在应用初始界面中加以介绍说明也是明智之举，但无论你的应用中有多少手势和隐藏操作，在初始的帮助中只能说明几个最重要的，剩余的就交给浅性引导了，主要包括如下几种方式：

- 露出功能的某一部分，提示前后、上下有隐藏的内容或操作。
- 到达界限值时给出文字、动画提示等，如拉到底部时提示向下拖动清除，拉到顶部时

提示返回上一级。

- 进入具有手势操作的界面时给出气泡、动画、图片等提示。
- 通过拟物的方式提示可以通过手势进行操作。如拉绳状、开关状、滑块状、掀起页角状等。

#### (4) 提供反馈

由于手势的不可见性和误操作性，除了在操作前进行引导外，在操作后也应提供一定的反馈。反馈方式可以采取声音、抖动等。例如，在点击屏幕时，系统会立即在交互的触点上绘制出一个可视化的触控涟漪（触控涟漪是触摸效果的核心视觉机制，如在点击屏幕时，出现的类似于墨水扩散那样的视觉效果）让用户感知到。为避免误操作而二次确认也有一定的必要，特别是较重要的手势，例如删除等，要让用户可以及时撤销操作。

### 5.3.4 感应器事件处理

Android 平台是通过使用硬件传感器创建创新应用程序的理想平台。大多数的 Android 设备都有内置的测量运动、方向和各种环境条件的传感器。这些传感器具有提供高精度和准确度的原始数据的能力，可用于监视设备在三维方向的移动和位置、或者监视设备周围环境的变化。

Android 平台下的传感器是通过监听机制来实现的，一般步骤如下：

#### ①创建 SensorManager 对象。

通过调用 `Context.getSystemService()` 方法传入参数 `SENSOR_SERVICE` 来获得 `SensorManager` 对象。通过 `SensorManager` 可以访问手持设备的传感器，同时该对象还提供了一些用于对捕获的数据进行计算处理的方法。

#### ②实现 SensorListener 接口。

这是开发传感器应用中主要的工作，实现 `SensorListener` 接口主要需要实现以下两种接口方法。

- `public void onAccuracyChanged (Sensor sensor, int accuracy)`

该方法在传感器的精确度发生变化时调用，`SensorManager` 提供了 4 种精确度，由高到低分别为：`SENSOR_STATUS_ACCURACY_HIGH`、`SENSOR_STATUS_ACCURACY_MEDIUM`、`SENSOR_STATUS_LOW` 和 `SENSOR_STATUS_UNRELIABLE`。

- `public void onSensorChanged (SensorEvent event)`

该方法在传感器的数据发生变化时调用，开发传感器应用的主要业务代码应该放在这里执行，如读取数据并根据数据的变化进行相应的操作等。

#### ③注册 SensorListener。

在这里调用步骤 1 中获得的 `SensorManager` 对象的 `registerListener (SensorEventListener listener, Sensor sensor, int rateUs)` 方法来注册监听器，其接收的参数为监听器的对象、传感器类型、传感器事件传递数据延时。



取消注册 `SensorListener` 时调用 `SensorManager` 的 `unregisterListener (SensorEventListener listener)` 方法。一般来讲,注册和取消注册的方法应该成对出现,如果 `Activity` 的 `onResume()` 方法中注册 `SensorListener` 监听,就应在 `onPause()` 方法中取消注册。

以下代码展示了如何用 `onSensorChanged()` 方法来监控光线传感器传回的数据,并将原始数据显示在一个 `TextView` 中:

```

01 public class SensorActivity extends Activity
02     implements SensorEventListener {
03     private SensorManager mSensorManager;
04     private Sensor mLight;
05
06     @Override
07     public final void onCreate(Bundle savedInstanceState) {
08         super.onCreate(savedInstanceState);
09         setContentView(R.layout.main);
10
11         mSensorManager = (SensorManager) getSystemService(
12             Context.SENSOR_SERVICE);
13         mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
14     }
15
16     @Override
17     public final void onAccuracyChanged(Sensor sensor, int accuracy) {
18         // 当传感器精度发生变化时处理响应事件
19     }
20
21     @Override
22     public final void onSensorChanged(SensorEvent event) {
23         // 利用光线传感器返回的返回值处理响应事件
24         float lux = event.values[0];
25     }
26
27     @Override
28     protected void onResume() {
29         super.onResume();
30         mSensorManager.registerListener(this, mLight,
31             SensorManager.SENSOR_DELAY_NORMAL);
32     }
33
34     @Override
35     protected void onPause() {
36         super.onPause();
37         mSensorManager.unregisterListener(this);
38     }
39 }

```

## 5.4 Widgets 设计技巧

本节主要介绍 Widgets 控件的设计技巧。

## 5.4.1 官方设计指引

以下是根据官方设计指引给出的 Widgets 设计建议。

### (1) 按钮状态

一般来说,按钮会有四种状态:不可点击状态、可点击状态、聚焦状态、按下状态。如果按钮此时处于不可用状态,那么一定要灰掉,或者隐藏按钮,否则会误导用户。

### (2) 菜单层次

菜单项以 5 ~ 7 个为宜,如果有二级菜单,就要注意合理的菜单分类,不能有太多层级的菜单,否则很难预期,也很难找到,寻找和返回都会变得很麻烦。

### (3) 文字长度

手机界面很小,每页只能显示 6 ~ 10 个列表,每行只能显示 10 ~ 16 个字,标题栏的字数以 5 个以内为宜,标签栏也以 2 ~ 3 个为宜。如果出现文字过长的情况,一定要定义一下处理方式,如果是选择型的,一般是截断或者打点缩略;如果是内容阅读型的,可以折行。但最合理的方式还是精简文字内容,缩短文字长度。

### (4) 文字表意

由于手机适用于碎片时间的片段式阅读方式,所以对手机界面上的文字表意性要求更高、更苛刻,一定要在用户看到的瞬间即可,准确地传达信息。除了表意清晰之外,还要求语言精简,避免啰嗦;使用用户的语言而不是程序的语言;产品文案体现产品性格。

### (5) 交互流程

做交互时一定要将任务流程的概念贯穿始终,用户是为了完成某个任务而使用软件的,交互设计师除了关注界面元素、跳转逻辑和交互反馈之外,还要关注用户任务,分得清主要任务和次要任务,给主要任务一个畅通无阻的清晰流程,不要给予太多可能的分支,而干扰主要流程。

### (6) 相关选项

相关选项一定要具有操作上的延续性,虽然手机屏幕看起来比电脑屏幕小得多,但是手机在屏幕上移动的代价却要比鼠标在电脑上移动的代价大得多,如果手机上相关选项离得很远,用户一是容易迷失,找不到下一步操作,二是需要移动手指,到屏幕另一端触发操作。

### (7) 数据载入

流量、电量、速度和稳定性是手机产品的四个硬指标,如果应用不能合理地帮助用户节约流量、电量,提升浏览速度和浏览体验,保证应用的稳定性能,就不要谈什么用户体验。可以利用预加载缓存、批量载入、动态刷新、服务端数据压缩等方式来保证“省、快、稳”基础体验。

### (8) 可点击范围

移动端有个神奇的数字“44”,根据食指的最小点触距离 7mm 和拇指的最小点触距离 9mm,可以推导出做设计时最小的点触距离是  $44 \times 32\text{px}$ 。可以设计一个精美的小图标,但是在定义它的点触大小时,却可以做放大处理,但千万不要设计一个过大的图标,而使点触范

围比图标还小，这样会给用户带来明显的误操作挫败感。

#### （9）标签页

标签页与内容需要有很好的联动关系，一般一个界面内有二级标签就足够复杂了，千万不要再有三级标签、四级标签。每个标签页都有自己特有的内容，当切换标签时，内容跟着切换。如果标签页是点击切换，内容部分可以整体刷新，如果标签页是滑动切换，内容页也要跟着滑动切换，千万不要一个点、一个滑。

#### （10）隐喻功能

手机产品交互设计要经历缩减、隐藏、附加、组织的过程，千万不要妄图把什么功能、什么操作都暴露出来，以彰显强大。需要把自己应用的所有功能及操作做个优先级设定，将那些常用的 20% 的功能放在界面的主要位置上，其他 80% 的操作放在次要位置，或合理归类组织后隐藏起来。

#### （11）空数据界面

应用刚推出时，是没有用户的，因此也没有数据；应用中的新注册用户也是没有数据的；应用中的用户可以清除个人数据。这三种情况下，用户都可能遇到空数据的界面。提倡的做法是提供一个情感化的界面，告诉用户当前没有内容；更具引导性的做法是引导用户去执行创建内容的操作。

#### （12）用户引导

设计人员都喜欢用漂亮的引导界面告诉用户新增的功能或隐藏的应用，但不是所有的应用及功能都需要花哨的引导。如果是通用的功能或非重点的模块就不需要引导；如果是功能告知则只需要轻量级的引导；如果是版本更新说明，说明书式的引导可以采用，但是要言简意赅。

#### （13）加载等待

手机产品只要是需要联网、交换数据，就都需要提供一个加载中状态，无论是旋转还是 Toast 对话框，需要给开发人员一个全局的定义，并且要告知加载中是模态（前台加载）的还是非模态的（后台加载）。且要考虑发生加载时间过长、网络开关没有打开、网络不通等情况时分别如何处理。

#### （14）返回按钮

在为 Android 做设计时，会涉及硬件交互，其中返回键的使用很重要，可以借鉴 Android 官方的一些指导原则，但是在具体开发时，还是会有很多特殊情况，例如单一实例的替换、键盘及一些中间状态。这时，返回键可能需要被定义一下，是回到前一个 Activity 实例（那就需要变成多实例了）还是该回到控件的初始状态（清空输入内容或恢复初始状态）。

#### （15）横屏模式

由于在横屏模式下，纵向空间变得格外宝贵，导航栏、标签栏、键盘都需要被压扁，横屏模式一定要考虑是简单拉伸适配还是重新设计。如果应用不适合在横屏模式下使用，就屏

蔽横盘，如果应用以应用的桌面组件都需要支持横盘模式（甚至是带侧滑键盘的横屏机器），就需要提供设计方案，甚至需要重新设计。

### 5.4.2 表单控件设计技巧

用户在体验产品时，经常看到错误的表单设计，信息混乱、步骤繁复、语言程序化、视觉焦点跳跃、校验顺序混乱、反馈不及时等都是常见的表单设计错误。本节给出一些表单设计建议，使开发者能最大限度地提升表单设计的体验，提升效率，提高满意度。

#### 1. 清晰的视觉纵线

用户在浏览信息时，如果没有足够多的强调元素，则会从上至下、从左至右地浏览，Web 端是一个“F”型视线，移动端更经常是“L”型视线（导航和重要操作往往在下边），那么如果表单的视觉浏览顺序符合这个“L”型规律，基本上就符合用户的心理预期，不需要任何的寻找及思考，就可以简单高效地执行完表单项的填写和提交。

图 5-23 是糯米应用的用户登录界面，在用户输入完用户名和密码之后，直接看到的操作按钮是“注册”，而不是“登录”。这种左右的布局方式，即使用颜色区隔，也阻挡不了用户的视线落到注册上，于是通过简单的眼动测试就可以发现，这时用户盯着“注册”按钮停顿思考一下是在所难免的。

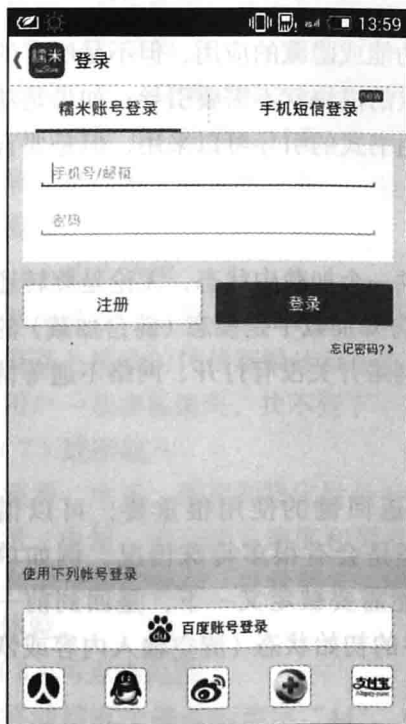


图 5-23 糯米应用的登录界面

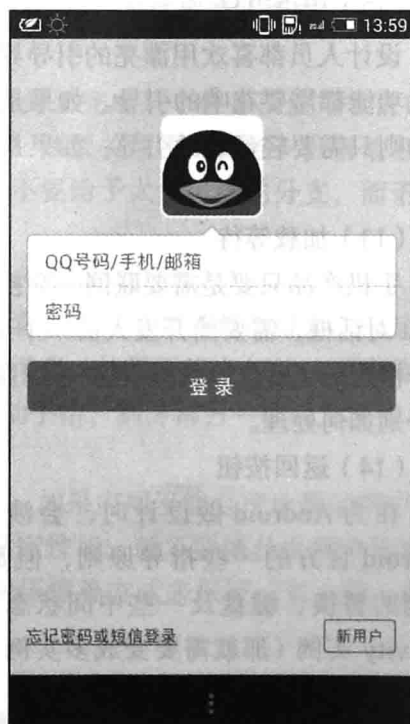


图 5-24 QQ 应用的登录界面

图 5-24 是 QQ 应用的用户登录界面，用户输入完 QQ 号和密码之后，直接看到的操作

按钮是一个和界面等宽的“登录”按钮。这是一个登录表单视觉纵线的例子，用户先关注到QQ号输入框，再关注到密码输入框，然后就自然而然地发现了“登录”按钮。类似这种登录界面的布局有很多，例如新浪微博、腾讯微信、百度云等。

## 2. 信息的分组

表单项并不是从上边罗列到下边就可以了，而是要经过信息组织的。同一类的表单可以放在一起，当表单太长时，可以拆分成不同的组，这样用户在填写时，类似于一种任务拆解，可以一组一组地完成填写。

分组的方式有两种：一种是以折叠列表的方式显示相关分组（如图 5-25 所示），一种是以分屏的方式显示相关分组（如图 5-26 所示）。

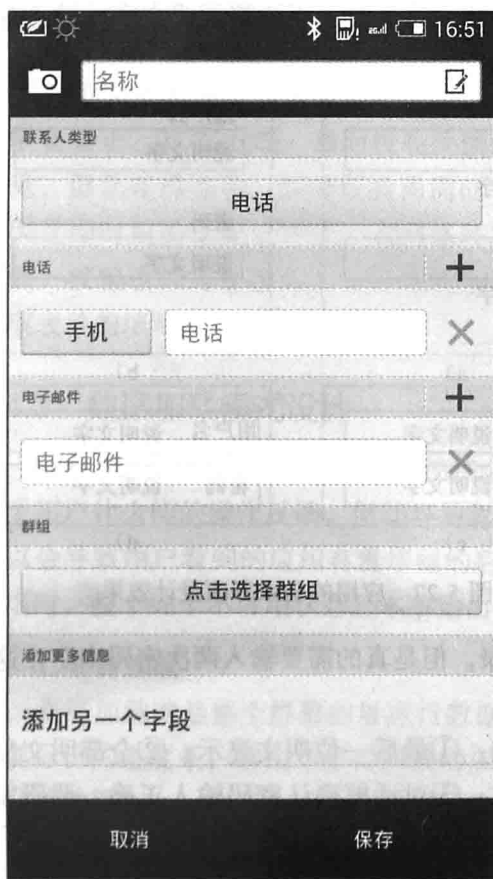


图 5-25 联系人应用的添加联系人界面

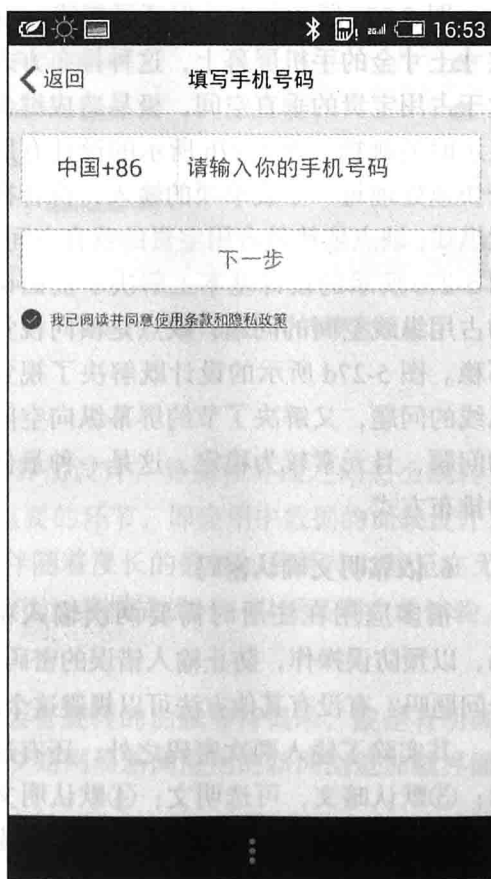


图 5-26 QQ 应用的注册界面

## 3. 极致的减法

表单中最好不要添加不需要的信息。那些确实需要，但是使用频率不高的信息可以隐藏，通过某个入口进行添加。例如，图 5-25 所示的联系人群组信息，不是每个联系人都一定要进行分组，因此，在表单界面通过添加接口来实现分组，有效地精简了表单界面。

#### 4. 利用选择代替输入

移动 APP 的输入成本非常高，尤其是触屏，输入效率和输入准确率都有很大的提升空间，在这种情况下，要尽量减少需要输入的内容，利用选择代替输入。例如性别、出生日期、城市等输入项，都是可以通过选择的形式来提交内容的。

在提供输入建议时，也是可以利用选择来代替输入的。例如当用户名已被注册时，系统提供几个用户名以供选择；在给自己打标签时，系统提供一些常用标签以供选择等。

#### 5. 标签及提示文字的排版方式

手机用户界面的空间非常有限，但是表单项往往需要通过标签告知表单类型，通过提示文字告知表单格式，那么标签及提示文字怎样排布才可以使信息呈现最友好呢？

图 5-27a 所示的设计保证了视线一直是纵向向下的，输入时，不遮挡说明文字；缺点是在寸土寸金的手机屏幕上，这种排布方式过于占用宝贵的垂直空间，极易造成键盘显示时的遮挡。图 5-27b 所示的设计有助于快速处理每一个表单项的输入，符合视觉纵线；缺点依然是占用宝贵的垂直空间。图 5-27c 所示的设计基本上解决了前面说的占用纵线空间的问题；缺点是横向视觉不稳。图 5-27d 所示的设计既解决了视觉纵线的问题，又解决了节约屏幕纵向空间的问题，且元素较为稳定，这是一种最佳的排布方式。

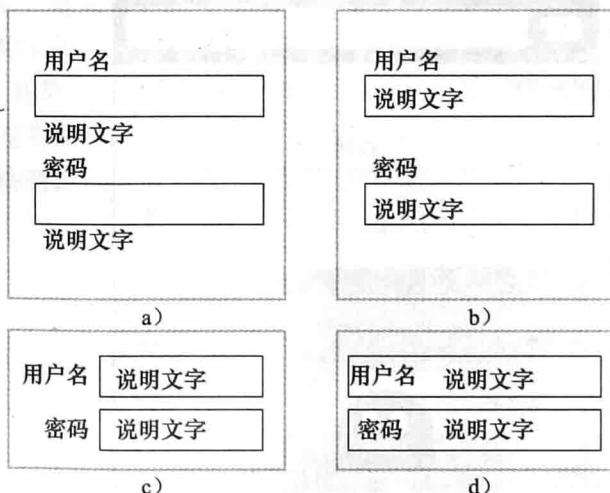


图 5-27 应用的登录界面设计效果

#### 6. 依靠明文确认密码

很多应用在注册时需要两次输入密码，以防误操作，防止输入错误的密码之后无法登录。但是真的需要输入两次密码来防止这个问题吗？有没有其他方法可以规避这个问题呢？

其实除了输入两次密码之外，还有这样几种办法：①最后一位明文显示；②全部明文显示；③默认暗文，可选明文；④默认明文，可选暗文；⑤对话框确认密码输入正确。调研发现，默认明文，可选暗文形式的接受度最高。

#### 7. 支持滑块输入

在 Android 原生系统中，滑块已经成为标准数据输入控件。滑块非常直观，对特定范围数据的输入非常适用，并具有外形美观、易于操作、占据屏幕空间不多等特点。图 5-28 是 Trulia 应用的搜索面板。

#### 8. 合理的键盘利用

不同的文本框类型可以调用不同的键盘。例如网址输入框，调用网址输入键盘可以方便



快捷地输入 .com；纯数字的输入框可以调用数字键盘；电话号码输入框可以调用电话号码键盘，除了数字之外，还有 \*、#、+、；、姓名等中文输入框可以调用中文键盘；邮箱输入框可以调用邮箱键盘，方便输入 @。

另外，键盘上右下角的功能键是可以被定义的，该功能键在填写表单时，与 PC 上的 Tab 键有点像，应该起着向下切换表单项的作用，当处于最后一个表单项时，该功能键就要变成对应的操作。

### 9. 即时的校验反馈

一种理想化的情况就是当输入完一个表单项时，系统可以立刻告诉用户这一项填写是否正确，而不是填完所有表单，提交之后，才告诉用户哪里需要修正。在 Web 端，即时校验反馈已经非常常见，但是在移动端，即时校验尚需时日。主要原因是即时校验受限于网速，当网络环境不好的时候，校验也许需要很久，会影响正在进行的下一项表单的填写。

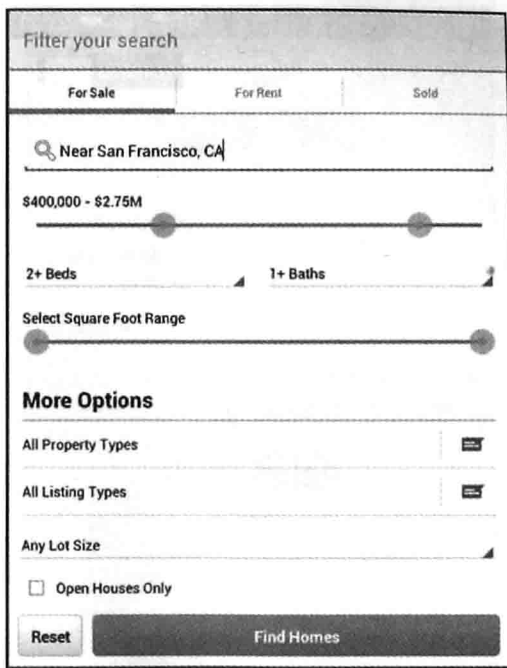


图 5-28 Trulia 应用的搜索面板

## 5.4.3 数据加载模式设计

设计师在进行应用的设计时，往往会更加专注于界面设计，界面和界面之间怎么跳转，给予用户什么样的操作反馈，但却容易忽略一个比较重要的环节，即应用中数据的加载设计，所以会导致用户看到的应用有着华丽的启动界面，却伴随着漫长的数据加载等待，甚至在没有网络时，整个处于不可用状态。本节给出一些常见的数据加载模式设计，以提高用户的体验。

### 1. 全屏加载

全屏加载就是整个屏幕白屏进行数据加载，一般会有旋转的加载等待提示，或是有明确进度标识的进度条，常用于手机网页的加载中。图 5-29 是网易新闻应用的新闻浏览加载界面的设计效果。

全屏加载的优点是能保证内容的整体性，全部加载完才能够系统化的阅读。缺点是有着非常强烈的等待感，3 秒以上的等待时间会使用户产生焦躁情绪。

### 2. 优先加载

如果一个页面有图片也有文字，在加载图片比较慢的情况下，可以先把文字都加载出来，从而保证用户可以顺畅阅读，然后再加载比较费流量的图片。如果使用这一种加载方式，则千万不能将重要信息全部放在顶层的图上，因加载不出来而导致无法继续操作。重要的操作也不能用图片按钮，否则也会有操作显示不出来的风险。图 5-30 所示的网易新闻应用

的阅读界面就采用了优先加载模式。

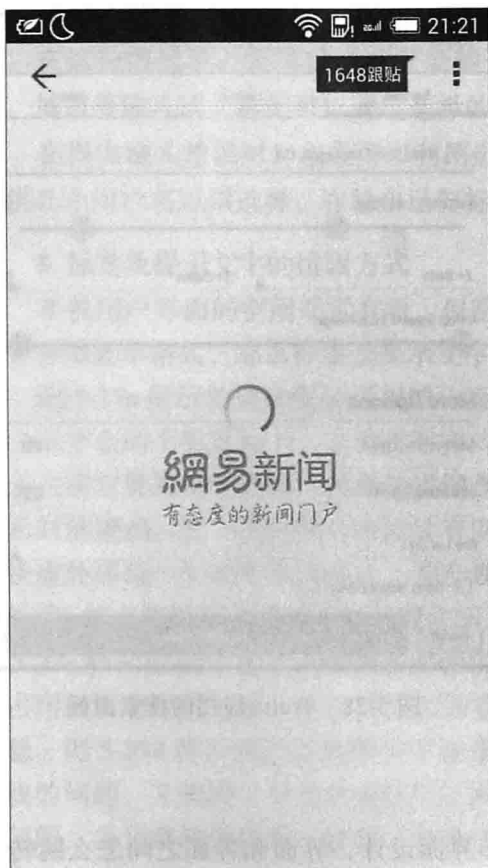


图 5-29 网易新闻浏览页面

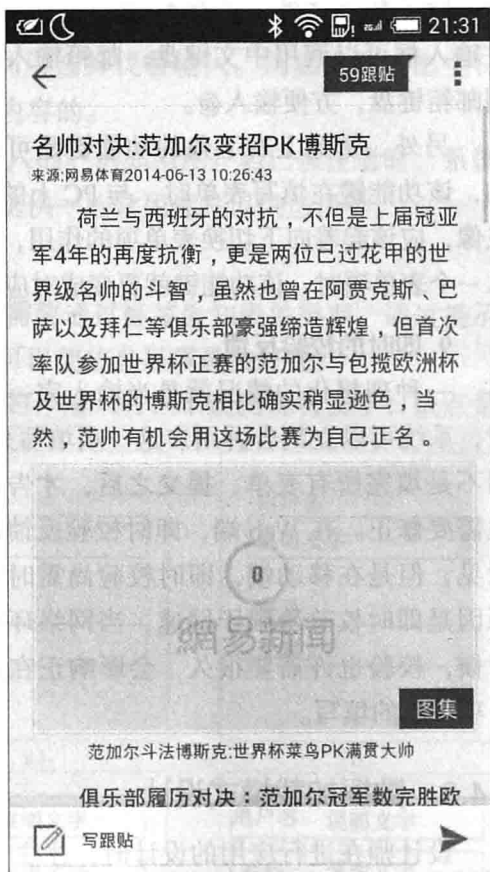


图 5-30 网易新闻浏览的优先加载

优先加载的优点是可以帮助用户快速阅读内容，了解信息。缺点是可能会丢失掉重要的关键信息，无法建立信息获取的闭环。这种加载模式更加适用于内容阅读型的手机应用。

### 3. 整页加载

如果当前页与下一页是整页切换，可以考虑采用整页加载的形式，但是要保证每个页面的数据量不是特别大。

整页加载的优点是能保证每个页面的完整性，体验比较整体。缺点是难以保证整页的加载效率，且有可能影响浏览的流畅度。这种加载模式一般适用于宫格图片模式、全屏图片模式、网状详情页模式等。

### 4. 自动加载

自动加载适用于长列表的情况，可以设定规则，例如默认加载 20 条，当滚动到第 20 条时，自动再加载 20 条。用这种方法可以营造一种无极限浏览的错觉，很容易吸引用户。图 5-31 是新浪微博的加载设计效果。



自动加载的优点是把用户代入无尽浏览模式,让用户一直向下滚动,不需要手动点击下一页。缺点是没有尽头,容易迷失,不方便快速索引定位到某个内容。这种模式适用于瀑布流、长列表、商品列表等情况。

### 5. 智能加载

当用户处于 Wi-Fi 网络环境时,不会受限于流量和访问速度,可以加载大图片、大图标,甚至直接播放视频动画。但是如果用户处于非 Wi-Fi 的网络模式下,则需要差异化地处理成小图或者无图模式,视频和动画直接用一个占位符标识,这种根据网络状况智能调整的加载方式叫做智能加载。

例如淘宝应用,当网络切换到 GPRS 或 3G 的时候,首先会提示用户网络发生变化,然后查看商品详情时,图片从自动下载变换成点击加载,并且加载的只是精简版图文详情,防止多图造成的流量浪费。

智能加载的优点是根据具体场景来控件流量和加载速度。缺点是不一定真实有效地命中用户需求,所以还是需要给予用户一定的查看详情的入口,或者是设置项。这种模式适用于有大量图片或视频的应用,如电子商务类或在线视频类的应用。

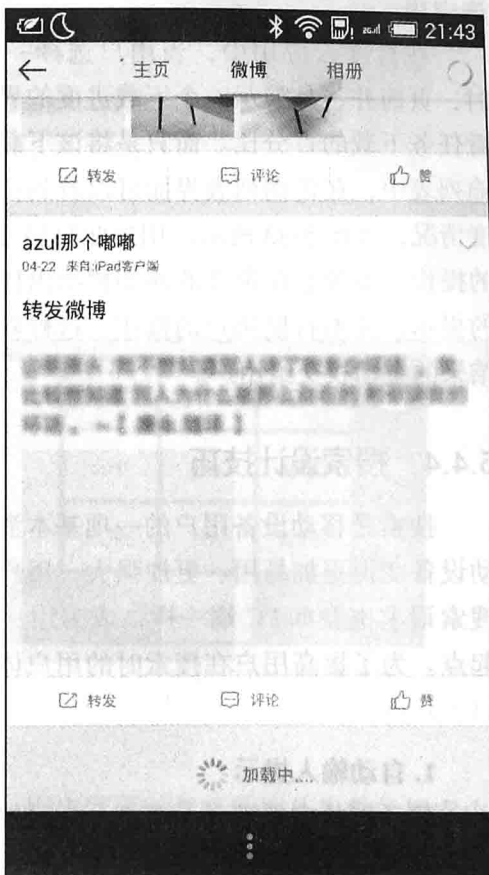


图 5-31 新浪微博的加载设计

### 6. 离线加载

当用户没有连接网络时,很多功能往往就不能用了,内容也无法加载,导致应用变得根本不可用,这时就要考虑“预加载+离线缓存”的设计了。首先在连接网络时将数据提前加载并缓存到本地,没有网络时直接加载已经缓存的内容。一般情况下用户可以选择是否开启 Wi-Fi 下的预加载功能,或者是否开启 Wi-Fi 下全部离线缓存的功能。这样就能保证在网络不稳定时也能加载完整的信息。图 5-32 是今日头条应用的侧向菜单,其中的“离线”选项用于设置在无网络情况下加载哪些内容。

离线加载的优点是解决了没有网络时获取数据的问题,并且节约了流量,保证了流畅。缺点是占用本地存储空间,而且有时候预加载的内容根本没有用到。这种模式适用于小说阅读、新闻阅读和视频类应用。

### 7. 支持异步任务

移动互联的核心就是为用户带来移动体验的方便和高效,这是移动互联网应用需要考虑的。所以在设计上应尽量让用户在短时间内熟悉应用,特别是某些等待界面需要再

设计，而不应直接将一个很枯燥的等待界面呈现在用户面前，否则用户将很快就会放弃该应用。

在百度云应用中，当用户选择一个文件下载时，页面并不是显示一个下载进度的界面，让用户看任务下载的百分比。而只是将该下载任务放在传输列表中，在传输列表界面可以看到任务下载的进度情况，如图 5-33 所示。用户此时仍可以完成其他的操作，系统会在顶部的通知栏给出任务正在进行的提示，并不打搅用户的操作。这样就把查看下载结果的主动权交给用户。

#### 5.4.4 搜索设计技巧

搜索是移动设备用户的一项基本活动。随着移动设备变得更加易用、更加强大，用户在移动端的搜索请求也会如 PC 端一样，成为用户移动旅程的起点。为了提高用户在搜索时的用户体验，应注意以下几点。

##### 1. 自动输入提示

移动端的虚拟键盘一直是一个科技界无法解决的难题，虚拟键盘的主要缺点包括输入定位无法反馈，所以无法形成高效的盲打；因虚拟键盘的空间限制，手指的单击经常造成误按。所以在设计应用程序时，只要遇到内容输入的控件，首先就要想到尽量让用户少输入。当搜索文本框获得焦点时，搜索框展开以显示历史搜索建议，或者智能地给出参考。如果需要，屏幕键盘也会显示。选择任意建议提交搜索。点击向上箭头来离开搜索并关闭建议和屏幕键盘。图 5-34 是 QQ 音乐应用的搜索界面，QQ 音乐的搜索先是把搜索的类型依次排列在列表中，当有搜索关键词输入时，会出现歌手的候选词。

##### 2. 语音输入

为避免手机输入的不方便，语音输入是绝佳的替代方式，用户可以在一个已准备好接受焦点的文本搜索框右侧点击“麦克风”按钮来激活语音搜索，通过内置麦克风输入搜索关键词提高搜索简易性。图 5-35 是 91 手机助手应用的语音搜索界面。

微信 4.5 版对语音输入功能做了淋漓尽致的发挥，除了语音对讲、语音搜索联系人之外，又推出了多套语音相关服务，如语音聊天室、语音提醒、语音记事本、语音输入、语音识别歌曲、语音搜索内容等。



图 5-32 今日头条应用的侧向菜单



图 5-33 百度云传输界面

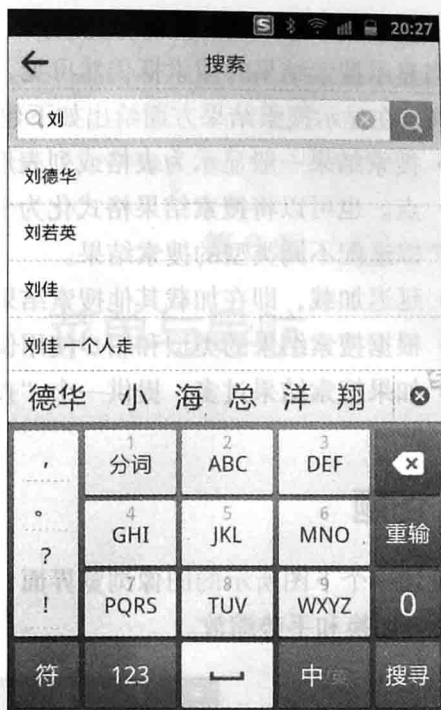


图 5-34 QQ 音乐应用的搜索界面

### 3. 保存搜索记录

通过保存用户的搜索记录并显示最近的搜索内容，用户就可以很容易地从先前的搜索内容中进行选择，而不需要再次输入相同的关键词。如果建议策略能命中用户的需求的话，能极大地提升用户的输入效率。

### 4. 将结果与本地结合

Google 的一份研究报告显示，1/3 的移动搜索与本地有关，或者有本地搜索意图。如果手机能定位，那么将帮助用户把搜索结果做一个本地化处理，并且能在搜索结果页直接启动导航或打开地图，提高用户搜索的效率和体验度。

### 5. 移动即时搜索

Google 已经有了即时搜索服务，如果把即时搜索放在手机端，则会颠覆搜索的模式。手机本来就是一个随时随地获取信息的利器，如果再结合即时搜索，保证搜索结果的新鲜感、时效性。对于热点新闻资讯来说会是非常有用的，对于基于位置的搜索，也不会因为搜索几年前的内容而导致搜到的商家已失效。



图 5-35 91 手机助手语音搜索界面

## 6. 显示搜索结果

当显示搜索结果时搜索框仍然可见，但默认失去焦点。屏幕键盘将隐藏以显示更多的搜索结果。在显示搜索结果方面给出如下建议：

- 搜索结果一般显示为表格或列表形式，如果是地图搜索，则直接在地图上显示搜索锚点。也可以将搜索结果格式化为卡片的样子来和搜索框内嵌样式相匹配。这种风格可以适配不同类型的搜索结果。
- 延迟加载，即在加载其他搜索结果的同时显示当前已经找到的结果。
- 根据搜索结果的类型和用户使用偏好提供多种视图。
- 如果搜索结果过多，提供一个“查看更多结果”按钮或类似的功能。

## 5.5 习题

设计一个下图所示的图像浏览界面，单击其中的一幅图像后，在新的界面显示大图，并支持滑屏切换和手势缩放。



## 对话框、菜单与导航

### 6.1 对话框设计

Android 提供了 Toast、Notification 和 AlertDialog 等控件来实现通知任务。

#### 6.1.1 Toast 通知

Toast 是一个简单的弹出式信息，最适用于确定用户正在关注屏幕时显示简短信息，例如“文件已保存”提示。它只占用信息所需的空间，用户当前的 Activity 仍然是可见且能够操作的。该通知自动地淡入淡出，不接受交互事件。图 6-1 展示了金山快盘网络连接错误的 Toast 通知的设计效果。

一般使用 Toast.makeText() 方法来实例化一个 Toast 对象。该方法需要三个参数：Context、文本信息和 Toast 的持续时间。它将返回一个正确初始化的 Toast 对象。可以用 show() 来显示该 Toast 通知，例如：

```
01 Context context = getApplicationContext();
02 CharSequence text = "Hello toast!";
03 int duration = Toast.LENGTH_SHORT;
04 Toast toast = Toast.makeText(context, text, duration);
05 toast.show();
```

#### 6.1.2 Notification 提示

Notification 显示一些简要的信息，用户在任何时候都可以从状态栏访问它们。它提供升

级、提醒以及一些重要但不至于直接打断用户的信息。将状态栏滑下可以打开通知抽屉。单击消息将会打开相关的应用。图 6-2 展示了 Android 4.4 平台的 Notification 通知栏。



图 6-1 金山快盘网络连接错误的 Toast 示例



图 6-2 Notification 下拉通知栏

Notification 适用于程序在后台服务工作中，需要通知用户某一事件时使用。该通知会向系统状态条添加一个图标（还可以附有文本消息）以及一条“通知”窗口内的可扩展消息。当用户选择了该可扩展消息，Android 发出一个有通知定义的 Intent（通常来启动一个 Activity）。也可以在 Notification 中设置声音、振动和设备闪光来提醒用户。

一个 Notification 通知必须包含以下所有内容（如图 6-3 所示）：

- 一个状态栏的图标（如图 6-3 中的⑤）。
- 一个标题（如图 6-3 中的②）和扩展视图中的扩展消息（如图 6-3 中的④）（除非另外自定义了一个的扩展视图）。
- 一个 PendingIntent，以供在通知被选中时放出。

创建一个 Notification 通知的一般步骤为：

① 获取 NotificationManager 的一个引用（Notification-Manager 是一个 Android 系统服务，它执行并管理所有 Notification）：

```
01 NotificationManager mNotificationManager = (NotificationManager)
02     getSystemService(Context.NOTIFICATION_SERVICE);
```



图 6-3 Notification 的典型组成

②实例化 Notification (Notification 的构造方法是: Notification(int icon, CharSequence tickerText, long when)):

```
01 int icon = R.drawable.notification_icon;
02 CharSequence tickerText = "Hello";
03 long when = System.currentTimeMillis();
04 Notification notification = new Notification(icon, tickerText, when);
```

③定义 Notification 的扩展消息和 Intent:

```
01 Context context = getApplicationContext();
02 CharSequence contentTitle = "My notification";
03 CharSequence contentText = "Hello World!";
04 Intent notificationIntent = new Intent(this, MyClass.class);
05 PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
06     notificationIntent, 0);
07 notification.setLatestEventInfo(context, contentTitle, contentText,
08     contentIntent);
```

setLatestEventInfo (Context, CharSequence, CharSequence, PendingIntent) 方法用来定义一个 Notification 所需的所有设置。

05 行的 PendingIntent 位于 android.app.PendingIntent 中, 该类用于处理即将发生的事情。PendingIntent 的重要特点是异步处理, 例如在通知 Notification 中用于跳转页面, 但不是马上跳转 (区别 Intent: Intent 位于 android.content.Intent, Intent 是及时启动, Intent 随所在的 Activity 消失而消失。)

④将 Notification 传递给 NotificationManager:

```
01 private static final int HELLO_ID = 1;
02 mNotificationManager.notify(HELLO_ID, notification);
```

用户发出通知后, 拖动状态栏图标, 显示如图 6-4 所示的通知内容。

NotificationManager 负责通知用户事件的发生, 有三个公共方法:

① cancel (int id) 移除一个已经显示的通知, 如果该通知是短暂的, 会隐藏视图; 如果通知是持久的, 会从状态栏中移除。

② cancelAll() 移除所有的已经显示的通知。

③ notify (int id, Notification notification)/ notify (String tag, int id, Notification notification) 提交一个通知并在状态栏中显示。如果拥有相同 ID 的通知已经被提交而且没有被移除, 该方法会用新的信息来替换之前的通知。

但是, 以下情况不应该使用 Notification 通知:

- 和用户没有直接关系或者不是时间敏感的事件。
- 如果该事件已经显示在屏幕上了, 就不要再使用通知了, 而是在应用界面中通知用户。
- 不要用技术细节来打扰用户, 例如保存、同步或者升级。如果应用可以自己处理, 就不要问用户。

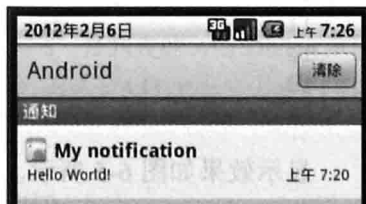


图 6-4 Notification 示例



- 不要用应用的错误消息打扰用户，如果应用可以快速恢复，那就不要问用户。
- 不要通知用户无法手动开始或者停止的服务。

### 6.1.3 AlertDialog 对话框

Dialog 对话框一般用于提示信息或与当前应用程序直接相关的小功能。Dialog 对话框一般是一个出现在当前 Activity 之上的小窗口。处于下面的 Activity 失去焦点，对话框接受所有的用户交互。

Android API 支持下列类型的对话框对象：

- **AlertDialog**：一个可以有 0 ~ 3 个按钮、一个单选框或复选框的列表的对话框。AlertDialog 可以创建大多数的交互界面，是推荐的类型。
- **ProgressDialog**：显示一个进度环或者一个进度条的对话框。由于它是 AlertDialog 的扩展，所以它也支持按钮。
- **DatePickerDialog**：是一个日期选择对话框，让用户选择一个日期。
- **TimePickerDialog**：是一个时间选择对话框，让用户选择一个时间。

还可以扩展 Dialog 类来自定义对话框。

#### 1. AlertDialog

AlertDialog 是 Dialog 的一个扩展，它能够创建大多数对话框用户界面。

下面的代码演示了使用 `AlertDialog.Builder(Context)` 创建带两个按钮的对话框的设计方法：

```
01 AlertDialog.Builder builder = new AlertDialog.Builder(this);
02 builder.setMessage("AlertDialog.Builder 提示对话框信息!!")
03     .setCancelable(false)
04     .setPositiveButton("OK", new DialogInterface.OnClickListener() {
05         public void onClick(DialogInterface dialog, int id) {
06             /* User clicked OK so do some stuff */
07         }
08     })
09     .setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
10         public void onClick(DialogInterface dialog, int id) {
11             dialog.cancel();
12         }
13     });
14 AlertDialog alert = builder.create();
```

显示效果如图 6-5 所示。

AlertDialog.Builder 中的重要方法有：

- **setTitle()**：为对话框设置标题。
- **setIcon()**：为对话框设置图标。
- **setMessage**：为对话框设置内容。
- **setView()**：给对话框设置自定义样式。
- **setItems()**：设置对话框要显示的一个列表，一般用于显示几个命令时使用。



图 6-5 带两个按钮的 AlertDialog 示例



- `setMultiChoiceItems()`: 用来设置对话框显示一系列的复选框。
- `setNeutralButton()`: 给对话框添加一个中性的按钮 (对每种按钮类型, 只能为 `AlertDialog` 创建一个)。
- `setPositiveButton()`: 给对话框添加 Yes 按钮。
- `setNegativeButton()`: 对话框添加 No 按钮。
- `create()`: 创建对话框。
- `show()`: 显示对话框。

下面的示例用 `setView()` 方法创建一个自定义的 `AlertDialog`:

```
01 LayoutInflater factory = LayoutInflater.from(this);
02 final View textEntryView = factory.inflate(
03     R.layout.alert_dialog_text_entry, null);
04 AlertDialog.Builder alertDialog = new AlertDialog.Builder(this);
05 alertDialog.setTitle("Android 提示");
06 alertDialog.setView(textEntryView);
07 alertDialog.setPositiveButton("OK",
08     new DialogInterface.OnClickListener() {
09         public void onClick(DialogInterface dialog, int whichButton) {
10             /* User clicked OK so do some stuff */
11         }
12     });
13 alertDialog.setNegativeButton("Cancel",
14     new DialogInterface.OnClickListener() {
15         public void onClick(DialogInterface dialog, int whichButton) {
16             /* User clicked cancel so do some stuff */
17         }
18     });
19 alertDialog.show();
```

显示效果如图 6-6 所示。

有兴趣的读者可以访问作者博客 (<http://blog.csdn.net/njcit/article/details/7760774>), 查看一篇关于 iPhone 效果的 `AlertDialog` 的文章。



图 6-6 带列表的 `AlertDialog` 示例

## 2. ProgressDialog

`ProgressDialog` 是 `AlertDialog` 的扩展, 它可以显示一个进度的动画——进度环或者进度条。这个对话框也可以提供按钮功能。

调用 `ProgressDialog.show()` 可以打开一个 `ProgressDialog`。例如:

```
ProgressDialog dialog = ProgressDialog.show(MyActivity.this, "",
    "Loading. Please wait...", true);
```

其中, 第一个参数是应用程序上下文, 第二个参数为对话框的标题 (这里为空), 第三个参数为对话框内容, 最后一个参数为该进度是否为不可确定的 (只与进度条的创建有关), 其显示效果如图 6-7 所示。

进度对话框的默认样式为一个旋转的环，如果希望显示进度值，需要使用 `setProgressStyle(int)` 方法将进度样式设置为 `STYLE_HORIZONTAL`。也可以使用 `setProgress(int)` 或者 `incrementProgressBy(int)` 来增加显示的进度，如图 6-8 所示。

Android 中还提供了 `DatePickerDialog` 和 `TimePickerDialog` 来显示日期和时间选择对话框，它们的使用与 `AlertDialog` 类似，这里不再赘述。

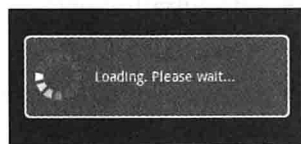


图 6-7 圆形 ProgressDialog 示例图

## 6.1.4 对话框的托管

Activity 提供了一种使用 `onCreateDialog(int)`、`showDialog(int)`、`onPrepareDialog(int, Dialog)`、`dismissDialog(int)` 等方法来管理创建、保存、回复对话框的机制。

### 1. onCreateDialog(int)

`onCreateDialog(int)` 方法用于在创建并显示 Android 对话框时调用，当使用该回调方法时，Android 系统会有效地将该 Activity 设置为每个对话框的所有者，从而自动管理每个对话框的状态，并挂靠到 Activity 上。这样，每个对话框就继承这个 Activity 的特定属性。



图 6-8 带百分比的 ProgressDialog

可以同时创建多个对话框，并为其设置不同的 `id` 参数以区分它们，然后可以通过 `showDialog()` 方法来显示。例如：

```
01 protected Dialog onCreateDialog(int id) {
02     Dialog dialog;
03     switch(id) {
04         case DIALOG_PAUSED_ID:
05             //do the work to define the pause Dialog
06             break;
07         case DIALOG_GAMEOVER_ID:
08             //do the work to define the game over Dialog
09             break;
10         default:
11             dialog = null;
12     }
13     return dialog;
14 }
```

### 2. showDialog(int)

当想要显示一个对话框时，调用 `showDialog()` 方法并传递一个唯一标识该对话框的整数。当对话框第一次被请求时，Android 从 Activity 中调用 `onCreateDialog()`，应该在这里初

始化这个对话框 Dialog。这个回调方法被传递给一个和 showDialog() 相同的 ID。当创建这个对话框后，在 Activity 的最后返回这个对象。

### 3. onPrepareDialog (int, Dialog)

如果需要在对话框被打开之前，程序根据具体的情况调整对话框的属性，可以在 onPrepareDialog() 中实现该功能。onPrepareDialog() 方法会在 showDialog() 方法之前被调用（而 onCreateDialog() 仅在对话框第一次打开时被调用）。该方法的参数 ID 含义和 showDialog() 方法一样。

### 4. dismissDialog (int)

dismissDialog() 方法的作用是关闭当前显示的对话框。该方法的参数 ID 含义和 showDialog() 方法一样。

dismissDialog() 和 removeDialog() 方法的区别为：dismissDialog() 方法使对话框消失，但对话框仍然处于内存中，只是不显示而已，如果再次调用 showDialog() 方法，则缓存在内存中的对话框会重新显示，而不需要重新创建它。removeDialog() 不仅使对话框消失，而且还从内存中将对话框清除，如果再次调用 showDialog() 方法来显示它，则在显示之前需要重新创建该对话框。

## 6.2 菜单设计

菜单是用户界面中最常见的元素之一，是程序重要的一部分，它提供给用户一个熟悉的接口以进入程序功能或是设置。Android 中的菜单有两种：一种是通过 MENU 键激活，叫 Options Menu；另一种是当用户长按一个 View 时激活，叫 Context Menu。

### 6.2.1 Options Menu

Options Menu 是作用于当前 Activity 全局的一种菜单，其中的菜单项命令对当前 Activity 的进程都有效。在大部分的手机中，用户按下 MENU 键后就会在屏幕下方显示 Options Menu。而用户再次按下 MENU 键或“返回”键就会关闭 Options Menu。图 6-9 演示了 Options Menu 的触发方式。

Options Menu 在屏幕底部最多只能显示 6 个菜单项，这些菜单项称为 Icon Menu，它只支持文字和图标，可以设置快捷键。如果 Options Menu 多于 6 个菜单项时会以“More”图标来显示，称为 Expanded Menu。它不支持 Icon 以及嵌套，其他的特性都和 Icon Menu 一样。

创建 Options Menu 有如下两种方法。

#### 1. 通过菜单资源创建 Options Menu

一般步骤如下：

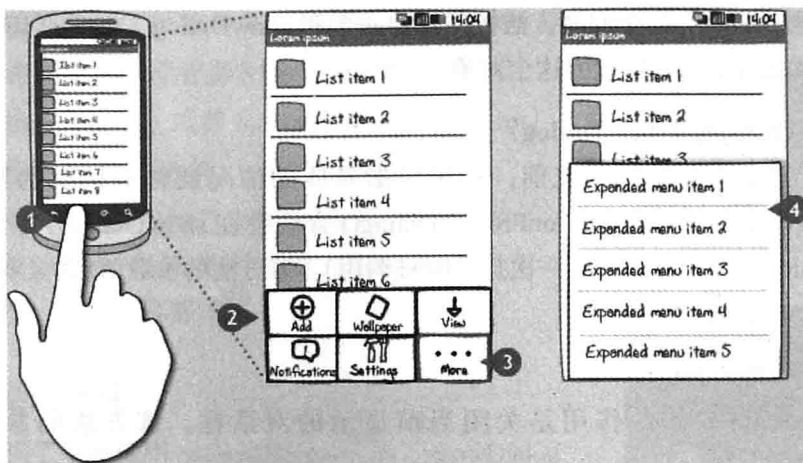


图 6-9 Options Menu 示例

①在 res/menu 中创建菜单资源文件 menu.xml，例如：

```
01 <menu xmlns:android="http://schemas.android.com/apk/res/android" >
02     <item
03         android:id="+id/action_about"
04         android:orderInCategory="100"
05         android:showAsAction="never"
06         android:title="关于" />
07 </menu>
```

<menu> 是根元素，没有属性，它包含 <item> 和 <group> 两种子元素。

- <item> 是一个菜单项，可以包含一个 <menu> 元素。<item> 必须为 <menu> 或 <group> 元素的子元素。<item> 元素的常见属性包括 id、menuCategory、title、icon 等。
- <group> 表示一个菜单组，相同的菜单组可以一起设置其属性。<group> 元素的常见属性包括 id、menuCategory、orderInCategory、checkableBehavior 等。

②在当前的 Activity 中加载菜单资源文件：

```
01 Override
02 public boolean onCreateOptionsMenu(Menu menu) {
03     super.onCreateOptionsMenu(menu);
04     getMenuInflater().inflate(R.menu.menu, menu);
05     return true;
06 }
```

getMenuInflater() 方法用于获取一个 MenuInflater 实例，MenuInflater.inflate(int menuRes, Menu menu) 方法使得菜单层次从一个指定的 XML 资源去填充，如果有错误则抛出 InflateException 异常。

③通过 onOptionsItemSelected (MenuItem) 回调方法捕捉菜单触发事件，例如：

```
01 @Override
```

```

02 public boolean onOptionsItemSelected(MenuItem item) {
03     switch (item.getItemId()) {
04         case R.id.action_about:
05             ...
06             break;
07     }
08     return true;
09 }

```

## 2. 在 Java 代码中创建选项菜单

一般步骤如下：

①在 Activity 中定义菜单项的识别序号常量，例如：

```
01 private static final int DELETE = Menu.FIRST + 1;
```

②使用 Menu.add() 方法在 onCreateOptionsMenu() 中构造菜单，例如：

```

01 @Override
02 public boolean onCreateOptionsMenu(Menu menu) {
03     menu.add(Menu.NONE, DELETE, 5, "删除").setIcon(
04         android.R.drawable.ic_menu_delete);
05     return true;
06 }

```

Menu.add(int groupId, int itemId, int order, int titleRes) 方法中的四个参数的含义是：

- groupId：组别，如果不分组的话就用 Menu.NONE。
- itemId：菜单项的 ID。
- order：定义菜单项在菜单中位置的顺序。
- titleRes：菜单的显示文本。

③通过 onOptionsItemSelected (MenuItem) 回调方法捕捉菜单触发事件。

④通过 onPrepareOptionsMenu (Menu) 更改 Menu Items 的属性 (可选)。

在 Android API 11 或更高版本中，Options Menu 中的项目被置于位于 Activity 顶部的动作栏中，取代了传统的标题栏的位置。有关动作栏的设计将在 6.3 节阐述。

下面给出两个常用的第三方 Options Menu 的设计效果，如图 6-10 所示。

### 6.2.2 Context Menu

Context Menu 是当用户长按一个注册了 Context Menu 的 View 时出现的浮动菜单项目列表。Context Menu 类似于桌面操作系统的右键菜单，对于不同的对象给出不同的菜单命令。图 6-11 演示了 Context Menu 的触发方式。

创建 Context Menu 和创建 Options Menu 类似，可以采用编写 XML 菜单资源或者在代码里动态创建。不同的是，创建 Context Menu 使用 onCreateContextMenuInfo() 回调方法，例如：

```
01 @Override
```

```

02 public void onCreateContextMenu(ContextMenu menu, View v,
03     ContextMenuInfo menuInfo) {
04     menu.setHeaderTitle("文件操作");
05     //add context menu item
06     menu.add(0, 1, Menu.NONE, "发送");
07     ...
08 }

```

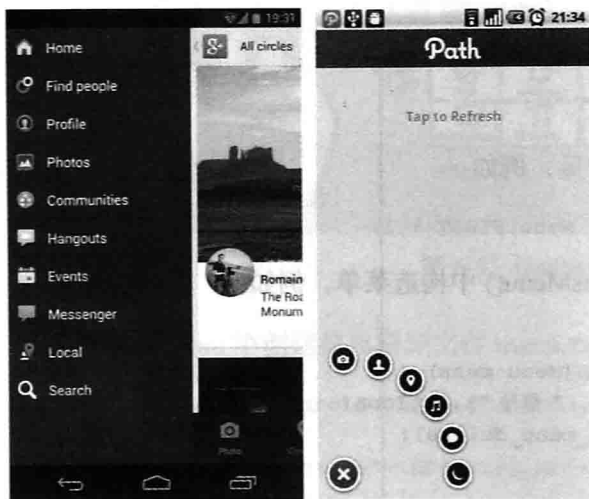
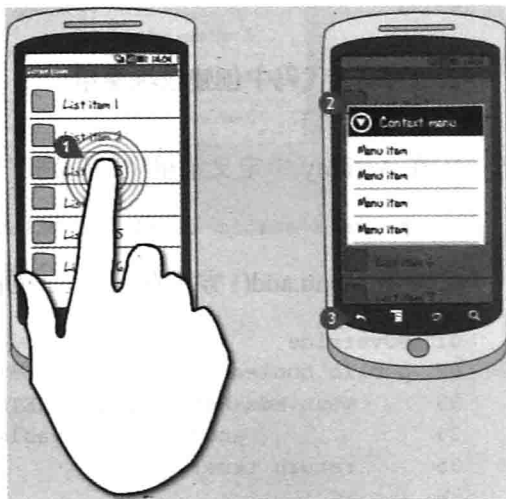
图 6-10 第三方 Options Menu 示例<sup>①</sup>

图 6-11 Context Menu 示例

然后使用 `onContextItemSelected()` 中响应 Context Menu 项（处理方法和 `onOptionsItemSelected`（MenuItem）一样），最后通过 `registerForContextMenu（View view）` 方法为 View 对象注册 Context Menu。

可以为 View 对象设置 `onLongClickListener` 监听来取代 Context Menu 的功能，并且可以实现比 Context Menu 更个性化的定制。

## 6.3 动作栏与导航设计

动作栏是 Android API 11 引入的概念，是 Activity 中的一种控件，用以代替传统的屏幕顶端的标题栏。

### 6.3.1 动作栏设计

动作栏是 Android API 11 引入的概念，是 Activity 中的一种控件，用以代替传统的屏幕顶端的标题栏。动作栏对于 Android 应用来说是最重要的设计元素，它通常在应用运行的所有生命周期都显示在屏幕顶部。

① SlidingMenu 项目主页 <https://github.com/jfeinstein10/SlidingMenu>；PathButton 项目主页 <https://github.com/dodola/PathButton>。

动作栏的主要作用包括：

- 突出重要的操作（例如“新建”和“搜索”），并且可以方便地使用。
- 在应用内提供统一的导航和视图切换体验。
- 较少使用的功能收集到其他操作菜单中，减少界面上的杂乱布局。
- 为应用提供一个展示其特点的空间。

图 6-12 是一个典型的动作栏设计效果，是默认的，动作栏包括了左侧的应用程序图标（图中的 1 部分），其右是 Activity 的标题或动作视图（图中的 2 部分），以及选项菜单（图中的 3 部分）和溢出菜单（图中的 4 部分）。

在上一节创建菜单资源时，如果为某一菜单项添加 `android:showAsAction="ifRoom"` 声明，则该菜单项将作为动作栏子项来显示，例如：

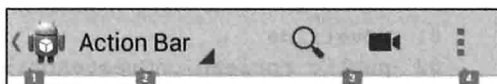


图 6-12 动作栏设计效果

```
01 <?xmlversion="1.0"encoding="utf-8"?>
02 <menu xmlns:android="http://schemas.android.com/apk/res/android">
03     <item android:id="@+id/menu_add"
04         android:icon="@drawable/ic_menu_save"
05         android:title="@string/menu_save"
06         android:showAsAction="ifRoom|withText"/>
07 </menu>
```

如果菜单项目同时提供了标题和图标，那么动作项目默认只显示图标。如果希望让动作项目包含文本，需要对 `android:showAsAction` 属性添加 `withText` 值，或是在程序代码中调用 `setShowAsAction()` 并使用 `SHOW_AS_ACTION_WITH_TEXT` 属性。06 行的代码即实现了该功能。如果 `android:showAsAction` 属性取值为 `always`，则总是将菜单项以动作栏子项显示。

也可以在使用 Java 动态生成菜单时使用如下语句实现相同的功能：

```
01 actionItem.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
```

显示动作栏的过程和显示菜单一样，在 Activity 启动时，系统将通过调用 `onCreateOptionsMenu()` 来为 Activity 生成选项菜单，同时根据菜单项的显示方式来生成动作栏。动作栏中的菜单项目和选项菜单中的其他项目都会调用相同的回调方法 `onOptionsItemSelected()` 来处理响应事件。

一般的，在 Activity 的 `onCreate()` 方法中通过 `getActionBar()` 方法来获取当前 ActionBar 的实例，然后通过 ActionBar 的 `hide()` 和 `show()` 方法来控制动作栏的隐藏和显示。

### 6.3.2 ActionMode 设计

ActionBar 多用于类似 Options Menu 菜单功能的设计，而 ActionMode 则多用于类似 Context Menu 菜单功能的设计。图 6-13 是文件管理器应用中勾选文件列表子项时的操作栏设计效果。

ActionMode 是一个浮于标题栏上的临时操作栏，用来放置一些特定的子任务。

ActionMode 一般在项目选择和文字选择时出现。ActionMode 在 Android API 11 之后才有支持。当用户激活 ActionMode 后，一个上下文操作栏会出现在屏幕的顶端，呈现出用户可以对当前选中项目进行的操作选项。

使用 ActionMode 的步骤如下：

①实现 ActionMode.Callback 接口。在它的回调方法中，可以设置操作的上下文操作栏。回调方法包括：

- onCreateActionMode (ActionMode mode, Menu menu)

在操作栏第一次被创建时调用。例如：



图 6-13 ActionMode 设计效果

```
01 @Override
02 public boolean onCreateActionMode(ActionMode mode, Menu menu) {
03     View v = LayoutInflater.from(MainActivity.this).inflate(
04         R.layout.actionbar_layout, null);
05     mActionText = (TextView) v.findViewById(R.id.action_text);
06     mActionText.setText(formatString(R.string.msg_select,
07         String.valueOf(getSelectedFiles())));
08     mode.setCustomView(v);
09     getMenuInflater().inflate(R.menu.action_menu, menu);
10     return true;
11 }
```

08 行通过 setCustomView() 方法定义操作栏的布局。09 行加载的菜单资源和 Options Menu 菜单资源一样。

- onPrepareActionMode (ActionMode mode, Menu menu) 在刷新菜单列表时被调用，一般使用 false 即可。
- onActionItemClicked (ActionMode mode, MenuItem item) 在菜单项被选中的时候被调用。例如：

```
01 @Override
02 public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
03     switch (item.getItemId()) {
04         case R.id.action_cut:
05             ...
06         default:
07             break;
08     }
09
10     mode.finish();
11     return true;
12 }
```

- onDestroyActionMode (ActionMode mode) 在退出或销毁操作栏时被调用。例如：

```
01 @Override
02 public void onDestroyActionMode(ActionMode mode) {
03     fileListAdapter.notifyDataSetChanged();
04     mActionMode = null;
05 }
```



②在需要显示上下文操作栏时调用 `startActionMode (ActionMode.Callback)`。例如，长按文件列表项时实现监听如下：

```
01 @Override
02 public boolean onItemLongClick(AdapterView<?> parent, View view,
03     int position, long id) {
04     // 激活 ActionMode
05     if (file.isDirectory() && mActionMode == null) {
06         mActionMode = startActionMode(mFolderOptActionMode);
07         mActionMode.setTitle(file.getName());
08         return true;
09     }
10     return false;
11 }
```

### 6.3.3 导航设计

Android 为 UI 导航提供了丰富的选择。

#### 1. 应用程序图标导航

默认情况下，应用程序图标出现在动作栏的左侧。通常的行为是在用户单击图标时，让应用程序回到父级 Activity 或是回到初始状态。

例如，在 Activity 的 `onStart()` 方法中，添加如下代码：

```
01 ActionBar actionBar = this.getActionBar();
02 actionBar.setDisplayOptions(ActionBar.DISPLAY_HOME_AS_UP,
03     ActionBar.DISPLAY_HOME_AS_UP);
```

此时单击应用程序的图标即返回父级 Activity。也可以调用 `setDisplayHomeAsUpEnabled(true)` 方法，这时，系统将为应用程序的图标增加一个表示向上一层动作的箭头。

当用户单击图标时，系统以 `android.R.id.home` 作为 ID 调用该 Activity 的 `onOptionsItemSelected()` 方法。例如，下面是一个 `onOptionsItemSelected()` 的实现，它将返回到 HomeActivity 用户界面：

```
01 @Override
02 public boolean onOptionsItemSelected(MenuItem item) {
03     switch (item.getItemId()) {
04         case android.R.id.home:
05             // app icon in Action Bar clicked; go home
06             Intent intent = new Intent(this, HomeActivity.class);
07             intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
08             startActivity(intent);
09             return true;
10         default:
11             return super.onOptionsItemSelected(item);
12     }
13 }
```

在 Intent 内设置 `FLAG_ACTIVITY_CLEAR_TOP` 使得要启动的 Activity 如果已经存在于

当前任务中, 则所有在其上的 Activity 将被销毁, 该 Activity 将回到最上层。

## 2. 标签页导航

标签页导航是一种比较理想的方式, 在窄屏显示时, 标签栏作为独立的一栏显示在屏幕的上方; 在宽屏显示时, 标签栏将和动作栏整合在一行上显示。例如, 在 Google Play 应用中, 如果当前是宽屏显示, 则显示图 6-14a 的设计效果; 如果是在窄屏显示, 则将中部的标签页显示在副操作栏中, 如图 6-14b 的设计效果。



图 6-14 Google Play 应用的副操作栏

标签栏导航设计的基本过程是:

①创建一个 `ActionBar.TabListener` 的实现来处理动作栏标签的交互事件。

主要是实现 `onTabSelected()`、`onTabUnselected()` 和 `onTabReselected()` 等方法。每一个回调方法都将传递收到了事件的 `ActionBar.Tab` 和一个 `FragmentManager` 用以执行 `Fragment` 事务 (如添加或是移除 `Fragment`)。例如:

```
01 public static class TabListener<T extends Fragment>
02     implements ActionBar.TabListener {
03     private Fragment mFragment;
04     private final Activity mActivity;
05     private final String mTag;
06     private final Class<T> mClass;
07
08     public TabListener(Activity activity, String tag, Class<T> clz) {
09         mActivity = activity;
10         mTag = tag;
11         mClass = clz;
12     }
13
14     public void onTabSelected(Tab tab, FragmentTransaction ft) {
15         if (mFragment == null) {
16             mFragment = Fragment.instantiate(mActivity, mClass.getName());
17             ft.add(android.R.id.content, mFragment, mTag);
18         } else {
19             ft.attach(mFragment);
20         }
21     }
22 }
```

```

23     public void onTabUnselected(Tab tab, FragmentTransaction ft) {
24         if (mFragment != null) {
25             ft.detach(mFragment);
26         }
27     }
28
29     public void onTabReselected(Tab tab, FragmentTransaction ft) {
30     }
31 }

```

该 `ActionBar.TabListener` 的实现添加了一个保存与标签关联的 `Fragment` 的构造方法，以使每一个回调方法可以添加或者移除 `Fragment`。有关 `Fragment` 的使用将在第 11 章介绍。

②对于要添加的每个标签页，通过调用 `setTabListener()` 来实例化 `ActionBar.Tab` 和设置 `ActionBar.TabListener`。还可以用 `setText()` 和 `setIcon()` 设置标签页的标题和图标。

③通过调用 `addTab()` 添加每个标签页到动作栏。

例如，以下的代码使用上面定义的监听器添加两个标签页：

```

01 @Override
02 protected void onCreate(Bundle savedInstanceState) {
03     super.onCreate(savedInstanceState);
04     ActionBar actionBar = getActionBar();
05     actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
06     actionBar.setDisplayShowTitleEnabled(false);
07
08     Tab tab = actionBar.newTab()
09         .setText("FRAGMENT A")
10         .setTabListener(new TabListener<ArtistFragment>(
11             this, "Fragment A", FragmentA.class));
12     actionBar.addTab(tab);
13
14     tab = actionBar.newTab()
15         .setText("FRAGMENT B")
16         .setTabListener(new TabListener<AlbumFragment>(
17             this, "Fragment B", FragmentB.class));
18     actionBar.addTab(tab);
19 }

```

### 3. 下拉列表导航

下拉列表导航的设计思想是在动作栏添加一个 `Spinner` 控件，通过 `Spinner` 项的选择实现导航。设计效果如图 6-15 所示。

设置下拉导航的基本过程是：

①创建一个适配 `Spinner` 的 `Adapter`，为下拉提供可选择项目的列表，以及当绘画列表中每个项目时所使用的布局。

②实现 `ActionBar.OnNavigationListener` 以定义用户从列表中选中一个项目时所发生的行为，例如：

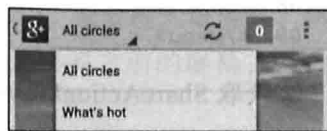


图 6-15 导航标签页设计效果

```

01 ActionBar.OnNavigationListener navigationListener =

```

```

02         new OnNavigationItemSelectedListener() {
03             @Override
04             public boolean onNavigationItemSelected(int itemPosition,
05                 long itemId) {
06                 ...
07                 return false;
08             }
09 };

```

③在 Activity 的 onCreate() 方法中使用 setNavigationMode() 为动作栏开启导航模式，例如：

```

01 ActionBar actionBar = getActionBar();
02 actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);

```

④用 setListNavigationCallbacks() 设置下拉列表的回调，例如：

```

01 actionBar.setListNavigationCallbacks(mSpinner Adapter,
02     navigationListener);

```

#### 4. 使用 ShareActionProvider 分享应用

如果应用允许对当前页的图像、视频等内容进行分享，可以在操作栏显示一个包含操作选择的下拉选项，并显示操作选择信息等。

ActionProvider 是 Android API 14 引入的概念，目的是能够更容易地实现一个在动作栏中友好和高效的共享动作项。ActionProvider 设计效果如图 6-16 所示。

设计 ActionProvider 的一般步骤如下：

①在菜单资源文件的 <item> 标签中加入 android:actionProviderClass="android.widget.ShareActionProvider" 属性，例如：

```

01 <?xml version="1.0" encoding="utf-8"?>
02 <menu xmlns:android="http://schemas.android.com/apk/res/android">
03     <item android:id="@+id/menu_share"
04         android:title="@string/share"
05         android:showAsAction="ifRoom"
06         android:actionProviderClass=
07             "android.widget.ShareActionProvider" />
08     ...
09 </menu>

```

②获取 ShareActionProvider 的实例，例如：

```

01 mShareActionProvider = (ShareActionProvider)
02     menu.findItem(R.id.menu_share).getActionProvider();

```

③调用 setShareIntent() 方法更新 ActionProvider，例如：

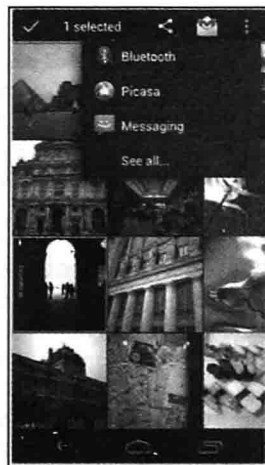


图 6-16 ActionProvider 设计效果

```
01 Intent intent = new Intent(Intent.ACTION_SEND);  
02 intent.setType("text/plain");  
03 intent.putExtra(Intent.EXTRA_TEXT, "New Share");  
04 provider.setShareIntent(intent);
```

ShareActionProvider 对象处理所有与菜单 id 为 menu-share 的项有关的用户交互，并且不需要处理来自 onOptionsItemSelected() 回调方法的单击事件。

默认情况下，ShareActionProvider 对象会基于用户的使用频率来保留共享目标的排列顺序。使用频率高的目标应用程序会显示在下拉列表的上面，并且最常用的目标会作为默认共享目标直接显示在动作栏。这种排序信息被保存在由 DEFAULT\_SHARE\_HISTORY\_FILE\_NAME 指定名称的私有文件中。如果只使用一种动作类型 ShareActionProvider 类或它的一个子类，那么应该继续使用这个默认的历史文件，而不需要做任何事情。但是，如果使用了不同类型的多个动作的 ShareActionProvider 类或它的一个子类，那么为了保持它们自己的排序信息，每种 ShareActionProvider 类都应该指定它们自己的排序文件。给每种 ShareActionProvider 类指定不同的历史文件，就要调用 setShareHistoryFileName() 方法，并且提供一个 XML 文件的名称（如 custom\_share\_history.xml）。

### 6.3.4 导航设计技巧

导航是用户体验的重要一环。不一致或者错误的导航对用户体验的影响非常严重。Android 2.3 以及更早版本的系统使用设备上的返回键在应用内导航。在 Android 3.0 引入了操作栏之后，一般使用操作栏上的应用图标和向左的箭头来导航。

#### 1. 向上与返回

“向上”按钮用来在应用内根据程序的逻辑层级进行导航。例如，屏幕 A 显示了一个项目列表，点击其中一项到达屏幕 B，显示该项目的详细信息，那么屏幕 B 应当提供一个“向上”按钮，让用户可以回到屏幕 A。如果某个屏幕已经是该应用的顶层了，那不需要“向上”按钮。

系统的“返回”键则用于按照屏幕切换历史返回到上一屏幕。如果之前的屏幕就是逻辑层次的上一层，那么“返回”和“向上”的行为是一样的。不过和“向上”不同的是，“返回”可能回到“主屏幕”或者其他的应用，“向上”回到的屏幕总是在当前应用中。图 6-17 演示了向上与返回的区别。

#### 2. 应用内导航

如果某个屏幕在应用的逻辑结构中的位置并不固定，可以从多个地方进入该屏幕，例如设置页面就可以从应用中的任何位置进入。那么，“向上”按钮会回到之前的屏幕，行为和“返回”完全一致。

如果通过标签或者左右滑动切换视图，或者使用下拉列表切换视图等，这些视图切换行为不会影响“向上”和“返回”按钮的行为，因为视图的切换不会影响当前屏幕在应用中的位置，也不会影响屏幕的导航历史。

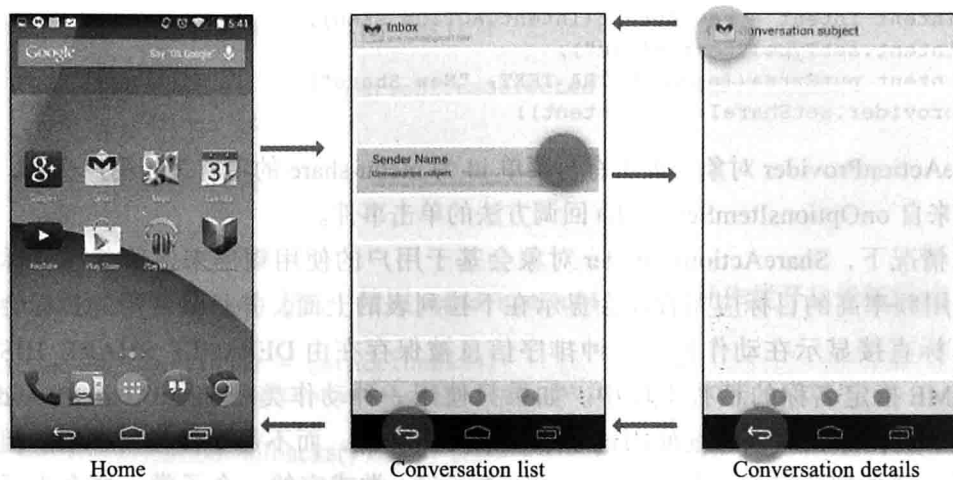


图 6-17 “向上”与“返回”导航的区别

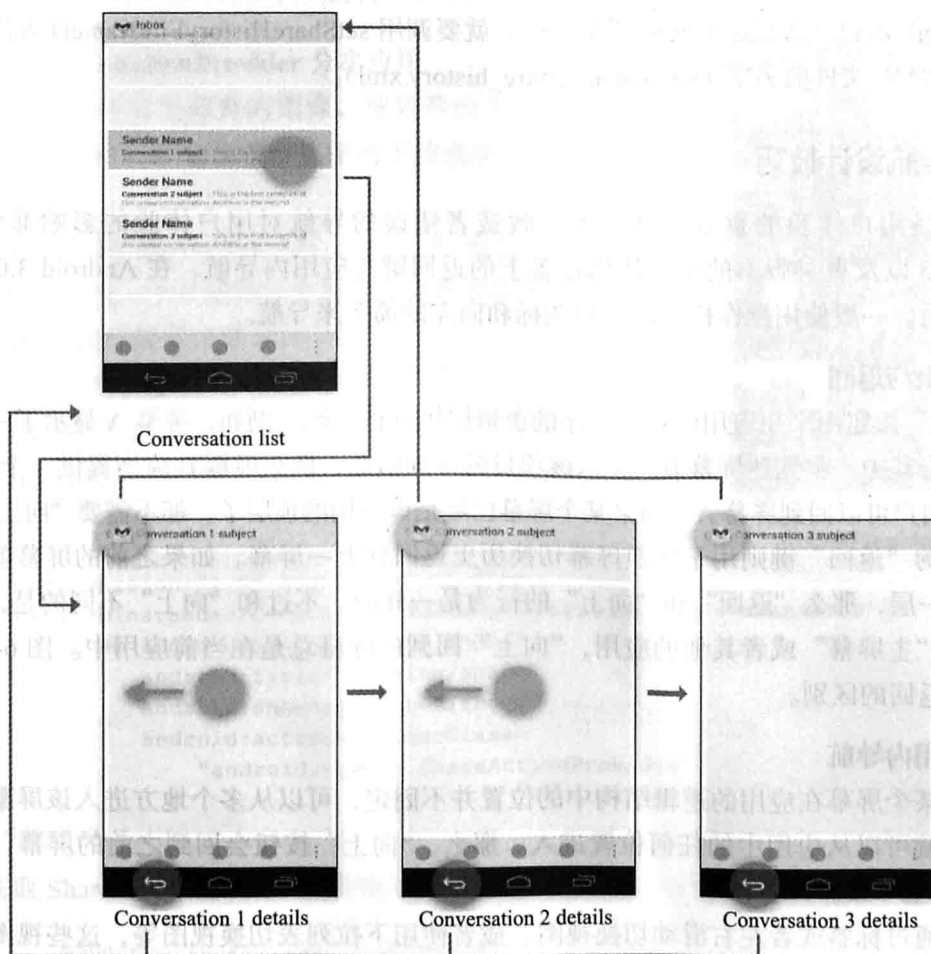


图 6-18 同级屏幕之间导航示意

如果应用可以从一个列表中进入某个条目的详细信息页面，那么应当考虑提供在相邻条目的详细信息页面之间的直接导航。例如在 Gmail 应用中，通过左右滑动邮件对话视图，可以在相同收件箱的上一条和下一条邮件对话之间导航。由于是在同一个屏幕中变换视图，所以这样的操作不会影响“向上”和“返回”的行为。设计效果如图 6-18 所示。

但是，浏览“相关”但是不是相邻的详细信息时，“向上”和“返回”的行为会发生变化。例如，在电子市场中浏览同一开发者的其他应用或者同一位歌手的不同专辑。这样的操作会导致屏幕历史发生变化，那么“返回”按钮就会回到上一个浏览的项目，而不是项目列表。“向上”按钮则仍然会回到项目列表。设计效果如图 6-19 所示。

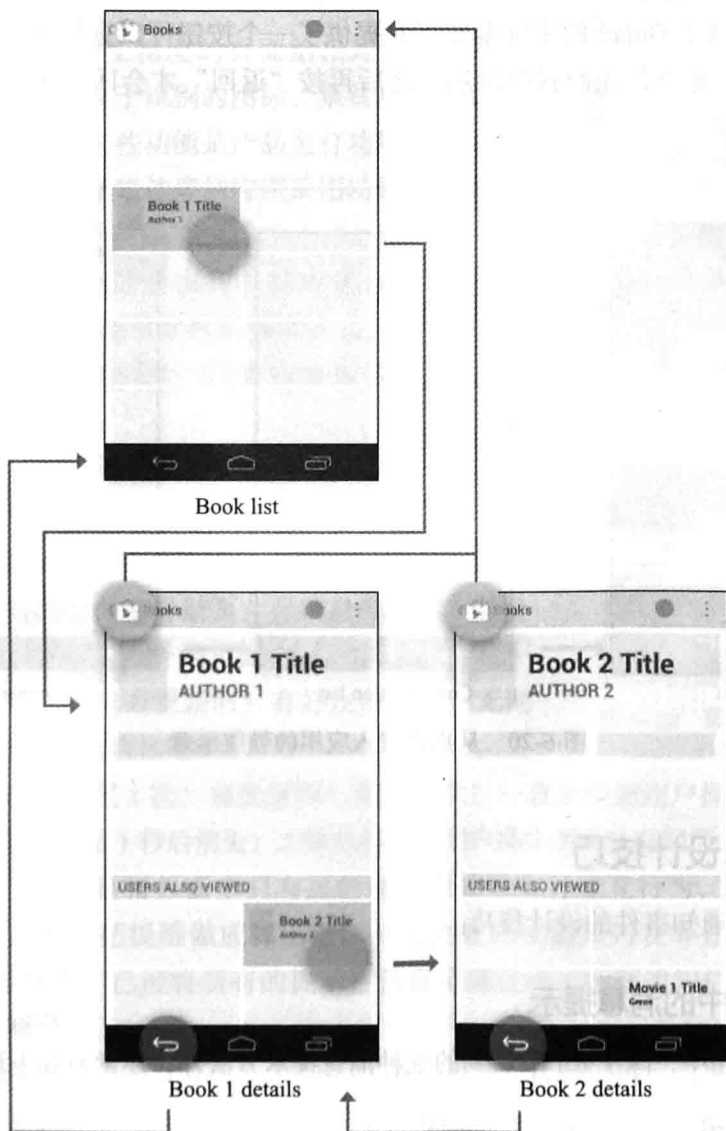


图 6-19 相关信息屏幕之间导航示意

### 3. 应用外导航

当其他应用通过 Intent 直接进入另一个应用的深层屏幕时,“返回”按钮将回到之前的应用中。“向上”则应当按照以下的规则处理:

- 如果该屏幕在应用中一般是通过一个项目列表进入的,例如某条项目的详细信息,那么“向上”应当回到项目列表。
- 如果不是上一种情况,“向上”就应当回到之前应用的首页。例如,从电子市场的图书详细信息页面通过“分享”进入 Gmail 的撰写屏幕时,“向上”会回到收件箱,而“返回”会回到电子市场。

如果应用可以通过通知栏或者主屏幕上的小工具进入,此时“向上”的行为和上面的描述完全一致。例如, Gmail 的主屏幕小工具提供了一个按钮可以直接进入撰写屏幕。按照导航原则,“返回”键会首先回到收件箱,之后再按“返回”才会回到主屏幕。设计效果如图 6-20 所示。

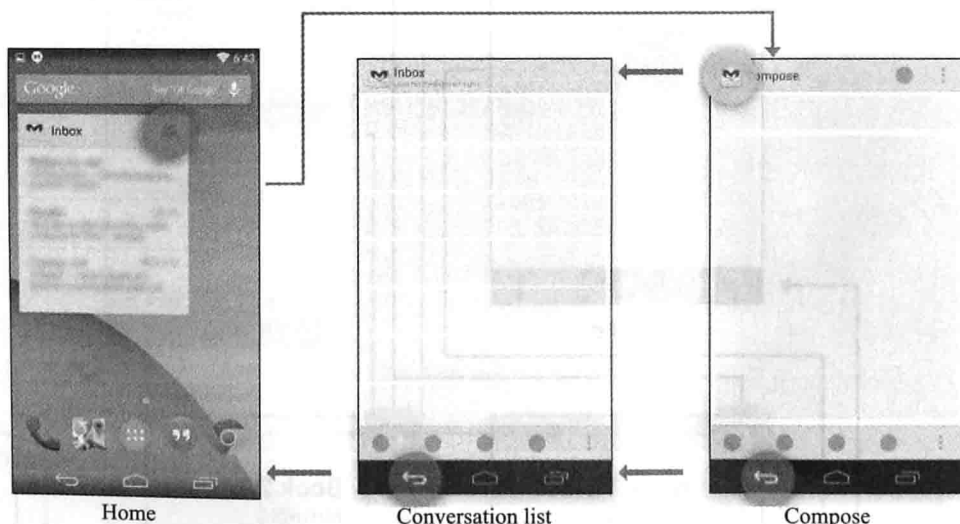


图 6-20 从桌面进入应用的导航示意

## 6.4 用户通知设计技巧

本节主要介绍通知事件的设计技巧。

### 6.4.1 Android 中的消息提示

在 Android 应用中,除了 6.1 节提到的三种消息提示方法外,还常有如下形式的消息提示。

#### 1. 桌面消息提示

在锁屏时,以桌面 Widgets 的形式给出消息提示,例如未接来电或短信等,如图 6-21



所示。

## 2. 标记消息提示

标记消息提示用于在相应项的右上角显示圆形消息数，提示用户该模块尚未处理的信息数。标记消息提示几乎都是通过用户的心理影响用户行为。很多时候，这种数字的激励方式确实对用户产生好的引导，让用户按照设计的轨迹操作。

标记消息提示属于弱提醒，图 6-22 是微信的标记消息提示。

## 3. 横幅消息提示

横幅消息提示用于在相应的页面给出用户操作提示。因为某些功能设计成不利于识别的图标，某些功能隐藏在手势操作之后不易察觉，某些功能是产品主打功能需要用户知晓等。在这种情况下，越来越多的应用采用功能提醒类的横幅消息做用户引导。

横幅消息提示属于适合出现在特定界面，具有较强引导性，但是又不打断当前操作的提醒方式。目前很多应用采用这样的设计，例如图 6-23 是新浪微博的横幅消息提示。



图 6-21 锁屏界面的未接来电提示

## 6.4.2 通知设计策略

在设计通知系统时，需要思考如下问题：

- 用户所在页面是九宫导航页？是消息列表页？是会话详情页？是应用之外的页面？
- 通知出现的时机是有更新时？有好友申请时？无网络时？累计 10 条消息时？
- 通知出现的频率仅 1 次？每次复现？重复 3 次？一直出现到用户操作为止？
- 通知消失的契机是 1 秒后消失？2 秒后消失？用户操作消失？有网消失？还是一直存在？
- 是否模态（模态是指界面中只有提醒弹出框才具有可交互行为，其他一切都不可操作；非模态不会把提醒做成弹出框，可能会处理成通知列表等方式来提醒用户。在 Android 系统中，已经将所有的提示性信息全部放到了标题通知栏里，避免了用户的任务被打断。）？
- 是否告知进度？告知百分比？告知运行中？不告知状态仅提示消息？

对照这些问题去比对，可以得出不同情况下，可以采纳的通知设计样式，举个简单的例子如下表 6-1 所示。



图 6-22 微信的标记消息提示



图 6-23 新浪微博的横幅消息提示

表 6-1 Activity 的生命周期回调方法

用户所在页面	出现时机	出现频率	消失契机	是否模态	告知进度	样式
锁屏界面	有新消息	1 次	3 秒后消失	是	否	桌面消息提示 +Notification
系统桌面	有新消息	一直存在，直到用户查看	用户查看后消失	否	否	标记消息提示 +Notification
应用主页	有新消息	重复 3 次	2 秒后消失	否	否	标记消息提示
我的主页	有好友申请	一直存在，直到用户处理	用户处理后消失	否	否	横幅消息提示

6.4.3 通知设计原则

通知的设计主要遵循以下几条原则。

(1) 引起用户的关注

通知系统如果完全无法引起用户的关注，例如做成闪图让用户误以为是广告，就达不到告知的作用，那通知就失败了。提供一段简短的文本作为通知的预览可以让用户大致了解通知的内容，从而帮助用户决定是否立刻查看该通知。

(2) 不使用户感到烦躁不安

标记提示并不是一种很友好的设计方式，因为用户不可能从一个数字气泡中推测出是

新消息还是好友请求，也不知道来自于谁，而必须点进去才知道。如果不点，屏幕就变成了“泡泡的世界”，实在让人不安。在应用内部，如果只是提醒进度或状态的通知，就不必要一直存在了，否则通知就变成了一种负担，尤其是那些不被用户操作消除的通知。

### （3）不希望中断用户的操作

要注意通知的时机和场合，如果为了引起用户的关注，而强硬地中断了用户的操作，牺牲了用户的体验，也是不可取的。用户可能对频繁出现的通知感到厌烦，所以应该让用户决定是否显示通知。因此，在应用程序的设置中应该让用户可以取消通知。

### （4）合并同类通知

如果一个应用程序发出了多个相同类型的通知，而且这些通知都还没被处理的话（被处理的通知会被移出通知抽屉），那么就将相同类型的通知合并为一个。合并后的通知会有一个总结性的描述，并且能让用户知道共合并了多少条通知。

### （5）引领用户到达该去的地方

如果应用发出的通知不但需要用户知晓，还需要用户执行操作，那么需要注意引导出现的时机，不要做成模态的，中断用户操作的。如果用户切换应用，需要帮用户记住当前状态，确保返回后，一切恢复到切换之前的状态。

### （6）用户需要时可再次找到提醒

好友申请、消息通知、信息流推送，这些通知，不能转瞬即逝，不管处理还是没处理，都可以再次找寻到通知所承载的内容才行。这样才能增强用户的安全感，保证信息的完整度。

### （7）使用不同的图标

为了让用户在通知栏看一眼就能知道是哪个应用程序发出的通知，应该采用有自己特色的图标。所以应该在设计应用程序的图标时，注意使用与其他 Android 应用有比较明显区别的通知图标。但需要注意的是不要用颜色来区分，因为通知图标通常都是黑白的。

### （8）自我清理

有些通知会在某个时间点出现告知用户一些相关的信息和提示，但是如果过了那个时间点，这个通知可能对用户来说就不重要了，此时就应该考虑自动删除该条通知。同样的，用户查看过的聊天消息或邮件，其通知也应该自动移除。

## 6.4.4 通知的导航机制

### （1）单条通知与合并通知

如果用户点击了一条通知，此时应该将相关的应用程序打开到可以对通知中提到的内容进行操作的状态。例如用户收到一封新邮件的通知，用户点开该通知后应该去到这封邮件的内容页。因为同类通知会被合并，如果用户点击了一个合并的通知，应该转到列表页面（内容页的上一层级）。如图 6-24 所示，当用户点击一条合并的新邮件通知后，即进入了收件箱界面。

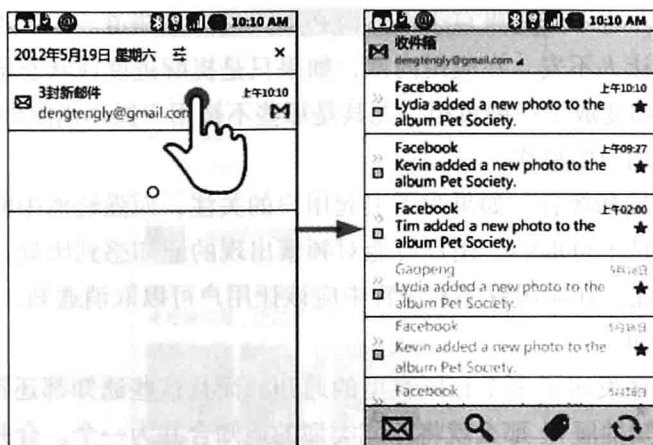


图 6-24 合并通知的导航机制

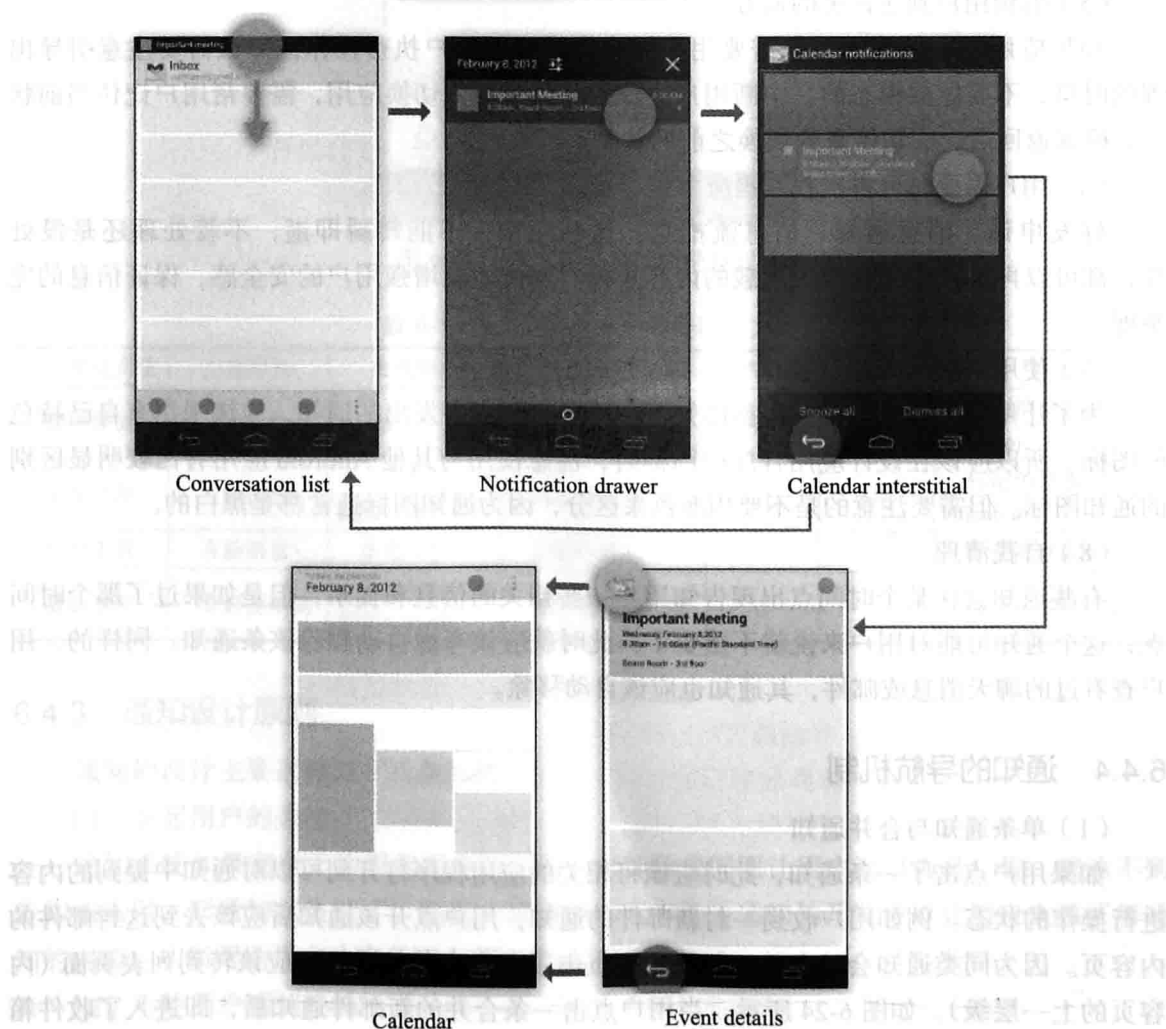


图 6-25 间接通知的导航机制

## （2）间接通知

如果应用程序需要同时展示多个事件的信息，可以使用一条通知将用户指引到一个中间界面。该界面会展示这些事件，并为用户提供进入应用程序的入口。这种类型的通知被称为间接通知。

例如一个用户在 Gmail 中收到一条 Calendar 发出的间接通知。点击这条通知后打开一个中间界面，在该界面中显示了几个事件的提醒，在该界面点击返回键则会回到 Gmail，但是如果用户点击了某个事件提醒，就会离开这个中间界面并打开 Calendar 应用程序以显示这个事件的详细内容。在这个事件的详细内容的界面下，点击向上和返回键都会去到 Calendar 应用的首页，如图 6-25 所示。

在间接通知的中间界面点击返回键会回到触发该通知的界面，返回路径中不会被插入其他界面。一旦用户通过中间界面进入了应用程序，“向上”和“返回”事件的逻辑就与标准通知一样了，即在应用程序之间进行导航，而不会返回到中间界面。

## （3）弹出通知

弹出通知会绕过抽屉通知直接出现在用户面前。一般情况下很少使用，只在需要及时地反馈并且必须打断用户的场合下才会使用。例如 Talk 应用使用这种形式的通知来提醒用户有好友邀请他加入视频聊天，因为这个邀请会在几秒后自动失效。

对于导航行为，弹出通知严格遵循间接通知的中间界面的导航逻辑。点击返回键会关闭弹出通知。如果用户从这条弹出通知进入了发出通知的应用程序，“向上”和“返回”事件的逻辑会与标准通知的逻辑保持一致，在应用程序内进行导航，如图 6-26 所示。

### 6.4.5 声音提醒

随着数字音频技术的发展，靠声音来与用户进行交互已经变得非常普遍且流行。比较常见的如新消息到达时的声音通知，Twitter 的新消息被赋予鸟儿鸣叫的声音等。人耳可以较容易地分别声音的强度、长短、音调、音色、节奏、旋律等特质。所以，声音一直被用来作为视觉的补充条件，提升用户体验。常见的声音通知方式如下。

#### （1）反馈音

在用手机进行输入时，不同的手机会伴随着按键动作产生不同的反馈音，类似的操作按键被点击，都可以有相应的反馈音。注册成功、升级成功也可以有成功提醒反馈音。

#### （2）警示音

当执行危险操作、登录失败、下载失败、电量不足、倒计时时间将至等情况时，会播放警示音告诉用户有危险。

#### （3）提示音

例如在新浪微博客户端执行刷新操作后，如果有新消息达到，即会有新消息提示音。这种提示音的好处就在于，用户可以执行操作之后去做其他事情，如果听到提示音，就知道消息发送成功或有新消息到达了，有效地告知了用户。

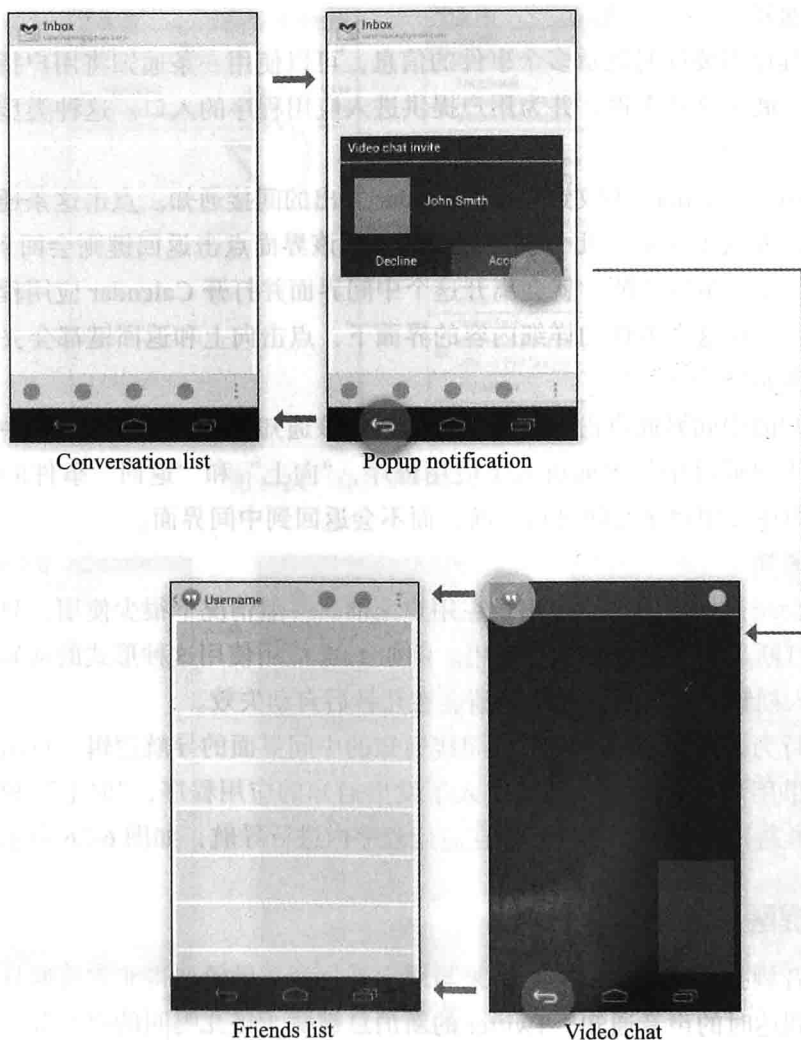


图 6-26 弹出通知的导航机制

#### (4) 拟物音

比如清除文件时，会播放倾倒垃圾时与垃圾桶碰撞的声音；QQ 好友上线会有咳嗽声提醒，有新的加好友请求会有敲门声提醒，这些都是拟物音。

#### (5) 人声

例如酷狗音乐应用通过模拟人声进行提醒；一些手机在电池需要充电时也会模拟人声告诉用户。

当然，尽管利用听觉可以做出很多有趣的交互体验效果，但是不能完全依赖听觉交互进行设计，因为在移动设备上听觉交互还有很多难题没有解决——环境，手机的使用环境复杂多样，在地铁上、路上等嘈杂的环境里，手机的声音可能就会变得微乎其微，听觉交互的可识别性将会大打折扣；当过度地依赖声音交互时，可能反而会形成一种噪音，干扰别人的生

活。所以，听觉提示要配合视觉提示一起使用，并且给予用户控制权，用户可以取消不需要的听觉反馈。

## 6.5 习题

设计一个下图所示的批量文件操作动作栏，并在执行删除命令前给出确认提示。



## 容器 UI 设计

### 7.1 导航类容器设计

导航类容器是指用于将多个用户界面组合在一个 Activity 中的控件，ViewPager、ViewFlipper、TabHost 等都是这类控件。

#### 7.1.1 使用 ViewPager 设计导航页

引导页是用户在首次安装并打开应用后，呈现给用户的说明书。目的是希望用户能在最短的时间内，了解这个应用的主要功能和操作方式，并迅速上手，开始体验之旅。在设计细节上，现在有越来越多的引导页运用线描的插画，这样既可以更方便、更准确地传达操作方式和使用场景等复杂信息，又可以增加亲和力，减少说教带来的用户反感。图 7-1 展示了 iReader 应用引导页的设计效果。

ViewPager 是 Android SDK 扩展包 android-support-v4.jar 中的控件，使用 ViewPager 能够实现只显示当前一组界面中的其中一个界面。当用户通过左右滑动界面时，当前的屏幕显示当前界面和下一个界面的一部分，滑动结束后，界面自动跳转到当前选择的界面中。

下面介绍使用 ViewPager 设计移动 APP 引导页的方法。

①在界面布局文件中加入 ViewPager 声明，例如：

```
01 <android.support.v4.view.ViewPager
02     android:id="@+id/guidePages"
03     android:layout_width="fill_parent"
04     android:layout_height="fill_parent" />
```



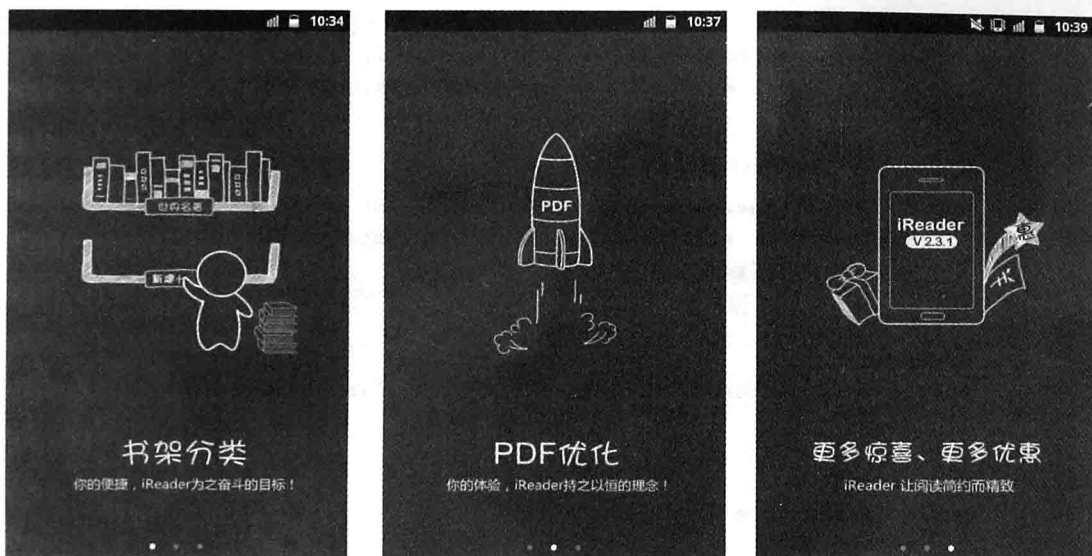


图 7-1 iReader 导航页示例

②初始化引导页视图集合，例如：

```
01 LayoutInflater inflater = getLayoutInflater();
02 View v1 = inflater.inflate(R.layout.item01, null);
03 View v2 = inflater.inflate(R.layout.item02, null);
04 View v3 = inflater.inflate(R.layout.item03, null);
05
06 ArrayList<View> pageViews = new ArrayList<View>();
07 pageViews.add(v1);
08 pageViews.add(v2);
09 pageViews.add(v3);
```

③为 ViewPager 实现 PagerAdapter，例如：

```
01 private class GuidePagerAdapter extends PagerAdapter {
02
03     @Override
04     public int getCount() {
05         return pageViews.size();
06     }
07
08     @Override
09     public boolean isViewFromObject(View arg0, Object arg1) {
10         return arg0 == arg1;
11     }
12
13     @Override
14     public int getItemPosition(Object object) {
15         return super.getItemPosition(object);
16     }
17 }
```

```
18     @Override
19     public void destroyItem(View arg0, int arg1, Object arg2) {
20         ((ViewPager) arg0).removeView(pageViews.get(arg1));
21     }
22
23     @Override
24     public Object instantiateItem(View arg0, int arg1) {
25         ((ViewPager) arg0).addView(pageViews.get(arg1));
26         return pageViews.get(arg1);
27     }
28
29     @Override
30     public void restoreState(Parcelable arg0, ClassLoader arg1) {
31     }
32
33     @Override
34     public Parcelable saveState() {
35         return null;
36     }
37
38     @Override
39     public void startUpdate(View arg0) {
40     }
41
42     @Override
43     public void finishUpdate(View arg0) {
44     }
45 }
```

④为 ViewPager 实现 PageChangeListener, 例如:

```
01 private class GuidePageChangeListener implements OnPageChangeListener {
02
03     @Override
04     public void onPageScrollStateChanged(int arg0) {
05     }
06
07     @Override
08     public void onPageScrolled(int arg0, float arg1, int arg2) {
09     }
10
11     @Override
12     public void onPageSelected(int arg0) {
13         for (int i = 0; i < imageViews.length; i++) {
14             imageViews[i].setBackgroundResource(
15                 R.drawable.page_indicator_focused);
16             if (arg0 != i) {
17                 imageViews[i]
18                     .setBackgroundResource(R.drawable.page_indicator);
19             }
20         }
21     }
22 }
```

### 7.1.2 使用 ViewFlipper 设计滑屏窗口

ViewFlipper 继承自 ViewAnimator，用于将多个视图组织在一起，并实现滑屏动画效果。ViewFlipper 一次仅能显示一个子视图。如果需要，可以设置 ViewFlipper 的间隔时间使子视图图像幻灯片一样自动显示。图 7-2 展示了 ViewFlipper 的设计效果。



图 7-2 ViewFlipper 设计效果

在图 7-2 中，左右滑动屏幕可以显示分组信息，上下滑动屏幕的下方，可以分别显示球队信息和比赛日程。为了实现滑屏及动画，ViewFlipper 常常和 OnGestureListener 结合使用。

下面介绍 ViewFlipper 的设计步骤：

①在界面布局文件中加入 ViewFlipper 声明，例如：

```
01 <ViewFlipper
02     android:id="@+id/mainViewFlipper"
03     android:layout_width="fill_parent"
04     android:layout_height="fill_parent" />
```

②设计子视图，并通过 ViewFlipper 的 addView() 方法将子视图添加到 ViewFlipper。

③让 Activity 实现 OnGestureListener 接口，并实现接口中的方法（参见 5.3.3 节），例如：

```
01 @Override
02 public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
03     float velocityY) {
04     if (e1.getX() - e2.getX() > FLING_MIN_DISTANCE) {
05         mainFlipper.setInAnimation(AnimationUtils.loadAnimation(this,
06             R.anim.push_left_in));
07         mainFlipper.setOutAnimation(AnimationUtils.loadAnimation(this,
08             R.anim.push_left_out));
09         mainFlipper.showNext();
10         return true;
11     }
12     ...
13     return false;
14 }
15 }
```

其中 05 行的 AnimationUtils 工具用于处理滑屏时的动画效果, 有关 Animation 的设计参见 9.2 节。

如果需要启动自动播放功能, 可以通过设置如下三个成员方法实现:

- etAutoStart(true) 设置是否启动自动播放功能, true 为自动播放, false 为不自动播放。
- setFlipInterval(intmilliseconds) 设置播放的时间间隔 (单位: 毫秒)。
- startFlipping() 开始自动播放。

### 7.1.3 使用 TabHost 设计标签页

#### 1. TabHost

在 6.3.3 节介绍了使用 ActionBar.TabListener 结合 Fragment 实现屏幕导航的方法, 本节主要介绍 TabHost 结合 TabActivity 实现标签页 UI 的方法。图 7-3 是 Android 4.4 中的联系人应用界面, 该界面展示了 TabHost 的设计效果。

TabHost 是一种选项卡容器, 即 UI 界面以选项卡的形式将多个 Activity 组合在一起, 通过单击选项卡标签显示不同的 View 或 Activity 的内容。

TabHost 的三要素是 TabWidget、FrameLayout 和 List<TabSpec>。TabHost 包含了两种子元素: 一些可以自由选择 Tab 和 Tab 对应的内容 TabContent。在布局的 <TabHost> 下它们分别对应 TabWidget 和 FrameLayout。TabWidget 是 TabHost 中的标签页控件, FrameLayout 是每个标签页内容的布局形式, List<TabSpec> 是标签页集合。

使用 TabHost 构建标签页面的一般步骤如下:

①构建布局, 如 2.2.4 节的帧布局示例。

在布局中, TabWidget 的 id 必须是 @android:id/tabs, FrameLayout 的 id 必须是 @android:id/tabcontent, TabHost 的 id 建议是 @android:id/tabhost。

②实现 TabActivity, 并在其 onCreate() 方法中通过 “TabHost tabHost = getTabHost();” 获取 TabHost 对象, 也可以使用 “TabHost tabHost = (TabHost) findViewById(android.R.id.tabhost);” 获取 TabHost 对象。

③定义 List<TabSpec>, 并指定标签页中的 Activity, 例如:

```
01 TabHost.TabSpec spec;
02 Intent intent = new Intent().setClass(this, ArtistsActivity.class);
03 spec = tabHost.newTabSpec("artists").setIndicator("Artists",
04         res.getDrawable(R.drawable.ic_tab_artists))
05         .setContent(intent);
06 tabHost.addTab(spec);
07
08 ...
09
```

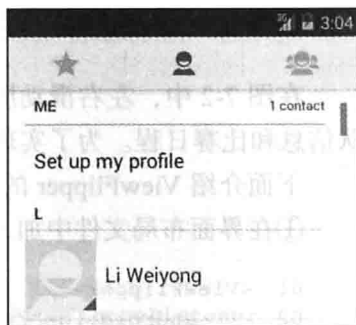


图 7-3 TabHost 设计效果

```
10 tabHost.setCurrentTab(1);
```

默认标签页上的图片和文字是重叠的,如图 7-4 右侧所示。

为了上下分行显示标签图片和文字(如图 7-4 左侧所示),先在 res/drawable 中定义图片资源,如 ic\_tab\_artists.xml,内容如下:

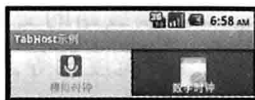


图 7-4 TabHost 上的标签

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <selector xmlns:android="http://schemas.android.com/apk/res/android">
03     <!-- When selected, use grey -->
04     <item android:drawable="@drawable/ic_tab_artists_grey"
05           android:state_selected="true" />
06     <!-- When not selected, use white -->
07     <item android:drawable="@drawable/ic_tab_artists_white" />
08 </selector>
```

然后在 getDrawable() 中引用该资源来实现标签页上的图标。

④遵循一般性,在 AndroidManifest.xml 中定义 TabActivity 的全屏显示风格,例如:

```
01 <activity android:name=".HelloTabWidget"
02           android:label="@string/app_name"
03           android:theme="@android:style/Theme.NoTitleBar">
```

TabActivity 继承自 Activity,其内部定义好了 TabHost,可以通过 getTabHost() 获取。

在实现 TabActivity 时,一般需要重载方法 setOnTabChangeListener() 来监听标签页的变动,例如:

```
01 tabHost.setOnTabChangeListener(new OnTabChangeListener() {
02     @Override
03     public void onTabChanged(String tabId) {
04         ...
05     }
06 });
```

## 2. Tabs 设计模式

使用 Tabs 将大量关联的数据或者选项划分成更易理解的分组,可以在不需要切换出当前上下文的情况下,有效地进行内容导航和内容组织。

### (1) Tabs 的特性

- Tabs 应该显示在一行内。
- Tabs 不应该被嵌套。也就是说,一个 Tab 里的内容不应包含另一组 Tabs。
- 一组 Tabs 至少包含 2 个 Tab,并且不多于 6 个 Tab。
- Tabs 控制的显示内容的定位要一致。
- Tabs 中当前可见内容要高亮显示。
- Tabs 应该归类并且每组 Tabs 中内容顺序相连。
- 保持 Tabs 和它们的内容相邻,可以明确两者间的关系,距离太远会让人误解。

### (2) Tabs 的分类

根据平台和使用环境,Tab 的内容可以表现为固定的 Tabs 或者是滚动(滑动)的 Tabs。

- 固定的 Tabs 同时显示所有 Tabs，最适合用于快速相互切换的 Tabs（例如，在地图中切换线路的交通方式）。视图的宽度限制了 Tabs 的最大数量。在固定的 Tabs 中每个 Tab 宽度相等，都是最宽的 Tab 标签的宽度。可以通过点击 Tab 或在内容区域中左右滑动来在固定的 Tabs 间进行导航。
- 滚动的 Tabs 用于显示 Tabs 的子集，可以在任何时候使用，并且可以包含更长的 Tab 标签和更多的 Tabs 数量，最适合用于触摸操作的浏览环境，并且用户不需要直接比较 Tab 标签。可以通过点击 Tab、在 Tab 上左右滑动或者在内容区域中左右滑动来在滚动的 Tabs 间进行导航。

### （3）Tabs 的尺寸规格

对于固定的 Tabs：

- Tab 的宽度为屏幕的 1/3。
- 激活的 Tab 的指示器高度为 2dp。
- 文本属性为 14sp、Roboto 和 Medium。
- 文本在 Tab 中居中。
- 激活的文字颜色为 #fff 或颜色选择中的次要颜色。
- 不可用的文字颜色为 60% 的 #fff。

对于滚动的 Tabs：

- Tab 宽度为 12dp+ 文本宽度 +12dp。
- 激活的 Tab 的指示器高度为 2dp。
- 文本属性为 14sp、Roboto 和 Medium。
- 激活的文字颜色为 #fff 或颜色选择中的次要颜色。
- 不可用的文字颜色为 60% 的 #fff。

Android 中，除了上面介绍的这些容器外，还有 ActivityGroup、SlidingDrawer 及 ViewFlow 等。

## 7.2 特定容器设计

特定容器是指为显示特定的内容而使用的控件，如 WebView、MapView 和 VideoView 等。

### 7.2.1 使用 WebView 显示网页

WebView 是 Android View 类的子类，可以在 Activity 布局中嵌入一个 WebView 控件来显示一个网页。图 7-5 展示了 WebView 的设计效果。



图 7-5 WebView 的设计效果

在应用中添加一个 WebView 的基本步骤如下:

①在界面布局文件中加入 WebView 声明, 例如:

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <WebView xmlns:android=http://schemas.android.com/apk/res/android
03     android:id="@+id/webview"
04     android:layout_width="fill_parent"
05     android:layout_height="fill_parent" />
```

②在 Activity 中绑定 WebView, 并使用 loadUrl() 方法载入网页, 例如:

```
01 private WebView webview;
02 ...
03 webview = (WebView) findViewById(R.id.webview);
04 webview.loadUrl("http://www.google.com");
05
06 webview.setWebChromeClient(new WebChromeClient() {
07     @Override
08     public void onProgressChanged(WebView view, int newProgress) {
09         if (newProgress == 100) {
10             // do something
11         } else {
12             // do something
13         }
14     }
15 });
16
```

在使用 WebView 的过程中需要注意以下几点:

① AndroidManifest.xml 中必须使用许可 android.permission.INTERNET, 否则会出现 Web page not available 异常。

②如果访问的页面中有 Javascript, 则 WebView 必须设置支持 Javascript。

```
01 webView.getSettings().setJavaScriptEnabled(true);
```

③如果希望单击页面中的链接继续在当前 Browser 中响应, 而不是启动 Android 系统中的浏览器响应该链接, 必须覆盖 WebView 的 WebViewClient 对象。例如:

```
01 webView.setWebViewClient(new WebViewClient(){
02     public boolean shouldOverrideUrlLoading(WebView view, String url) {
03         view.loadUrl(url);
04         return true;
05     }
06 });
```

④如果在浏览网页时不做任何处理, 单击系统返回键会使整个浏览器调用 finish() 而结束自身。如果希望浏览的网页回退而不是退出浏览器, 需要在当前 Activity 中处理并屏蔽返回按键事件。例如:

```

01 public boolean onKeyDown(int keyCode, KeyEvent event) {
02     if ((keyCode == KeyEvent.KEYCODE_BACK) && mWebView.canGoBack()) {
03         mWebView.goBack();
04         return true;
05     }
06     return super.onKeyDown(keyCode, event);
07 }

```

## 【 Web App 】

伴随着移动互联网的发展，未来的移动 APP 是 Web App 的天下还是 Native App 的天下？设计师是应该努力把客户端的体验提升到最优，还是在网页应用层面上做更多的设计？这些争论一直不断。

Web App 无需安装，对设备碎片化的适应能力优于 Native App，它只需要通过 XHTML、CSS 和 JavaScript 就可以在任意移动浏览器中执行。随着 WebKit 浏览体验的升级，使得支持 WebKit 浏览内核的移动设备开发的 Web App，也有了如 Native App 一般流畅的用户体验。

Web App 的优势包括：开发成本低；适配多种移动设备；开发成本低；跨平台和终端；迭代更新容易；无需安装成本。

Web App 的劣势包括：浏览的体验短期内还无法超越原生应用；不支持离线模式（HTML5 将会解决这个问题）；消息推送不够及时；调用本地文件系统的能力弱。

Google 的 Chrome OS 和 Android 都是操作系统，但走的是两条路。Chrome OS 走的是 Web App 的路。从 Chrome OS 大会上发布的 Chrome Web App 能看出来，Google 想为未来的 Chrome OS 做铺垫，鼓励更多的开发者开发出具有应用程序体验的 Web App，正如 Chrome OS 官网上说的——“Nothing but the web”。而 Android 走的是 Native App 的路。Android 作为手机平台的操作系统，明显更注重应用程序开发，这一点从 Android Market 可以看出来。前微软首席架构师 Ray Ozzie 的评价更加一针见血：“Google 的战略中 Android（以 App 为主）是在赌过去，而 Chrome OS（完全基于 Web）则是在赌未来。”

## 7.2.2 使用 MapView 显示地图

Android 平台为使用 Google 地图应用开发提供了良好的支持，可以通过引用 Google 地图扩展包 com.google.android.maps 来创建地图服务应用程序。图 7-6 展示了 MapView 的设计效果。

使用 MapView 设计地图应用的基本步骤如下：

① 申请 Google Map API Key，申请方法参见作者博客（<http://blog.csdn.net/njcit/article/details/8747477>）。申请到的 API Key 是一个类似图 7-7 所示的字符串。



图 7-6 MapView 的设计效果



② 在 Android SDK Manager 中安装 Google Play services SDK, 包括 Android Support Library 和 Google Play services。

③ 在 AVD 上安装 “google play 服务” 和 “google play 商店”, 并升级到最新。

④ 在项目的布局文件中加入 MapFragment 声明, 例如:

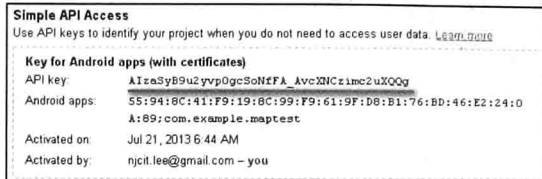


图 7-7 API Key 获取界面

```
01 <fragment
02     android:id="@+id/map"
03     android:layout_width="match_parent"
04     android:layout_height="match_parent"
05     class="com.google.android.gms.maps.MapFragment" />
```

05 行的 `com.google.android.gms.maps.MapFragment` 是地图显示控件。MapFragment 是 Fragment 的子类, 是放置地图的容器。注意, 不能只写成 MapFragment。

⑤ 导入 Google Play services 类库, 方法如下:

在 Eclipse 中, 执行菜单命令 `File → Import → Android → Existing Android Code Into Workspace`, 将位于 `<android-sdk-folder>\extras\google\google_play_services\libproject\google-play-services_lib` 下的项目导入 Eclipse。导入完成后, 将类库添加到项目的 Library 中。

⑥ 配置 AndroidManifest.xml, 核心代码如下:

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
03     ... >
04
05     <!-- Copied from Google Maps Library/AndroidManifest.xml. -->
06     <uses-sdk
07         android:minSdkVersion="14"
08         android:targetSdkVersion="17" />
09     <uses-permission android:name
10         ="android.permission.ACCESS_NETWORK_STATE" />
11     <uses-permission android:name="android.permission.INTERNET" />
12     <uses-permission android:name
13         ="com.google.android.providers.gsf.permission.READ_GSERVICES" />
14     <uses-permission android:name
15         ="android.permission.WRITE_EXTERNAL_STORAGE" />
16     <uses-permission android:name
17         ="android.permission.ACCESS_COARSE_LOCATION" />
18     <uses-permission android:name
19         ="android.permission.ACCESS_FINE_LOCATION" />
20     <uses-feature
21         android:glEsVersion="0x00020000"
22         android:required="true" />
23     <!-- End of copy. -->
24
25     <application
26         android:hardwareAccelerated="true"
27         ... >
```

```
28         <meta-data
29             android:name="com.google.android.maps.v2.API_KEY"
30             android:value="AIzaSyBn_IihHqF3FM_RysF5GTeBwak-Wpd4fE4" />
31         <meta-data
32             android:name="com.google.android.gms.version"
33             android:value="@integer/google_play_services_version" />
34
35         <activity
36             ...
37         </activity>
38     </application>
39
40 </manifest>
```

程序的 09 ~ 19 行定义了相关权限。20 ~ 22 行是对 OpenGL ES V2 的特性支持。28 ~ 33 行声明 API Key。其中的 value 即为前面的操作获取的 API Key。注意，不同的开发设备其获取的 API Key 不同，如果没有 API Key 或 API Key 错误，地图将不能显示。

如果需要改变地图的显示类型（包括交通模式、街景模式和卫星模式），可以使用如下方法：

```
01 GoogleMap mMap = ((MapFragment) getFragmentManager()
02     .findFragmentById(R.id.map)).getMap();
03 mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
```

还可以使用如下方法在地图上添加标记：

```
01 MarkerOptions markerOpt = new MarkerOptions();
02 markerOpt.position(new LatLng(纬度值, 经度值));
03 markerOpt.title("标记地点");
04 markerOpt.draggable(true);
05 mMap.addMarker(markerOpt);
```

### 7.2.3 使用 VideoView 播放视频

VideoView 是 Android 提供的一个视频播放控件。在布局文件中使用 VideoView 结合 MediaController 来实现对视频播放的控制。图 7-8 展示了 VideoView 的设计效果。



图 7-8 VideoView 的设计效果

使用 VideoView 设计视频播放的基本步骤如下：

①在界面布局文件中加入 VideoView 声明，例如：

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <VideoView xmlns:android=http://schemas.android.com/apk/res/android
03     android:id="@+id/videoview"
04     android:layout_width="fill_parent"
05     android:layout_height="fill_parent" />
```

②在 Activity 中绑定 VideoView，并使用 setVideoURI() 方法载入视频，例如：

```
02 VideoView videoView = (VideoView)findViewById(R.id.video_view);
03 videoView.setMediaController(new MediaController(this));
04 Uri uri = Uri.parse(Environment.getExternalStorageDirectory().getPath()
05     + "/Videos/Test_Movie.wmv");
06 videoView.setVideoURI(uri);
07 videoView.requestFocus();
```

VideoView 中的常用方法有：

- getCurrentPosition() 获得当前的播放位置。
- getDuration() 获得播放视频的总时间。
- setMediaController(MediaControllercontroller) 设置视频控制器。
- setOnCompletionListener(MediaPlayer.OnCompletionListenerl) 注册在视频文件播放完毕时调用的回调方法。
- setOnErrorListener(MediaPlayer.OnErrorListenerl) 注册在设置或播放过程中发生错误时调用的回调方法。如果未指定回调方法，或回调方法返回 false，VideoView 会通知用户发生了错误。
- setOnPreparedListener(MediaPlayer.OnPreparedListenerl) 注册在视频文件加载完毕可以播放时调用的回调方法。
- setVideoPath(Stringpath) 设置视频文件的路径名。
- setVideoURI(Uriuri) 设置视频文件的统一资源标识符。
- stopPlayback() 停止回放视频文件。

## 7.3 第三方容器控件设计

本节主要介绍两个开源的第三方容器控件的使用。

### 7.3.1 使用 SlidingMenu 设计菜单容器

SlidingMenu<sup>①</sup>是一个开源的菜单容器控件，提供了体验良好的滑入 / 滑出效果，并以其定

① 项目主页 <https://github.com/jfeinstein10/SlidingMenu>。

制灵活、丰富的阴影和渐变效果，广泛应用在多种移动 App 中。图 7-9 展示了 SlidingMenu 的设计效果。

### 1. SlidingMenu 的设计步骤

使用 SlidingMenu 设计滑屏菜单的基本步骤如下：

①在 Eclipse 中导入 ActionBarSherlock<sup>①</sup> 和 SlidingMenu 项目库，并将这两个项目作为自身项目的 Library。

如果提示 `getSupportActionBar()` 方法不能用，需要修改 `SlidingFragmentActivity` 继承自 `SherlockFragmentActivity`。

②设计 SlidingMenu 菜单容器所在的 Fragment，主要包括：

- 设计 SlidingMenu 的布局文件，这里可以是 `ListView` 或其他 `View`。
- 在 Fragment 中，设计 SlidingMenu 菜单项的单击响应事件。

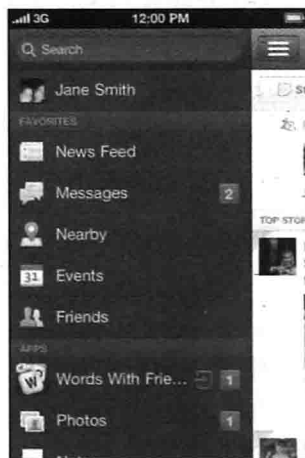


图 7-9 SlidingMenu 的设计效果

③ SlidingMenu 一般由标题栏左侧的应用程序图标激活，因此，需要对标题栏进行如下设置：

```
01 private void initActionBar(){
02     ActionBar actionBar = getSupportActionBar();
03     actionBar.setCustomView(R.layout.actionbar_layout);
04     actionBar.setDisplayShowCustomEnabled(true);
05     actionBar.setDisplayShowHomeEnabled(false);
06     mAppButton = (ImageButton) findViewById(R.id.iv_app_icon);
07     mAppButton.setOnClickListener(this);
08 }
```

06 行绑定的按钮就是标题栏左侧的应用程序图标按钮，单击该按钮后激活 `toggle()` 方法来显示 SlidingMenu。`toggle()` 是 `SlidingFragmentActivity` 中提供的方法。

④在 Activity 的 `onCreate()` 方法中初始化 SlidingMenu，例如：

```
01 private void initSlideMenu(){
02     FragmentModel fragmentModel = mControlCenter.getTouTiaoFragmentModel();
03     switchContent(fragmentModel);
04     SlidingMenu sm = getSlidingMenu();
05     sm.setMode(SlidingMenu.LEFT_RIGHT);
06     setBehindContentView(R.layout.left_menu_frame);
07     sm.setSlidingEnabled(true);
08     sm.setTouchModeAbove(SlidingMenu.TOUCHMODE_FULLSCREEN);
09     sm.setShadowWidthRes(R.dimen.shadow_width);
10     sm.setShadowDrawable(R.drawable.shadow);
11     getSupportActionBar().setDisplayHomeAsUpEnabled(true);
12     getSupportFragmentManager()
13         .beginTransaction()
14         .replace(R.id.left_menu_frame, new NavigationFragment())
```

① 项目主页 <http://actionbarsherlock.com/>。

```

15     .commit();
16     sm.setBehindOffsetRes(R.dimen.slidingmenu_offset);
17     sm.setBehindScrollScale(0);
18     sm.setFadeDegree(0.25f);
19 }

```

initSlideMenu() 方法主要用来设置 SlideMenu 的宿主 Fragment，以及滑入 / 滑出时的动画效果等。

## 2. SlidingMenu 的局限性

设计人员在使用 SlidingMenu 设计多视图应用时，发现存在功能不易发现且更低效率的用户体验。举个例子，2013 年 9 月，Facebook 在更新 Facebook App 时，在不同的版本使用中分别使用 Tab 标签式或 SlidingMenu 抽屉式导航方式，如图 7-10 所示。通过测试发现，人们使用图 7-10a 所示应用时停留在应用上的时间比图 7-10b 所示的应用要多，与应用的交互也较多，这也符合人们“看不到的不记挂”的思维模式。



a)



b)

图 7-10 Facebook 的两种导航设计效果

那么，到底什么时候适合用 SlidingMenu 侧导航呢？一般来说，如果应用的主要功能和内容都在一个页面里，而只是一些用户设置和选项需要显示在其他页面里。出于让主页面看上去干净美观的目的可以把这些辅助功能放在侧边栏里。而如果应用有不同的视图，且它们是平级的，需要用户同等地对待，那么侧边栏将会浪费掉大多数的用户对于侧边栏中入口的潜在参与度和交互程度。

### 7.3.2 使用 TimesSquare 设计日期

TimesSquare<sup>①</sup> 是一个开源的日期容器控件，能够方便地在用户界面中实现日期选择功能。

图 7-11 展示了 TimesSquare 的设计效果。

使用 TimesSquare 设计日期容器的基本步骤如下：

①在 Eclipse 中导入 TimesSquare 项目库，并将该项目作为自身项目的 Library。

②在界面布局文件中加入 TimesSquare 声明，例如：

```
01 <com.squareup.timesquare.CalendarPickerView
02     android:id="@+id/calendar_view"
03     android:layout_width="match_parent"
04     android:layout_height="0dp"
05     android:layout_weight="1"
06     android:paddingLeft="16dp"
07     android:paddingRight="16dp"
08     android:paddingBottom="16dp"
09     android:scrollbarStyle="outsideOverlay"
10     android:clipToPadding="false"
11     android:background="#FFFFFF" />
```

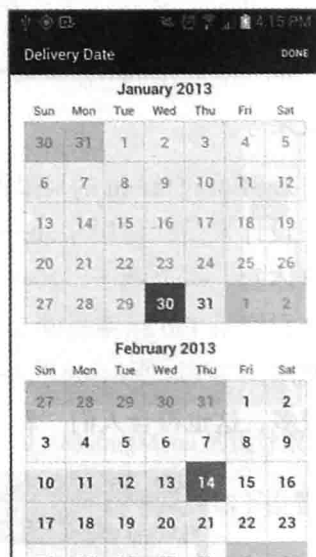


图 7-11 TimesSquare 的设计效果

③在 Activity 中绑定 CalendarPickerView，并使用 init() 方法初始化日期信息和显示模式，例如：

```
01 CalendarPickerView calendar = (CalendarPickerView)
02     findViewById(R.id.calendar_view);
03 calendar.init(lastYear.getTime(), nextYear.getTime())
04     .inMode(SelectionMode.SINGLE)
05     .withSelectedDate(new Date());
```

④在 Activity 中，通过 CalendarPickerView 实例的 getSelectedDate() 方法获取选择的日期信息。

## 7.4 引导页设计技巧

对于移动 APP 来说，“引导”的本质就是一种小提示，当用户第一次使用应用时显示出来，向用户推荐一些常用功能的操作方式，或引领它们完成一个预设的教学目标。简单却用心的引导方式可以很有效地在初次体验中提升用户满意度。

下面给出三种常见的引导页设计模式。

### 1. 游历式

游历可以带来终极的引导体验——通过对关键屏和重要功能的连续展示，在最短的时间

① 项目主页 <https://github.com/square/android-times-square>。

内引领用户对应用进行全面的探索。图 7-12 是城市生活应用的引导页，这是一个典型的游历式引导页。



图 7-12 游历式引导页

游历式引导页的设计原则是：

- 告诉用户现在处于指南的哪个位置；
- 每个页面只讲清楚一个功能；
- 最好可以选择性的跳过；
- 只能出现一次。

莫贝网的“移动端引导设计技巧 1：前置的引导页”一文（网址：<http://www.guimobile.net/design-skills-front-guide-page.html>）为这种引导页设计模式提供了更为丰富的介绍。

## 2. 半透明标注式

与其他具有引导性的设计模式相比，半透明标注的方式对于触屏设备来说有些独特。它通常会在应用的首屏出现，以一个覆盖在真实界面上方的半透明层为背景。通过半透明标注的模式，它们快速并且可视化地向用户展示了怎样对内容进行导航。应该以正确的方式使用半透明标注，而不是试图通过这种效果来弥补应用界面本身的拙劣设计。当用户开始产生相应的交互行为时，要及时地移除标注。图 7-13 是 Piictu 应用的引导页，这是一个典型的半透明标注式引导页。

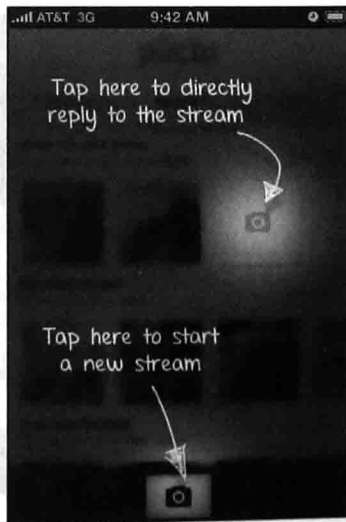


图 7-13 半透明标注式引导页

半透明标注式引导页的设计原则是：

- 可以引导用户执行主要操作；
- 不要强迫用户执行操作；
- 如果任务是多步骤，那么要关注每一步的设计；
- 主要功能可以在设计上区别于其他。

### 3. 嵌入式

与其他模式不同，“嵌入”不会在目标界面加载之前呈现。在这种模式中，相关的引导内容会融入到界面设计当中，直到被真正的内容覆盖移除掉。在一个界面中可以存在多个嵌入式引导提示。在设计应用的界面时，要对这些嵌入式元素进行差异化处理，通过图片等方式使它们能与普通内容很好地区别开。图 7-14 是 Mini Diary 应用的引导页，这是一个典型的嵌入式引导页。

嵌入式引导页的设计原则是：

- 注意动画持续的时间和出现的时机；
- 用来提醒隐藏的操作或重要的功能；
- 不要影响主要任务流程；
- 判断需要提醒的时机；
- 持续 3~5 秒自动消失。



图 7-14 嵌入式引导页

## 7.5 习题

设计一个下图左侧所示的个人微博资料界面，单击操作栏应用程序图标左侧的箭头，显示下图右侧所示的侧边栏菜单。





## 自定义控件设计

### 8.1 概述

Android 框架中定义的所有控件都继承自 View，这些控件包括 Button、TextView 等基本 Widgets，包括 ListView、MapView 等容器控件，以及 LinearLayout、RelativeLayout 等布局控件。

在 Android 中，可以通过 3 种方法设计一个与众不同的控件：

- 为原生控件应用个性化的主题与样式。
- 直接继承自 View，并重写 onDraw() 方法绘制控件。
- 重构现有的 View 子类。

无论使用哪种方式来设计个性化的控件，这些控件都必须满足：

- 符合 Android 标准。
- 提供能够在 AndroidXML 布局中工作的自定义样式属性。
- 发送可访问的事件。
- 与多个 Android 平台兼容。

### 8.2 定制控件

定制控件的一般步骤如下：

- ① 创建一个基于 View 类的子类。
- ② 设计一个构造方法以便从 XML 文件中提取属性和参数，也可以自定义这些属性和参数。

③设计事件监听器，如属性的访问和修改方法，以及一些控件本身的功能上的代码。

④重载 `onMeasure()` 方法和 `onDraw()` 方法，绘制 `View`。

⑤其他有必要重载的 `on***()` 系列方法。

下面以一个用于在 Android 中显示 Gif 动画的自定义控件 `GifView`<sup>⑨</sup> 为例，来详细介绍绘制控件的方法和步骤。

## 1. 拓展 View 类

实现扩展自 `View` 的 `GifView`，构造方法如下：

```
01 public class GifView extends View {
02     public GifView(Context context) {
03         this(context, null);
04     }
05
06     public GifView(Context context, AttributeSet attrs) {
07         this(context, attrs, R.styleable.CustomTheme_gifViewStyle);
08     }
09
10     public GifView(Context context, AttributeSet attrs, int defStyle) {
11         super(context, attrs, defStyle);
12         setViewAttributes(context, attrs, defStyle);
13     }
14     ...
15 }
```

为了让 `Activity` 能够与 `GifView` 交互，必须提供一个能够获取 `Context` 和作为属性的 `AttributeSet` 对象（`AttributeSet` 是节点的属性集合）的构造方法。该构造方法允许布局编辑器建立和编辑一个 `GifView` 的实例，如代码中的 06~08 行及 10~13 行。

## 2. 自定义属性

为了在布局文件中指定 `GifView` 并正确设置其属性和样式参数，需要为 `GifView` 定义一组用来控制其外观和行为的属性，这些属性必须满足：

- 在资源元素 `<declare-styleable>` 中为自定义控件定义属性。
- 在 XML 布局中指定属性的值。
- 在运行时取得属性值。
- 将取回的属性值应用到 `View` 中。

`<declare-styleable>` 资源通常放在 `res/values/attrs.xml` 文件里。如下是 `GifView` 项目的 `attrs.xml` 资源：

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <resources>
03
```

⑨ 项目主页 <https://github.com/RoiSoleil/GifView>。

```

04     <declare-styleable name="GifView">
05         <attr name="gif" format="reference" />
06         <attr name="paused" format="boolean" />
07     </declare-styleable>
08     <declare-styleable name="CustomTheme">
09         <attr name="gifViewStyle" format="reference" />
10     </declare-styleable>
11
12 </resources>

```

attrs.xml 文件声明了两个样式实体, GifView 样式包含自定义属性 gif 和 paused, CustomTheme 样式包含自定义属性 gifViewStyle。一般来说, 样式实体的名字和声明的自定义控件类名是相同的。

一旦定义了自定义属性, 就可以在 XML 布局文件中像内建属性一样使用它们。唯一的区别是, 自定义属性属于不同的命名空间。它们属于 `http://schemas.android.com/apk/res/[your package name]` 以取代默认的 `http://schemas.android.com/apk/res/android` 命名空间。下面的代码演示了如何为 GifView 使用这些自定义属性:

```

01 <?xml version="1.0" encoding="utf-8"?>
02 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
03     xmlns:custom=
04         "http://schemas.android.com/apk/res/com.example.customviews">
05     <org.roisoleil.gifview.GifView
06         custom:gif="@drawable/pic"
07         custom:paused="true" />
08 </LinearLayout>

```

上例中, 为了不重复较长的命名空间, 使用了一个别名为 custom 的 xmlns 指示来指向命名空间 `http://schemas.android.com/apk/res/org.roisoleil.gifview`。在向布局中添加自定义控件的 XML 标签时, 使用了自定义控件类的完全表述, 即 `org.roisoleil.gifview.GifView`。如果自定义的控件是一个内部类, 则必须使用外部类的名字进一步限定它。例如, GifView 类有一个叫做 GifWidget 的内部类。为了使用这个类中的自定义控件, 必须使用标签 `org.roisoleil.gifview.GifView$GifWidget`。

### 3. 获取自定义属性

当在 XML 布局中引用了自定义控件以后, XML 标签中所有的属性都从资源包中读取出来并作为一个 AttributeSet 传递给 View 的构造方法。通常的做法是, 将 AttributeSet 传递给 `obtainStyledAttributes()` 方法。这个方法传回一个 TypedArray 数组, 包含了已经解除引用和样式化的值。下面是 GifView 类读取这些属性的示例:

```

01 private void setViewAttributes(Context context, AttributeSet attrs,
02     int defStyle) {
03     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
04         setLayerType(View.LAYER_TYPE_SOFTWARE, null);
05     }

```

```

06     final TypedArray array = context.obtainStyledAttributes(attrs,
07         R.styleable.GifView, defStyle, R.style.Widget_GifView);
08     mMovieResourceId = array.getResourceId(R.styleable.GifView_gif, -1);
09     mPaused = array.getBoolean(R.styleable.GifView_paused, false);
10     array.recycle();
11     if (mMovieResourceId != -1) {
12         mMovie = Movie.decodeStream(getResources().openRawResource(
13             mMovieResourceId));
14     }
15 }

```

第 10 行非常重要，`TypedArray` 对象是一个共享的资源，使用完毕必须回收它。

#### 4. 设置属性和事件

属性是控制 `GifView` 的行为和外观的有效方式，为了提供动态的行为，需要实现每个自定义属性的一对 `getter` 和 `setter` 方法。下面的代码用于设置 `GifView` 的动画是否显示：

```

01 public void setPaused(boolean paused) {
02     this.mPaused = paused;
03     if (!paused) {
04         mMovieStart = android.os.SystemClock.uptimeMillis()
05             - mCurrentAnimationTime;
06     }
07     invalidate();
08 }
09
10 public boolean isPaused() {
11     return this.mPaused;
12 }

```

`setPaused()` 方法调用了 `invalidate()`，以保证 `View` 行为是可靠的。必须在改变这个可能改变外观的属性后废除这个 `View`，这样系统才知道需要重绘。同样，如果属性的变化可能影响尺寸或者 `View` 的形状，则需要请求一个新的布局。

自定义控件需要支持和重要事件交流的事件监听器。例如，`GifView` 实现了如下的 3 个方法来监听事件：

```

01 @Override
02 public void onScreenStateChanged(int screenState) {
03     super.onScreenStateChanged(screenState);
04     mVisible = screenState == SCREEN_STATE_ON;
05     invalidateView();
06 }
07
08 @Override
09 protected void onVisibilityChanged(View changedView, int visibility) {
10     super.onVisibilityChanged(changedView, visibility);
11     mVisible = visibility == View.VISIBLE;
12     invalidateView();
13 }

```

```

14
15 @Override
16 protected void onWindowVisibilityChanged(int visibility) {
17     super.onWindowVisibilityChanged(visibility);
18     mVisible = visibility == View.VISIBLE;
19     invalidateView();
20 }

```

## 5. 处理布局事件

为了正确地绘制自定义控件，需要知道它的大小。复杂的自定义控件经常需要根据它的大小和在屏幕上的图形区域执行多次布局计算。虽然 View 类有很多处理尺寸大小的方法，但是大部分方法需要重载。如果控件不需要特别控制它的大小，只需要重载方法 `onSizeChanged()`。`onSizeChanged()` 在控件第一次分配大小时调用，如果控件因为任何原因改变了大小也会再次调用。在该方法里计算位置、大小和其他一些与视图大小相关的值，而不是每次绘制时重新计算。下面是 GifView 处理计算的方法 `onLayout()` 中的代码片段：

```

01 @Override
02 protected void onLayout(boolean changed, int l, int t, int r, int b) {
03     super.onLayout(changed, l, t, r, b);
04     mLeft = (getWidth() - mMeasuredMovieWidth) / 2f;
05     mTop = (getHeight() - mMeasuredMovieHeight) / 2f;
06     mVisible = getVisibility() == View.VISIBLE;
07 }

```

如果需要良好地控制控件的布局参数，需要实现 `onMeasure()` 方法。这个方法的参数是 `View.MeasureSpec` 值，告诉控件的父元素想让控件多大，并且告诉这个大小是否是最大值或只是一个建议。作为优化，这些值保存为整数的封装类型，可以用 `View.MeasureSpec` 里的静态方法解析每个整数里面的信息。下面是 GifView 实现 `onMeasure()` 的示例：

```

01 @Override
02 protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
03     if (mMovie != null) {
04         int movieWidth = mMovie.width();
05         int movieHeight = mMovie.height();
06         int maximumWidth = MeasureSpec.getSize(widthMeasureSpec);
07         float scaleW = (float) movieWidth / (float) maximumWidth;
08         mScale = 1f / scaleW;
09         mMeasuredMovieWidth = maximumWidth;
10         mMeasuredMovieHeight = (int) (movieHeight * mScale);
11         setMeasuredDimension(mMeasuredMovieWidth, mMeasuredMovieHeight);
12     } else {
13         setMeasuredDimension(getSuggestedMinimumWidth(),
14                             getSuggestedMinimumHeight());
15     }
16 }

```

`onMeasure()` 方法没有返回值，相反，它通过调用 `setMeasureDisimension()` 方法传递结

果。调用 `onMeasure()` 方法是强制的，如果省略该方法，`View` 类会抛出 `Runtime Exception`。

## 6. 绘制控件

一旦有了创建的对象和定义了测绘布局的代码，就可以实现方法 `onDraw()`。例如：

```
01 @Override
02 protected void onDraw(Canvas canvas) {
03     if (mMovie != null) {
04         if (!mPaused) {
05             updateAnimationTime();
06             drawMovieFrame(canvas);
07             invalidateView();
08         } else {
09             drawMovieFrame(canvas);
10         }
11     }
12 }
```

## 7. 处理交互事件

绘制好控件后，还需要响应用户的输入。与其他许多 UI 框架一样，Android 也支持输入事件模型。用户的操作会转变成触发回调方法的事件，可以重载该回调方法来定制应用如何响应用户。在 Android 系统中最常用的输入事件是触摸事件，它会触发 `onTouchEvent(android.view.MotionEvent)`，重载该方法可以处理该事件，例如：

```
01 @Override
02 public boolean onTouchEvent(MotionEvent event) {
03     return super.onTouchEvent(event);
04 }
```

## 8.3 重载控件

重载控件是一种更为简单的自定义控件的方式，如果已经存在一个与需求十分相似的控件，可以继承它并按需重载该控件的某些行为。这种自定义控件的方式在移动 APP 开发中变得十分流行。

`ViewFlow`<sup>②</sup> 是 Android 平台上一个视图切换的效果库，相当于 Android UI 部件提供水平滚动的 `ViewGroup`，使用 `Adapter` 进行条目绑定。图 8-1 展示了 `ViewFlow` 的设计效果。



图 8-1 View Flow 的设计效果

### 8.3.1 重构 `AdapterView`

下面以 `ViewFlow` 为例，介绍重载控件的方法和步骤。

② 项目主页 <https://github.com/pakerfeldt/android-viewflow>。

## 1. 拓展 AdapterView<Adapter> 类

ViewFlow 继承自 AdapterView<Adapter>, 代码如下:

```

01 public class ViewFlow extends AdapterView<Adapter> {
02     ...
03     public ViewFlow(Context context) {
04         super(context);
05         mSideBuffer = 3;
06         init();
07     }
08
09     public ViewFlow(Context context, int sideBuffer) {
10         super(context);
11         mSideBuffer = sideBuffer;
12         init();
13     }
14
15     public ViewFlow(Context context, AttributeSet attrs) {
16         super(context, attrs);
17         TypedArray styledAttrs = context.obtainStyledAttributes(attrs,
18             R.styleable.ViewFlow);
19         mSideBuffer = styledAttrs.getInt(R.styleable.ViewFlow_sidebuffer,
20             3);
21         init();
22     }
23 }

```

构造方法中的 AttributeSet 用于初始化 ViewFlow 的属性, 其中的 init() 方法用于初始化 ViewFlow 子视图等相关对象。

## 2. 核心方法重载

### (1) 重载布局方法

和定制控件一样, 也需要对重构控件的绘制进行重载, 包括:

- 重载 onMeasure(int, int)

核心代码如下:

```

01 @Override
02 protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
03     super.onMeasure(widthMeasureSpec, heightMeasureSpec);
04
05     final int width = MeasureSpec.getSize(widthMeasureSpec);
06     final int widthMode = MeasureSpec.getMode(widthMeasureSpec);
07     if (widthMode != MeasureSpec.EXACTLY) {
08         throw new IllegalStateException(
09             "ViewFlow can only be used in EXACTLY mode.");
10     }
11
12     final int heightMode = MeasureSpec.getMode(heightMeasureSpec);

```

```

13     if (heightMode != MeasureSpec.EXACTLY) {
14         throw new IllegalStateException(
15             "ViewFlow can only be used in EXACTLY mode.");
16     }
17
18     //The children are given the same width and height as the workspace
19     final int count = getChildCount();
20     for (int i = 0; i < count; i++) {
21         getChildAt(i).measure(widthMeasureSpec, heightMeasureSpec);
22     }
23
24     if (mFirstLayout) {
25         scrollTo(mCurrentScreen * width, 0);
26         mFirstLayout = false;
27     }
28 }

```

系统调用 `measure()` 方法时回调此方法，表明此时系统正在计算视图的大小。

- 重载 `onLayout(boolean, int, int, int, int)`

核心代码如下：

```

01 @Override
02 protected void onLayout(boolean changed, int l, int t, int r, int b) {
03     int childLeft = 0;
04
05     final int count = getChildCount();
06     for (int i = 0; i < count; i++) {
07         final View child = getChildAt(i);
08         if (child.getVisibility() != View.GONE) {
09             final int childWidth = child.getMeasuredWidth();
10             child.layout(childLeft, 0, childLeft + childWidth,
11                 child.getMeasuredHeight());
12             childLeft += childWidth;
13         }
14     }
15 }

```

系统调用 `layout()` 方法时回调此方法，表明此时系统正在给 `ViewFlow` 的子视图分配空间，该方法必须在 `onMeasure()` 之后回调。

## (2) 重载适配器方法

`ViewFlow` 继承自 `AdapterView<Adapter>`，因此，有必要对其中的适配器方法进行重载，包括：

- 重载 `getAdapter()/setAdapter()` 方法
- 重载 `getSelectedView()` 方法
- 重载 `setSelection(int)` 方法

核心代码如下：



```

01 @Override
02 public void setSelection(int position) {
03     if (mAdapter == null || position >= mAdapter.getCount())
04         return;
05
06     ArrayList<View> recycleViews = new ArrayList<View>();
07     View recycleView;
08     while (!mLoadedViews.isEmpty()) {
09         recycleViews.add(recycleView = mLoadedViews.remove());
10         detachViewFromParent(recycleView);
11     }
12
13     for (int i = Math.max(0, position - mSideBuffer); i < Math.min(
14         mAdapter.getCount(), position + mSideBuffer + 1); i++) {
15         mLoadedViews.addLast(makeAndAddView(i, true,
16             (recycleViews.isEmpty() ? null : recycleViews.remove(0))));
17         if (i == position)
18             mCurrentBufferIndex = mLoadedViews.size() - 1;
19     }
20     mCurrentAdapterIndex = position;
21
22     for (View view : recycleViews) {
23         removeDetachedView(view, false);
24     }
25     requestLayout();
26     setVisibleView(mCurrentBufferIndex, false);
27     if (mViewSwitchListener != null) {
28         if (mIndicator != null) {
29             mIndicator.onSwitched(mLoadedViews.get(mCurrentBufferIndex),
30                 mCurrentAdapterIndex);
31         }
32         mViewSwitchListener
33             .onSwitched(mLoadedViews.get(mCurrentBufferIndex),
34                 mCurrentAdapterIndex);
35     }
36 }

```

### (3) 重载触摸事件处理方法

这里主要针对 onTouchEvent (MotionEvent) 方法进行了重载，核心代码如下：

```

01 @Override
02 public boolean onTouchEvent(MotionEvent ev) {
03     ...
04     if (getChildCount() == 0)
05         return false;
06
07     if (mVelocityTracker == null) {
08         mVelocityTracker = VelocityTracker.obtain();
09     }
10     mVelocityTracker.addMovement(ev);
11 }

```

```

12     switch (action) {
13     case MotionEvent.ACTION_DOWN:
14         if (!mScroller.isFinished()) {
15             mScroller.abortAnimation();
16         }
17
18         mLastMotionX = x;
19         mTouchState = mScroller.isFinished() ? TOUCH_STATE_REST
20             : TOUCH_STATE_SCROLLING;
21         break;
22     ...
23     case MotionEvent.ACTION_CANCEL:
24         mTouchState = TOUCH_STATE_REST;
25     }
26     return true;
27 }

```

onTouchEvent() 方法主要处理滑屏时的指示器计算，即如何显示另一个子视图。

### 8.3.2 应用控件

下面简单介绍对重构控件 ViewFlow 的应用步骤。

①在布局文件中加入对 ViewFlow 的声明，例如：

```

01 <cn.liweiyong.view.ViewFlow
02     android:id="@+id/viewflow"
03     android:layout_width="fill_parent"
04     android:layout_height="fill_parent"
05     android:duplicateParentState="true" >

```

②在 Activity 的 onCreate() 方法中绑定 ViewFlow，并为其设置适配器，例如：

```

01 viewFlow = (ViewFlow) findViewById(R.id.viewflow);
02 ViewFlowAdapter adapter = new ViewFlowAdapter(this);
03 viewFlow.setAdapter(adapter);

```

其中的 ViewFlowAdapter 是一个继承自 BaseAdapter 的适配器。

③ ViewFlow 通常会和一个实现了 ViewSwitchListener 的 FlowIndicator 一起使用，以响应滑屏事件发生时的子视图切换。因此，需要实现一个实现 FlowIndicator 的 TitleFlowIndicator，其中的核心方法是 setViewFlow(ViewFlow)，代码如下：

```

01 @Override
02 public void setViewFlow(ViewFlow view) {
03     viewFlow = view;
04     invalidate();
05 }

```

④在 Activity 中绑定 TitleFlowIndicator，并设置适配器，例如：

```

01 TitleFlowIndicator indicator = (TitleFlowIndicator)
02     findViewById(R.id.viewflowindic);
03 indicator.setTitleProvider(adapter);

```

⑤最后，为 ViewFlow 绑定 TitleFlowIndicator，例如：

```

01 viewFlow.setFlowIndicator(indicator);

```

这样，便实现了一个标题和内容联动的滑屏容器。

## 8.4 绘制 UI

在 Android 中，可以使用 Paint 类提供的方法在 Canvas 中绘制个性化的 UI 界面。绘图分为 2D 和 3D 两种，2D 是由 Skia 实现的，也就是框架图上的 SGL。下面以一个圆形路径文字的绘制为例，介绍使用 Skia 绘图的方法和步骤。

①定义一个继承自 View 的类并实现其构造方法，例如：

```

01 public class GraphicsView extends View {
02
03     public GraphicsView(Context context) {
04         super(context);
05         ...
06     }

```

②重载 View 的 onDraw() 方法，例如：

```

01 @Override
02 protected void onDraw(Canvas canvas) {
03     canvas.drawPath(circle, cPaint);
04     canvas.drawTextOnPath(QUOTE, circle, 0, 20, tPaint);
05 }

```

需要对 onDraw() 方法中的 Canvas 和 Paint 进行初始化，例如：

```

01 Paint cPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
02 cPaint.setStyle(Paint.Style.STROKE);
03 cPaint.setColor(Color.LTGRAY);
04 cPaint.setStrokeWidth(3);

```

③应用 View 实例，例如：

```

01 public class MainActivity extends Activity {
02     @Override
03     public void onCreate(Bundle savedInstanceState) {
04         super.onCreate(savedInstanceState);
05         setContentView(new GraphicsView(this));
06     }
07 }

```

运行效果如图 8-2 所示。

在绘制 UI 的过程中，有两个重要的类 Canvas 和 Paint。

Canvas 类代表可以在其上绘图的画布。Canvas 类主要实现屏幕的绘制过程，其中包含很多实用方法，例如绘制线条、矩形、圆及其他任意图形。

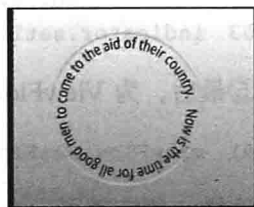


图 8-2 Graphics View 运行效果

Canvas 的主要方法包括：

- ① drawRect(RectF rect, Paint paint): 绘制区域。
- ② drawPath(Path path, Paint paint): 绘制一个路径。
- ③ drawBitmap(Bitmap bitmap, Rect src, Rect dst, Paint paint): 绘制贴图。
- ④ drawLine(float startX, float startY, float stopX, float stopY, Paint paint): 绘制直线。
- ⑤ drawPoint(float x, float y, Paint paint): 绘制点。
- ⑥ drawText(String text, float x, float y, Paint paint): 渲染文本。
- ⑦ drawTextOnPath(String text, Path path, float hOffset, float vOffset, Paint paint): 在路径上绘制文本。

Canvas 是一块画布，而 Paint 类就是绘画的工具，Paint 类包含样式和颜色以及绘制几何形状、文本和位图的其他信息，决定了被绘制对象在 Canvas 中如何显示。

Paint 类的常用方法有：

- ① setARGB(int a, int r, int g, int b): 设置 Paint 对象颜色。
- ② setAlpha(int a): 设置 alpha 不透明度，范围为 0~255。
- ③ setAntiAlias(boolean aa): 是否抗锯齿。
- ④ setColor(int color): 设置颜色。
- ⑤ setTextSize(float text Size): 设置字体大小。
- ⑥ Shader(shader): 设置阴影。
- ⑦ setTextAlign(Paint.Align align): 设置文本对齐。

## 8.5 开源 UI 工具

以下 10 款 Android 移动 APP 开发工具是开发者广泛应用的开源 UI 设计工具：

- ActionBarSherlock: 一个独立的动作栏设计库，设计效果如图 8-3 所示。
- NineOldAndroids: 实现非常复杂的动画效果。
- PullToRefresh: 下拉列表即可刷新当前页面内容的效果。
- ProgressWheel: 一款能够取代 Android 原生 Indeterminate 式、可显示具体进度的滚动式进度条。
- ViewPagerIndicator: 基于 PatrikAkerfeld, 兼容 ViewPager 和 ActionBarSherlock 的 Android 分页指针小部件。

- **AndroidUniversalImageLoader**: 一款为 Android 量身打造的开源 UI 组件。
- **ColorPicker**: Android 平台的颜色拾取器。
- **SegmentedRadioButton**: 一款可以在 Android 上实现 iOS 上分段控制效果的 UI 工具。
- **PhotoView**: 支持通过单点 / 多点触摸来进行图片缩放。
- **SmartImageView**: 取代 Android 自带的 ImageView 组件。

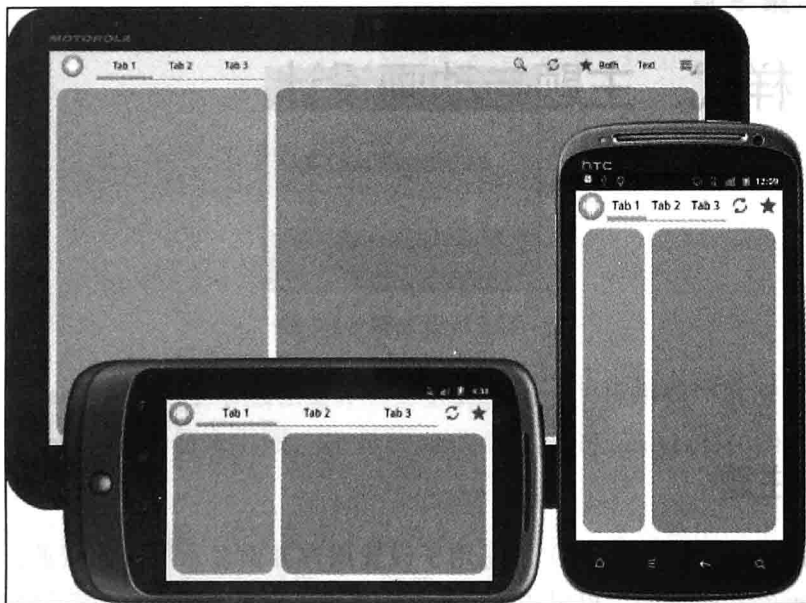
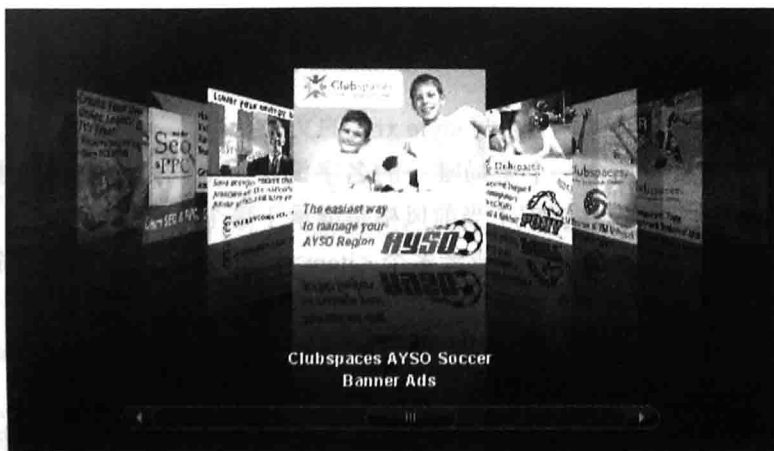


图 8-3 ActionBarSherlock 设计效果

## 8.6 习题

设计一个继承自 Gallery 的自定义组件，实现下图所示的 UI 效果。



## 样式、主题与动画设计

### 9.1 样式与主题

样式 (Style) 和主题 (Theme) 都是用于设置用户界面显示风格的资源, `themes.xml` 和 `style.xml` 位于 `res/values` 包中。可以用 Android 提供的一些默认的 Style 和 Theme 资源, 也可以自定义 Style 和 Theme 资源。

#### 9.1.1 Style

Style 是一个包含一种或者多种格式化属性的集合, 针对的是窗体中的元素, 可以将其作为一个单位用在布局 `.xml` 的单个元素当中, 用于改变指定控件或者布局的样式。Style 可以指定诸如高度、补白、字体颜色、字体大小、背景颜色等属性。

##### 1. 设计 Style

设计 Style 的一般步骤如下:

- ①在 `res/values` 目录下新建一个名叫 `style.xml` 的文件。
- ②使用 `<style>` 元素并结合一个全局唯一的名字设计一个风格或主题, 通过这个名字来应用风格, 而可选择的父类属性标识了当前风格是继承于哪个风格。
- ③在 `<style>` 元素内部, 声明一个或者多个 `<item>`, 每个 `<item>` 定义一个名字属性, 并且在元素内部定义该风格的值。
- ④在 XML 布局中使用定义的 Style 资源。

例如, 图 9-1 所示对话框的风格就是一个自定义的样式。

图 9-1 所示对话框中的两个 `TextView` 和 `Button` 均使用了样式设计, 其中的标题文字样

式定义如下：

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <resources xmlns:android="http://schemas.android.com/apk/res/android">
03     <style name="text_18_ffffff">
04         <item name="android:textSize">18.0dip</item>
05         <item name="android:textColor">#ffffff</item>
06     </style>
07 </resources>
```

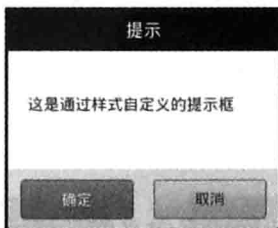


图 9-1 样式设计效果

每个 `<resources>` 元素的子节点在编译时都被转换为一个应用程序资源对象，可以通过 `<style>` 元素的 `name` 属性的值来引用。03 行的 `style` 可以通过 `@style/text_18_ffffff` 来在一个布局 XML 中引用。

`text_18_ffffff` 样式为标题文字设置了字体的大小和颜色。在布局中应用样式时就不需要再为标题控件 `TextView` 设置相应的属性了，例如：

```
01 <TextView
02     android:id="@+id/title"
03     style="@style/text_18_ffffff"
04     android:layout_width="fill_parent"
05     android:layout_height="40.0dip"
06     android:gravity="center"
07     android:text="@string/title_alert"
08     android:visibility="visible" />
```

而对 `Button` 样式的设计，主要是通过设置按钮获取焦点、被按下和释放时设置不同的按钮背景来实现，例如 `Button` 的样式设置如下：

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <resources xmlns:android="http://schemas.android.com/apk/res/android">
03     <style name="button">
04         <item name="android:background">@drawable/custom_button</item>
05     </style>
06 </resources>
```

其中的 `custom_button` 是位于 `res/drawable` 中的 xml 文件，部分代码如下：

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <selector xmlns:android="http://schemas.android.com/apk/res/android">
```

```

03     <item android:state_enabled="false">
04         <shape>
05             <gradient android:angle="270"
06                 android:endColor="#76007B16"
07                 android:startColor="#766CAC83" />
08             <stroke android:width="3px"
09                 android:color="#032F28" />
10             <corners android:radius="7px" />
11             <size android:height="50px"
12                 android:width="250px" />
13         </shape>
14     </item>
15     ...
16 </selector>

```

## 2. 继承 Style

通过继承机制，可以利用已有的样式来定义新的样式。所定义的新的样式不仅拥有新定义的样式 Item，而且还同时拥有旧的样式 Item。一般来说，称已存在的用来派生新样式的样式为父 Style，新定义的样式称为子 Style。例如：

```

01 <style name="pickprof_guide_text">
02     <item name="android:textSize">16.0sp</item>
03     <item name="android:textColor">#ff333333</item>
04 </style>
05 <style name="pickprof_guide_text_small"
06     parent="@style/pickprof_guide_text">
07     <item name="android:textSize">13.0sp</item>
08 </style>

```

`<style>` 中的 `parent` 属性是可选的，用来指定另外一个 Style 的资源 ID，前者继承后者的所有属性。可以像 07 行那样覆写继承来的 Style 属性。06 行的 `@` 表明引用的资源是前边定义过的，这个资源可以是在前面的样式定义中，或者是 Android 平台内部的样式。如果资源前有 `?`，表明引用的资源的值在当前的主题当中定义过。

继承一般有两种形式，如果继承自 Android 内部的样式，一般用 `parent` 属性来表明，例如：

```

01 <style name="XDialog" parent="@android:Theme.Dialog">
02     <item name="android:windowBackground">@drawable/pop_frame</item>
03 </style>

```

如果要继承自定义的 Style，一般不需要通过 `parent` 属性，只要在 Style 的 `name` 中以需要继承的 Style 的 `name` 开始，后连接新的 Style 的 `name`，中间用 `“.”` 隔开。注意：这种方式只适用于自定义的 Style 继承。例如：

```

01 <style name="Animation" />
02 <style name="Animation.Dialog">
03     <item name="windowEnterAnimation">@anim/dialog_enter</item>
04     <item name="windowExitAnimation">@anim/dialog_exit</item>
05 </style>

```



类似这样的继承可以有很多次，只要更换“.”前的名称即可。

### 9.1.2 Theme

Theme 是一种使 Android 应用保持统一风格的机制，是一个包含一种或者多种格式化属性的集合，Theme 资源的设计和 Style 一样，不同的是，Theme 针对的是整个 Activity 或整个应用程序的 UI，而 Style 针对的是某一个单独的 View。当一个 Style 资源被当作一个 Theme 来应用时，此 Activity 或应用程序中的每个 View 都将会应用其所能支持的每个 Style 属性。

图 9-2 是一个应用 Android 4.4 默认样式和应用了 customTheme 样式的应用设置界面的对比。

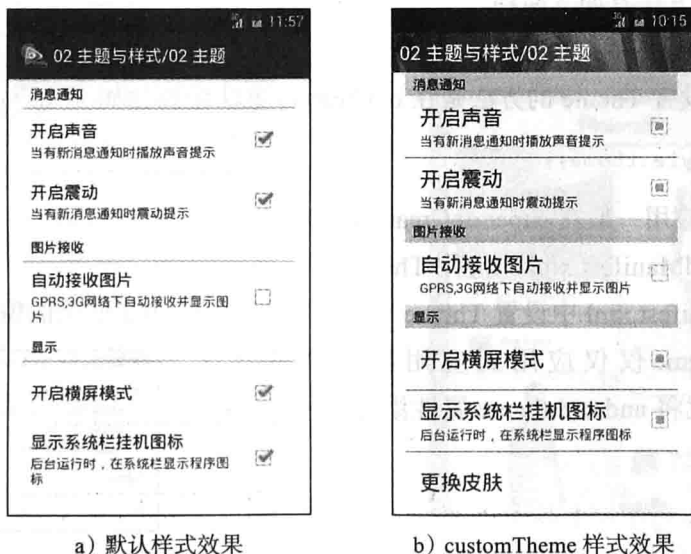


图 9-2 主题设计效果

#### 1. 设计 Theme

设计 Theme 的方法和设计 Style 的方法一样，首先需要在 res/values 目录下新建一个名叫 theme.xml 的文件，然后设计和 Style 类似的一些样式属性。例如：

```

01 <style name="Default.NoTitleBar"
02     parent="@android:style/Theme.Light.NoTitleBar">
03     <item name="android:textColorPrimaryInverse">@android:color/black />
04     <item name="android:windowBackground">@color/window_bg />
05     <item name="android:windowContentOverlay">@null />
06     <item name="android:windowTitleSize">42.0dip />
07     <item name="android:windowTitleStyle">@style/CustomWindowTitleText />
08     <item name="android:windowTitleBackgroundStyle">
09         @style/CustomWindowTitleBackground />

```

```

10     <item name="android:checkboxStyle">@style/customCheckBox />
11     <item name="android:listViewStyle">@style/customListView />
12     </style>
13     <style name="customTheme" parent="style/Default.NoTitleBar">
14         <item name="android:windowNoTitle">false />
15     </style>

```

注意，一些 Style 属性只能被当作一个 Theme 来应用，而不受任何 View 元素的支持。例如那些用于隐藏应用程序标题、隐藏状态栏或改变 window 的背景的 Style 属性。这些 Style 属性不属于任何 View 对象。例如，windowNoTitle 和 windowBackground 只有在作为 Theme 应用于一个 Activity 或应用程序时才是有效的 Style 属性。

## 2. 应用 Theme

应用 Theme 的方法有如下两种。

### (1) 在 Activity 中应用 Theme

在 Activity 中设置 Theme 的方法是在 onCreate() 方法中添加如下代码：

```
setTheme(R.style.theme);
```

注意，Theme 应用一般在 super.onCreate() 前设置。

### (2) 在 AndroidManifest.xml 中应用 Theme

在 AndroidManifest.xml 中设置 Theme 时，可以使用如图 9-3 所示的窗口。

如果希望 Theme 仅仅应用到应用程序中的某个 Activity 上，那么就将 android:theme 属性添加到 <activity> 标签中。例如：

```

01 <activity android:name=".themeTest"
02     android:theme="@style/theme"
03     android:label="@string/app_name">
04     <intent-filter>
05         ...
06     </intent-filter>
07 </activity>

```

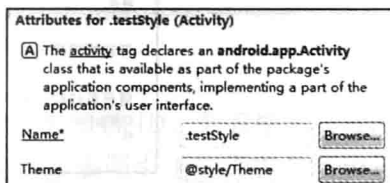


图 9-3 设置 Theme

## 3. Android 中的 Theme 和 Style

正像 Android 提供的其他内建资源一样，Android 平台提供了大量的 Style 和 Theme 供应用程序使用，而不用自己编写它们。可以在 R.style 类中找到所有可用的 Style。

Android 为在 Ice Cream Sandwich 版本的 SDK 上设计的应用提供了 3 种系统主题：

- 浅色 Holo 主题
- 深色 Holo 主题
- 浅色 Holo 主题配搭深色的操作栏（如图 9-4 所示）

以下是 Android 系统本身的 themes.xml 中比较常见的属性：

```

01 <item name="windowBackground">@android:drawable/screen_background_dark
02 </item>
03 <item name="windowFrame">@null</item>
04 <item name="windowNoTitle">false</item>
05 <item name="windowFullscreen">false</item>
06 <item name="windowIsFloating">false</item>
07 <item name="windowContentOverlay">@android:drawable/title_bar_shadow
08 </item>
09 <item name="windowTitleStyle">@android:style/WindowTitle</item>
10 <item name="windowTitleSize">25dip</item>
11 <item name="windowTitleBackgroundStyle">
12 @android:style/WindowTitleBackground </item>
13 <item name="android:windowAnimationStyle">
14 @android:style/Animation.Activity </item>

```

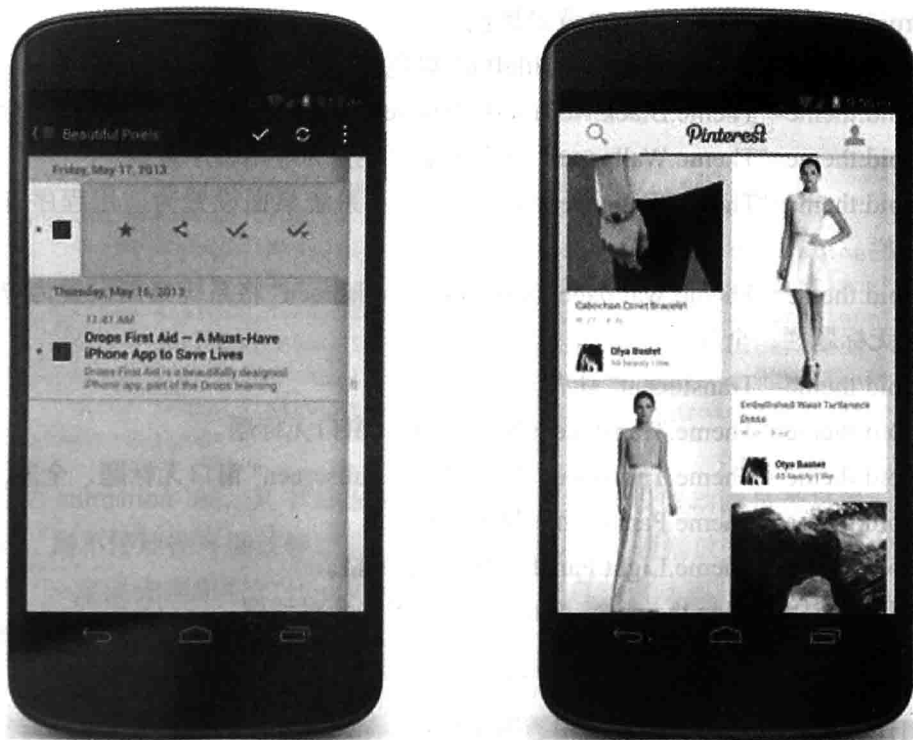


图 9-4 浅色 Holo 主题示例

可以通过类似 "@android:style/Theme.NoTitleBar" 的方法应用 Theme\_NoTitleBar theme 等主题。如果准备扩展一个 Theme，可以将其作为自定义 Theme 的 parent。例如，可以修改传统的对话框 Theme，以使用自己的背景图像：

```

01 <style name="CustomDialogTheme" parent="@android:style/Theme.Dialog">
02     <item name="android:windowBackground">
03         @drawable/custom_dialog_background</item>
04 </style>

```

然后在 Android Manifest 中使用 CustomDialogTheme 代替 Theme.Dialog, 例如:

```
01 <activity android:theme="@style/CustomDialogTheme">
```

系统自带样式包括:

- android:theme="@android:style/Theme.Dialog" 将一个 Activity 显示为对话框模式。
- android:theme="@android:style/Theme.NoTitleBar" 不显示应用程序标题栏。
- android:theme="@android:style/Theme.NoTitleBar.Fullscreen" 不显示应用程序标题栏, 并全屏。
- android:theme="Theme.Light" 背景为白色。
- android:theme="Theme.Light.NoTitleBar" 白色背景并无标题栏。
- android:theme="Theme.Light.NoTitleBar.Fullscreen" 白色背景, 无标题栏, 全屏。
- android:theme="Theme.Black" 背景黑色。
- android:theme="Theme.Black.NoTitleBar" 黑色背景并无标题栏。
- android:theme="Theme.Black.NoTitleBar.Fullscreen" 黑色背景, 无标题栏, 全屏。
- android:theme="Theme.Wallpaper" 将系统桌面设置为应用程序背景。
- android:theme="Theme.Wallpaper.NoTitleBar" 将系统桌面设置为应用程序背景, 且无标题栏。
- android:theme="Theme.Wallpaper.NoTitleBar.Fullscreen" 将系统桌面设置为应用程序背景, 无标题栏, 全屏。
- android:theme="Translucent" 让背景透明。
- android:theme="Theme.Translucent.NoTitleBar" 窗口无标题。
- android:theme="Theme.Translucent.NoTitleBar.Fullscreen" 窗口无标题、全屏。
- android:theme="Theme.Panel" 面板显示风格。
- android:theme="Theme.Light.Panel" 白板显示风格。

例如, 可以使用 Dialog theme 将 Activity 变得像一个对话框:

```
01 <activity android:theme="@android:style/Theme.Dialog">
```

或者想让背景变成透明, 那就使用透明主题:

```
01 <activity android:theme="@android:style/Theme.Translucent">
```

## 9.2 动画设计

Android 平台提供了一套完整的动画框架, 使得应用开发者可以用它来实现各种动画效果。例如, 按钮的弹入弹出效果、Activity 的切换动画、文本图片的旋转效果等。在 Android API 11 版本以前支持两种动画, 分别为补间动画和帧动画, 在 Android API 11 版本中新加入的动画叫属性动画。下面分别介绍这 3 种动画。

### 9.2.1 帧动画

帧动画是一种传统动画，它的设计思想是像放电影一样，在 `onDraw()` 方法中使用 `invalidate()` 方法不断刷新 View，以显示不同的图片序列。

设计帧动画的一般步骤如下：

①在项目的 `res/drawable` 目录下存放类似图 9-5 所示的帧动画 png 图像素材。

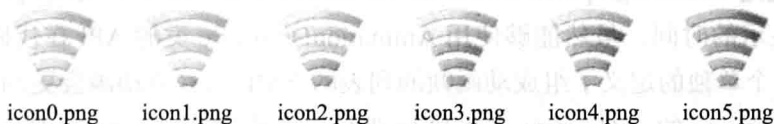


图 9-5 帧动画素材

②在 `res/drawable` 目录下新建帧布局文件，例如：

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <animation-list
03     xmlns:android="http://schemas.android.com/apk/res/android"
04     android:oneshot="true" >
05     <item android:drawable="@drawable/icon0" android:duration="150" />
06     <item android:drawable="@drawable/icon1" android:duration="150" />
07     <item android:drawable="@drawable/icon2" android:duration="150" />
08     <item android:drawable="@drawable/icon3" android:duration="150" />
09     <item android:drawable="@drawable/icon4" android:duration="150" />
10     <item android:drawable="@drawable/icon5" android:duration="150" />
11 </animation-list>
```

根标签为 `animation-list`，其中 `android:oneshot` 属性用于设置动画是否只展示一遍，如果设置为 `false`，则不停地循环播放帧动画。注意，不管动画是否播放完毕，如果想要第二次启动该帧动画，一定要先调用它的 `stop()` 方法才可以再次启动该动画。通过 `item` 标签对动画中的每一个图片进行声明，`android:duration` 表示展示所用的图片的时间长度。

③在布局文件中添加一个 `ImageView` 对象，并将帧布局文件作为其 `android:src` 属性的取值。

④触发帧动画。通过用户界面事件触发帧动画。例如，通过触摸事件触发，代码如下：

```
01 AnimationDrawable animation;
02
03 public void onCreate(Bundle savedInstanceState) {
04     super.onCreate(savedInstanceState);
05     setContentView(R.layout.main);
06     ImageView rocketImage = (ImageView) findViewById(R.id.rocket_image);
07     rocketImage.setBackgroundResource(R.drawable.rocket_thrust);
08     animation = (AnimationDrawable) rocketImage.getBackground();
09 }
10
11 public boolean onTouchEvent(MotionEvent event) {
```

```

12     if (event.getAction() == MotionEvent.ACTION_DOWN) {
13         animation.start();
14         return true;
15     }
16
17     return super.onTouchEvent(event);
18 }

```

帧动画是通过 `android.graphics.drawable.AnimationDrawable` 类来实现的，在该类中保存了帧序列以及显示的时间。虽然能够使用 `AnimationDrawable` 类的 API 在代码中定义动画的帧，但是，用一个单独的定义了组成动画帧的列表的 XML 来完成动画会更加简单。

注意，在 `Activity` 的 `onCreate()` 方法执行期间，不能调用 `AnimationDrawable` 对象上的 `start()` 方法，因为 `AnimationDrawable` 对象在这时还没有与窗口绑定。如果要立即播放动画，而不需要交互，可以在 `Activity` 的 `onWindowFocusChanged()` 回调方法中调用 `start()` 方法，`onWindowFocusChanged()` 方法会在窗口获取焦点时被 Android 系统调用。

如果希望帧动画播放完毕自动结束，可以在帧布局的最后添加如下的一个 `<item>`：

```

01 <item android:drawable="@android:id/empty" android:duration="150" />

```

## 9.2.2 补间动画

补间动画是一种通过指定 `View` 对象开始和结束的状态，然后通过系统自动生成需要显示的过渡效果（如移动、缩放、旋转、改变透明度等）的动画。即通过动画资源中预先定义一组指令，这些指令指定了图形变换的类型、触发时间、持续时间。

和帧动画一样，一般也通过在 `res/anim` 中创建动画 XML 资源来设计动画效果，下面是补间动画的框架：

```

01 <?xml version="1.0" encoding="utf-8"?>
02 <set xmlns:android="http://schemas.android.com/apk/res/android"
03     android:interpolator="@[package:]anim/interpolator_resource"
04     android:shareInterpolator=["true" | "false"] >
05     <alpha
06         android:fromAlpha="float"
07         android:toAlpha="float" />
08     <scale
09         android:fromXScale="float"
10         android:toXScale="float"
11         android:fromYScale="float"
12         android:toYScale="float"
13         android:pivotX="float"
14         android:pivotY="float" />
15     <translate
16         android:fromXDelta="float"
17         android:toXDelta="float"
18         android:fromYDelta="float"

```

```

19         android:toYDelta="float" />
20     <rotate
21         android:fromDegrees="float"
22         android:toDegrees="float"
23         android:pivotX="float"
24         android:pivotY="float" />
25     <set>
26         ...
27     </set>
28 </set>

```

动画资源文件必须包含一个根元素，这个元素既可以是一个单独的 `<alpha>`、`<scal>`、`<translate>`、`<rotate>` 的插值元素，也可以是拥有这些元素（包括 `<set>` 元素）组合的 `<set>` 元素。

但是补间动画只能应用于 View 对象，并且只支持一部分属性，如支持缩放旋转而不支持背景颜色的改变。

默认情况下，所有的动画指令都是同时发生的，为了让它们按顺序发生，需要设置一个特殊的属性 `startOffset`。动画的指令定义了想要发生什么样的转换，当它们发生时，应该执行多长时间，转换可以是连续的也可以是同时的。例如，让文本内容从左边移动到右边，然后旋转 180 度，或者在移动的过程中同时旋转，每个转换需要设置一些特殊的参数，如开始和结束的大小尺寸变化，开始和结束的旋转角度等，也可以设置一些基本的参数，如开始时间与周期。如果让几个转换同时发生，可以给它们设置相同的开始时间，如果按序列的话，计算开始时间加上其周期。

以下是补间动画的通用 XML 属性：

- ① `duration` 动画持续时间，时间以毫秒为单位。
- ② `fillAfter` 当设置为 `true`，该动画转化在动画结束后被应用，即动画结束时画面停留在最后一帧。
- ③ `fillBefore` 当设置为 `true`，该动画转化在动画开始前被应用，即动画结束时画面停留在第一帧。
- ④ `interpolator` 设置动画渲染器，有 3 种渲染器：
  - `accelerate_decelerate_interpolator` 动画加速减速器。
  - `accelerate_interpolator` 动画加速器。
  - `decelerate_interpolator` 动画减速器。
- ⑤ `repeatCount` 动画的重复次数。
- ⑥ `repeatMode` 动画重复的行为，默认值是 `restart`，如果设置为 `reverse`，表示偶数次显示动画时会做与动画文件定义的方向相反的动作。
- ⑦ `startOffset` 动画之间的时间间隔，从上次动画停多少时间开始执行下个动画。
- ⑧ `zAdjustment` 动画的 `ZOrder` 的改变，0 表示保持 `ZOrder` 不变，1 表示保持在最上层，-1 表示保持在最下层。

## 1. 移动补间动画

移动补间动画通过实现画面转换位置的移动而生成动画效果，例如：

```
01 <translate
02     android:repeatCount="2"
03     android:fromXDelta="-30"
04     android:fromYDelta="-30"
05     android:toXDelta="-80"
06     android:toYDelta="200"
07     android:duration="3000"
08 />
```

Translate 动画的常见属性有：

- ① fromXDelta 属性为动画起始时 X 坐标上的位置。
- ② toXDelta 属性为动画结束时 X 坐标上的位置。
- ③ fromYDelta 属性为动画起始时 Y 坐标上的位置。
- ④ toYDelta 属性为动画结束时 Y 坐标上的位置。

## 2. 缩放补间动画

缩放补间动画通过改变 View 对象的尺寸而生成动画效果，例如：

```
01 <scale
02     android:interpolator=
03         "@android:anim/accelerate_decelerate_interpolator"
04     android:repeatCount="1"
05     android:fromXScale="0.5"
06     android:fromYScale="0.5"
07     android:toXScale="1.4"
08     android:toYScale="1.4"
09     android:pivotX="50%"
10     android:pivotY="50%"
11     android:fillAfter="false"
12     android:duration="3000"
13 />
```

Scale 动画常见的属性有：

- ① fromXScale 动画起始时 X 坐标上的伸缩尺寸。
- ② toXScale 动画结束时 X 坐标上的伸缩尺寸。
- ③ fromYScale 动画起始时 Y 坐标上的伸缩尺寸。
- ④ toYScale 动画结束时 Y 坐标上的伸缩尺寸。

以上四种属性值，0.0 表示收缩到没有，1.0 表示正常无伸缩，值小于 1.0 表示收缩，值大于 1.0 表示放大。

- ① pivotX 动画相对于物件的 X 坐标的开始位置。
- ② pivotY 动画相对于物件的 Y 坐标的开始位置。

以上两个属性值从 0% ~ 100% 中取值，50% 为物件的 X 或 Y 方向坐标上的中点位置。



### 3. 旋转补间动画

旋转补间动画通过旋转 View 对象而生成动画效果, 例如:

```
01 <rotate
02     android:interpolator="@android:anim/accelerate_interpolator"
03     android:repeatCount="2"
04     android:fromDegrees="0"
05     android:toDegrees="+270"
06     android:pivotX="50%"
07     android:pivotY="50%"
08     android:duration="3000"
09 />
```

Rotate 动画的常见属性有:

- ① fromDegrees 动画起始时物件的角度。
- ② toDegrees 动画结束时物件旋转的角度, 可以大于 360°。

当角度为负数时表示逆时针旋转, 当角度为正数时表示顺时针旋转 (负数 from——to 正数: 顺时针旋转), (负数 from——to 负数: 逆时针旋转), (正数 from——to 正数: 顺时针旋转), (正数 from——to 负数: 逆时针旋转)。

### 4. 透明补间动画

透明补间动画通过修改 View 对象的透明度而生成动画效果, 例如:

```
01 <alpha
02     android:fromAlpha="0.1"
03     android:toAlpha="1.0"
04     android:duration="3000"
05 />
```

Alpha 动画的常见属性有:

- ① fromAlpha 动画起始时的透明度: 0.0 表示完全透明。
- ② toAlpha 动画结束时的透明度: 1.0 表示完全不透明。

### 5. 应用动画

应用动画之前, 需要使用 AnimationUtils.loadAnimation() 方法装载补间动画资源文件, 例如:

```
01 Animation animation = AnimationUtils.loadAnimation(this, R.anim.animfile);
02 animation.setRepeatCount(Animation.INFINITE); // 循环显示。
```

然后调用 View 控件的 startAnimation() 方法或 Animation 类的 start() 方法来启动动画。例如, 有一个 EditText 控件, 将补间动画资源文件中设置的补间动画应用到 EditText 控件上的方式有两种:

- ①使用 EditText 类的 startAnimation 方法:

```
01 editText.startAnimation(animation);
```

## ②使用 Animation 类的 start 方法:

```
01 editText.setAnimation(animation);  
02 animation.start();
```

## 6. 在 Activity 中定义补间动画

Android 中提供了与补间动画的 4 种动画类型相对应的类: AlphaAnimation、ScaleAnimation、TranslateAnimation 和 RotateAnimation, 每个子类都在父类的基础上增加了各自独有的属性。

下面的示例演示了在代码中定义补间动画的方法:

```
01 private Animation mAnimation_Alpha;  
02 private Animation mAnimation_Scale;  
03 private Animation mAnimation_Translate;  
04 private Animation mAnimation_Rotate;  
05 mAnimation_Alpha=new AlphaAnimation(0.1f, 1.0f);  
06 mAnimation_Scale =new ScaleAnimation(0.0f, 1.4f, 0.0f, 1.4f,  
07     Animation.RELATIVE_TO_SELF, 0.5f,Animation.RELATIVE_TO_SELF, 0.5f);  
08 mAnimation_Translate=new TranslateAnimation(30.0f, -80.0f, 30.0f, 300.0f);  
09 mAnimation_Rotate=new RotateAnimation(0.0f, +350.0f,  
10     Animation.RELATIVE_TO_SELF,0.5f,Animation.RELATIVE_TO_SELF,0.5f);
```



**注意** 不管动画如何移动或调整尺寸, 拥有动画的 View 对象的边界都不会自动地调整来适应变化, 即使动画超出了 View 对象的边界也不会被裁剪, 但是如果动画超出了它的父容器的边界, 那么它将会被裁剪。

### 9.2.3 属性动画

属性动画更改的是对象的实际属性, 而补间动画中改变的是 View 的绘制效果。例如, 对 Button 实施大小缩放的补间动画, 无论在补间动画中如何缩放 Button 的大小, Button 的有效单击区域还是原来的区域, 其位置与大小都不变。而如果对 Button 实施属性动画, 则改变的是 Button 的实际属性。

在属性动画中, 可以对动画应用以下特性:

- 持续时间: 动画的持续时间。默认长度是 300 毫秒。
- 时间插值: 该值能够作为计算当前动画运行时间的方法的属性值来指定, 它决定动画的变化频率。
- 重复次数和行为: 该属性能够指定在动画结束时是否重新播放动画, 以及重复播放的次数。还能够指定动画是否能够反向回播, 如果设置了反向回播, 那么动画就会先向前再向后重复播放, 直到达到播放次数。

- 动画集合：即可以同时为一个对象应用几个动画，这些动画可以同时播放也可以对不同动画设置不同开始偏移。
- 帧刷新延迟：指定动画帧的刷新频率。默认是每 10 秒钟刷新一次，但是应用程序最终的刷新帧的速度依赖于系统的繁忙程度以及系统能够提供的底层定时器的反应速度。

### 1. ValueAnimator

ValueAnimator 是用于处理动画属性值的主要属性动画时序引擎。它有所有的计算动画值的核心功能，并包含了每个动画的时序细节、动画是否重复的信息、监听接收更新事件和设置评估定制类型的能力。ValueAnimator 对象保持着动画的时间轨迹，如动画的运行时间及动画属性的当前值。

要启动一个动画，就要创建一个 ValueAnimator 对象，并且要给该对象设置想要的动画的属性的开始和结束值，以及动画的持续时间。在调用 start() 方法启动动画时，整个动画期间，ValueAnimator 对象会根据动画的持续时间和已经执行的时间比计算出一个时间因子（0~1），然后根据 TimeInterpolator 计算出另一个因子，最后 TypeAnimator 通过该因子计算出属性值。

因此，应用属性动画有两个步骤：

- ① 计算属性值，由 ValueAnimator 完成。
- ② 根据属性值执行相应的动作，如改变对象的某一属性。需要实现 ValueAnimator.OnUpdateListener 接口，例如：

```
01 ValueAnimator animation =ValueAnimator.ofFloat(0f, 1f);
02 animation.setDuration(1000);
03 animation.addUpdateListener(newAnimatorUpdateListener() {
04     @Override
05     public void onAnimationUpdate(ValueAnimator animation) {
06     }
07 });
08 animation.setInterpolator(newCycleInterpolator(3));
09 animation.start();
```

Animator.AnimatorListener 中含有下面四种操作：

- ① onAnimationStart(): 动画开始时被调用。
- ② onAnimationEnd(): 动画结束时被调用，它不管动画是如何结束的。
- ③ onAnimationRepeat(): 动画重复播放时被调用。
- ④ onAnimationCancel(): 动画被取消播放时被调用。

如果要在该动画结束后运行其他动画或者操作，那么需要在 AnimatorListener 接口的 onAnimationEnd 方法中完成。下面的例子是小球落到地面后又弹起的动画：

```
01 public void onAnimationEnd(Animation animation) {
```

```

02     if(animation.hashCode() == animationBottom.hashCode()){
03         imageView.startAnimation(animationTop);
04     }else if(animation.hashCode() == animationTop.hashCode()){
05         imageView.startAnimation(animationBottom);
06     }
07 }

```

`ValueAnimator.AnimatorUpdateListener` 中的 `onAnimationUpdate()` 回调方法用于监听属性的值更新时执行相应的操作，这是 `ValueAnimator` 必须要监听的事件。可以继承 `AnimatorListenerAdapter` 而不是实现 `AnimatorListener` 接口来简化操作，该类对 `AnimatorListener` 中的方法都定义了一个空方法体，这样就只需要定义想监听的事件而不用实现每个方法。例如：

```

01 ObjectAnimator oa=ObjectAnimator.ofFloat(tv, "alpha", 0f, 1f);
02 oa.setDuration(3000);
03 oa.addListener(new AnimatorListenerAdapter(){
04     public void onAnimationEnd(Animator animation){
05     }
06 });
07 oa.start();

```

`ValueAnimator` 封装了一个 `TimeInterpolator`，`TimeInterpolator` 定义了属性值在开始值与结束值之间的插值方法。`ValueAnimator` 还封装了一个 `TypeAnimator`，根据开始、结束值与 `TimeInterpolator` 计算得到的值计算出属性值。

## 2. ObjectAnimator

`ObjectAnimator` 继承自 `ValueAnimator`，用来指定一个对象及该对象的一个属性，当属性值计算完成时自动设置为该对象的相应属性，即完成了属性动画的全部两步操作。实际应用中一般都会用 `ObjectAnimator` 来改变某一对象的某一属性，但用 `ObjectAnimator` 有一定的限制（如在目标对象上需要指定用于呈现的 `accessor` 方法），要想使用 `ObjectAnimator`，应该满足以下条件：

①动画效果的属性必须有一个 `set<propertyName>` 格式的设置器方法。因为在动画处理期间，`ObjectAnimator` 对象会自动地更新对应的动画效果属性，所以它必须使用该设置器方法来访问对应的属性。例如，如果属性名是 `foo`，那么就需要有一个 `setFoo()` 方法。

②如果只在 `ObjectAnimator` 类的一个工厂方法中指定了一个 `values` 参数，那么该值会被假定为动画的结束值。因此，该对象的动画效果属性就必须要有个获取方法，用于获得动画的开始值。该获取方法必须使用 `get<propertyName>()` 格式。例如，属性是 `foo`，就必须有一个 `getFoo()` 方法。

③动画属性的获取和设置方法取决于 `ObjectAnimator` 对象的开始和结束值的数据类型。例如，构建一个 `ObjectAnimator` 对象：`ObjectAnimator.ofFloat(targetObject, "propName", 1f)`。因为 `targetObject` 属性的设置方法为 `targetObject.setPropName(float)` 和 `targetObject.getPropName(float)`，因此，`ObjectAnimator` 调用 `ofFloat()` 方法。

④根据属性或对象上的动画效果，可能需要调用 View 对象上的 `invalidate()` 方法，在更新动画效果时，强制屏幕重绘自己。在 `onAnimationUpdate()` 回调方法中做这件事情。例如，一个绘图对象的颜色属性的动画效果，在队形重绘自己时，才会将变化结果更新到屏幕上。在 View 对象上的所有属性的设置器，如 `setAlpha()` 和 `setTranslationX()` 会正确地让 View 对象失效。

如果上述条件不满足，则不能用 `ObjectAnimator`，而应该用 `ValueAnimator` 代替。

下面的代码把一个 `TextView` 的透明度在 3 秒内从 0 变至 1：

```
01 tv=(TextView)findViewById(R.id.textview1);
02 btn=(Button)findViewById(R.id.button1);
03 btn.setOnClickListener(new OnClickListener() {
04     @Override
05     public void onClick(View v) {
06         ObjectAnimator oa=ObjectAnimator.ofFloat(tv, "alpha", 0f, 1f);
07         oa.setDuration(3000);
08         oa.start();
09     }
10 });
```

根据应用动画的对象或属性的不同，可能需要在 `onAnimationUpdate` 方法中调用 `invalidate()` 方法刷新视图。

## 9.3 应用风格设计

本节主要介绍移动 APP 产品风格的设计技巧。

### 1. 唯一主色调

应用的主界面保持一个主色调，这样能够很好地表达界面层次、重要信息，并且能展现良好的视觉效果。现在越来越多唯一主色调风格的设计多采用简单的色阶，并配套灰阶来展现信息层次，但是绝不采用更多的颜色。卡塔尔航空公司就是这样的设计案例，如图 9-6 所示。整个界面以粉色作为主色调，从标题栏到标签页，从操作按钮到提示信息，除了黑白灰之外，全部采用粉色设计，这种简洁的配色风格，反倒起到了很好的信息传达效果，也具有良好的视觉表现力，设计师在内容排版上的技巧非常值得学习。

可以说唯一主色调的设计手法真正做到了移动端应用的最小化设计，减少冗余信息的干扰，使用户专注于主要信息的获取。

### 2. 多彩色风格

与唯一主色调形成对比的，就是 Metro（Metro 是微软在 Windows Phone 7 中正式引入的一种界面设计语言，也是 Windows 8 的主要界面显示风格）引领的多彩色风格，这种风格使得不同页面、不同信息组块采用撞色多彩色的方式来设计，甚至同一个界面的局部都可以采用多彩撞色，也产生了不少优秀的设计。

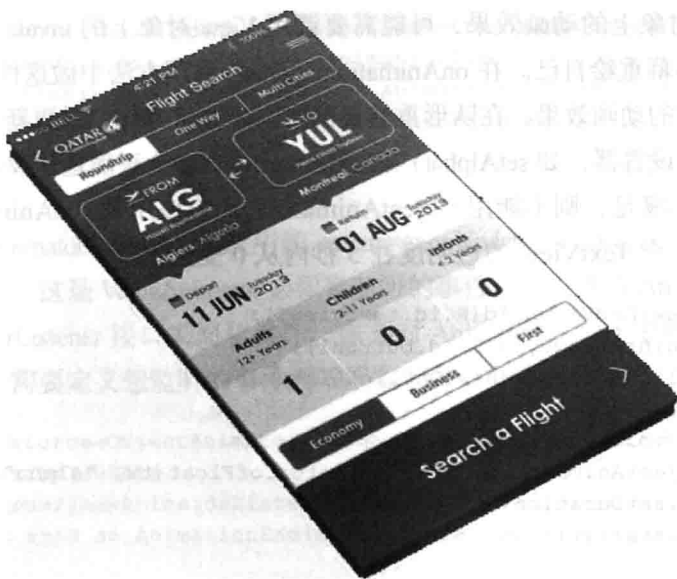


图 9-6 卡塔尔航空公司应用的主色调

优衣库出品的 RECIPE 是一个让人眼前一亮的设计案例，如图 9-7 所示。多彩色的设计风格融入到整个 APP 中，不论是切换标签页，还是在内容组块中滚动，都会变更不同的主题色。色彩切换时，还会有淡入淡出的效果，让切换变得自然而不生硬。RECIPE 的番茄钟计时器模块，会一边计时一边播放优美的美食背景音乐，并同时切换不同的主体颜色，随着主体颜色的变更，所有的前景文案、图片也会变更为该色系，再加上清晰度极高的美食图片，真的是视觉加听觉的双重享受。



图 9-7 RECIPE 应用的多彩色

可是这对于一些内容型的应用也许并不适用，例如 Google Keep 的多彩卡片，在内容阅读上反而会起到反效果。百度云记事本第一版设计也是多彩色的，但是后来考虑到文字记事比较多，为提供良好的文字阅读体验，还是把多彩色变为灰白色微质感的设计。

### 3. 数据可视化

现在，越来越多的应用开始尝试运用数据可视化、信息图表化来呈现信息，让界面上不仅仅是列表，还有更多直观的饼图、扇形图、折线型、柱状图等丰富的表达方式。虽然表面上看起来不是很难的事情，但是若真想实现，背后的复杂程度不容小觑。

墨迹天气采用了多种图形来表达天气的信息，如图 9-8 所示。用不同的天气图形来表示

天气的阴晴状况，用柱状图来表示温度的区间，用曲线图来表示温度的变化。

移动 APP 利用数据可视化，可以在更小的屏幕空间内，更立体化地展示内容。现今数据可视化不只是静态展现数据，用户希望通过互动及时获取数据流，动态数据可视化 (Dynamic Data Visualization) 将更加强调数据实时更新的图形，以及动态的图形化表达。

#### 4. 卡片化设计

卡片也是一种采用较多的设计语言形式，但无法考究这种卡片的设计是从 Metro 的 Tiles 流行起来的，还是从 Pinterest 的瀑布流流行起来的。总之，Google 的移动端产品设计已经全面卡片化了，甚至 Web 端也沿用了这种统一的设计语言。

Google Now 使用卡片式列表框架将用户需要的信息展示在首页，将搜索结果前置，省去输入、点击和页面跳转的步骤，让用户更快捷地获取所需要的信息，如图 9-9 所示。Google Now 的卡片流突出信息本身，用大图和标题文字吸引用户，强化了无尽浏览的体验。Google Now 中，有的卡片是用来做用户教育的，有的卡片是用来告知天气的，有的卡片是呈现联系人列表的，有的卡片是显示待办事项的。不同的卡片都遵循在一个统一宽度和样式的卡片内进行发挥和设计。既保证了卡片和卡片之间的独立性，又保证了服务和服务的统一化设计。

#### 5. 突出内容

移动 APP 产品的最终发展路径势必会与 Web 产品一样，突出内容，为用户提供更好的服务。与内容相比，所有的设计和包装，都不外乎是一种表现手法，而真正具有价值的应用，一定是内容取胜的。Facebook 用 190 亿美金收购了 Whatsapp，这个全是用控件搭建的应用，并不是因为它的设计多出众，而是因为它的服务足够有价值。

如图 9-10a 所示的 Artsy 应用的图片瀑布流完全没有用线和面来区分信息组块，而只是利用内容本身做排版，用户可以将注意力更集中于图片内容上。如图 9-10b 所示的 Prismatic 应用利用字体排版，尽可能将内容前置，弱化图标和操作，让用户集中注意力于内容阅读上。而图 9-10c 所示的 MR Porter 应用则利用商品图片、名称和价格直接做设计，让用户聚焦于商品本身。

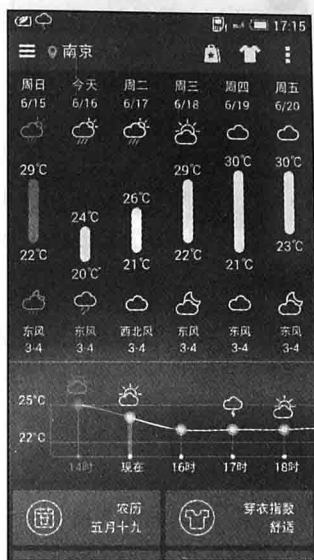


图 9-8 墨迹天气应用的图形化界面



图 9-9 Google Now 应用的卡片界面





图 9-10 应用界面示例

## 6. 圆形的运用

圆形是最容易让人觉得舒服的形状，尤其是在充满各种方框的手机屏幕内，增加一些圆润的形状点缀，立刻就会增加活泼的气息。起初 iPhone 的拨号数字键盘都是矩形设计，等到 iOS7，就变成了圆形，可以说是对传统电话的致敬，也可以说是增强了界面的柔和感。当然，也要相应地处理圆形的实际点触区域，不要因为设计成圆形而使点击区域也变小，导致点击准确率下降，美观度虽得到了提升易用性却受到了影响。

Tumblr 应用把要创建的内容的类型选择用蒙层加圆形选项按钮来设计，让选择变得专注而明确，又不那么死板，如图 9-11 所示。Beats Music 应用把喜欢的标签设计成了圆形，这就比普通的列表、矩形标签的感觉要好很多，更有趣且更具探索性。

## 7. 大视野背景图

不论是大屏电子设备还是汽车的全景天窗，甚至是落地的阳台玻璃，人们总在追求更大的显示区域和更佳的显示效果，大视野在同类产品中总是能带来更突出的体验，在移动 APP 中也是如此。用通栏的图片

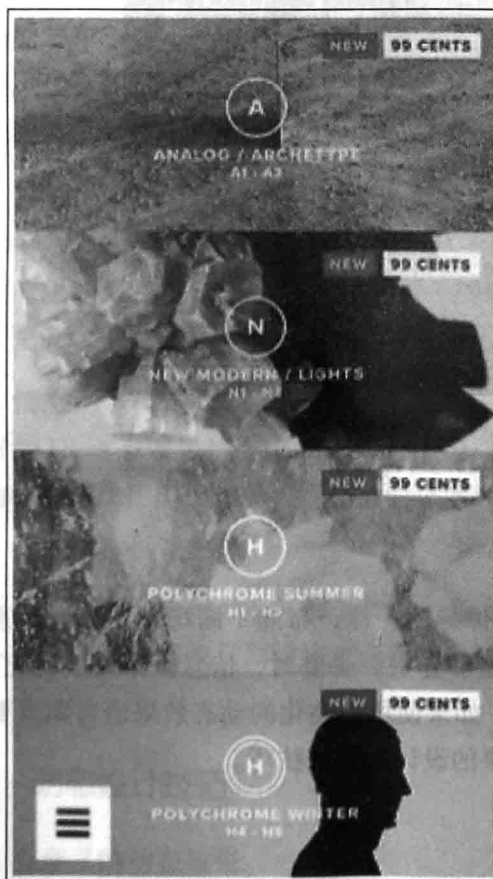


图 9-11 Tumblr 应用的界面

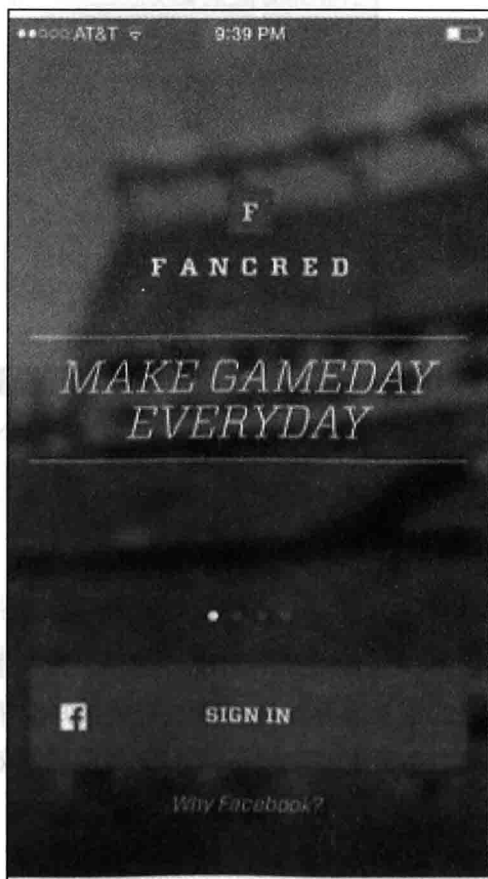


作为背景,也成为当前的一个流行趋势,或者是作为整个应用的背景,或者是作为内容区块的背景,既提升了视觉表现力度,又丰富了应用情感化元素。通过将一些信息或操作浮动在图片上的设计方法,对字体和排版设计要求更高,难度也更大,但极容易渲染出氛围。

大视野背景图风格也分为两种,一种像 Secret 应用那样的设计,如图 9-12a 所示。Secret 应用内容区块中采用大视野背景图,好处是可以利用图片做区块分割,难度是图片拼接后的效果不一定好看,所以可能还需要配合描边、留白等设计手段优化拼接。另外一种就是像 VSCO 应用的设计,如图 9-12b 所示,全屏背景图覆盖状态栏,前景做内容排版、导航、操作。这种设计非常具有生命力,所以也将这种风格叫有计划设计,或回归大自然的设计手法。但风险也是很明显的,即前景的信息排布设计有很大挑战,毕竟花花绿绿的背景太过于干扰注意力,导致前景的文字内容可读性变弱。所以需要将重要操作用明确的按钮区隔出来,阅读型文字与背景图要用明显的反色,或者将文字浮在半透明蒙层上,解决可读性问题。



a) Secret 应用界面



b) VSCO 应用界面

图 9-12 应用界面示例

## 8. 任务窗口模式

手机越来越大, 视野越来越广, 但在给用户带来更优越体验的同时也带来了烦恼。面对大屏手机单手操作和手指操作范围有限的劣势, 如何让用户灵活地操作成为设计师思考的问题; 而另一方面, 设计能否因为大屏带来体验上更大的突破呢?

设计师们通过悬浮窗口进行了很大胆的尝试, 利用小窗口镜像操作大屏幕, 甚至可以同时展示多窗口, 执行多任务, 这不但解决了大屏手机手指操作受限的问题, 而且拓展了大屏手机的功能, 使屏幕空间得以充分利用, 给大屏手机带来全新的交互体验。

多任务窗口模式跳出了传统手机单屏使用的思路, 例如三星 Galaxy Note 3 允许多窗口同时存在, 可以实现同屏显示两个应用, 如图 9-13 所示。例如, 同时显示网页浏览器和音乐播放器, 用户可以在主屏幕选择运行几个特定的任务。多任务窗口充分利用大屏手机的显示空间, 提升用户的操作效率。



图 9-13 GalaxyNote3 的多窗口界面

## 9.4 动态效果设计

设计师设计移动 APP 的动态效果时, 需要像导演一样, 布局不同场景中界面元素的登场方式, 不同元素之间的出现、运动、消失要足够连贯。必要时, 信息框架需要优化重构。动态效果的语义需要保持一致, 避免认知混淆。如果使用拟物化的动态效果语言则需要符合物理定律。本节主要介绍移动 APP 界面动态效果的设计原则和技巧。

### 9.4.1 动态设计原则

#### 1. 个性

这是 UI 动态效果设计中最基本的原则, 甚至可以说是动态效果设计的最高原则。UI 动

态效果设计就是要摆脱应用“开袋即食”的粗犷设定，设计独特的动态效果，创造引人入胜的效果。在确保 UI 风格一致性的前提下，表达出应用的鲜明个性，这就是 UI 动态效果设计“个性化”要做的事情。同时，还应令动态效果的细节符合那些约定俗成的交互规则，这样动态效果就具备了“可预期性”，用户不会有“出戏”的感觉，如此一来，UI 动态效果设计便有助于强化用户的交互经验，保持应用黏度。

## 2. 导向

动态效果应当通过使用体验安抚用户，令他们轻松愉悦。设计师需要将屏幕视作一个物理空间，将 UI 元素看作物理实体，它们能在这个物理空间中打开、关闭，任意移动、完全展开或者聚焦为一点。动态效果应当随动作移动而自然变化，不论是在动作发生前、过程中还是动作完成以后，为用户做出应有的引导。UI 动态效果就应该如同导游一样，为用户指引方向，防止用户感到无聊，减少额外的图形化说明。

## 3. 背景

动态效果应当为内容赋予背景，通过背景来表现内容的物理状态和所处环境。在摆脱模拟物品细节和纹理的设计束缚之后，UI 设计甚至可以自由地表现与环境设定矛盾的动态效果。为对象添加拉伸或者形变的效果，或者为列表添加俏皮的惯性滚动都不失为增加整体体验的有效手段。

## 4. 共鸣

动态效果应该具有直觉性和共鸣性。UI 动态效果的目的是与用户互动，并产生共鸣，而非令他们困惑甚至感到意外。UI 动态效果和用户操作之间的关系应该是互补的，两者共同促成交互完成。

## 5. 情感

好的 UI 动态效果是能够唤起积极的情绪反应的，平滑流畅的滚动能带来舒适感，而有效的动作执行往往能带来令人兴奋的愉悦感和快感。

## 6. 克制

滥用特效会让用户分心，应平衡好。动态效果是用来保持用户的关注点，引导用户操作的，不要为了有动态效果而设置动态效果。过度表现和过多的转场动画会令用户烦躁，甚至沮丧。

### 9.4.2 动态设计技巧

#### 1. 有意义的动态效果

动态效果可以在用户认知当中构成一种漂亮的、非“计算机化”的沟通范式。通过动态化的处理引导用户聚焦界面的关键部位，以使体验更加流畅。如果得以正确运用，便可以代

替通常需要大段文字才能阐述的含义。最简单而经典的例子就是内容飞入垃圾桶的动态效果，无需多做解释，用户完全理解其中的含义。图 9-14 演示的是 Yep! 应用设计效果。

## 2. 流畅过渡

移动 APP 越来越强调沉浸式的体验，过渡的流畅是用户对于动态效果的认识里最容易想到也最被认可的一点，通过界面及其元素的出现和消失，以及大小、位置和透明度的变化，使用户和产品的交互过程更流畅。转场动态效果（如图 9-15 所示）需要更加极致和平滑。平滑无缝的切换和体验，需要信息没有阻力的传递，用户需要集中注意力，专注于目标任务进行一系列操作。优先注重转场动态效果的移动产品会有极大的产品竞争力。

由于转场元素在整屏范围里移动，所以它们需要以协调的方式运动。起到引导视觉焦点作用的元素，其整个移动过程都要有意义、有秩序。随机的动画会分散注意力。如果转场的所有元素都协调得很好，用户对于该应用的理解也会增强。在设计转场动态效果时要考虑如下因素：

- 除非该动画是被限制在某一个坐标轴上或者是与其他元素一起随某个点协调移动，否则尽量避免线性路径。



图 9-14 Yep! 应用

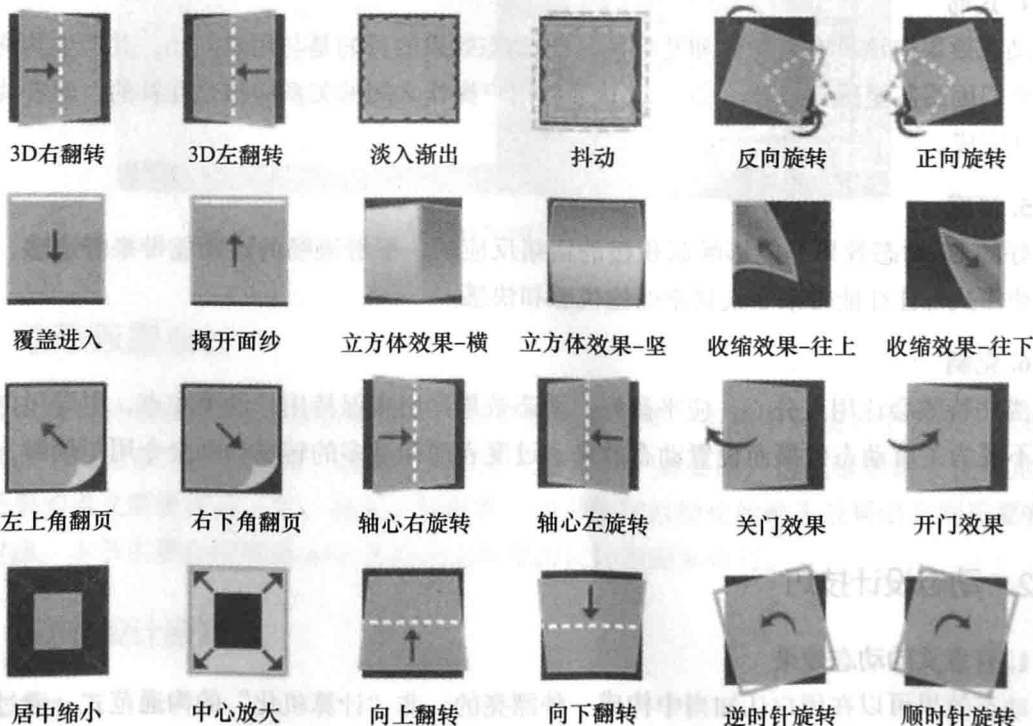


图 9-15 转场动态效果

- 确保元素移动的方向在整个转场过程中都是协调一致的。避免冲突的动作和重叠的运动路径。
- 如果所有运动的元素都在屏幕上按路径移动，需要让用户清楚地知道应该看哪里。
- 通过新旧元素的连贯性的动画来表现空间上的关系。
- 通过和谐一致的动画引导用户的注意力。
- 避免混乱不连贯的动画，元素以随机方向离开或进入会造成用户的困惑。

### 3. 反馈高效

高效的反馈可以说是移动 APP 最原始的需求，它通过动态效果让用户了解程序当前状态，同时对用户操作（平移、放大、缩小、删除）做出及时反馈。例如，在用户点击下载按钮后，需要给用户展示程序当前的状态（未下载、下载中、下载完成），如果不把反馈给用户，用户可能就觉得应用卡住了。同样地，对平移、放大等操作，及时、友好的反馈也是必要的。抖动是增强反馈的方法之一，用动效反馈替代图形文字的静态提示，更加自然和引人注意。图 9-16 演示了 91 桌面卸载应用的动态反馈效果。



图 9-16 91 桌面卸载应用的动态效果

### 4. 层级展现

随着移动 APP 越来越复杂，承载的功能越来越多，原来的三层结构原则已经不能完全适用，合理清晰的结构层级对用户理解应用和使用应用有着至关重要的作用。通过焦点缩

放、覆盖、滑出等动态效果帮助用户构建空间感受，例如利用动效使界面中的部分信息隐藏，当进行某些操作后隐藏的内容会动态展开，从而达到简化初始界面的目的，使界面简洁大气。图 9-17 演示的 AppFlow 应用的页面跳转就是这样。



图 9-17 AppFlow 应用页面跳转

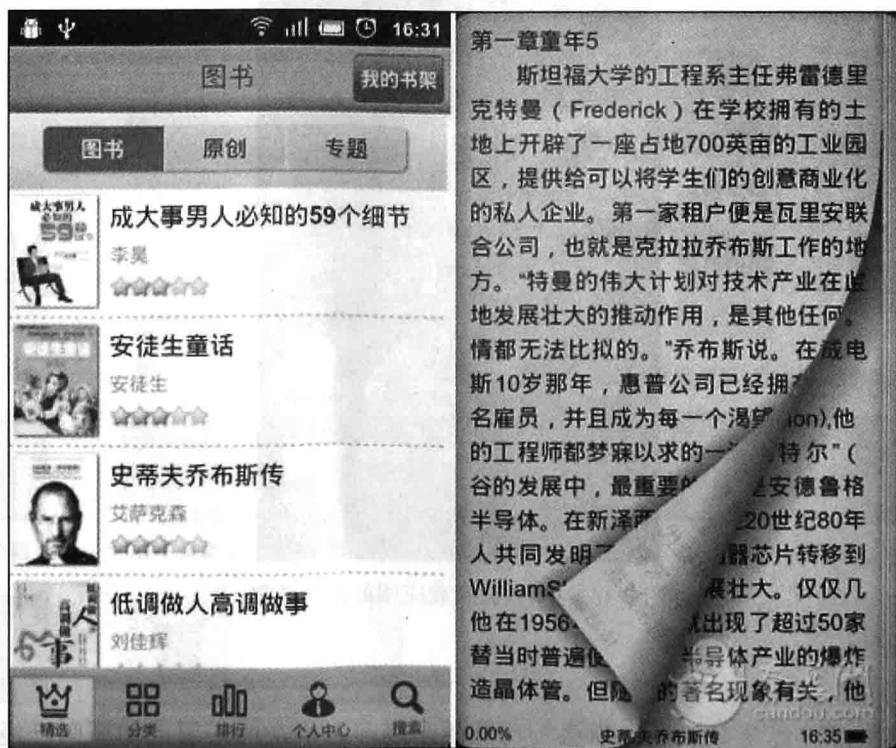


图 9-18 iReader 应用翻页的动态效果

## 5. 增强操纵

一些动态效果通过对现实世界进行模拟而减少不必要的提示,迎合用户的意识认知,使产品的交互方式更接近真实世界。用户通过对现实世界的认知来理解动态效果,增强了用户对应用的操纵感和带入感。图 9-18 所示的 iReader 应用翻页的设计,可以让用户感觉纸面在翻动。

## 9.5 习题

设计一个下图所示的界面切换动画。





## 桌面 UI 设计

### 10.1 设计简单的桌面组件

微型桌面应用程序 (App Widgets) 是一个可以嵌入到其他应用程序 (如主屏幕), 并能定期更新其视图的桌面小部件。一个能容纳其他 App Widgets 的应用程序, 称之为 App Widgets 宿主。图 10-1 包括了 Android 设备中常见的 App Widgets。



图 10-1 桌面 App Widgets 设计效果

本节通过实现一个日期桌面应用 (Month Calendar Widget<sup>①</sup>) 来介绍 App Widgets 的设计方

① 项目主页 <https://github.com/romannurik/Android-MonthCalendarWidget>。



法。日期桌面应用的设计效果如图 10-2 所示。

创建一个简单的 App Widgets 一般需要三个部分：桌面应用的布局、桌面应用的元数据描述以及桌面应用的事件处理。下面分别介绍这些任务的实现。

### 10.1.1 RemoteViews

为了让 App Widgets 能进行显示，需要为 App Widgets 提供一个布局文件。标准的 App Widgets 显示界面由三个组件组成：一个有界限的封装盒，一个框架和图形控制等元素。好的设计往往在封装盒和框架之间有一些留白，框架内边界和 Widget 控件间也有一些空白，如图 10-3 所示。

和 Activity 的布局一样，也需要把布局文件保存在 res/layout 目录下。但是 App Widgets 的布局文件必须在 XML 文件中定义。因为 App Widgets 的布局是基于 RemoteViews 对象的，所以它并不能支持所有的 View。

RemoteViews 类在 android.widget.RemoteViews 包下，是一个能够显示在其他进程中的远程视图。App Widgets 中的视图都是通过 RemoteViews 表现的。

在 RemoteViews 的构造方法中，通过传入布局文件的 ID 来获取布局文件对应的 RemoteViews 视图。然后，调用 RemoteViews 中的方法能对布局中的控件进行设置。例如，可以调用 `setTextViewText()` 来设置 TextView 控件的文本，可以调用 `setOnClickListener()` 来设置 Button 的单击响应事件等。

RemoteViews 支持的布局包括 `FrameLayout`、`LinearLayout`、`RelativeLayout` 和 `GridLayout`。支持的控件包括 `AnalogClock`、`Button`、`Chronometer`、`ImageButton`、`ImageView`、`ProgressBar`、`TextView`、`ViewFlipper`、`ListView`、`GridView`、`StackView` 和 `AdapterViewFlipper`。RemoteViews 不支持这些类的派生。

在本应用中，App Widgets 的布局包括两种，分别是：

- `loading.xml`：在首次将本桌面组件添加到桌面时，需要初始化一些日期相关数据。为了提高用户的体验，告诉用户需要进行等待，此时，AppWidgets 显示一个等待的对话框（通过 `ProgressBar` 来实现）。
- `widget.xml`：这是 AppWidgets 真正的显示界面，包括上下两个部分。上面通过 3 个按



图 10-2 Month Calendar Widget 设计效果

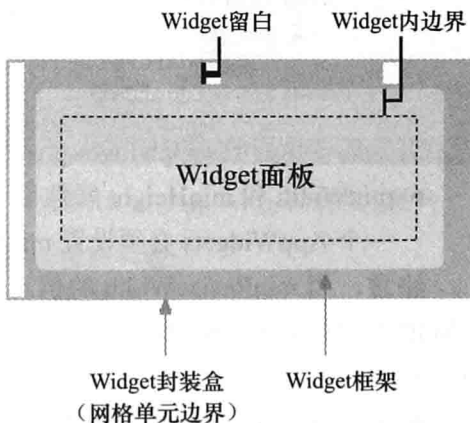


图 10-3 App Widgets 显示界面设计示意图

钮来实现月份的上下切换，下面是使用 TextView 实现的星期、日期文本。

布局文件的编写同一般用户界面的编写没什么区别，这里不再赘述。

### 10.1.2 AppWidgetProviderInfo

AppWidgetProviderInfo 用于定义 App Widgets 的基本属性，如显示的最小尺寸、初始布局资源、更新频率和 Configuration Activity 等。AppWidgetProviderInfo 在 res/xml 目录中定义。

AppWidgetProviderInfo 的定义必须在一个只有单一的 <appwidget-provider> 元素的 XML 资源文件中进行。Month Calendar Widget 的 AppWidgetProviderInfo 设计如下：

```

01 <appwidget-provider
02     xmlns:android="http://schemas.android.com/apk/res/android"
03     android:initialLayout="@layout/loading"
04     android:minHeight="180dp"
05     android:minResizeHeight="40dp"
06     android:minResizeWidth="180dp"
07     android:minWidth="180dp"
08     android:previewImage="@drawable/widget_preview"
09     android:resizeMode="vertical|horizontal"
10     android:updatePeriodMillis="1800000"
11     android:widgetCategory="keyguard|home_screen" />

```

以下是一些关于 <appwidget-provider> 属性的介绍：

- minWidth 和 minHeight 属性用于说明 App Widgets 在屏幕上至少要占用多大空间。每一个 App Widgets 必须设置 minWidth 和 minHeight，表明它默认状态下的最小空间。

注意，当 minResizeWidth 的值比 minWidth 大时，minResizeWidth 无效；当 resizeMode 的取值不包括 horizontal 时，minResizeWidth 无效；当 minResizeHeight 的值比 minHeight 大时，minResizeHeight 无效；当 resizeMode 的取值不包括 vertical 时，minResizeHeight 无效。

- updatePeriodMillis 属性用于说明 App Widgets 框架请求 AppWidgetProvider 的 onUpdate() 方法来更新 App Widgets 的频率。但是无法保证应用严格按照该频率更新数据，一般来说，应尽量减少更新的频率。有时可能一个小时才更新一次，以便节约电池。也可以提供一个配置，让用户自己设置更新的频率。

注意，如果在手机处于休眠状态时，对 App Widgets 进行更新的时间到了，这时设备将唤醒以便进行 App Widgets 的更新。如果不想手机在处于休眠时还进行 App Widgets 更新，可以通过一个 Alarm 进行更新。用 AlarmManager 来设置一个定期发送 AppWidgetProvider 的 Intent 的 Alarm，且将 Alarm 的类型设置为 ELAPSED\_REALTIME 或 RTC，这两种类型的 Alarm 只有在系统处于 awake 状态才会发送。此时要将 android:updatePeriodMillis 设置为 0。

- initialLayout 属性用于设置 App Widgets 的布局文件。
- configure 属性用于说明在 App Widgets 被添加到 AppWidgetsHost 时，哪个 ConfigureActivity 将首先启动，这是一个可选属性。ConfigureActivity 用的是完整的名字，因为它将在 APK 包外被引用。

- `previewImage` 属性在 Android API 11 版本中才被添加, 用于指明 AppWidgets 的预览图片, 帮助用户选中该 AppWidgets 的图标, 并打算添加该 AppWidgets 时进行显示, 以便用户了解该 AppWidgets 的界面。如果没提供预览图标的话, 显示的将是 AppWidgets 的启动图标。该属性和 `AndroidManifest.xml` 中的 `<receiver>` 元素的 `android:previewImage` 的属性一致。图 10-4 就是程序面板中显示的 AppWidgets 的预览图片。
- `resizeMode` 属性在 Android API 12 中才被添加, 用于说明 AppWidgets 重新调整大小的规则。通过该属性, 可以设置在什么方向允许调整 AppWidgets 的大小, 可以是垂直、水平, 或同时垂直和水平两个方向。用户可以按住 AppWidgets 来显示大小调整拖柄, 通过在水平或垂直方向拖动拖柄来调整 AppWidgets 在垂直或水平方向的尺寸。`resizeMode` 属性的值可以是 `horizontal`、`vertical`、`none` 和 `horizontal|vertical`。
- `icon` 属性用于说明 AppWidget 在 AppWidgets 列表中显示的图标, 它应该和 `AndroidManifest.xml` 中的 `<receiver>` 元素的 `android:icon` 的属性一致。
- `label` 属性用于说明 AppWidgets 在 AppWidgets 列表中显示的名字, 它应该和 `AndroidManifest.xml` 中的 `<receiver>` 元素的 `android:label` 的属性一致。

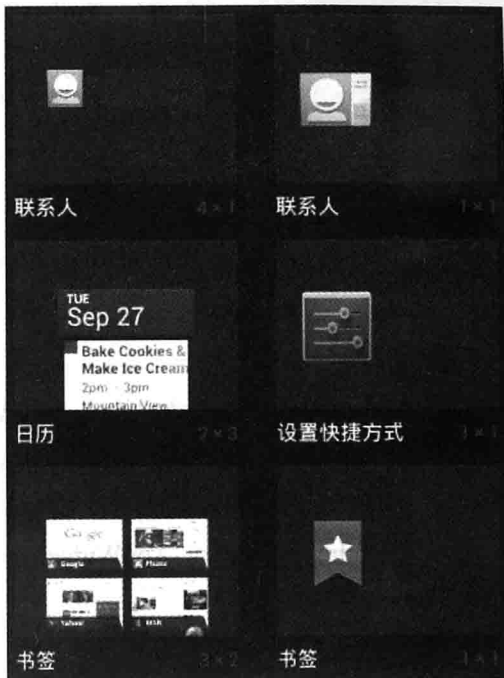


图 10-4 App Widgets 的预览图片

### 10.1.3 AppWidgetProvider

`AppWidgetProvider` 继承于 `BroadcastReceiver`, 它对 App Widgets 的广播进行了简单分类, 并封装了处理的统一接口, 通过这些方法可以很方便和 App Widgets 进行交互。`AppWidgetProvider` 基于广播事件。当 App Widgets 进行更新、启用、禁用和删除时, 在 `AppWidgetProvider` 中将收到其对应的广播, 并调用相应的回调方法进行处理。

Month Calendar Widget 应用的 `AppWidgetProvider` 框架如下:

```

01 public class MonthCalendarWidget extends AppWidgetProvider {
02
03     @Override
04     public void onUpdate(Context context, AppWidgetManager appWidgetManager,
05         int[] appWidgetIds) {
06     }
07
08     @Override
09     public void onReceive(Context context, Intent intent) {

```

```

10     }
11
12     @Override
13     public void onAppWidgetOptionsChanged(Context context,
14         AppWidgetManager appWidgetManager,
15         int appWidgetId, Bundle newOptions) {
16     }
17
18     private void drawWidget(Context context, int appWidgetId) {
19     }
20
21     private void redrawWidgets(Context context) {
22     }
23
24 }

```

其中的 `drawWidget()` 方法和 `redrawWidgets()` 方法用于绘制 App Widgets 的用户界面。

`AppWidgetProvider` 只接受和 App Widgets 相关的广播，例如 App Widgets 更新、删除、启用和禁用的广播。当收到以上广播后，将分别调用以下方法：

- `onUpdate(Context, AppWidgetManager, int[])`

代码如下：

```

01 @Override
02 public void onUpdate(Context context, AppWidgetManager appWidgetManager,
03     int[] appWidgetIds) {
04     super.onUpdate(context, appWidgetManager, appWidgetIds);
05
06     for (int appWidgetId : appWidgetIds) {
07         drawWidget(context, appWidgetId);
08     }
09 }

```

当系统以 `AppWidgetProviderInfo` 中的 `updatePeriodMillis` 属性定义的频率请求更新 App Widgets 时，将调用该方法。如果没有定义 `Configuration Activity`，当用户添加该 App Widgets 时，也会调用该方法，主要用于处理初始化工作，例如设置 View 的事件监听者、启动一个临时 Service 等。如果定义了 `Configuration Activity`，在该方法中常定义一个 `PendingIntent` 来启动这个 Activity 并使用 `setOnClickPendingIntent(int, PendingIntent)` 方法将其附着到这个 App Widgets 的按钮上。

06~07 行是在一个循环中对 `appWidgetIds` 这个数组的每项依次进行操作的，该数组包括了通过该 `AppWidgetProvider` 创建的所有 App Widgets 实例的 id。通过该方法用户可以创建该 App Widgets 的多个实例，并同时对它们进行更新。然而，只有一个 App Widgets 实例的 `updatePeriodMillis` 的进度表来对所有该 App Widgets 实例的更新进行管理。例如，一个 App Widgets 的更新频率是 2 小时一次，首先添加了一个实例，然后隔了一个小时，又添加它的另一个实例，这时它的更新还是通过第一个 App Widgets 的更新进度表来

处理，第二个将被忽略掉（它们将每隔两小时更新，而不是两个更新进度表叠加变成每隔一小时更新）。

因为 AppWidgetProvider 扩展自 BroadcastReceiver，所以，不能保证回调方法完成调用后，AppWidgetProvider 还在继续运行。如果 App Widgets 的初始化需要多达几秒的时间，而且希望 AppWidgetProvider 的进程能够长久运行，那么可以考虑在 onUpdate() 中启动一个 Service，在该 Service 中可以更新 App Widgets。这样就不用担心 AppWidgetProvider 因为 ANR 错误而被迫关闭。

- onReceive(Context, Intent)

核心代码如下：

```
01 @Override
02 public void onReceive(Context context, Intent intent) {
03     super.onReceive(context, intent);
04
05     String action = intent.getAction();
06
07     if (ACTION_PREVIOUS_MONTH.equals(action)) {
08         ...
09         redrawWidgets(context);
10
11     } else if (ACTION_NEXT_MONTH.equals(action)) {
12         ...
13         redrawWidgets(context);
14
15     } else if (ACTION_RESET_MONTH.equals(action)) {
16         ...
17         redrawWidgets(context);
18     }
19 }
```

在收到任何广播时，该方法都会被调用。一般来说不用重载该方法，因为 AppWidget Provider 已经提供了默认的实现，它对广播进行分类，并调用其他几个对应的回调方法。

- onAppWidgetOptionsChanged(Context, AppWidgetManager, int, Bundle)

代码如下：

```
20 @Override
21 public void onAppWidgetOptionsChanged(Context context,
22     AppWidgetManager appWidgetManager,
23     int appWidgetId, Bundle newOptions) {
24     super.onAppWidgetOptionsChanged(context, appWidgetManager,
25         appWidgetId, newOptions);
26     drawWidget(context, appWidgetId);
27 }
```

该回调方法在 API Level 16 中被引入。在第一次放置 App Widgets 及调整 App Widgets 尺寸时，该方法就会被调用。可以使用该回调方法来实现显示和隐藏 App Widgets 的内容。

本例中重载了上面的三个方法，在 `AppWidgetProvider` 中，还提供了下面常用的回调方法：

- `onDeleted(Context, int[])`

当 App Widgets 从 App Widgets Host 移除时，将调用该方法。例如：

```
01 @Override
02 public void onDeleted(Context context, int[] appWidgetIds) {
03     Log.d(TAG, "onDeleted");
04     final int N = appWidgetIds.length;
05     for (int i=0; i<N; i++) {
06         ExampleAppWidgetConfigure.deleteTitlePref(context,
07             appWidgetIds[i]);
08     }
09 }
```

- `onEnabled(Context)`

如果用户向 App Widgets Host 加入 App Widgets 时，在 App Widgets 宿主中还没有该 App Widgets 实例时，就会调用该方法。在该方法中可以做些初始化工作，如果需要打开一个新的数据库或者执行其他对于所有的 App Widgets 实例只需要发生一次的设置，即可在该方法中处理。例如：

```
01 @Override
02 public void onEnabled(Context context) {
03     Log.d(TAG, "onEnabled");
04     PackageManager pm = context.getPackageManager();
05     pm.setComponentEnabledSetting(
06         new ComponentName("com.example.android.apis",
07             ".appwidget.ExampleBroadcastReceiver"),
08         PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
09         PackageManager.DONT_KILL_APP);
10 }
```

- `onDisabled(Context)`

当用户把 App Widgets 从 App Widgets Host 中移除时，并且它是 App Widgets Host 中的唯一的该 App Widgets 实例时，就会调用该方法。在该方法中可以清理在 `onEnabled(Context)` 中做的工作，例如清理临时的数据库。

### 10.1.4 声明 App Widgets

创建了 `AppWidgetProviderInfo` 并实现了 `AppWidgetProvider` 之后，还需要在 `AndroidManifest.xml` 文件中声明该 `AppWidgetProvider` 类，例如：

```
01 <receiver android:name=".MonthCalendarWidget">
02     <intent-filter>
03         <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
04     </intent-filter>
05     <meta-data
```

```

06         android:name="android.appwidget.provider"
07         android:resource="@xml/widget_info" />
08     </receiver>

```

<receiver> 元素的 android:name 属性必须进行设置，该属性说明了将使用哪个 AppWidgetProvider 来提供 App Widgets。

<intent-filter> 元素必须包含 android:name 属性为 "android.appwidget.action.APPWIDGET\_UPDATE" 的 Action。该属性说明了 AppWidgetProvider 可以接收 ACTION\_APPWIDGET\_UPDATE 广播。该广播是唯一必须要声明接收的广播（其他广播包括删除 ACTION\_APPWIDGET\_DELETED、启用 ACTION\_APPWIDGET\_ENABLED 和禁用 ACTION\_APPWIDGET\_DISABLED）。AppWidgetManager 能自动地将其他所有的 App Widgets 广播发送到 AppWidgetProvider。

在 <meta-data> 元素中，必须要指定 AppWidgetProviderInfo 资源文件，并定义以下 2 个属性：

- android:name 属性：用于定义 metadata 元素的名字。必须将该属性设置为 “android.appwidget.provider” 以表明该 <meta-data> 元素是用于描述 AppWidgetProviderInfo 资源文件位置的。
- android:resource 属性：用于说明 AppWidgetProviderInfo 资源文件的位置。

## 10.2 配置和管理桌面组件

对桌面组件的配置和管理主要是指对组件元数据的设置、更新和删除等操作。

### 10.2.1 Configuration Activity

如果想让用户在添加一个新的 App Widgets 时，能对该 App Widgets 进行一些个性化的配置的话，可以通过编写一个 App Widgets 的 Configuration Activity 来实现。该 Activity 是可选的，当用户添加 App Widgets 时，该 Activity 将被启动。通过它可以在 App Widgets 被创建时做一些对 App Widgets 的设置。这里的设置是指和 App Widgets 的事务相关的设置，不是设置 AppWidgetProviderInfo 的内容。

Configuration Activity 和一般的 Activity 一样，在 AndroidManifest.xml 文件中进行声明。App Widgets Host 通过 Action 为 ACTION\_APPWIDGET\_CONFIGURE 的 Intent 启动 Configuration Activity，所以 Configuration Activity 必须要能接收该 Action。例如：

```

01 <activity android:name=".ExampleAppWidgetConfigure">
02     <intent-filter>
03         <action
04             android:name="android.appwidget.action.APPWIDGET_CONFIGURE"/>
05     </intent-filter>
06 </activity>

```



同时，在 `AppWidgetProviderInfo` 文件中使用 `android:configure` 属性指明该 `Configuration Activity`。

在实现一个 `Configuration Activity` 时，需要注意：

- 通过 `AppWidgetsHost` 调用 `ConfigurationActivity`，`ConfigurationActivity` 应该总是能返回一个执行结果。返回结果应该包含通过 `Intent` 传给 `ConfigurationActivity` 的要添加的 `AppWidgets` 的 ID(该 ID 通过 `EXTRA_APPWIDGET_ID` 保存在 `Intent` 的 `extras` 中)。
- 如果 `AppWidgets` 有 `ConfigurationActivity`，那么当 `AppWidgets` 被创建时，`AppWidgetProvider` 的 `onUpdate()` 方法将不会被调用，且 `ConfigurationActivity` 必须负责请求 `AppWidgetManager` 对 `AppWidgets` 进行首次更新。然而以后只要更新时间到了，系统还是会发送 `ACTION_APPWIDGET_UPDATE` 广播，因此 `AppWidgets` 的 `onUpdate()` 方法还是会被调用，以进行 `AppWidgets` 更新。系统只是在 `AppWidgets` 被创建时，不发送 `ACTION_APPWIDGET_UPDATE` 广播。

以下是在 `Configuration Activity` 中更新 `App Widgets` 和退出 `Configuration Activity` 的主要步骤。

①在启动 `Configuration Activity` 的 `Intent` 中得到 `App Widgets` 的 ID。例如：

```
01 Intent intent = getIntent();
02 Bundle extras = intent.getExtras();
03 if (extras != null) {
04     mAppWidgetId = extras.getInt(AppWidgetManager.EXTRA_APPWIDGET_ID,
05     AppWidgetManager.INVALID_APPWIDGET_ID);
06 }
```

②进行 `App Widgets` 配置的处理。

③当 `App Widgets` 的配置事务被处理完后，调用 `AppWidgetManager.getInstance(context)` 来得到 `AppWidgetManager` 的一个实例。例如：

```
01 AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(context);
```

④调用 `updateAppWidget(int, RemoteViews)` 方法，通过 `RemoteViews` 对象来更新 `App Widgets`。例如：

```
01 RemoteViews views = new RemoteViews(context.getPackageName(),
02     R.layout.example_appwidget);
03 appWidgetManager.updateAppWidget(mAppWidgetId, views);
```

⑤将执行结果放在 `Intent` 的附加数据中并通过 `Intent` 返回结果，结束 `Configuration Activity`。例如：

```
01 Intent resultValue = new Intent();
02 resultValue.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWidgetId);
03 setResult(RESULT_OK, resultValue);
04 finish();
```



注意，当 Configuration Activity 首次启动时，应该将 Activity 的返回结果设置为 RESULT\_CANCELED。这样如果用户中途退出了 Configuration Activity，那么系统将通知 App Widgets Host 该配置过程被取消，这样 App Widgets 将不被添加。

## 10.2.2 AppWidgetManager

AppWidgetManager 是 Android 平台上 App Widgets 的管理类，一般通过 getInstance() 方法获取一个实例。AppWidgetManager 提供的一些方法可以绑定 App Widgets、通过 Provider 名称获取对应的 ID 以及获取一个 App Widgets Provider 信息等，并向 AppWidgetProvider 发送通知。

这些方法包括：

- bindAppWidgetId(int appWidgetId, ComponentName provider) 通过给定的 ComponentName 绑定 appWidgetId。例如，通过下面的方法向 Launcher 添加 Widget：

```
01 private boolean addAppWidget(SQLiteDatabase db, ContentValues values,
02     ComponentName cn, int spanX, int spanY) {
03     boolean allocatedAppWidgets = false;
04     final AppWidgetManager appWidgetManager =
05         AppWidgetManager.getInstance(mContext);
06
07     try {
08         int appWidgetId = mAppWidgetHost.allocateAppWidgetId();
09         values.put(Favorites.ITEM_TYPE, Favorites.ITEM_TYPE_APPWIDGET);
10         values.put(Favorites.SPANX, spanX);
11         values.put(Favorites.SPANY, spanY);
12         values.put(Favorites.APPWIDGET_ID, appWidgetId);
13         db.insert(TABLE_FAVORITES, null, values);
14         allocatedAppWidgets = true;
15         appWidgetManager.bindAppWidgetId(appWidgetId, cn);
16     } catch (RuntimeException ex) {
17         Log.e(TAG, "Problem allocating appWidgetId", ex);
18     }
19
20     return allocatedAppWidgets;
21 }
```

- getAppWidgetIds(ComponentName provider) 通过给定的 ComponentName 获取 AppWidgetId。例如下面的方法用于检测 Widget 是否存在：

```
01 private boolean hasInstances(Context context) {
02     AppWidgetManager appWidgetManager =
03         AppWidgetManager.getInstance(context);
04     int[] appWidgetIds = appWidgetManager.getAppWidgetIds(THIS_APPWIDGET);
05     return (appWidgetIds.length > 0);
06 }
```

- getAppWidgetInfo(int appWidgetId) 通过 AppWidgetId 获取 AppWidget 信息。例如，下面的方法查询指定 appWidgetId 的 AppWidgetProviderInfo 对象，即在 XML 文件配置

的 `<appwidget-provider/>` 节点信息。

```
01 AppWidgetProviderInfo appWidgetProviderInfo =
02     appWidgetManager.getAppWidgetInfo(appWidgetId);
```

- `getInstalledProviders()` 返回一个 `List<AppWidgetProviderInfo>` 的信息。例如：

```
01 List<AppWidgetProviderInfo> widgets =
02     AppWidgetManager.getInstance(mLauncher).getInstalledProviders();
```

- `getInstance(Context context)` 获取 `AppWidgetManager` 实例使用的上下文对象。
- `updateAppWidget(int[] appWidgetIds, RemoteViews views)` 通过 `appWidgetId` 对传进来的 `RemoteView` 进行修改，并重新刷新 `AppWidget` 组件。
- `updateAppWidget(ComponentName provider, RemoteViews views)` 通过 `ComponentName` 对传进来的 `RemoteView` 进行修改，并重新刷新 `AppWidget` 组件。
- `updateAppWidget(int appWidgetId, RemoteViews views)` 通过 `appWidgetId` 对传进来的 `RemoteView` 进行修改，并重新刷新 `AppWidget` 组件。

下面是一个简单的更新 `App Widgets` 中显示日期文本信息的示例：

```
01 int appWidgetId = appWidgetIds[i];
02 RemoteViews views = new RemoteViews(context.getPackageName(),
03     R.layout.firstappwidget);
04 java.text.DateFormat df = new java.text.SimpleDateFormat("hh:mm:ss");
05 views.setTextViewText(R.id.tvMsg, "当前时间: " + df.format(new Date()));
06 appWidgetManager.updateAppWidget(appWidgetId, views);
```

## 10.3 设计集合桌面组件

集合桌面组件是指使用 `ListView`、`GridView`、`StackView` 或 `AdapterViewFlipper` 来设计用户界面的 `App Widgets`。集合桌面组件是 Android API 11 版本引入的功能。

### 10.3.1 Collection Views

集合 `App Widgets` 通过 `RemoteViewsService` 提供的 `RemoteViewsFactory` 来显示远程的数据集。用于数据集显示的 `View`，称为 `Collection Views`，它可以为以下类型之一。

#### 1. ListView

该 `View` 以垂直可滚动列表的形式进行数据集中所有数据项的显示。最具代表性的是 `Gmail App Widgets`。

#### 2. GridView

该 `View` 以水平和垂直都可以滚动的网格状形式进行数据集中所有数据项的显示。最具代表性的是 `BookMarks App Widgets`。

### 3.StackView

该 View 以卡片集的形式进行数据集中所有数据项的显示。可以对最卡片进行上翻/下翻操作来浏览卡片。最具代表性的是 YouTube 和 Books App Widgets。

### 4.AdapterViewFlipper

被适配器支持的简单 ViewAnimator 实现了两个或多个 View 之间的动画。每次只显示一个子 View。

图 10-5 是提供了多种集合样式的 ContactsWidgetICS<sup>①</sup>项目的预览效果。

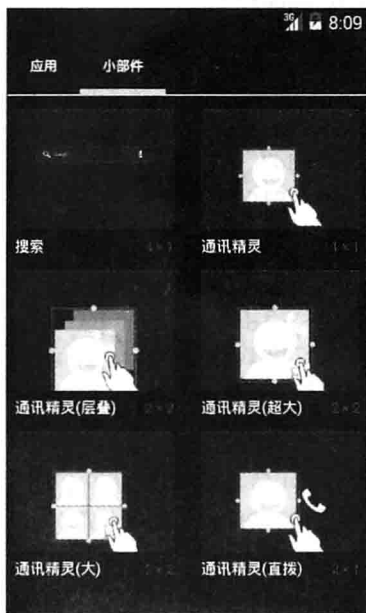


图 10-5 ContactsWidgetICS 桌面应用预览图

本节以创建 ContactsWidgetICS 的层叠桌面应用（如图 10-6 所示）为例，介绍集合桌面组件的设计。



图 10-6 ContactsWidgetICS 层叠设计效果图

① 项目主页 <https://github.com/yuyang226/ContactsWidgetICS>。

在项目的 res/layout 中创建 contact\_manager\_stack.xml 布局文件，代码如下：

```

01 <?xml version="1.0" encoding="utf-8"?>
02 <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
03     android:layout_width="match_parent"
04     android:layout_height="wrap_content"
05     android:layout_margin="@dimen/widget_margin"
06     android:padding="0dp" >
07
08     <RelativeLayout
09         android:id="@+id/superLayout"
10         android:layout_width="match_parent"
11         android:layout_height="wrap_content"
12         android:layout_margin="@dimen/widget_margin"
13         android:horizontalSpacing="@dimen/widget_spacing"
14         android:padding="0dp"
15         android:verticalSpacing="@dimen/widget_spacing" >
16
17         <StackView
18             android:id="@+id/contactList"
19             android:layout_width="match_parent"
20             android:layout_height="match_parent"
21             android:gravity="center"
22             android:loopViews="true" />
23
24         <ImageButton
25             android:id="@+id/buttonPeople"
26             android:layout_width="wrap_content"
27             android:layout_height="wrap_content"
28             android:layout_alignParentBottom="true"
29             android:layout_alignParentRight="true"
30             android:background="@android:color/transparent"
31             android:contentDescription="@string/open_the_people_app"
32             android:paddingBottom="5dp"
33             android:paddingRight="5dp"
34             android:src="@drawable/people_app"
35             android:visibility="gone" />
36     </RelativeLayout>
37
38     <TextView
39         android:id="@+id/empty_view"
40         android:layout_width="match_parent"
41         android:layout_height="match_parent"
42         android:gravity="center"
43         android:text="@string/empty_view_text"
44         android:textColor="@color/highlighted_text_holo_dark"
45         android:textSize="20sp"
46         android:textStyle="bold" />
47
48 </FrameLayout>

```

其中的 `ImageButton` 用于显示一个联系人图标, 单击该图标即进入系统通讯录。`TextView` 是在通讯录中的联系人为空时显示的提示信息。

除了定义 App Widgets 布局文件以外, 还必须另外定义一个布局文件用于显示数据项, 如 Contacts Widget ICS 中的 `contact_entry_large_stack.xml`。

和设计简单的桌面组件一样, 也需要在项目的 `res/xml` 中设计 `AppWidgetProviderInfo` 文件 `widgetinfo_stack.xml`, 代码如下:

```
01 <appwidget-provider
02     xmlns:android="http://schemas.android.com/apk/res/android"
03     android:minWidth="@dimen/grid_size_large"
04     android:minHeight="@dimen/grid_size_large"
05     android:configure="com.gmail.yuyang226.contactswidget.pro" +
06         ".ui.ContactsWidgetStackConfigurationActivity"
07     android:updatePeriodMillis="180000"
08     android:previewImage="@drawable/preview_stack"
09     android:initialLayout="@layout/contact_manager_stack"
10     android:resizeMode="vertical|horizontal"
11     android:minResizeWidth="@dimen/grid_size_large"
12     android:minResizeHeight="@dimen/grid_size_large" >
13 </appwidget-provider>
```

如果需要在没有任何用户交互下, App Widgets 自动顺序地递增它的 View, 则需要在 `widgetinfo_stack.xml` 文件中添加 `android:autoAdvanceViewId="@id/stack_view"` 属性设置。这个设置应用于该 View 的 ID, 上例中为 `contactList`。

### 10.3.2 RemoteViewsService

由于 Collection Views 显示的是由远程数据所组成的数据集, 因此, 需要一个适配器来将数据绑定到 Collection Views, 该适配器要负责将数据集的单个数据项和 Collection Views 中的单个 View 对象进行绑定。因为 Collection Views 的数据项是在后台提供的, 所以 Android Framework 必须使用另外一种框架来支持在 App Widgets 中使用它们。

在 App Widgets 应用中, 使用 `RemoteViewsFactory` 来代替传统的适配器, `RemoteViewsFactory` 提供了和 `Adapter` 类似的接口。当请求数据集中的一个数据项时, `RemoteViewsFactory` 将以一个 `RemoteViews` 的形式返回数据集的一个数据项。为了能在 App Widgets 中使用 Collection Views, 则必须实现一个 `RemoteViewsService` 实例和 `RemoteViewsFactory` 接口。

`RemoteViewsService` 是一个管理 `RemoteViews` 的 Service, 远程的适配器可以通过它请求并获得 `RemoteViews` 对象。

一般的, 当 App Widgets 中包含 `GridView`、`ListView`、`StackView` 等 Collection Views 时, 才需要使用 `RemoteViewsService` 来进行更新、管理。

`RemoteViewsService` 更新 Collection Views 的一般步骤如下:

①通过 `setRemoteAdapter` 设置 `RemoteViews` 对应 `RemoteViewsService`。

②在 RemoteViewsService 中，实现 RemoteViewsFactory 接口。然后，在 RemoteViewsFactory 接口中对 Collection Views 的各个子项进行设置。

下面是 ContactsWidgetICS 项目中 RemoteViewsService 的核心代码：

```
01 public class ContactsWidgetService extends RemoteViewsService {
02
03     public ContactsWidgetService() {
04         super();
05     }
06
07     @Override
08     public RemoteViewsFactory onGetViewFactory(Intent intent) {
09         return new GridRemoteViewsFactory(
10             this.getApplicationContext(), intent);
11     }
12
13     ...
14 }
```

为了实现将带 Collection Views 的 App Widgets 绑定到 RemoteViewsService 上，必须在 AndroidManifest.xml 文件中声明含有 BIND\_REMOTEVIEWS 权限的 Service。这就防止其他应用来任意的访问本 App Widgets 中的数据。AndroidManifest.xml 的描述如下：

```
01 <service
02     android:name=
03         "com.gmail.yuyang226.contactswidget.pro.ContactsWidgetService"
04     android:exported="false"
05     android:permission="android.permission.BIND_REMOTEVIEWS" />
```

### 10.3.3 RemoteViewsFactory

RemoteViewsFactory 是 RemoteViewsService 中的一个接口。RemoteViewsFactory 为 Collection Views 与其数据提供了适配接口。RemoteViewsFactory 中的数据可以是数组，也可以是来自于 ContentProvider 数据库。在 RemoteViewsFactory 中，主要负责为数据集的每个数据项提供一个 RemoteViews 对象。实现 RemoteViewsService 的主要工作集中在 RemoteViewsFactory 接口的设计中。

RemoteViewsFactory 提供了一系列方法来管理 Collection Views 中的每一项。例如：

①通过 RemoteViews getViewAt(int position) 方法获取 Collection Views 中的子视图，视图是以 RemoteViews 对象返回的。

②通过 int getCount() 方法获取 Collection Views 中所有子项的总数。

下面是 ContactsWidgetICS 项目中 RemoteViewsFactory 的核心代码：

```
01 class GridRemoteViewsFactory implements
02     RemoteViewsService.RemoteViewsFactory {
```

```

03  ...
04
05  public GridRemoteViewsFactory(Context context, Intent intent) {
06      mContext = context;
07      mAppWidgetId = intent.getIntExtra(
08          AppWidgetManager.EXTRA_APPWIDGET_ID,
09          AppWidgetManager.INVALID_APPWIDGET_ID);
10      ...
11      widgetEntryLayoutId = intent.getIntExtra(
12          ContactsWidgetProvider.CONTACT_ENTRY_LAYOUT_ID,
13          R.layout.contact_entry);
14  }
15
16  public void onCreate() {
17      mWidgetItems.clear();
18      mWidgetItems.addAll(new ContactAccessor().getContacts(
19          getContentResolver(), this.mContext,
20          this.mAppWidgetId, this.imageSize));
21  }
22
23  public void onDestroy() {
24      for (Contact contact: mWidgetItems) {
25          if (contact.getPhoto() != null
26              && !contact.getPhoto().isRecycled()) {
27              contact.getPhoto().recycle();
28          }
29      }
30      mWidgetItems.clear();
31      ContactAccessor.clearImageCache();
32  }
33
34  public int getCount() {
35      return mWidgetItems.size();
36  }
37
38  public RemoteViews getViewAt(int position) {
39      RemoteViews rv = new RemoteViews(mContext.getPackageName(),
40          widgetEntryLayoutId);
41      ...
42      return rv;
43  }
44
45  }

```

当 `RemoteViewsFactory` 首次被创建时, `onCreate()` 方法将被调用, 它在该方法中主要做一些初始化工作。其中的 18~20 行通过调用 `ContactAccessor` 的 `getContacts()` 方法来获取联系人信息, 然后通过 `getContactsByGroup()` 方法中的 `CursorLoader()` 方法异步载入 URI 为 `ContactsContract.Data.CONTENT_URI` 的 `ContentProvider` 数据。

### 10.3.4 子视图事件

在简单的 App Widgets 中，一般通过 `setOnClickPendingIntent()` 方法来设置 View 控件被单击时发送的 Intent。但是这种方法对带 Collection Views 的 App Widgets 并不适用。替代策略是首先使用 `setOnClickFillInIntent()` 方法为数据集的数据项 View 统一设置其 PendingIntent 模板。然后在 RemoteViewsFactory 中为数据集的数据项 View 的 PendingIntent 模板的设置填充 Intent。这样当数据项 View 被单击时，发送的 Intent 将是其 PendingIntent 模板和其填充 Intent 合成的 Intent。

本节以单击 ContactsWidgetICS 项目中层叠桌面应用样式的联系人头像的事件设计为例，说明如何为数据项 View 添加单击行为。在该应用中，当单击联系人头像 ImageView 控件时，显示图 10-7 所示的界面。

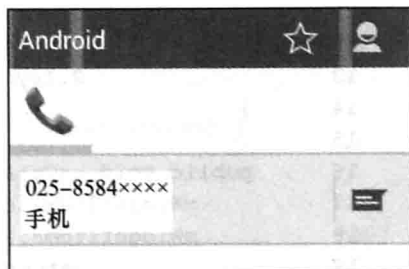


图 10-7 联系人头像事件响应界面

其主要设计流程如下：

①在 RemoteViewsFactory 中设置数据项 View 的填充 Intent。例如：

```
01 public RemoteViews getViewAt(int position) {
02     RemoteViews rv = new RemoteViews(mContext.getPackageName(),
03         widgetEntryLayoutId);
04     if (mWidgetItems == null || mWidgetItems.isEmpty()) {
05         return rv;
06     }
07     Contact contact = mWidgetItems.get(position);
08     if (ContactsWidgetConfigurationActivity.loadShowName(
09         this.mContext, this.mAppWidgetId)) {
10         rv.setViewVisibility(R.id.contactEntryText, View.VISIBLE);
11         rv.setTextViewText(R.id.contactEntryText,
12             contact.getDisplayName());
13     } else {
14         rv.setViewVisibility(R.id.contactEntryText, View.GONE);
15     }
16
17     Bitmap photo = contact.getPhoto();
18     if (photo != null) {
19         rv.setImageViewBitmap(R.id.contactPhoto, photo);
20     } else {
21         rv.setImageViewResource(R.id.contactPhoto, R.drawable.icon);
22     }
23
24     Intent fillInIntent = new Intent();
25     fillInIntent.putExtras(new Bundle());
26     fillInIntent.setData(contact.getContactUri());
27     rv.setOnClickFillInIntent(R.id.contactPhoto, fillInIntent);
28
29     boolean supportDirectDial = false;
```



```

30     boolean supportDirectSms = false;
31     if (this.canDirectDial && contact.getPhoneNumbers() != null
32         && !contact.getPhoneNumbers().isEmpty()) {
33         supportDirectDial = true;
34         fillInIntent = new Intent();
35         fillInIntent.putExtras(new Bundle());
36         fillInIntent.putExtra(ContactsWidgetProvider.INTENT_TAG_ACTION,
37             ContactsWidgetProvider.DIRECT_DIAL_ACTION);
38         String phoneNumber = contact.getPhoneNumbers().get(0).getNumber();
39         fillInIntent.setData(Uri.parse("tel:" + phoneNumber));
40         rv.setTextViewText(R.id.contactPhoneNumberText, phoneNumber);
42         rv.setOnClickFillInIntent(this.viaContactIcon
42             ? R.id.contactPhoto : R.id.dialerButton, fillInIntent);
43
44         if (directSms) {
45             supportDirectSms = true;
46             fillInIntent = new Intent();
47             fillInIntent.putExtras(new Bundle());
48             fillInIntent.putExtra(ContactsWidgetProvider.INTENT_TAG_ACTION,
49                 ContactsWidgetProvider.DIRECT_SMS_ACTION);
50             fillInIntent.setData(Uri.parse("smsto:" + phoneNumber));
51             rv.setOnClickFillInIntent(R.id.smsButton, fillInIntent);
52         }
53     }
54
55     rv.setVisibility(R.id.dialerButton, supportDirectDial
56         && !viaContactIcon ? View.VISIBLE : View.GONE);
57     ...
58
59     return rv;
60 }

```

必须在 RemoteViewsFactory 中对数据集的每个数据项 View 设置填充 Intent，以用于区分是哪个数据项 View。当用户单击数据项 View 时，其 PendingIntent 模板和其填充 Intent 被合成一个最终的 Intent，并进行广播发送。

②设置数据项 View 的 PendingIntent 模板，例如：

```

01 public static void updateAppWidget(Context context,
02     AppWidgetManager appWidgetManager, int appWidgetId,
03     int widgetEntryLayoutId, boolean canLaunchPeopleApp,
04     Rect imageSize) {
05     AppWidgetProviderInfo widgetProviderInfo = appWidgetManager
06         .getAppWidgetInfo(appWidgetId);
07     ...
08     RemoteViews rv = new RemoteViews(context.getPackageName(), layoutId);
09     rv.setRemoteAdapter(R.id.contactList, intent);
10     if (isKeyguard || canLaunchPeopleApp) {
11         Intent launchPeopleIntent = new Intent(context,
12             ContactsWidgetProvider.class);

```

```

13         launchPeopleIntent
14             .setAction(ContactsWidgetProvider.LAUNCH_PEOPLE_ACTION);
15         PendingIntent pi = PendingIntent.getBroadcast(context, 0,
16             launchPeopleIntent, PendingIntent.FLAG_UPDATE_CURRENT);
17         rv.setOnClickPendingIntent(R.id.buttonPeople, pi);
18     }
19     rv.setVisibility(R.id.buttonPeople,
20         canLaunchPeopleApp ? View.VISIBLE : View.GONE);
21     rv.setEmptyView(R.id.contactList, R.id.empty_view);
22     Intent toastIntent = new Intent(context, ContactsWidgetProvider.class);
23     toastIntent.setAction(
24         ContactsWidgetProvider.SHOW_QUICK_CONTACT_ACTION);
25     toastIntent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
26         appWidgetId);
27     toastIntent.putExtra(CONTACT_ENTRY_LAYOUT_ID, widgetEntryLayoutId);
28     intent.setData(Uri.parse(intent.toUri(Intent.URI_INTENT_SCHEME)));
29     PendingIntent toastPendingIntent = PendingIntent.getBroadcast(context,
30         0, toastIntent, PendingIntent.FLAG_UPDATE_CURRENT);
31     rv.setPendingIntentTemplate(R.id.contactList, toastPendingIntent);
32
33     appWidgetManager.updateAppWidget(appWidgetId, rv);
34     appWidgetManager.notifyAppWidgetViewDataChanged(appWidgetId,
35         R.id.contactList);
36 }

```

Contacts WidgetProvider 设置了一个 PendingIntent。容器中的单独项没有自己的 PendingIntent。取而代之的是容器作为一个整体设置一个 PendingIntent 模板，对于数据集的单个数据项 View 只能进行填充 Intent 设置，以便让每个数据项 View 都有个性化的单击行为。用户单击数据项 View 时，其 PendingIntent 模板和其填充 Intent 被合成一个 Intent，该 Intent 被激活并被发送。

09 行调用 setRemoteAdapter() 方法告诉 Collection Views 能在哪里取得它的数据。在调用 setRemoteAdapter() 方法时，必须传递一个用于启动 RemoteViewsService 的 Intent 和一个 App Widgets ID 用于说明对哪个 App Widgets 进行更新。

### ③接收广播。

数据项 View 发送的 Intent 广播被 StackWidgetProvider 接收，并传递到它的 onReceive() 中处理该事件。如果 Intent 的 Action 是 SHOW\_QUICK\_CONTACT\_ACTION，则从 Intent 的 Extra 中提取其数据项 View 的索引，然后通过下面的代码显示图 10-7 所示的界面。

```

01 QuickContact.showQuickContact(context,
02     intent.getSourceBounds(),
03     uri,
04     ContactsContract.QuickContact.MODE_SMALL, null);

```

01 行的 QuickContact 是为了应用程序能够快速方便地访问联系人，并且快速地运用联系人的信息执行相应操作而设计的。

## 10.4 桌面组件设计规范

桌面组件是为了将应用中最重要或者随着时间变化的信息以最快、最容易的方式在用户主界面上显示。本节介绍如何设计一个在主界面上和其他控件和谐搭配的桌面应用组件。

### 10.4.1 桌面组件的种类

#### 1. 信息组件

信息组件一般用来显示一些重要的信息，并且会随着时间而改变。最好的例子就是天气组件、时钟组件和比分牌组件等。触摸信息组件一般都会打开关联的应用，显示更加详细的信息。图 10-8 就是典型的桌面信息组件。



图 10-8 桌面信息组件

#### 2. 列表组件

列表组件用于显示一系列信息，例如相册应用中的图片列表、新闻阅读应用中的文章列表或者通信应用中的短消息列表。列表组件一般有两种用途：浏览列表和在应用中打开指定项目的详细信息。列表组件可以纵向滑动。图 10-9 就是典型的桌面列表组件。

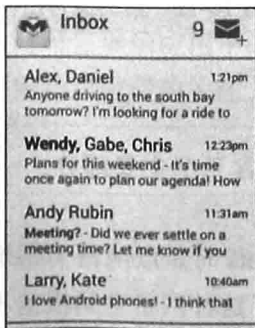


图 10-9 桌面列表组件

#### 3. 控制组件

控制组件主要用于方便用户，使他们在主屏幕上就能开关一些常用的设置而不用进入应用，也可以将其理解为应用的遥控器。控制组件不一定会打开关联应用的详细视图，如果操作会产生结果，那么则应该打开。例如搜索组件要显示结果列表。图 10-10 就是典型的桌面控制组件。



图 10-10 桌面控制组件

#### 4. 混合组件

有些组件可以综合以上所述的组件特点组合成一种新型的组件。在设计自己的组件时，应当以一种组件为基础，然后根据需要增加其他类型组件的元素。例如图 10-11 所示的音乐播放器组件是以控制组件为基础，不过组件上也能显示当前正在播放的音乐，它将控制组件和信息组件的元素结合在了一起。



图 10-11 桌面混合组件

### 10.4.2 桌面组件的尺寸

典型的 Android App Widgets 包含三个主要部分：一个有界限的封装盒、一个框架和图形控制等元素（见 10.1.1 节）。

默认的主屏幕在放置 App Widgets 时是以网格为基本单位，而不是以像素，这里的网格是指拥有一定的像素的长方形区域。这种网格空间根据不同设备有所改变。例如，许多手持设备提供的是  $4 \times 4$  的网格，平板电脑设备则是  $8 \times 7$ 。

添加 App Widgets 后，如果 App Widgets 的最小长宽和这些网格单元的尺寸不匹配，那么该 App Widgets 将收缩到最接近的单元尺寸，即它会自动伸展占据最小数量的网格，同时也要满足 `minWidth` 和 `minHeight` 的限制。使用 `nine-patch` 背景图片和富有弹性的布局会使 App Widgets 更好地匹配网格。

因为桌面布局方向可以变化，按照拇指规则，应该假设最坏情况单元尺寸是 74 像素高和宽。不过，必须从最后的尺寸中减去 2，以将像素计算过程中产生的任何的整数舍入误差

考虑在内。通过计算 ( 网格数  $\times 74$ )  $- 2$  可以得到像素密度无关的最小宽度和高度。为遵循该式子, 应该使用 72dp 为一个单元高度, 294dp 为四个单元宽度。图 10-12 演示了这种尺寸的使用。

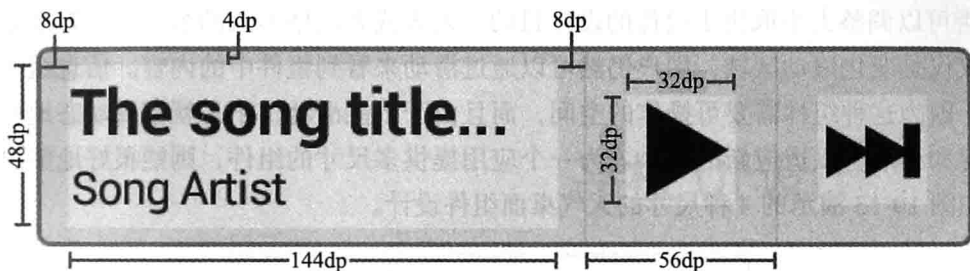


图 10-12 音乐控制部件的尺寸设计

图 10-12 所示音乐空间的尺寸计算方法如下:

$$\text{minWidth} = 144\text{dp} + (2 \times 8\text{dp}) + (2 \times 56\text{dp}) = 272\text{dp}$$

$$\text{minHeight} = 48\text{dp} + (2 \times 4\text{dp}) = 56\text{dp}$$

从 Android 3.1 开始, App Widgets 可以被用户设置尺寸, 这意味着 minWidth 和 minHeight 只是默认状态下的有效数据。所以可以使用 minResizeWidth 和 minResizeHeight 设置最小尺寸, 这是用户可以设置的最小尺寸, 小于它们则不可用。

### 10.4.3 桌面组件设计技巧

#### 1. 内容有吸引力

就像报纸头版的大幅广告, 组件可以展示应用中的信息, 并与应用中的详细信息建立联系。因此, 组件的设计要能够吸引用户对应用的关注, 并保证应用显示的内容已经比组件显示得更多。

#### 2. 灵活的导航

除了纯粹的信息内容, 还应该考虑在组件上增加应用中常用操作的导航。这样可以使用户直接从主屏幕进入常用功能, 更快地完成操作。可以放在组件上的导航链接包括:

- 创建功能: 有些操作使用户在应用中新建内容, 例如新建文档或者编写新信息。
- 打开应用的顶层视图: 触摸信息元素一般将用户带到详细信息视图。应该提供一个导航到应用顶层视图的按钮, 这样用户就不用把一个应用的图标放在主屏幕上了。

#### 3. 组件大小可调整

从 Android 3.1 版开始, Android 就提供了可调整大小的桌面组件。用户可以调整组件的宽度和高度。用户可以决定组件是可以随意调整大小还是只能在指定方向上调整大小, 也不需要为固定大小的组件提供调节能力。这种可调整大小的组件的好处是:

- 可以调节组件中显示信息的数量。
- 可以自由地调节主屏幕上组件和图标的布局。

#### 4. 多尺寸的组件

是否可以调整大小取决于组件的设计目的。列表或者网格形式的组件比较容易处理，调整大小仅仅是变化滚动区域，用户仍然可以通过滑动来看到组件中的内容。信息组件则比较难处理，因为这种组件需要可操作的空间，而且由于不能滑动，于是就需要动态地调整组件中的内容和布局，以适应新的大小。为一个应用提供多尺寸的组件，则能很好地解决这一问题。例如如图 10-13 演示的 4 种尺寸的天气桌面组件设计。

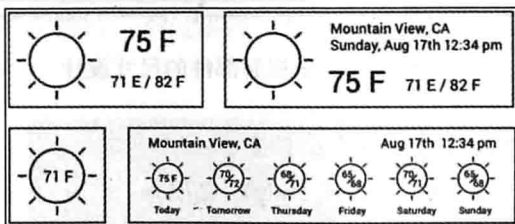


图 10-13 4 种尺寸的天气桌面组件

#### 5. 手势设计的限制

因为组件是放在主屏幕上的，所以必须和主屏幕的导航手势共存。组件手势的限制是相对于全屏应用来说的。全屏应用可以使用多种手势在视图之间切换，但是组件的手势不能和主屏幕导航使用的相同。在组件设计中，只有两种可用的手势：触摸和纵向滑动。

## 10.5 习题

设计一个下图所示的能够显示未接电话、未读信息的桌面应用组件。



# 平板 UI 设计

## 11.1 Fragment 概述

Android 平板设计的构建模块是 Fragment。Fragment 是 Android API 11 引入的概念，主要用在平板电脑的用户界面设计中，以支持更动态、更灵活的界面设计。

### 11.1.1 Fragment 布局特性

以一个典型的新闻阅读应用程序为例，如果设备是平板电脑，可以用一个 Fragment 显示标题列表，另一个 Fragment 显示选中标题的内容，这两个 Fragment 都在同一 Activity 上并排显示，如图 11-1a 所示显示效果。如果设备是手机，那么就用一个 Activity 显示标题列表，用另一个 Activity 显示新闻内容，如图 11-1b 所示显示效果。

通过图 11-1 所示的例子可以看出，使用 Fragment 布局用户界面非常的灵活和方便。Fragment 是一个自成体系的布局组成部分，它可以根据屏幕的方向和尺寸改变自身的尺寸和布局位置。Fragment 为开发者和设计师提供了一种全新的方法，让他们设计的应用变得有弹性、可堆叠，从而适应不同设备的屏幕规格。屏幕组件可以自由拉伸、堆叠、缩放和隐藏。

最常见的 Fragment 布局方式是分割视图。在屏幕左边显示列表，右边显示列表中被选中项目的详细信息。保持左边列表中被选中的状态，使得面板间的关系更加明确。这一布局在新闻应用程序和电子邮件客户端中较为常见，如图 11-2 所示。

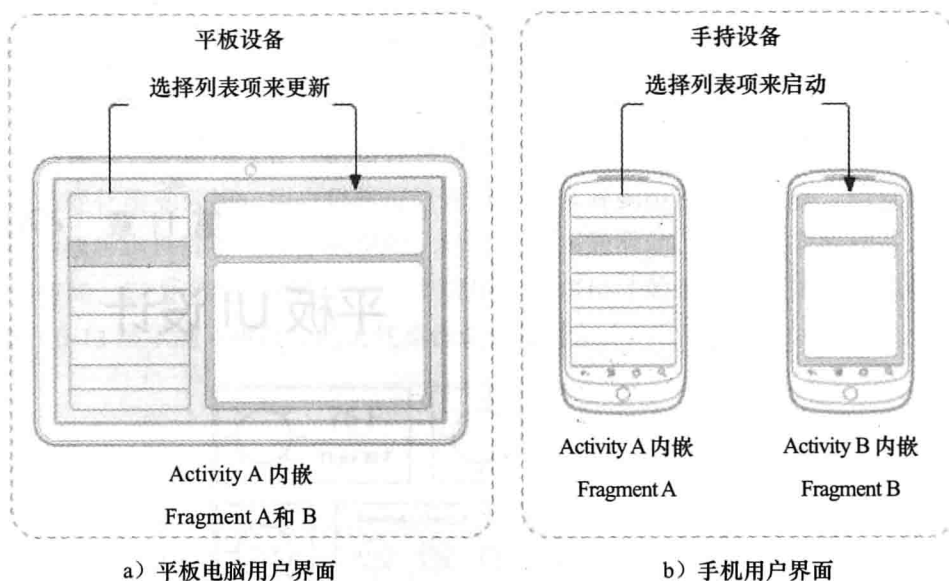


图 11-1 平板电脑与手机用户界面的对比

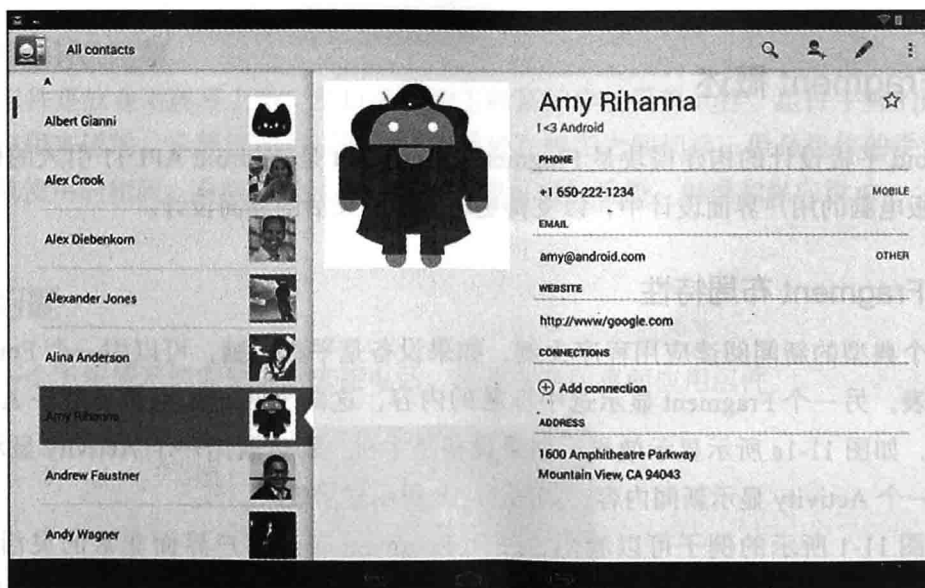


图 11-2 联系人平板应用设计效果图

### 11.1.2 Fragment 与 Activity

Fragment 是 Activity 的用户界面中的一部分或一种行为。可以在一个单独的 Activity 中将多个 Fragment 组合成为一个多区域的 UI，也可以在多个 Activity 中复用一个 Fragment。可以将 Fragment 认为是模块化的 Activity 片段，它具有自己的生命周期处理自己的输入事



件，并可以在 Activity 运行时动态地添加或删除 Fragment。

Fragment 不能独立存在，它必须嵌入到 Activity 中，而且 Fragment 的生命周期直接受所在的 Activity 的影响。例如，当 Activity 被暂停时，它拥有的所有的 Fragment 也都暂停了。当 Activity 销毁时，它拥有的所有 Fragment 也都被销毁。然而，当 Activity 处于运行状态时（在 Activity 生命周期方法 `onResume()` 之后，`onPause()` 之前），可以单独地操作每个 Fragment，例如添加或删除它们。当执行上述针对 Fragment 的事务时，可以将事务添加到一个回退栈中，该栈被 Activity 管理，栈中的每一个条目都是一个 Fragment 的一次事务。有了这个栈，就可以反向执行 Fragment 的事务，这样就可以允许用户通过按返回键回退一项 Fragment 事务（即向后导航）。

当将一个 Fragment 作为 Activity 布局的一部分时，它必须部署在 Activity 的视图的 ViewGroup 里面，同时，Fragment 必须定义它自己的视图布局。可以通过在 Activity 布局文件中以 `<fragment>` 元素的方式插入 Fragment，也可以在代码中创建 Fragment，然后将其加入到 ViewGroup 控件中。然而，Fragment 并不是 Activity 布局中必须的部分，也可以使用一个没有 UI 的 Fragment，它可以隐藏在后台为 Activity 工作。

## 11.2 创建 Fragment

Event Locations<sup>①</sup>应用是美国 1993 年创办的一个杂志的在线版。本节主要介绍通过 County 来搜索婚礼举办地信息功能的实现，应用界面如图 11-3 所示。

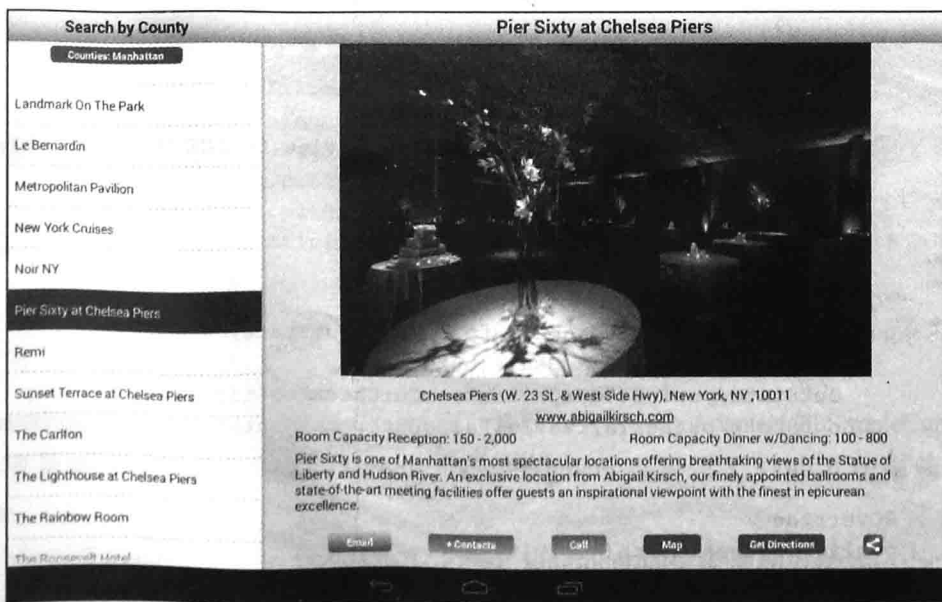


图 11-3 Event Locations 应用宽屏用户界面

### 11.2.1 创建 ListFragment

Event Locations 主界面包括左右两个 Fragment，左侧以列表的形式显示当前城市的商务区，可以采用 Fragment 的子类 ListFragment 来实现。右侧是单击左侧商务区后显示的婚礼举办地信息，采用 Fragment 来实现。

下面，创建一个 ListFragment 的实例 MasterFragmentCounty 来显示左侧的用户界面。MasterFragmentCounty 的核心代码如下：

```
01 public class MasterFragmentCounty extends ListFragment {
02     ...
03     public void setUpdateListener(OnTutSelectedListener l) {
04         UI_UPDATE_LISTENER = l;
05     }
06
07     @Override
08     public void onActivityCreated(Bundle savedInstanceState) {
09         super.onActivityCreated(savedInstanceState);
10
11         ...
12         View detailsFrame = getActivity().findViewById(R.id.details);
13         mDualPane = detailsFrame != null
14             && detailsFrame.getVisibility() == View.VISIBLE;
15
16         if (getResources().getConfiguration().orientation ==
17             Configuration.ORIENTATION_PORTRAIT) {
18             ...
19             showDetailsNew(mCurCheckPosition);
20         }
21     }
22
23     if (mDualPane) {
24         getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
25     }
26
27 }
28
29 @Override
30 public void onSaveInstanceState(Bundle outState) {
31     super.onSaveInstanceState(outState);
32     outState.putInt("curChoice", mCurCheckPosition);
33     outState.putSerializable("account", accounts);
34 }
35
36 @Override
37 public void onAttach(Activity activity) {
38     super.onAttach(activity);
39     try {
40         UI_UPDATE_LISTENER = (OnTutSelectedListener) activity;
41     } catch (Exception e) {
42     }
```

```

43
44     }
45
46     @Override
47     public void onItemClick(ListView l, View v, int pos, long id) {
48         ...
49         showDetailsNew(pos);
50     }
51
52     void showDetails(int index) {
53         mCurCheckPosition = index;
54
55         if (mDualPane) {
56             getListView().setItemChecked(index, true);
57             DetailFragmentCounty details = (DetailFragmentCounty)
58                 getFragmentManager().findFragmentById(R.id.details);
59
60             if (details == null || details.getShownIndex() != index) {
61                 details = DetailFragmentCounty.newInstance(index);
62                 if (accounts != null) {
63                     DetailFragmentCounty.setData(accounts, index);
64                 }
65                 FragmentTransaction ft = getFragmentManager()
66                     .beginTransaction();
67                 ft.replace(R.id.details, details);
68                 ft.setTransition(
69                     FragmentTransaction.TRANSIT_FRAGMENT_FADE);
70                 ft.commit();
71             }
72
73         } else {
74             DetailFragmentCounty.setData(accounts, index);
75             Intent intent = new Intent();
76             intent.setClass(getActivity(), DetailActivity.class);
77             intent.putExtra("index", index);
78             intent.putExtra("option", Common.SEARCH_BY_COUNTY);
79             startActivity(intent);
80         }
81     }
82
83 }

```

MasterFragmentCounty 扩展自 ListFragment，显示一个由一个 Adapter(如 SimpleCursorAdapter) 管理的项目的列表，类似于 ListActivity。ListFragment 也提供了一些方法来管理内嵌的 ListView。

在 MasterFragmentCounty 中，主要实现两个 Fragment 的生命周期方法 onAttach(Activity) 和 onActivityCreated(Bundle)。onAttach() 方法主要实现 OnTutSelectedListener 的赋值。onActivityCreated() 方法主要实现绑定 ListFragment 中内嵌的 ListView，并设置该 ListView 的单击事件。

另外，在代码的 13~14 行为判断当前 UI 是否为双窗体的。如果 `mDualPane` 为 `true`（即横屏模式），则 UI 界面采用 `Fragment` 来组织，使用 `showDetails()` 方法中的 56~70 行代码来构造 `DetailFragmentCounty`，显示被单击 `Item` 的详细信息。如果 `mDualPane` 为 `false`（即竖屏模式），则调用 `DetailsActivity` 响应 `MasterFragmentCounty` 的 `onListItemClick()` 事件。

### 11.2.2 创建 Fragment

创建 `Fragment` 和创建 `Activity` 类似，一般要实现 `Fragment` 的布局 XML 文件，并通过 `onCreateView()` 方法加载布局文件，最后将 `Fragment` 添加到 `Activity` 中。

创建 `Fragment` 的实现 `DetailFragmentCounty` 来显示右侧的用户界面。`DetailFragmentCounty` 的核心代码如下：

```

01 public class DetailFragmentCounty extends DetailFragmentBase {
02     ...
03     public static DetailFragmentCounty newInstance(int index) {
04         DetailFragmentCounty f = new DetailFragmentCounty();
05         Bundle args = new Bundle();
06         args.putInt("index", index);
07         f.setArguments(args);
08         return f;
09     }
10
11     @Override
12     public void onAttach(Activity activity) {
13         super.onAttach(activity);
14         try {
15             UI_UPDATE_LISTENER = (OnTutSelectedListener) activity;
16         } catch (Exception e) {
17
18         }
19
20     }
21
22     public View onCreateView(LayoutInflater inflater, ViewGroup container,
23         Bundle savedInstanceState) {
24         if (container == null) {
25             return null;
26         }
27
28         if (accounts != null) {
29             final HorizontalPager pager = new HorizontalPager(getActivity(),
30                 null);
31             ...
32             return pager;
33         } else {
34             viewer = (AccountViewCounty) inflater.inflate(
35                 R.layout.account_view_county, container, false);

```

```

36         if (account != null) {
37             viewer.setAccount(accounts.get(ind));
38         }
39         return viewer;
40     }
41 }
42
43 }

```

在 DetailFragmentCounty 中, 主要实现两个 Fragment 的生命周期方法 onAttach(Activity) 和 onCreateView(LayoutInflater, ViewGroup, Bundle)。onAttach(Activity) 方法主要实现 OnTutSelectedListener 的赋值。onCreateView() 方法主要加载 account\_view\_county 布局 (代码第 34~35 行), 并通过适配器适配界面的文件数据信息。

### 11.2.3 添加 Fragment 到 Activity

在 Activity 布局中添加 Fragment 有如下两种方法。

#### (1) 在 Activity 的布局文件中声明 Fragment

可以像为 View 一样为 Fragment 指定布局属性。例如, 下面是 Event Locations 使用的包含 Fragment 的布局文件 by\_site\_county.xml 的部分代码:

```

01 <?xml version="1.0" encoding="utf-8"?>
02 <RelativeLayout
03     xmlns:android="http://schemas.android.com/apk/res/android"
04     android:layout_width="fill_parent"
05     android:layout_height="fill_parent"
06     android:orientation="vertical" >
07     ...
08
09     <fragment
10         android:id="@+id/master"
11         android:layout_width="350dp"
12         android:layout_height="match_parent"
13         android:layout_below="@+id/titleBar"
14         android:layout_marginTop="43dp"
15         android:layout_weight="1"
16         class="us.eventlocations.androidtab.fragments
17             .MasterFragmentCounty" />
18
19     <FrameLayout
20         android:id="@+id/details"
21         android:layout_width="925dp"
22         android:layout_height="match_parent"
23         android:layout_marginTop="0dp"
24         android:layout_toRightOf="@id/divider" />
25
26 </RelativeLayout>

```

<fragment> 中的 class (或 android:name) 属性指定了布局中实例化的 Fragment 类。

当系统创建 Activity 布局时, 它实例化布局文件中指定的每一个 Fragment, 并为它们调用 onCreateView() 方法, 以获取每一个 Fragment 的布局。系统直接在 <fragment> 元素的位置插入 Fragment 返回的 View。

每个 Fragment 都需要一个唯一的标识 ID, 如果重启 Activity, 系统可用来恢复 Fragment (并且可用来捕捉 Fragment 的事务处理)。为 Fragment 提供 ID 有如下三种方法。

- 用 android:id 属性提供一个唯一的标识。
- 用 android:tag 属性提供一个唯一的字符串 (不仅在没有界面的 Fragment 中, 在有界面的 Fragment 中也可以使用 tag 来作为唯一标志, 这样在需要获取 Fragment 对象时, 可以调用 findFragmentTag())。
- 如果上述两个属性都没有, 系统会使用其容器视图的 ID。

## (2) 在代码中添加 Fragment 到一个 ViewGroup

只需要简单指定用来放置 Fragment 的 ViewGroup, 就可以在 Activity 运行的任何时候, 将 Fragment 添加到 Activity 布局中。具体步骤如下:

①首先从 Activity 中取得 FragmentTransaction 的实例, 一般方法是:

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

FragmentTransaction 中提供了对 Activity 中的 Fragment 进行添加、移除, 或者替换 Fragment 等操作的方法。

②然后调用 FragmentTransaction 的 add() 方法添加 Fragment, 并指定要添加的 Fragment 以及要将其插入到哪个视图之中, 例如:

```
01 DetailFragmentCounty fragment = new DetailFragmentCounty();
02 fragmentTransaction.add(R.id.fragment_container, fragment);
03 fragmentTransaction.commit();
```

传入 add() 方法的第一个参数是 Fragment 被放置的 ViewGroup, 它由资源 ID 指定, 第二个参数就是要添加的 Fragment。



**注意** 一旦通过 FragmentTransaction 对 Fragment 做出了改变, 就必须调用方法 commit() 提交这些改变。

---

## 11.2.4 使用 Support Library

Fragment 是 Android API 11 引入的概念, 如果要在 API 11 之前的版本使用 Fragment, 就

必须在项目中导入 Support Library。Android Support Library 提供了包含一个 API 库的 JAR 文件，当应用运行在 Android 早期版本时，Support Library 允许应用使用最近版本的 Android API。

在项目中添加 Support Library 的步骤如下：

①使用 SDK Manager 下载 Android Support 包，如图 11-4 所示。

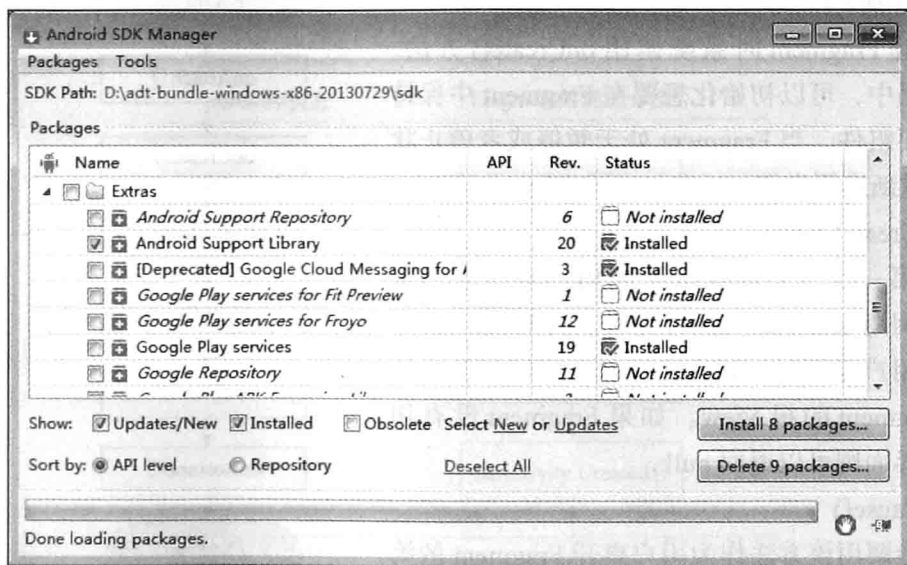


图 11-4 Android SDK Manager

②在项目的顶层目录下创建 libs 目录。

③找到想要引入库的 JAR 文件，然后将其复制到 libs 目录。

例如，支持 API Level 4 的库就位于 <sdk>/extras/android/support/v4/android-support-v4.jar，其中 <sdk> 代表着安装 Android SDK 的目录。

④修改 AndroidManifest.xml 文件，设置最低级别为 API level 4，目标 API level 为最新版本，例如：

```
<uses-sdk android:minSdkVersion="4" android:targetSdkVersion="19" />
```

注意，当使用 Support Library 创建有关 Fragment（包含在 android.support.v4.app.\* 中）的 Activity 时，必须继承 FragmentActivity 类，而不是传统的 Activity 类。

## 11.3 管理 Fragment

Android 中，使用 FragmentManager 来管理 Activity 中的 Fragment。

### 11.3.1 Fragment 的生命周期

和 Activity 一样，Fragment 也需要通过实现相应的生命周期方法来管理自身的状态。

Fragment 的生命周期非常类似于 Activity，在创建一个 Fragment 实例（继承 Fragment 或其子类）时，需要实现类似 Activity 一样的回调方法，例如 onCreate()、onStart()、onPause() 和 onStop()。Fragment 的生命周期如图 11-5 所示。

以下简要叙述了几个生命周期方法的作用。

- onCreate()

当创建 Fragment 时系统调用 onCreate() 方法。在实现代码中，可以初始化想要在 Fragment 中保持的那些必要组件，当 Fragment 处于暂停或者停止状态之后可重新启用它们。

- onCreateView()

- 在第一次为 Fragment 绘制用户界面时系统会调用 onCreateView() 方法。为 Fragment 绘制用户界面时，该方法必须要返回所绘出的 Fragment 的根 View。如果 Fragment 没有用户界面则可以返回 null。

- onPause()

系统会调用该方法作为用户离开 Fragment 的第一个处理（但这并不总意味着 Fragment 被销毁）。在当前用户会话结束之前，通常要在这里提交任何应该持久化的变化（因为用户可能不再返回）。

大部分应用程序都应该至少为每个 Fragment 实现如上三个方法，当然 Fragment 还要比 Activity 多几个生命周期回调方法，这些额外的方法是为了与 Activity 的交互而设立的，包括：

- onAttach()

当 Fragment 被绑定到 Activity 时调用（在这个方法中可以获得所在的 Activity）。

- onActivityCreated()

当 Activity 的 onCreate() 方法返回时被调用。

- onDestroyView()

当与 Fragment 关联的视图体系正被移除时被调用。

- onDetach()

当 Fragment 被从 Activity 中删掉时被调用。

Activity 的生命周期与 Fragment 的生命周期之间的关系如图 11-6 所示。可以看到

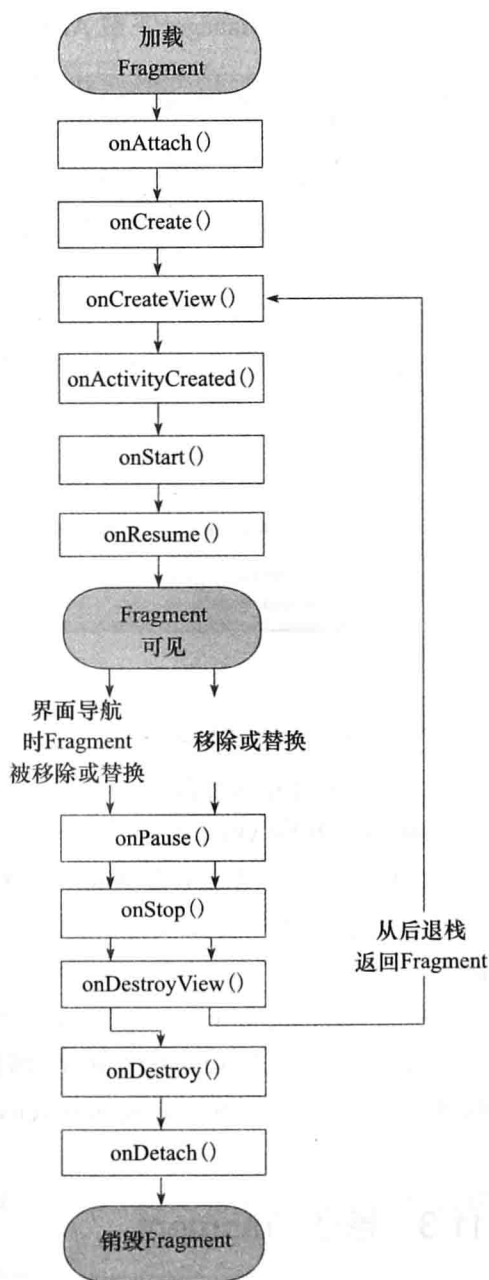


图 11-5 Fragment 的生命周期



Activity 的每个连续状态是如何决定 Fragment 可能接收到哪个回调方法的。例如，当 Activity 接收到它的 onCreate() 回调时，Activity 之中的 Fragment 接收到的仅仅是 onActivityCreated() 回调。一旦 Activity 处于 Resumed 状态，则可以在 Activity 中自由地添加或者移除 Fragment。因此，只有当 Activity 处于 Resumed 状态时，Fragment 的生命周期才可以独立变化。当 Activity 离开 Resumed 状态时，Fragment 再一次被 Activity 推入它的生命周期中。

类似于 Activity，Fragment 也有如下三种状态：

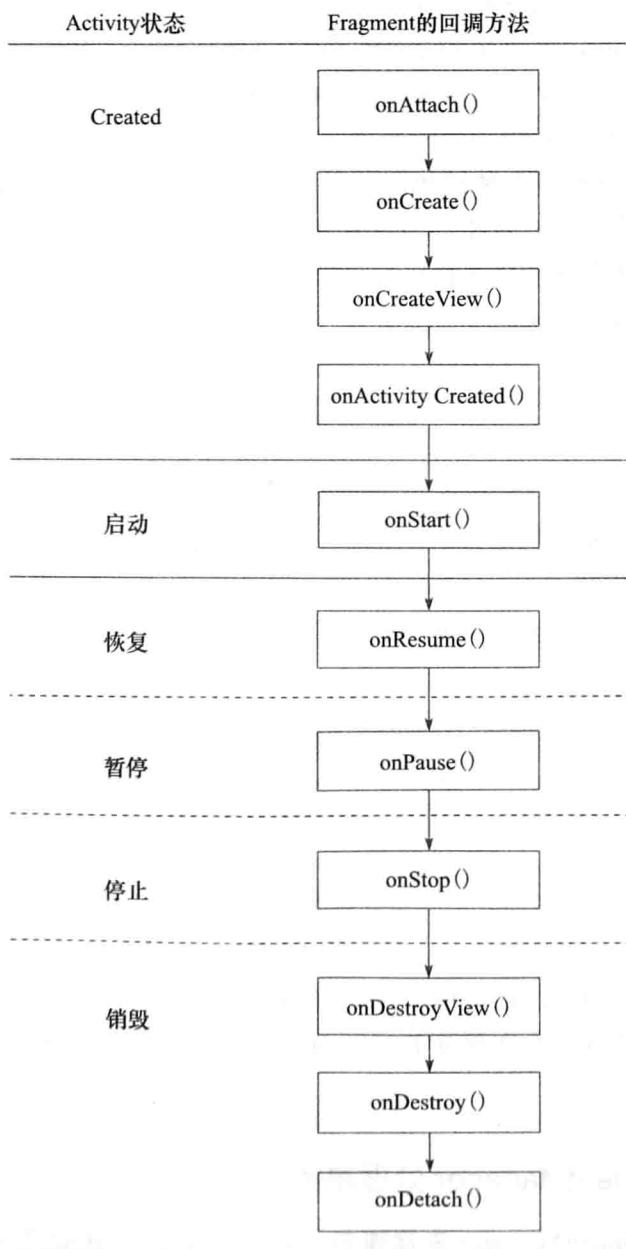


图 11-6 Activity 的生命周期对 Fragment 的生命周期的影响

- Resumed 状态: Fragment 在运行中的 Activity 可见。
- Paused 状态: 另一个 Activity 处于前台且得到焦点, 但是这个 Fragment 所在的 Activity 仍然可见 (前台 Activity 部分透明, 或者没有重载全屏)。
- Stopped 状态: Fragment 不可见。要么宿主 Activity 已经停止, 要么 Fragment 已经从 Activity 上移除, 并被添加到后台栈中。一个停止的 Fragment 仍然存在于内存中 (所有状态和成员信息仍然由系统保留着)。但是, 它对用户来讲已经不再可见, 并且如果 Activity 被杀掉, 它也将被杀掉。

同样类似于 Activity, 可以把 Fragment 的状态保存在一个 Bundle 中, 在 Activity 被重建时就需使用它来恢复。可以在 onSaveInstanceState() 方法中保存状态, 并在 onCreate()、onCreateView() 或 onActivityCreated() 中恢复。但是, Android 只保证在 onDestroy() 之前调用 onSaveInstanceState(), 不保证具体的执行顺序, 因此有可能在调用 onSaveInstanceState() 时, 相关的 View 容器已经无效。因此, 不应在 onSaveInstanceState() 中处理 View 的数据。同样, 不应将 Fragment 重创建后已不存在的对象进行保留, Bundle 携带的数据应尽可能地少, 以减少内存泄漏的风险。如果确实需要进行 View 的处理, 可以通过 FragmentManager 触发 onSaveInstanceState() 方法:

```
01 getFragmentManager().saveFragmentInstanceState(this);
```

saveFragmentInstanceState() 将返回 Fragment.SavedState 对象, 该 State 对象中包含 Fragment 的状态 (含有在 onSaveInstanceState 中保存的 Bundle), 通过 setInitialSavedState() 可以在新的 Fragment 中进行恢复。

Event Locations 应用 MasterFragmentCounty 的 onSaveInstanceState() 方法的代码如下:

```
01 @Override
02 public void onSaveInstanceState(Bundle outState) {
03     super.onSaveInstanceState(outState);
04     outState.putInt("curChoice", mCurCheckPosition);
05     outState.putSerializable("account", accounts);
06 }
```

如上方法的作用是存储当前显示的婚礼举办地的信息。

Fragment 与 Activity 的生命周期中最大的不同就是存储到后退栈中的过程。Activity 是在停止时自动被系统压入停止栈, 并且栈是被系统管理的; 而 Fragment 是被压入 Activity 所管理的一个后退栈, 并且只有在删除 Fragment 后并明确调用 addToBackStack() 方法时才被压入。

### 11.3.2 使用 FragmentManager 处理事务

Android 使用 FragmentManager 来管理 Fragment。获取 FragmentManager 实例的方法是在 Activity 中调用 getFragmentManager() 方法。

FragmentManager 的功能包括:

- 使用 `findFragmentById()` 或 `findFragmentByTag()` 获取 Activity 中存在的 Fragment。  
例如:

```
01 DetailFragment details = (DetailFragment)
02     getSupportFragmentManager().findFragmentById(R.id.details);
```

- 使用 `popBackStack()` 从后台栈弹出 Fragment。例如:

```
01 getSupportFragmentManager().popBackStack(null,
02     FragmentManager.POP_BACK_STACK_INCLUSIVE);
```

- 使用 `addOnBackStackChangeListener()` 注册一个监听后台栈变化的监听器。例如:

```
01 getSupportFragmentManager().addOnBackStackChangeListener(
02     new FragmentManager.OnBackStackChangeListener() {
03         public void onBackStackChanged() {
04             // Update your UI here.
05         }
06     });
```

- 使用 FragmentManager 打开一个 FragmentTransaction 来执行 Fragment 的事务, 例如添加或删除 Fragment。

在 Activity 中使用 Fragment 的一大特点是能根据用户的输入对 Fragment 进行添加、删除、替换以及执行其他动作的能力, 以响应用户的互动。提交给 Activity 的每一系列变化被称为事务, 并且可以用 FragmentTransaction 中的方法处理。也可以将每一个事务保存在由 Activity 管理的后台栈中, 并且允许用户导航回退 Fragment 变更。

每项事务是在同一时间内要执行的一系列的变更。可以为一个给定的事务用相关方法设置想要执行的所有变化, 例如 `add()`、`remove()` 和 `replace()`。然后调用 `commit()` 方法将事务提交给 Activity。

然而, 在调用 `commit()` 之前, 可以用 `addToBackStack()` 将事务添加到一个后退栈中, 该后退栈属于所在的 Activity 管理, 并且允许用户通过按返回键回退到前一个 Fragment 状态。

下面的代码演示了如何使用一个 Fragment 代替另一个 Fragment, 同时在该后退栈中保存被代替的 Fragment 状态的方法。

```
01 FragmentManager manager = getSupportFragmentManager();
02     FragmentTransaction transaction = manager.beginTransaction();
03     Fragment fragment = new DetailFragment();
04     transaction.replace(R.id.account_name, fragment);
05     transaction.addToBackStack(null);
06     transaction.commit();
```

在如上的例子中, Fragment 替换了当前在布局容器中用 `R.id.account_name` 标识的所有 Fragment, 替代的事务被保存在后台栈中, 因此用户可以回退该事务, 可通过按返回键还

原之前的 Fragment。

如果添加多个变更事务（例如另一个 `add()` 或者 `remove()`）并调用 `addToBackStack()`，那么在调用 `commit()` 之前的所有应用的变更被作为一个单独的事务添加到后台栈中，并且返回键可以将它们一起回退。

事务中动作的执行顺序可以随意，但是将变更添加到 `FragmentManager` 中的顺序应注意以下两点：

- 必须在最后调用 `commit()`；
- 如果将多个 `Fragment` 添加到同一个容器中，那么添加顺序决定了它们在视图层次里显示的顺序。

在执行删除 `Fragment` 事务时，如果没有调用 `addToBackStack()`，那么事务一提交，`Fragment` 就会被销毁，而且用户也无法回退它。然而，当移除一个 `Fragment` 时，如果调用了 `addToBackStack()`，那么之后 `Fragment` 会被停止，如果用户回退，它将被恢复过来。

每一个 `Fragment` 事务都是在提交之前通过调用 `setTransition()` 来应用一系列事务动作的。调用 `commit()` 后，事务并不会马上执行，而是采取预约方式。它会在 `Activity` 的 UI 线程中等待，直到线程能执行时才执行。如果必要，可以在 UI 线程中调用 `executePendingTransactions()` 方法来立即执行由 `commit()` 提交的事务。但一般不需这样做，除非有其他线程在等待事务的执行。

注意，只能在 `Activity` 保存状态之前用 `commit()` 提交事务。如果尝试在那时之后提交，则会抛出一个异常。这是因为如果 `Activity` 需要被恢复，提交后的状态会被丢失。对于这类丢失提交的情况，可使用 `commitAllowingStateLoss()`。

### 11.3.3 Fragment 之间的通信

尽管 `Fragment` 的实现是独立于 `Activity` 的，可以被用于多个 `Activity`，但是每个 `Activity` 所包含的是同一个 `Fragment` 的不同的实例。`Fragment` 可以通过调用 `getActivity()` 方法很容易地得到它所在的 `Activity` 的对象，然后就可以查找 `Activity` 中的控件（通过 `findViewById()` 方法）。例如：

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

同样的，`Activity` 也可以通过调用 `FragmentManager` 中的方法查找它所包含的 `Fragment`。例如：

```
01 DetailFragmentCounty details = (DetailFragmentCounty)
02     fragmentManager().findFragmentById(R.id.details);
```

在有些情况下，可能需要 `Fragment` 与 `Activity` 共享事件。一个较好的方法是在 `Fragment` 内部定义一个回调接口，然后在宿主 `Activity` 中实现它。当 `Activity` 通过接口接收到回调时，可以在必要时与布局中的其他 `Fragment` 共享信息。

例如，在 Event Locations 的主界面 Activity 中有两个 Fragment，一个显示城市中的婚礼举办地列表信息（MasterFragmentCounty），另一个显示婚礼举办地的介绍（DetailFragmentCounty），然后 MasterFragmentCounty 必须要告诉 Activity 列表项何时被选中，这样，Activity 可以通知 DetailFragmentCounty 显示该婚礼举办地的详细信息。该功能在本例中的设计步骤如下：

①在 MasterFragmentCounty 的 onItemClick() 方法中，设置左侧列表项的单击事件，核心代码如下：

```
01 @Override
02 public void onItemClick(ListView l, View v, int pos, long id) {
03     ...
04     showDetailsNew(pos);
05 }
```

②实现 showDetailsNew() 方法，核心代码如下：

```
01 void showDetailsNew(int index) {
02     mCurCheckPosition = index;
03     if (mDualPane) {
04         DetailFragmentCounty details = (DetailFragmentCounty)
05             getFragmentManager().findFragmentById(R.id.details);
06         if (details == null || details.getShownIndex() != index) {
07             details = DetailFragmentCounty.newInstance(index);
08             if (accounts != null) {
09                 DetailFragmentCounty.setData(accounts, index);
10             }
11             FragmentTransaction ft = getFragmentManager()
12                 .beginTransaction();
13             ft.replace(R.id.details, details);
14             ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
15             ft.commit();
16         }
17     } else {
18         ...
19     }
20 }
```

Show DetailsNew() 方法的功能是通过 07 行生成 DetailFragmentCounty 实例，并通过 setData() 方法传递索引信息，最后绑定布局中的 FrameLayout 容器，并将 DetailFragmentCounty 实例显示在其中。

这种方式虽然可以实现两个 Fragment 之间的通信，但是非常不好。因为每次单击左侧 ListView 列表项时，都要在右侧重新生成一个 Fragment 实例，浪费系统资源。

下面介绍一种基于接口通信的方式实现 Fragment 之间通信的方法。

首先在 MasterFragmentCounty 内部声明接口 OnChangeListenerLocationListener，代码如下：

```
01 public interface OnChangeListenerLocationListener {
```

```
02     void onChangeLocation(String name);
03 }
04
05 public static void setOnChangeLocationListener(
06     OnChangeLocationListener l) {
07     mChangeLocList = l;
08 }
09
10 @Override
11 public void onItemClick(ListView list, View view, int pos, long id) {
12     ...
13     if (mChangeLocList != null)
14         mChangeLocList.onChangeLocation(mDirList.get(pos));
15 }
```

然后使 `DetailFragmentCounty` 实现 `OnChangeLocationListener` 接口，并在方法 `onChangeLocation()` 中重新获取新的婚礼举办地索引，并根据该索引显示具体信息。

也可以让 `MainActivity` 实现 `OnChangeLocationListener` 接口，在方法 `onChangeLocation()` 中通知 `DetailFragmentCounty` 来自于 `MasterFragmentCounty` 的事件。核心代码如下：

```
01 public static class MainActivity extends Activity
02     implements MasterFragmentCounty.OnChangeLocationListener {
03     ...
04
05     public void onChangeLocation(String name) {
06         ...
07         Bundle args = new Bundle();
08         args.putString(MasterFragmentCounty.index, name);
09         FragmentTransaction transaction =
10             getSupportFragmentManager().beginTransaction();
11         transaction.replace(R.id.fragment_container, MasterFragmentCounty);
12         transaction.addToBackStack(null);
13         transaction.commit();
14     }
15 }
```

当 `Fragment` 添加到 `Activity` 中时，会调用 `Fragment` 的 `onAttach()` 方法，该方法适合检查 `Activity` 是否实现了 `OnChangeLocationListener` 接口，检查方法就是对传入的 `Activity` 的实例进行类型转换，如下所示：

```
01 public static class MasterFragmentCounty extends ListFragment {
02     OnChangeLocationListener mListener;
03     ...
04     @Override
05     public void onAttach(Activity activity) {
06         super.onAttach(activity);
07         try {
08             mListener = (OnChangeLocationListener) activity;
09         } catch (ClassCastException e) {
```

```

10         throw new ClassCastException(activity.toString()
11             + " must implement OnChangeListener");
12     }
13 }
14 }

```

如果 Activity 没有实现 OnChangeListener 接口, 那么 Fragment 会抛出一个 ClassCastException 异常。如果实现了该接口, mListener 成员变量会保留一个 Activity 的 OnChangeListener 实现的引用。由此 MasterFragmentCounty 可以通过调用由 OnChangeListener 接口定义的方法与 Activity 共享事件。例如, 每次用户单击 DirListActivity 中的列表项时, 系统都会调用 Fragment 的 onItemClick() 事件, 然后 Fragment 调用 onChangeLocation() 来与 Activity 共享事件。

## 11.4 PreferenceFragment

SharedPreferences 是一种轻量级的数据存储机制, 它将一些简单数据类型 (如 boolean 型、int 型、float 型、long 型以及 String 型等) 的数据以键值对的形式存储在应用程序的私有 Preferences 目录的 XML 文件中。这种 Preferences 机制广泛应用于存储应用程序中的配置信息。

在设计手机应用时, 多使用 PreferenceActivity 来构建应用的配置界面, 为了体现平板布局的灵活性, 在平板应用中, 一般使用 PreferenceFragment 来构建应用的配置界面, 如图 11-7 所示。

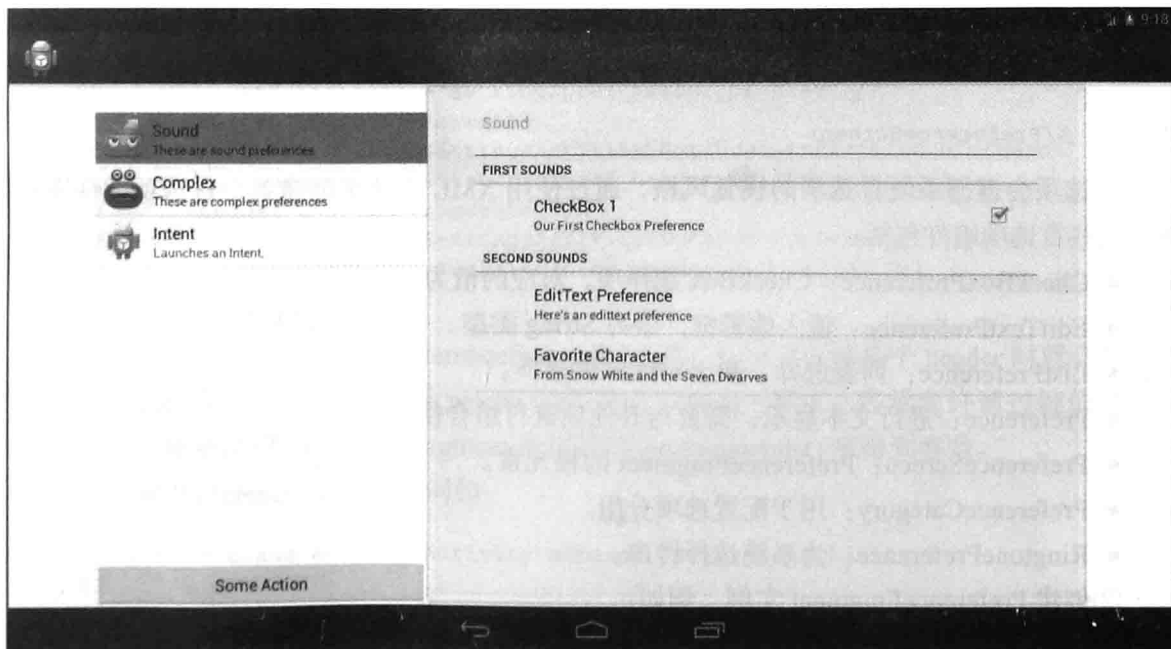


图 11-7 PreferenceFragment 设计效果

PreferenceFragment 以一个列表来展示首选项对象的层级关系，这些首选项将自动地保存为 SharedPreferences，使用户可以用它们来进行交互。为了能够重新获得 SharedPreferences 的实例，该 Fragment 中的层级首选项将会在同一个包下面使用带有一个上下文的 PreferenceManager.getDefaultSharedPreferences() 方法获取 SharedPreferences 实例。

使用 PreferenceFragment 创建一个应用的系统设置界面的基本步骤如下：

①在项目的 res/xml 中新建一个 preferences1.xml 文件用于定义界面的设置选项，例如：

```

01 <?xml version="1.0" encoding="utf-8"?>
02 <PreferenceScreen
03     xmlns:android="http://schemas.android.com/apk/res/android" >
04
05     <CheckBoxPreference
06         android:key="chb1"
07         android:summary="@string/checkbox1_summary"
08         android:title="@string/checkbox1_title" >
09     </CheckBoxPreference>
10
11     <EditTextPreference
12         android:key="address1"
13         android:summary="@string/edittext1_summary"
14         android:title="@string/edittext1_title" >
15     </EditTextPreference>
16
17     <ListPreference
18         android:entries="@array/entries"
19         android:entryValues="@array/entry_values"
20         android:key="list1"
21         android:summary="@string/list1_summary"
22         android:title="@string/list1_title" >
23     </ListPreference>
24
25 </PreferenceScreen>

```

首选项会遵循系统首选项的视觉风格，通过使用 XML 文件来创建各个首选项的视图层级。这些首选项组件包括：

- **CheckBoxPreference**：CheckBox 选择项，对应的值为 true 或 false。
- **EditTextPreference**：输入编辑框，值为 String 类型，会弹出对话框供输入。
- **ListPreference**：列表选择，弹出对话框供选择。
- **Preference**：进行文本显示，需要与其他项进行组合使用。
- **PreferenceScreen**：PreferenceFragment 的根元素。
- **PreferenceCategory**：用于配置选项分组。
- **RingtonePreference**：为系统选择铃声。

②创建 PreferenceFragment 实例，例如：

```

01 public static class Pref1Fragment extends PreferenceFragment{
02     @Override

```



```

03     public void onCreate(Bundle savedInstanceState) {
04         super.onCreate(savedInstanceState);
05         addPreferencesFromResource(R.xml.preferences1);
06     }
07 }

```

通过 `addPreferenceFromResource(int)` 方法加载一个 XML 文件来获得配置界面。XML 文件的根元素为 `PreferenceScreen`。可以调用 `setPreferenceScreen(PreferenceScreen)` 方法来指定一个以 `PreferenceScreen` 为根元素的对象。

为了指定一个 `Intent` 来查询带首选项的 `Activity`，使用了 `addPreferenceFromIntent()` 方法。每个 `Activity` 可以在 `AndroidManifest.xml` 文件中指定 `meta-data` 来指向一个 XML 文件资源。这些资源文件将被填充到单独的首选项层次结构，并且通过该 `Fragment` 来展示。例如：

```

01 <PreferenceScreen android:title=" 设置 wifi">
02     <intent android:action="android.settings.WIFI_SETTINGS" />
03     <EditTextPreference android:title=" 请输入 Wifi地址 "
04         android:key="inputwifi"
05 </PreferenceScreen>

```

③在项目的 `res/xml` 中新建一个 `preference_headers.xml` 文件用于包含 `PreferenceFragment` 实例，例如：

```

01 <preference-headers
02     xmlns:android="http://schemas.android.com/apk/res/android" >
03     <header android:fragment=
04         "cn.liwy.SettingsActivity$GeneralPreferenceFragment"
05         android:title="@string/pref_header_general" />
06     <header android:fragment=
07         "cn.liwy.SettingsActivity$DisplayPreferenceFragment"
08         android:title="@string/pref_header_display" />
09     <header android:fragment=
10         "cn.liwy.SettingsActivity$NotificationPreferenceFragment"
11         android:title="@string/pref_header_notifications" />
12     <header android:fragment=
13         "cn.liwy.SettingsActivity$DataSyncPreferenceFragment"
14         android:title="@string/pref_header_data_sync" />
15 </preference-headers>

```

每一个 `header` 声明一个 `PreferenceFragment` 实例，当用户选择某个 `header` 时就会打开其对应的 `PreferenceFragment`。如果 `header` 标签内嵌 `<extras>` 节点，表示允许通过键值对的形式传参，一般是使用 `Bundle`。`Fragment` 通过调用 `getArguments()` 来得到参数。

④实现 `PreferenceActivity`，例如：

```

01 public class SettingsActivity extends PreferenceActivity {
02     ...
03     @Override
04     public void onBuildHeaders(List<Header> target) {
05         loadHeadersFromResource(R.xml.preference_headers, target);
06     }

```

```
07
08     @Override
09     protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         if (hasHeaders()) {
12             final Button button = new Button(this);
13             button.setText("Some action");
14             setListFooter(button);
15         }
16     }
17 }
```

04 行的 `onBuildHeaders()` 回调方法用来指定头文件，即加载界面对应的 `Fragment` 组。一般通过 `onHeaderClick()` 方法来响应当用户从 `headers` 列表选择一个 `Item` 时，系统打开相关的 `PreferenceFragment`。例如：

```
01 public void onHeaderClick(Header header, int position) {
02     if (header.fragment != null) {
03         if (mSinglePane) {
04             int titleRes = header.breadCrumbTitleRes;
05             int shortTitleRes = header.breadCrumbShortTitleRes;
06             if (titleRes == 0) {
07                 titleRes = header.titleRes;
08                 shortTitleRes = 0;
09             }
10             startWithFragment(header.fragment, header.fragmentArguments,
11                               null, 0, titleRes, shortTitleRes);
12         } else {
13             switchToHeader(header);
14         }
15     } else if (header.intent != null) {
16         startActivity(header.intent);
17     }
18 }
```

注意，当使用 `Preference Headers` 时，`PreferenceActivity` 子类一般不需要重写 `onCreate()` 方法，因为这一 `Activity` 的任务仅仅是载入 `headers` 而已。

## 11.5 平板 UI 设计技巧

随着 Android 系统从 1.5 版到 4.4 版的不断更新，在经历了 Android 3.x（Honeycomb）时代短暂的分离之后，手机和平板平台终于再次在 Android 4.0（Ice Cream Sandwich）版本中迎来了统一。

### 11.5.1 Google 的准则

Google 向开发者发布了 10 条在 Android 平板上开发平板应用的准则，下面是这 10 条准则的详细描述。

### 1. 保证符合 App 的通用开发准则

在谈 Android 平板 App 的开发准则之前，首先要保证 App 符合通用的开发准则，这些准则是所有 Android 设备上的 App 都必须满足的。另外，为了测试开发者设计的 App 是否符合这些准则，开发者需要通过模拟 App 的运行环境进行测试，而如何设置测试环境以及测试应满足哪些规范，开发者同样需要注意。

### 2. 针对平板屏幕大的特性优化 App

如果把为 Android 手机开发的 App 放到 Android 平板上运行，由于屏幕尺寸变大，会出现不同程度的拉伸变形。这时候，对于一些小平板，例如 7 寸的 Nexus 7，开发者只需要做适当的微调，例如放大字体、放大元素、增加元素和边框（padding）、元素和元素（margin）之间的间隙等，就能够满足需求了。

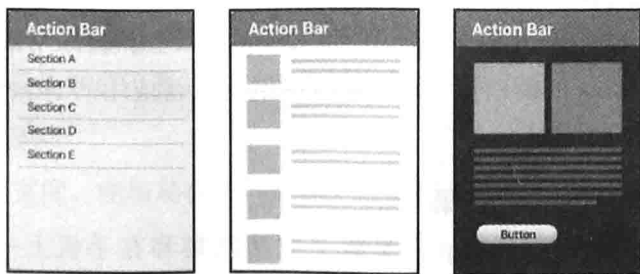
但是，对于一些大平板，例如 Google 即将推出的 10 寸 Nexus 平板，仅做如上微调就无法起作用了。例如一个列表控件，在 7 寸平板上微调一下就能用，但在 10 寸平板上，微调可能出现大片空白，也可能列表每列的字数增加到超过 100 字（每列字数在 50 ~ 75 为宜），这种设计中的拉伸变形应尽量避免。这时开发者应该变废为宝，把多余的空间好好利用起来做些其他事情。

### 3. 利用好平板上多出来的空间

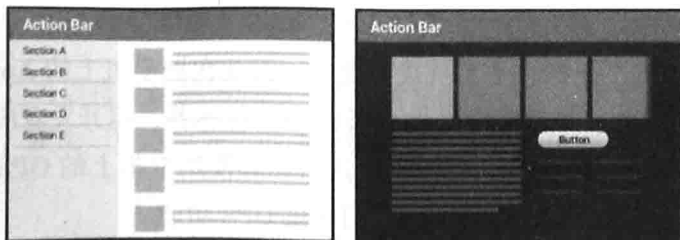
平板，尤其是 10 寸大平板，比起手机的屏幕来，多的地方不是一点半点。要怎么利用好这些多出来的地方呢？这里有一些建议：

- 看是否能够添一些新的内容，或者换一种方式呈现已有的内容；
- 试着将多个界面拼到一起，或者增加一个导航栏，方便用户在多个界面之间随意切换。

如图 11-8a 所示是手机 App 中的 3 个界面，对应导航、内容条目、内容三个层级的界面。而如果是在平板上，可以将 3 个界面合并成图 11-8b 所示的效果。



a) 手机应用的三个界面



b) 平板应用的两个整合界面

图 11-8 利用平板上的多余空间示例

在设计代码时，除了将每一个页面作为一个 Activity 子类，还可以考虑将页面里的内容板块化，每个内容板块单独做一个 Fragment 子类，从而提升代码的重用性。

4. 针对特定屏幕密度使用特定的图标和位图

为了 App 获得最佳效果，App 中的按钮图标或位图必须和特定的屏幕密度匹配。因此对于同一个按钮图标，开发者应该针对 Android 屏幕的 4 种屏幕密度准备 4 张图。

表 11-1 是 4 种典型的控制按钮对应 4 种屏幕密度的参考分辨率。

表 11-1 控制按钮的密度参考

Density	Launcher	Action Bar	Small/Contextual	Notification
mdpi	48 × 48 px	32 × 32 px	16 × 16 px	24 × 24 px
hdpi	72 × 72 px	48 × 48 px	24 × 24 px	36 × 36 px
tvdpi	(use hdpi)	(use hdpi)	(use hdpi)	(use hdpi)
xhdpi	96 × 96 px	64 × 64 px	32 × 32 px	48 × 48 px

5. 调整字体大小和触控按钮

针对 UI 上的标签、触控按钮等所有元素都必须一个一个地调整字体大小，保证在一个标签中，字与字之间没有间隔。触控按钮的标准大小为 48dp（最小为 32dp）。对于一些特定群体的用户，可以适当增大触控按钮。如果触控按钮上的图标很小，一定要将图标居中，同时可以适当扩宽有效的触控区域。

6. 调整桌面 Widgets 的大小

如果开发者要为 App 做一个桌面 Widgets，则它的尺寸、伸缩范围也必须按照屏幕尺寸做适当调整，确保 App 的 Widgets 能够拉伸到 420dp 及以上。确保 Widgets 上含有的图片能够正确地被渲染。Widgets 的留白使用系统默认的取值。将应用的 targetSdkVersion 参数设置为 14，或者更高。

7. 根据平板特性调整 App 功能集

一般情况下，至少要保证 App 在平板上的功能集和在手机上一样大。在某些特殊情况下，如硬件不支持或用户使用平板场景的限制等，可以考虑去掉或者替换掉某些功能。

例如，平板和手机不一样，不支持拨号业务，因此在把手机上的 App 移植到平板上时，应把相关功能去掉。另外，虽然很多平板上都有 GPS 传感器，但开发者应考虑到，用户在使用 GPS 功能时，大多是出门在外，在行走，这时比起使用平板上的 GPS，用户更喜欢用小巧的手机上的 GPS。因此，平板 App 不用专门提供 GPS 功能。

最后，如果开发者在 UI 设计上省去了某项功能，那就一定要保证用户不会通过其他方式获得此项功能。而且，如果因硬件受限要对某项功能降级，确保要降的漂亮。

## 8. 避免利用那些平板可能不支持的硬件功能

手机和平板在传感器、照相机、拨号功能的硬件支持方面可能采取的是不同的策略。为了保证 App 尽量简单通用，一个 APK 就能把事情搞定，最好不要调用平板上一些可能不具有的硬件功能。例如 `android.hardware.telephony`、`android.hardware.camera`、`android.hardware.camera.front`。如果非要调用，一定要事先声明 `android:required="false"`。

## 9. 针对屏幕尺寸进行声明

为保证 App 能适用于各种平板的屏幕，在代码的声明部分，一定要通过 `<supports-screens>` 元素列举各种屏幕尺寸。

## 10. 在 Google Play 上发布 App 的注意事项

Google 鼓励开发者针对所有尺寸的设备（包括手机和平板）只发布一个 APK。如果有手机和平板两个版本，也不要建两个 Google Play 页面，一个就够了，否则会降低品牌影响力。

同时，App 如果有平板上的版本，至少要在 Google Play 页面的屏幕截图区域，放上一张 App 在平板运行的照片，在 App 描述部分也要提到支持平板，在 App 的宣传片包含 App 在平板上运行的镜头。

另外，开发者一定要通过 Google 给的开发者接口去查查你的 App 是否屏蔽了平板设备，确保去掉这个屏蔽。有能力的开发者最好专门为 App 的平板版本做宣传。

## 11.5.2 横竖屏布局设计

在设计平板应用时，要预先为不同的屏幕尺寸和方向设计好应用的布局；要仔细考虑在不同的屏幕方向时，何种复合视图的布局是应用最好的选择；要寻找各种可能，将应用的视图组合成复合视图；要确保在屏幕旋转后，仍然提供相同的功能。具体措施如下。

### 1. 拉伸 / 压缩

调整左边列表的宽度，使布局在不同方向上都保持平衡且合理，如图 11-9 所示。

### 2. 堆放

重新排列面板的堆放方式，以适应不同的屏幕方向，如图 11-10 所示。

### 3. 展开 / 折叠

当屏幕旋转时，可以将左边的列表的一部分折叠起来，只显示重要的信息。并且提供一个“展开”的控件，使用户可以将左边的列表重新调整成原来的宽度，如图 11-11 所示。

### 4. 显示 / 隐藏

旋转屏幕后，让右边的面板占满整个屏幕。通过操作栏中的“向上”按钮隐藏 / 显示左边的列表，如图 11-12 所示。



图 11-9 拉伸 / 压缩复合视图

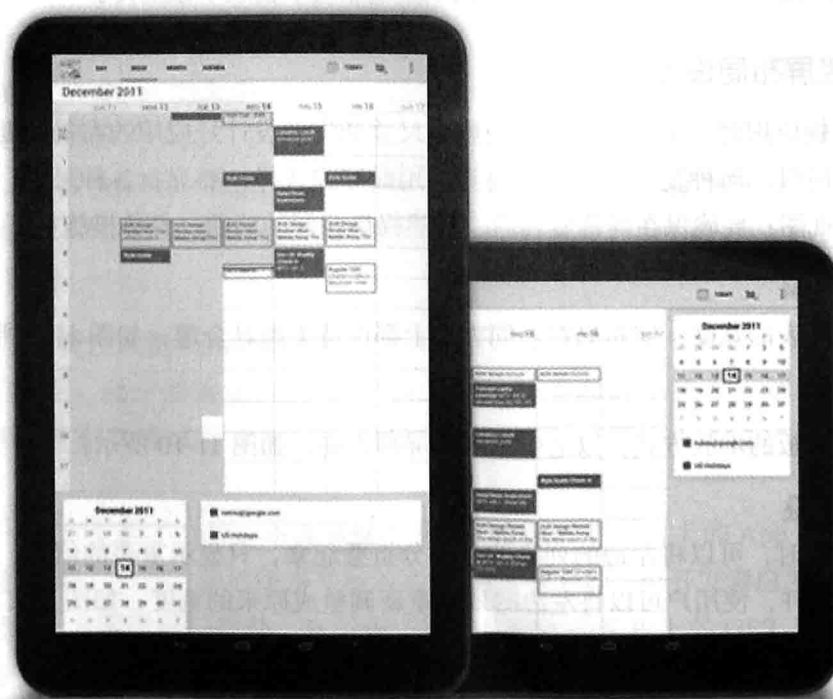


图 11-10 重新堆放复合视图



图 11-11 展开 / 折叠复合视图



图 11-12 显示 / 隐藏复合视图

### 11.5.3 常见平板布局

Android 平板电脑上的应用程序仍然是一个相对较新的空间，一些品牌都还处在试水阶段。以下是一些可以启发灵感的应用程序案例。

#### 1. YouTube

Google 的 YouTube 是一款典型的 Android 平板应用程序设计，它展示了平板布局所有设计模式和 UI 元素，如图 11-13 所示。



图 11-13 YouTube 界面布局

#### 2. CNN

CNN 这款应用程序很好地运用了触摸手势（包括弹出和查看更多内容），拆分视图和字体应用也不错，如图 11-14 所示。



图 11-14 CNN 界面布局



### 3. CNBC

CNBC 是最具视觉冲击力的应用程序之一，包含了丰富的图形和渐变的动态效果，如图 11-15 所示。



图 11-15 CNBC 界面布局

### 4. FlightTrack

FlightTrack 程序包含了漂亮的地图、微妙的动画和标准的蜂巢 UI 元素，数据量大的应用程序也可以设计得很典雅，如图 11-16 所示。



图 11-16 FlightTrack 界面布局

## 11.6 习题

设计一个下图所示布局的平板文件管理器。



## Android UI 综合应用

### 12.1 项目概述

Apollo 音乐播放器<sup>①</sup>是一款基于 Android 平台的开源媒体文件播放应用软件，支持 MP3、WAV、AAC 等多种文件格式，并具备 Internet 流功能，是国外著名固件团队 CyanogenMod 10 的推荐播放器。

Apollo 音乐播放器的主要功能包括：

- 将专辑、艺术家、流派和播放列表快捷方式添加到主屏幕
- 自定义主题和布局
- 歌词支持
- 将歌曲标记为收藏
- 通知播放控制
- 文本和语音搜索
- 滑动切换标签页
- 自动获取专辑封面和艺术家照片
- 耳机线控
- 将歌曲设置为铃声
- 锁屏窗口小部件
- 后台播放

① 项目主页 [https://github.com/Splitter/android\\_packages\\_apps\\_apolloMod](https://github.com/Splitter/android_packages_apps_apolloMod)。

- 支持播放大多数主流的音乐格式
- 拖放播放列表和队列
- 本地音乐管理

## 12.2 用户界面设计

用户界面设计是屏幕产品的重要组成部分。界面设计是一个复杂的有不同学科参与的工程，认知心理学、设计学、语言学等在此都扮演着重要的角色。用户界面设计的三大原则是：置界面于用户的控制之下；减少用户的记忆负担；保持界面的一致性。用户界面设计在工作流程上分为结构设计、交互设计、视觉设计三个部分。

### 12.2.1 结构设计

结构设计（Structure Design）也称概念设计（Conceptual Design），是指对界面内容进行分组，对界面中的信息、数据进行设计使之结构化呈现的过程。结构设计是界面设计的骨架，好的结构设计能使界面信息传达更加清晰、快捷。

通过对用户研究和任务分析，制定出产品的整体架构。基于纸质的低保真原型（Paper Prototype）可提供用户测试并进行完善。在结构设计中，目录体系的逻辑分类和文本标注是用户易于理解和操作的重要前提。应用的结构主要由内容和展示给用户的功能决定。

#### 1. 基本结构

典型的 Android 应用由顶层视图和详细信息 / 编辑视图组成。如果出现层次过深且结构复杂的层级结构，则使用分类目录视图连接顶层和详细信息。基本结构如图 12-1 所示。

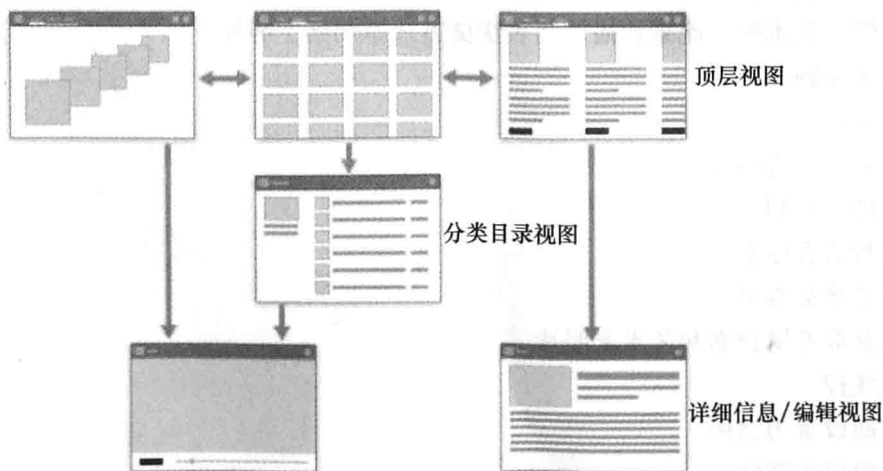


图 12-1 典型应用程序的基本结构

- 顶层视图包含了应用中几个不同的视图。这些视图可以是对于相同数据的不同展示方

式，也可以是应用中各种功能的集合。

- 分类目录视图可以进一步显示数据。
- 详细信息 / 编辑视图中，用户可以创造或者编辑数据。

## 2. 顶层视图

应用的主页的设计需要仔细推敲。人们第一次启动应用时，就将看到该界面，所以应当考虑新用户和老用户的使用习惯。在设计时需要根据应用的主要功能目标来设计应用的主页。

### (1) 显示内容

许多应用主要是用来展示内容的。那么不要使用只有分类导航的界面，而是直接将内容展示在主页上，让人们可以立即看到应用的核心。选择一个视觉上适于所需要展示的数据类型的布局，还要考虑屏幕尺寸。图 12-2 演示了 Apollo 应用的主界面。

Apollo 的主界面通过导航的方式显示艺术家、专辑和播放列表信息，并展示了个性化的推荐和促销内容。

### (2) 操作栏设计

操作栏主要用于导航和操作，在应用的每一个屏幕都要显示操作栏，这样能保持统一的导航体验并且一直显示重要的操作。

顶层的操作栏设计需要考虑以下要求：

- 用操作栏显示应用的图标或者标题。
- 如果顶层是由多个视图组成的，或者当前视图需要在不同的用户账户间切换，那么应当在操作栏加入视图切换菜单，使用户更容易导航。
- 如果应用让人们编写内容，那么应当在顶层就可以直接访问这些内容。
- 如果要提供搜索，那么把搜索放在操作栏中，这样人们可以不用导航而是直接搜索。

### (3) 控件设计

顶层屏幕向用户展示了应用的主要功能。所以有时顶层屏幕会包含多个不同的视图，在设计时要确保用户可以轻松地在多个视图之间切换，Apollo 应用中使用了滚动标签来实现该功能。滚动标签上的条目可能被滑出屏幕，所以，在设计时需要保证用户可以通过左右滑动实现在不同视图间切换。

Apollo 应用通过弹出的菜单让用户选择不同的操作，如图 12-3 所示。



图 12-2 Apollo 应用主界面

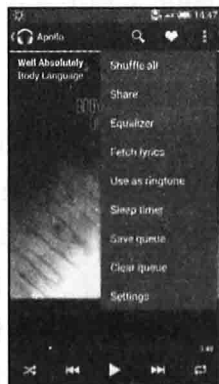


图 12-3 Apollo 应用的弹出菜单

弹出式菜单也可以通过导航抽屉来实现。导航抽屉从屏幕旁边滑入，为用户提供不同视图的选择。它可以放置很多项目，并且可以从屏幕的任何位置调用滑入。导航抽屉可以显示应用的多个顶层视图，还可以显示下一级视图，所以比较适合复杂的应用。

一般的，适合使用导航抽屉的情景包括：

- 不希望标签栏占用过多的垂直空间
- 应用有很多顶层视图
- 应用需要从低层视图直接切换到其他顶层视图
- 应用需要在多个没有直接联系的视图之间切换
- 应用可能会有较深的导航路径

### 3. 分类目录

由数据驱动的应用都是先在整理好的分类目录中浏览，然后再进入详细信息进行查看和管理。可以通过扁平化应用的深度，降低导航的难度。虽然从顶层到详细信息视图的垂直步骤是根据应用的内容而定的，但是仍然有几种方式可以简化认知的难度。

#### (1) 使用标签整合多个分类类别和数据视图

如果分类是相似的或者类别比较少时，使用标签整合多个分类类别和数据视图比较好。因为使用标签可以减少一层的导航，并且使数据一直保持在用户关注的中心。在丰富的内容中导航更像是一种随意的浏览而不是有明确目的导航。

如果分类之间是相似的、可预见的或者关系接近的，则可以使用滚动标签。保证滚动标签中项目的数量不要太多，否则难以操作，常见的规则是 5 ~ 7 个项目。如果分类的类别之间不是很接近，则应当使用固定标签，这样就可以同时看到所有的分类。

#### (2) 允许穿过多个层级的操作

用快捷的方式可以使用户轻松地完成他们所要执行的操作。为了能够控制顶层列表中的数据项，需要在数据项边上显示一个明显的指示，这样，在点击数据项后显示下拉菜单。这样使得人们不需要进入多个层级就可以直接执行操作。图 12-4 显示了 Apollo 应用中音乐专辑长按的弹出菜单，它允许用户在专辑视图中直接操作歌曲，这样就不需要进入歌曲的详细信息视图，简化了操作。

#### (3) 同时对多个项目进行操作

尽管分类视图一般是给人们导航到详细信息用的，但是，提供一些操作使其可以直接控制多条数据也是有必要的。例如，如果允许用户在详细信息视图删除某个项目，那么应当允许用户在分类视图中一次性删除多个项目。在设计应用时，需要仔细考虑哪些操作可以作用于多个项目，通过多选界面提供这些操作，让用户可以在分类视图中直接使用。

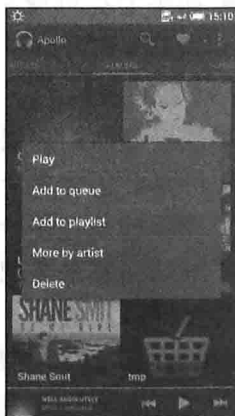


图 12-4 专辑长按的弹出菜单

#### 4. 详细信息

详细信息视图用于显示和处理数据。详细信息视图的布局根据需要显示的数据的不同而不同。图 12-5 是 Apollo 应用中音乐播放列表的详细信息界面。

点击播放列表中的音乐条目，显示如图 12-6 所示的音乐播放的详细信息界面。



图 12-5 播放列表界面

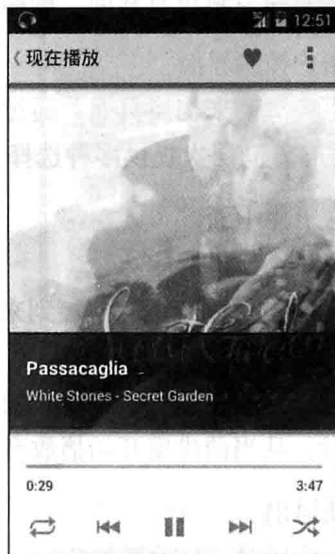


图 12-6 音乐播放界面

在设计详细信息视图时，需要考虑应用所进行的操作，并进行布局，使详细信息视图之间的导航变得简单。如果用户经常顺序浏览多个项目，那么应当让他们在详细信息视图中直接做到。例如，在 Apollo 应用中，点击图 12-2 所示的应用主界面下方的播放控制区域，就可以转到图 12-6 所示的音乐播放界面。

### 12.2.2 交互设计

交互设计 (Interactive Design) 是一种目标导向设计，所有的工作内容都是围绕着用户行为去设计的。交互设计师通过设计用户的行为，让用户更方便、更有效率地完成产品业务目标，获得愉快的用户体验。

在 Apollo 应用的交互设计中，主要做了如下交互设计。

#### 1. 突出内容

在 Apollo 应用的界面中，使用用户的语言，以用户熟悉的维度来组织内容，例如，在主界面以音乐专辑的形式给出音乐信息，这样更容易查找目标信息，提升内容的利用率。某专辑的 UI 设计如图 12-7 所示，这样的设计删除了无关的多余内容，让内容更简洁清晰和具体。



图 12-7 音乐专辑布局界面

## 2. 有清楚的错误提示

在进行误操作后，系统提供有针对性的提示，包括：

- 在音乐播放前，先检查可播放性。
- 在选择主题前，先检查网络可用性。
- 如果音乐专辑没有封面，则给出默认的封面，而不是黑框。

## 3. 让用户控制界面

通过手势滑动来切换页面，通过应用图标来导航视图，在整个应用中，提供了统一的用户控制，并在不同层次提供多种选择，给不同层次的用户提供多种可能性。

## 4. 允许工作中断

这些中断设计包括：

- 若在音乐播放的过程中接到来电提醒，则自动暂停音乐播放，完成电话事件后，继续音乐播放。
- 若在音乐播放的过程中手机没有电了，则自动存储当前音乐的播放进度，下次进入应用时，从当前进度开始播放音乐。

## 5. 方便退出

在退出应用时，分为两种情况：

- 如果用户是在暂停音乐后点击“返回”或 Home 按钮来退出的，则退出应用，停止音乐播放。
- 如果用户是在音乐播放的过程中点击“返回”或 Home 按钮来退出的，则退出应用，但是音乐将转入后台继续播放，并在系统通知栏显示如图 12-8 所示的通知信息。



图 12-8 通知栏消息界面

## 12.2.3 视觉设计

在结构设计的基础上，参照目标群体的心理模型和任务达成进行视觉设计（Visual Design）。包括色彩、字体、页面等。视觉设计要达到用户愉悦使用的目的。

在 Apollo 应用的视觉设计中，主要做了如下视觉设计。

### 1. 界面清晰明了

应用界面整体风格一致、协调，并允许用户通过设置选项进行界面定制。

### 2. 依赖认知而非记忆

通过隐喻的图标来帮助用户了解应用的信息。例如，在图 12-9 的音乐播放界面中，



①处为下拉菜单, ②处为专辑封面, ③处为音乐动效, ④处为音乐循环模式。

### 3. 注重共性, 抓住个性

分析了解同类的音乐播放 App 及各自图标设计的定位, 找到设计方向的共性及其自身软件的独特个性。例如, 在应用主界面中, 将音乐播放控制面板至于底层, 无论是在应用的主导航视图 (如图 12-2 所示), 还是点击专辑后进入的音乐播放列表界面 (如图 12-5 所示), 播放控制面板始终处于可用状态。

### 4. 其他

包括:

- 完善视觉的清晰度。条理清晰, 图片、文字的布局 and 隐喻不需要让用户去猜。
- 同样功能用同样的图形。
- 整体软件不超过 5 个色系, 近似的颜色表示近似的意思。



图 12-9 音乐播放界面

## 12.3 用户界面功能实现

在本节主要实现 Apollo 音乐播放器的如下功能:

- 应用主界面容器设计
- 歌曲列表界面设计
- 系统设计界面设计
- 桌面应用组件设计

### 12.3.1 主界面设计

Apollo 音乐播放器的主界面包括三个部分 (如图 12-2 所示), 顶部是一个个性化的标题栏, 中间主体是一个基于 ViewPager 的可左右滑动的容器页面, 底部是一个音乐播放的微播放控制面板。

#### 1. 主界面布局设计

主界面 MusicLibrary 的布局文件 library\_browser.xml 的核心代码如下:

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <RelativeLayout
03     xmlns:android="http://schemas.android.com/apk/res/android"
04     android:layout_width="match_parent"
```

```

05     android:layout_height="wrap_content" >
06
07     <com.andrew.apolloMod.ui.widgets.ScrollableTabView
08         android:id="@+id/scrollingTabs"
09         android:layout_width="match_parent"
10         android:layout_height="@dimen/tab_height" />
11
12     <android.support.v4.view.ViewPager
13         android:id="@+id/viewPager"
14         android:layout_width="match_parent"
15         android:layout_height="wrap_content"
16         android:layout_below="@+id/scrollingTabs" />
17
18     <LinearLayout
19         android:layout_width="match_parent"
20         android:layout_height="@dimen/bottom_action_bar_height"
21         android:layout_alignParentBottom="true" >
22
23         <fragment
24             android:id="@+id/bottomactionbar_new"
25             android:name
26             ="com.andrew.apolloMod.ui.fragments.BottomActionBarFragment"
27             android:layout_width="match_parent"
28             android:layout_height="match_parent"
29             android:layout_weight="1" />
30     </LinearLayout>
31
32 </RelativeLayout>

```

其中 07~10 行声明了容器头部的标签页，12~16 行是主体内容的承载容器，18 行的线性布局载入了 23~29 行描述的底部微播放控制面板。

## 2. 主界面 Activity 设计

主界面 Activity 主要实现以下功能。

- 加载容器子视图
- 实现动作栏
- 实现主菜单
- 绑定音乐服务

### (1) 加载容器子视图

加载容器子视图的过程包括实现 ViewPager 的适配器 PagerAdapter、实现子视图 Fragment，核心代码如下：

```

01 public void initPager() {
02     PagerAdapter mPagerAdapter = new PagerAdapter(getFragmentManager());

```

```

03     ...
04     if (tabs_set.contains(getResources().getString(R.string.tab_recent)))
05         mPagerAdapter.addFragment(new RecentlyAddedFragment(bundle));
06     ViewPager mViewPager = (ViewPager) findViewById(R.id.viewPager);
07     mViewPager.setAdapter(mPagerAdapter);
08     initScrollableTabs(mViewPager);
09     ThemeUtils.initThemeChooser(this, mViewPager, "viewpager",
10         THEME_ITEM_BACKGROUND);
11 }

```

04~05 行是依次向容器 ViewPager 中添加子视图 Fragment, 08 行的 initScrollableTabs() 用于初始化容器顶部的标签栏, 并实现标签栏和容器内容的联动, 代码如下:

```

01 public void initScrollableTabs(ViewPager mViewPager) {
02     ScrollableTabView mScrollingTabs = (ScrollableTabView)
03         findViewById(R.id.scrollingTabs);
04     ScrollingTabsAdapter mScrollingTabsAdapter =
05         new ScrollingTabsAdapter(this);
06     mScrollingTabs.setAdapter(mScrollingTabsAdapter);
07     mScrollingTabs.setViewPager(mViewPager);
08 }

```

## (2) 实现动作栏

实现动作栏的方法 initActionBar() 主要用于个性化动作栏, 代码如下:

```

01 private void initActionBar() {
02     ActionBar actBar = getActionBar();
03     int upId = Resources.getSystem().getIdentifier("up", "id", "android");
04     ImageView actionBarUp = (ImageView) findViewById(upId);
05     ThemeUtils
06         .setActionBarBackground(this, actBar, "action_bar_background");
07     ThemeUtils.initThemeChooser(this, actionBarUp, "action_bar_up",
08         THEME_ITEM_BACKGROUND);
09     actBar.setDisplayUseLogoEnabled(true);
10     actBar.setDisplayShowTitleEnabled(false);
11 }

```

## (3) 实现主菜单

首先创建主菜单资源 actionbar\_top.xml, 包括显示在动作栏中的查找菜单和 3 个溢出菜单, 代码如下:

```

01 <?xml version="1.0" encoding="utf-8"?>
02 <menu xmlns:android="http://schemas.android.com/apk/res/android" >
03     <item
04         android:id="@+id/action_search"
05         android:icon="@drawable/apollo_holo_light_search"
06         android:orderInCategory="100"

```

```

07         android:showAsAction="always"
08         android:title="@string/cd_search"/>
09     <item
10         android:id="@+id/action_overflow"
11         android:icon="@drawable/apollo_holo_light_overflow"
12         android:orderInCategory="100"
13         android:showAsAction="always"
14         android:title="@string/cd_overflow">
15         <menu>
16             <item
17                 android:id="@+id/action_settings"
18                 android:title="@string/settings"/>
19             <item
20                 android:id="@+id/action_equalizer"
21                 android:title="@string/equalizer"/>
22             <item
23                 android:id="@+id/action_shuffle_all"
24                 android:title="@string/shuffle_all"/>
25         </menu>
26     </item>
27
28 </menu>

```

在 MusicLibrary 中实现主菜单的 3 个回调方法 onCreateOptionsMenu(Menu)、onPrepareOptionsMenu(Menu) 和 onOptionsItemSelected(MenuItem)，其中的 onPrepareOptionsMenu() 代码如下：

```

01 @Override
02 public boolean onPrepareOptionsMenu(Menu menu) {
03     MenuItem search = menu.findItem(R.id.action_search);
04     MenuItem overflow = menu.findItem(R.id.action_overflow);
05     ThemeUtils.setActionBarItem(this, search, "apollo_search");
06     ThemeUtils.setActionBarItem(this, overflow, "apollo_overflow");
07     return super.onPrepareOptionsMenu(menu);
08 }

```

该方法主要是根据当前应用的主题来设置主菜单的样式。

### 3. 微播放控制面板设计

主界面下方的微播放控制面板通过布局文件 library\_browser.xml 加载 BottomActionBarFragment，该 Fragment 主要实现面板中播放、上一首和下一首 3 个按钮功能的实现。在 onCreateView(LayoutInflator, ViewGroup, Bundle) 方法中，通过 LayoutInflator.inflate() 方法加载 Fragment 的布局文件 bottom\_action\_bar.xml，在该文件中包含了自定义的 LinearLayout 对象 BottomActionBar，该对象实现 OnClickListener 接口，实现方法如下：

```

01 @Override
02 public void onClick(View v) {
03     switch (v.getId()) {
04         case R.id.bottom_action_bar:
05             Intent intent = new Intent();
06             intent.setClass(v.getContext(), AudioPlayerHolder.class);
07             intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
08                 Intent.FLAG_ACTIVITY_NEW_TASK);
09             v.getContext().startActivity(intent);
10             break;
11         default:
12             break;
13     }
14 }

```

这样,当单击下方的微播放控制面板时,即打开图 12-6 所示的音乐播放主界面。

### 12.3.2 歌曲列表界面设计

歌曲列表界面如图 12-10 的中部所示,是一个以列表的形式显示的界面。

列表的每个子项包括 3 个部分,如图 12-11 所示。

其中,①是歌曲名称,②是演唱者,③是内容菜单触发按钮,单击③显示图 12-12 所示的内容菜单。

歌曲列表界面设计步骤如下:

①创建一个继承自 RefreshableFragment 的 Tracks Fragment, RefreshableFragment 是实现 refresh() 抽象方法的 Fragment。

②在 TracksFragment 的 onCreateView(LayoutInflator, ViewGroup, Bundle) 方法中加载歌曲列表界面布局文件 listview.xml。

③在 TracksFragment 中重载 onCreateLoader(int,Bundle) 方法来加载 URI 为 Audio.Media.EXTERNAL\_CONTENT\_URI 中的音乐文件,并将结果通过 onLoadFinished(Loader<Cursor>, Cursor) 方法赋给 Cursor 对象 data。

④为 listview.xml 中的 ListView 定义列表子项布局文件 listview\_items.xml 和适配器 TrackAdapter, Track Adapter 的核心代码如下:

```

01 public class TrackAdapter extends SimpleCursorAdapter {

```



图 12-10 歌曲列表界面

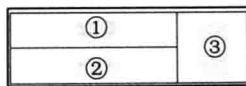


图 12-11 歌曲列表子项的构成

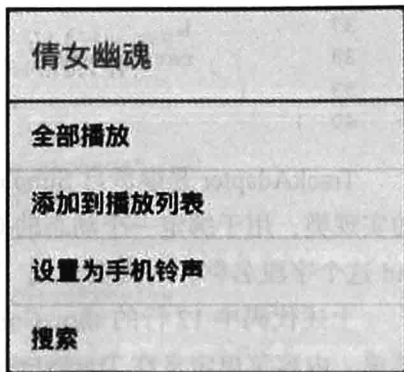


图 12-12 歌曲列表子项的内容菜单

```

02     ...
03     public TrackAdapter(Context context, int layout, Cursor c, String[] from,
04         int[] to, int flags) {
05         super(context, layout, c, from, to, flags);
06     }
07
08     private final View.OnClickListener showContextMenu =
09         new View.OnClickListener() {
10             @Override
11             public void onClick(View v) {
12                 v.showContextMenu();
13             }
14         };
15
16     @Override
17     public View getView(int position, View convertView, ViewGroup parent) {
18         final View view = super.getView(position, convertView, parent);
19         Cursor mCursor = (Cursor) getItem(position);
20         ViewHolderList viewholder;
21
22         if (view != null) {
23             viewholder = new ViewHolderList(view);
24             holderReference = new WeakReference<ViewHolderList>(viewholder);
25             view.setTag(holderReference.get());
26         } else {
27             viewholder = (ViewHolderList) convertView.getTag();
28         }
29
30         String trackName = mCursor.getString(TracksFragment.mTitleIndex);
31         viewholder.mViewHolderLineOne.setText(trackName);
32         String artistName = mCursor.getString(TracksFragment.mArtistIndex);
33         ...
34         } else {
35             holderReference.get().mPeakOne.setImageResource(0);
36             holderReference.get().mPeakTwo.setImageResource(0);
37         }
38         return view;
39     }
40 }

```

TrackAdapter 是继承自 SimpleCursorAdapter 的适配器，SimpleCursorAdapter 是 ListAdapter 的实现类，用于绑定一个动态的 Cursor。使用 SimpleCursorAdapter 所适配的数据表一定要有 `_id` 这个字段名称，否则会出现“找不到 `_id` 字段”的错误。

上述代码中 12 行的 `showContextMenu()` 方法用于单击图 12-11 所示的③部分时显示内容菜单，内容菜单定义在 TracksFragment 中，核心代码如下：

```

01 @Override
02 public void onCreateContextMenu(ContextMenu menu, View v,
03     ContextMenuInfo menuInfo) {

```

```

04 menu.add(0, PLAY_SELECTION, 0,
05         getResources().getString(R.string.play_all));
06 ...
07 AdapterContextMenuInfo mi = (AdapterContextMenuInfo)menuInfo;
08 mSelectedPosition = mi.position;
09 mCursor.moveToPosition(mSelectedPosition);
10
11 try {
12     mSelectedId = mCursor.getLong(mMediaIdIndex);
13 } catch (IllegalArgumentException ex) {
14     mSelectedId = mi.id;
15 }
16
17 String title = mCursor.getString(mTitleIndex);
18 menu.setHeaderTitle(title);
19 super.onCreateContextMenu(menu, v, menuInfo);
20 }

```

⑤实现列表子项的 `onClick()` 回调监听方法和内容菜单的 `onContextItemSelected(MenuItem)` 方法。

### 12.3.3 系统设置界面设计

执行主菜单中的设置菜单命令, 显示图 12-13 所示的系统设置界面。

系统设置界面设计步骤如下:

①在项目的 `Res` 的 `xml` 文件夹下新建系统设置界面布局文件 `settings.xml`, 代码如下:

```

01 <?xml version="1.0" encoding="utf-8"?>
02 <PreferenceScreen
03     xmlns:android="http://schemas.android.com/apk/res/android" >
04     <PreferenceCategory android:title="@string/header_interface" >
05         <PreferenceScreen
06             android:icon="@drawable/apollo_settings_themes"
07             android:key="@string/key_themes_preferences"
08             android:title="@string/themes" >
09             <ListPreference
10                 android:key="@string/key_themes_package"
11                 android:summary="@string/apollo_themes"
12                 android:title="@string/select_theme" />
13
14             <com.andrew.apolloMod.preferences.ThemePreview
15                 android:key="@string/key_themes"
16                 android:layout="@layout/theme_preview" />
17         </PreferenceScreen>
18
19         <MultiSelectListPreference
20             android:capitalize="words"

```

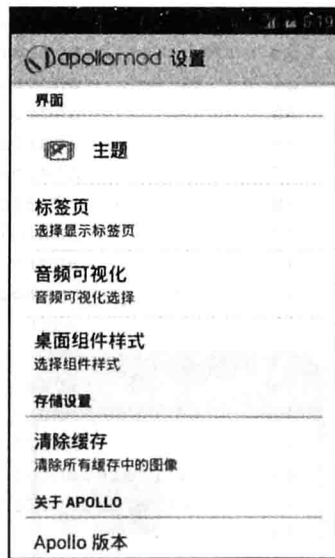


图 12-13 Apollo 应用设置界面

```

21         android:defaultValue="@array/tab_titles"
22         android:dialogTitle="@string/tab_visibility_title"
23         android:entries="@array/tab_titles"
24         android:entryValues="@array/tab_titles"
25         android:key="@string/key_tabs_enabled"
26         android:summary="@string/tab_visibility_summary"
27         android:title="@string/tab_visibility_title" />
28
29     <ListPreference
30         android:capitalize="words"
31         android:defaultValue="@string/visual_none"
32         android:dialogTitle="@string/visualizer_title"
33         android:entries="@array/visualization_types"
34         android:entryValues="@array/visualization_types"
35         android:key="@string/key_visualization_type"
36         android:summary="@string/visualizer_summary"
37         android:title="@string/visualizer_title" />
38     <ListPreference
39         android:capitalize="words"
40         android:defaultValue="@string/widget_style_light"
41         android:dialogTitle="@string/widget_style_title"
42         android:entries="@array/widget_style_types"
43         android:entryValues="@array/widget_style_types"
44         android:key="@string/key_widget_style"
45         android:summary="@string/widget_style_summary"
46         android:title="@string/widget_style_title" />
47 </PreferenceCategory>
48 <PreferenceCategory android:title="@string/settings_storage_category" >
49
50     <!-- Delete cache -->
51     <Preference
52         android:key="delete_cache"
53         android:summary="@string/settings_delete_cache_summary"
54         android:title="@string/settings_delete_cache_title" />
55 </PreferenceCategory>
56 <PreferenceCategory android:title="@string/about" >
57     <Preference
58         style="?android:preferenceInformationStyle"
59         android:enabled="false"
60         android:key="@string/key_build_version"
61         android:summary="1.0"
62         android:title="@string/version" />
63 </PreferenceCategory>
64
65 </PreferenceScreen>

```

整个设置界面包括界面设置、存储设置和关于 Apollo 三个部分。

②实现 PreferenceActivity 的扩展类 SettingsHolder，并在 onCreate(Bundle) 方法中加载配置文件，部分代码如下：

```

01 int preferencesResId = R.xml.settings;
02 addPreferencesFromResource(preferencesResId);

```



③ 在 MusicLibrary 的 onOptionsItemSelected(MenuItem) 回调方法的 R.id.action\_settings 分支调用如下语句来启动系统设置界面:

```
01 startActivityForResult(new Intent(this, SettingsHolder.class), 0);
```

④ 在 MusicLibrary 的 onActivityResult(int, int, Intent) 回调方法响应系统设置的更改, 代码如下:

```
01 protected void onActivityResult(int requestCode, int resultCode,
02     Intent data) {
03     Intent i = getBaseContext().getPackageManager()
04         .getLaunchIntentForPackage(getBaseContext().getPackageName());
05     i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
06     startActivity(i);
07 }
```

### 12.3.4 桌面应用组件设计

Apollo 音乐播放器的桌面应用组件包括  $1 \times 1$ 、 $4 \times 1$  和  $4 \times 2$  三种风格。本节主要介绍图 12-14 所示的  $4 \times 2$  风格桌面应用组件的设计。

设计步骤如下:

① 在项目的 Res/layout 文件夹中创建桌面应用组件布局文件 fourbytwo\_app\_widget.xml。

② 在项目的 Res/xml 文件夹中创建桌面应用组件 AppWidget-ProviderInfo 文件 appwidget4x2\_info.xml, 代码如下:

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <appwidget-provider
03     xmlns:android="http://schemas.android.com/apk/res/android"
04     android:initialLayout="@layout/fourbytwo_app_widget"
05     android:minHeight="110dp"
06     android:minWidth="250dp"
07     android:updatePeriodMillis="0" />
```



图 12-14 桌面应用组件界面

③ 创建 AppWidgetProvider 实例 AppWidget42, 核心代码如下:

```
01 public class AppWidget42 extends AppWidgetProvider {
02     ...
03     public static synchronized AppWidget42 getInstance() {
04         if (sInstance == null) {
05             sInstance = new AppWidget42();
06         }
07         return sInstance;
08     }
09
10     @Override
11     public void onUpdate(Context context, AppWidgetManager appWidgetManager,
```

```

12         int[] appWidgetIds) {
13             defaultAppWidget(context, appWidgetIds);
14             Intent updateIntent = new Intent(ApolloService.SERVICECMD);
15             updateIntent.putExtra(ApolloService.CMDNAME,
16                 AppWidget42.CMDAPPWIDGETUPDATE);
17             updateIntent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_IDS,
18                 appWidgetIds);
19             updateIntent.addFlags(Intent.FLAG_RECEIVER_REGISTERED_ONLY);
20             context.sendBroadcast(updateIntent);
21         }
22
23     private void linkButtons(Context context, RemoteViews views,
24         boolean playerActive) {
25         Intent intent;
26         PendingIntent pendingIntent;
27         final ComponentName serviceName = new ComponentName(context,
28             ApolloService.class);
29
30         if (playerActive) {
31             intent = new Intent(context, AudioPlayerHolder.class);
32             pendingIntent = PendingIntent.getActivity(context, 0, intent, 0);
33             views.setOnClickPendingIntent(R.id.four_by_two_albumart,
34                 pendingIntent);
35             views.setOnClickPendingIntent(R.id.four_by_two_info,
36                 pendingIntent);
37         } else {
38             intent = new Intent(context, MusicLibrary.class);
39             pendingIntent = PendingIntent.getActivity(context, 0, intent, 0);
40             views.setOnClickPendingIntent(R.id.four_by_two_albumart,
41                 pendingIntent);
42             views.setOnClickPendingIntent(R.id.four_by_two_info,
43                 pendingIntent);
44         }
45         ...
46     }
47 }

```

其中的 `onUpdate()` 方法自动响应音乐播放进程的界面更新, `linkButtons()` 方法实现界面按钮的响应事件处理。

④在 `AndroidManifest.xml` 中注册 `AppWidget42`, 代码如下:

```

01 <receiver
02     android:name="com.andrew.apolloMod.app.widgets.AppWidget42"
03     android:label="@string/apollo_4x2" >
04     <intent-filter>
05         <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
06     </intent-filter>
07
08     <meta-data
09         android:name="android.appwidget.provider"

```

```

10         android:resource="@xml/appwidget4x2_info" />
11     </receiver>

```

## 12.4 UI 测试

UI 测试是指测试用户界面的风格是否满足客户要求、文字是否正确、页面是否美观、文字和图片组合是否完美、操作是否友好等。UI 测试的目标是确保用户界面会通过测试对象的功能来为用户提供相应的访问或浏览功能。确保用户界面符合公司或行业的标准。包括用户友好性、人性化、易操作性测试。

本节通过百度移动云测试中心主要测试 Apollo 音乐播放器在多设备、多屏幕尺寸、多平台版本等方面的 UI 设计情况。

百度移动云测试中心简称 MTC (Mobile Testing Center), 该中心为开发者提供了上百种主流厂商的移动终端设备及增强模拟器, 涵盖了 Top 100 Android 真机和各种配置的模拟器, 方便开发者进行实时的手机应用开发和测试工作。百度移动云测试服务通过“两种类型”、“三个维度”、“六种测试”十分钟内完成上百种主流手机环境下的自动化测试, 全面检测 Android 应用。

下面介绍基于百度移动云测试中心对 Apollo 音乐播放器进行 UI 测试的流程。

具体实现步骤如下:

①登录百度移动云测试中心 (<http://mtc.baidu.com/>), 单击 Native App 测试按钮, 进入图 12-15 所示的设置测试项界面。

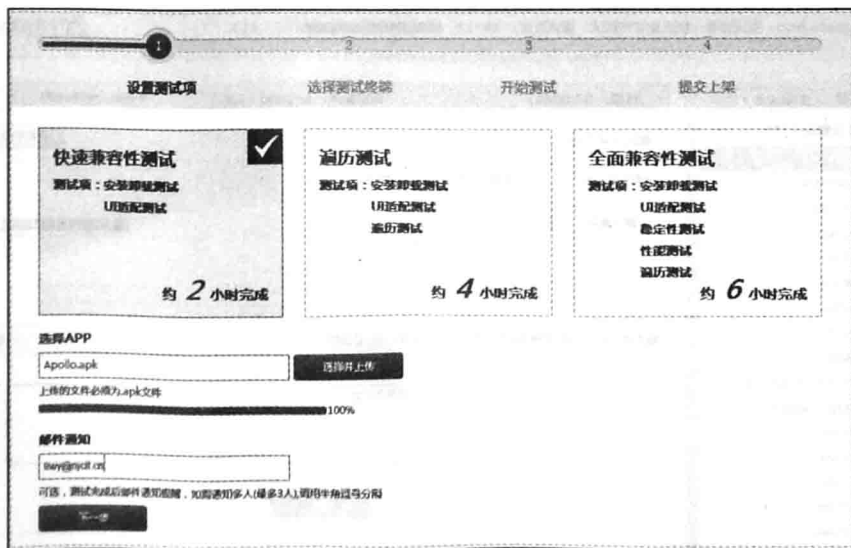


图 12-15 设置测试项界面

②单击“选择并上传”按钮, 提交 Apollo.apk, 在邮件通知中输入邮件地址, 单击“下一步”按钮, 进入图 12-16 所示的选择测试终端界面。

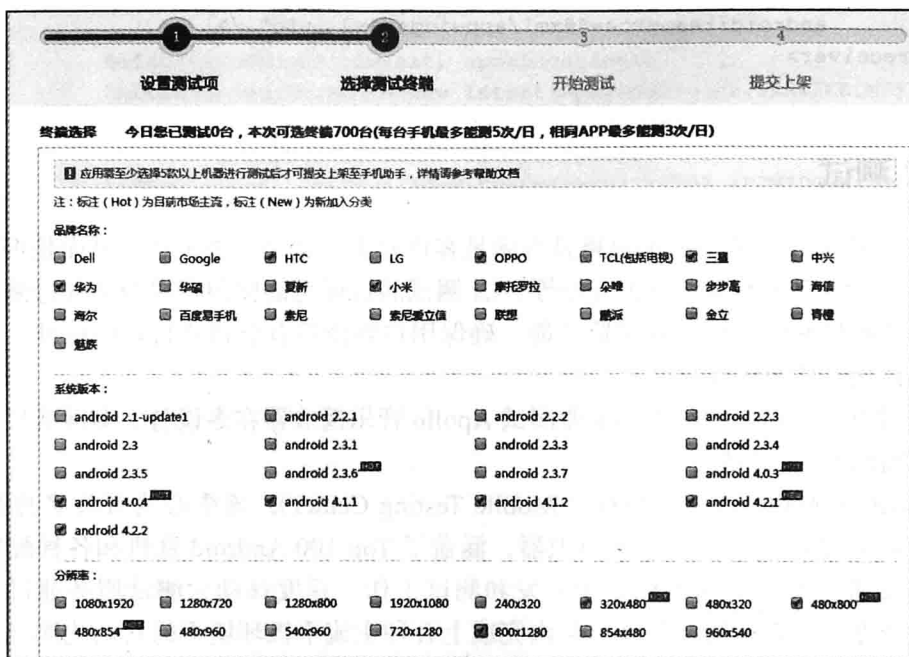


图 12-16 选择测试终端

③在品牌选择、系统版本和分辨率中选择目标测试终端, 单击“启动测试”按钮, 进入图 12-17 所示的测试执行界面。

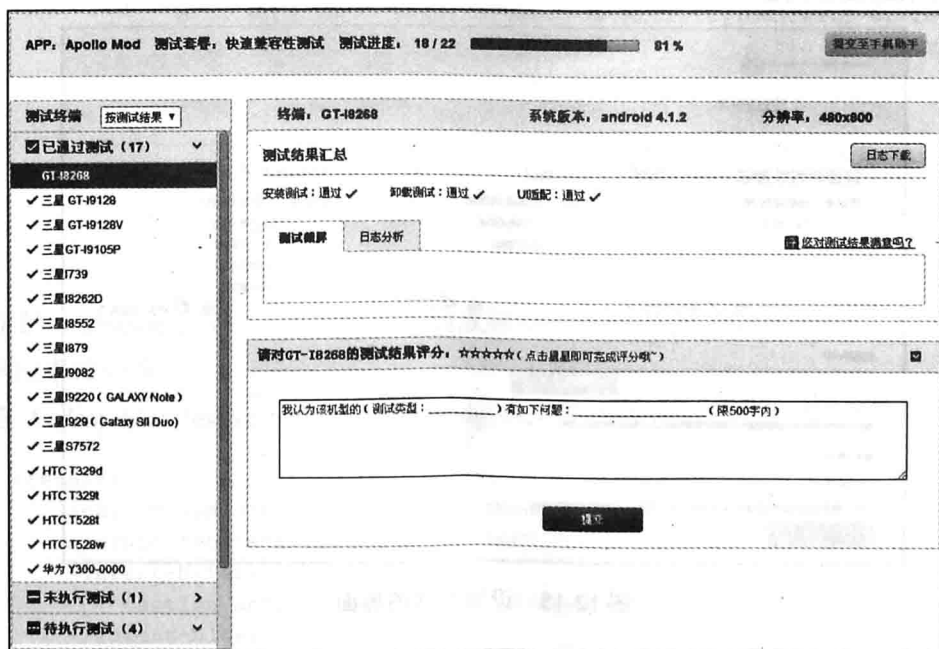


图 12-17 测试执行

④测试完毕后，单击查看测试报告，显示如图 12-18 所示的测试基础信息界面。

[综合概述](#)
[测试基础信息](#)
[下载报告](#)
[查看测试详站](#)
[提交至手机助手](#)

APP名称: Apollo Mod(1.0)	版本: 1.0	测试日期: 2014-02-08	测试网络环境: WLAN
选择终端数: 22款	实际终端执行数: 21款	测试终端通过数: 21款	
品牌覆盖: 三星 HTC 华为			
系统版本覆盖: android 4.1.2 android 4.0.4 android 4.1.1 android 4.2.1			
分辨率覆盖: 480x800 800x1280			
测试套餐: 快速兼容性测试		测试项: 安装卸载测试 UI适配性测试	

**测试通过 (21)**

三星I9220 ( GALAXY Note ) ★★★★★	三星I9228 ( Galaxy Note ) ★★★★★	三星I929 ( Galaxy SII Duo ) ★★★★★
三星I9108 ★★★★★	三星I9220 ( GALAXY Note ) ★★★★★	HTC T329t ☆☆☆☆☆
HTC T329d ☆☆☆☆☆	三星S7572 ☆☆☆☆☆	HTC T528t ☆☆☆☆☆
HTC T528t ☆☆☆☆☆	华为 Y300-0000 ☆☆☆☆☆	更多

图 12-18 测试基础信息

⑤如果测试通过，单击“提交至手机助手”按钮即可打开应用的信息编辑页面，输入应用相关的信息后，单击“提交”按钮即可完成应用的发布。

## 参考文献

- [1] Greg Nudelman. Android 应用 UI 设计模式 [M]. 袁国忠, 译. 北京: 人民邮电出版社, 2013.
- [2] Theresa Neil. 移动应用 UI 设计模式 [M]. 王军锋, 译. 北京: 人民邮电出版社, 2013.
- [3] Jakob Nielsen. 贴心设计: 打造高可用性的移动产品 [M]. 牛化成, 译. 北京: 人民邮电出版社, 2013.
- [4] Chris Haseman. Android 应用开发 [M]. 付弘宇, 译. 北京: 人民邮电出版社, 2013.
- [5] Reto Meier. Android 4 高级编程 [M]. 余建伟, 译. 2 版. 北京: 清华大学出版社, 2013.
- [6] Juhani Lehtimki. 精彩绝伦的 Android UI 设计 [M]. 王东明, 译. 北京: 机械工业出版社, 2013.
- [7] Google. API Guides [EB/OL]. <http://developer.android.com/develop/index.html>, 2013.
- [8] 莫贝网. 手持移动设备界面设计专业网站 [EB/OL]. <http://www.guimobile.net/>, 2014.
- [9] Eric Freeman. Head First 设计模式 [M]. O' Reilly Taiwan 公司, 译. 北京: 中国电力出版社, 2007.

## 推荐阅读



### 设计搜索体验：搜索的艺术与科学

书号：978-7-111-44925-6 作者：Tony Russell-Rose Tyler Tate 定价：59.00元

这是一本从实践角度系统讲解搜索产品体验设计的著作，是搜索体验设计领域的标杆性著作，由资深搜索体验设计专家撰写，腾讯用户体验与设计部“CDC翻客”团队翻译，Amazon畅销书。整本书以如何构造以用户为中心的搜索解决方案为主题，从搜索产品设计和用户体验两个维度诠释了以用户为中心的搜索体验设计原则和技巧，为设计精准、流畅和优良的搜索体验提供完美经验和解决方案。

本书分三个部分探讨搜索的艺术和科学。第一部分侧重于理论，提出了一个信息检索的概念性框架，第1章强调用户的多样性；第2章阐释信息搜索模型、信息觅食、意义建构；第3章分析搜索情境的四个层面：信息检索、信息搜寻、工作任务和文化；第4章分析搜索与发现的模式。第二部分将理论付诸实践，将第一部分介绍的原理应用到搜索用户界面设计，第5章探索制定搜索请求的过程，研究如何将搜索者从模棱两可转变为明确、具体的搜索；第6章探究人们搜索时搜索结果的形式反应；第7章从设计基础到更高级的话题详细介绍了分面搜索；第8章研究移动信息搜索背后的动机，评论了移动搜索的设计原则，调查了移动设计的特定解决方案；第9章探究社会搜索的领域。第三部分（第10章）关注未来信息时代关于搜索体验的一些想法。



### 设计原本——计算机科学巨匠Frederick P. Brooks的反思（经典珍藏）

作者：Frederick P. Brooks ISBN：978-7-111-41626-5 定价：79.00元

图灵奖得主、软件工程之父《人月神话》作者Brooks 经典著作，揭秘软件设计本质！

程序员、项目经理和架构师终极修炼必读！

如果说《人月神话》结束了软件工业的神话时代，粉碎了“银弹”的幻想，从此人类进入了理性统治一切的工程时代，那么《设计原本》则再次唤醒了人类心中沉睡多年的激情，引导整个业界突破理性主义的无形牢笼，鼓励以充满大胆创新为本的设计作为软件工程核心动力的全新思维。可以说，不读《人月神话》，则会在幻想中迷失；而不读《设计原本》，则必将在复杂低效的流程中落伍！《设计原本》开启了软件工程全新的“后理性时代”，完成了从破到立的圆满循环，具有划时代的重大里程碑意义，是每位从事软件行业的程序员、项目经理和架构师都应该反复研读的经典著作。

全书以设计理念为核心，从对设计模型的探讨入手，讨论了有关设计的若干重大问题：设计过程的建立、设计协作的规划、设计范本的固化、设计演化的管控，以及设计师的发现和培养。书中各章条分缕析、层层推进，读来却丝毫没有书匠气，因为作者不仅运用了大量的图表和案例，并且灵活地运用苏格拉底提问式教学法，即提出问题，摆出事实，让读者自己去思考和取舍，而非教条式地给出现成答案。同时，作者的观点也十分鲜明：反对理性模型的僵化条框，拥抱基于实践的随机应变；质疑一切从头做起而不问前人工作的愚昧自大，主张从前人的失败中把握其选择的脉络，站到巨人的肩膀上而取得设计改进。Brooks每提出一个论点，就举出至少一个例子，将理论扎实地建立在实践的基础之上。他的素材丰富多彩：从CPU体系结构到厨房改造装修，从任务控制、指令尺寸到管理委员会人数，万事万物无不可视作设计案例。本书浓缩了设计百科，又把握了设计脉搏，真无愧“设计原本”之名！