



Android



- 书中实例的源代码文件
- 赠送网络、多媒体、物联网、智能家居等热门领域开发实例源代码

智能穿戴设备开发

从入门到精通

张明星 孙娇 编著

编排科学 通过基本理论、实例分析、综合实战等内容，
引领读者在实践中掌握所学知识

内容全面 全面、深入讲解传感器开发、蓝牙技术、
数据传输等核心技术

实用性强 通过实例详细讲解实际开发项目经验和技巧，
内容更贴近实战



10:42

"ok glass"

ok glass,google...
Ask a question



Web results
for
awesome

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE





Android

智能穿戴设备开发

从入门到精通

张明星 孙娇 编著



10:42

"ok glass"

ok glass,google...
Ask a question



Web results
for
awesome



中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

内 容 简 介

本书循序渐进地讲解了在 Android 系统中开发穿戴设备的各种必备知识及其应用。书中几乎涵盖了 Android 穿戴设备应用开发方面的所有重点内容。全书共分 14 章,依次讲解了 Android 开发技术基础,Android 技术核心框架分析,HTTP 数据通信,使用 Socket 实现数据通信,下载远程数据,上传数据,传感器技术,人工智能技术,语音识别和手势识别,蓝牙技术基础,Android 蓝牙模块详解,蓝牙 4.0 BLE 详解,以及智能心率计和计步器的设计全过程。

本书适用于 Android 初学者、Android 应用开发、Android 穿戴设备开发、Android 底层开发人员和 Android 源码分析人员学习,也可作为培训学校和大中专院校相关专业的教学用书。

图书在版编目(CIP)数据

Android 智能穿戴设备开发从入门到精通 / 张明星,
孙娇编著. — 北京:中国铁道出版社,2014.11
ISBN 978-7-113-19212-9

I. ①A… II. ①张… ②孙… III. ①移动终端—应用
程序—程序设计 IV. ①TN929.53

中国版本图书馆 CIP 数据核字(2014)第 207814 号

书 名: Android 智能穿戴设备开发从入门到精通
作 者: 张明星 孙 娇 编著

责任编辑: 于先军	读者热线电话: 010-63560056
特邀编辑: 王惠凤	封面设计: 多宝格
责任印制: 赵星辰	

出版发行: 中国铁道出版社(北京市西城区右安门西街 8 号)	邮政编码: 100054)
印 刷: 北京鑫正大印刷有限公司	
版 次: 2014 年 11 月第 1 版	2014 年 11 月第 1 次印刷
开 本: 787mm×1092mm 1/16	印张: 29.25 字数: 682 千
书 号: ISBN 978-7-113-19212-9	
定 价: 59.80 元(附赠光盘)	

版权所有 侵权必究

凡购买铁道版图书,如有印制质量问题,请与本社读者服务部联系调换。电话:(010) 51873174

打击盗版举报电话:(010) 51873659

2007年11月5日,谷歌公司发布了Android系统的第一个版本。Android是一款基于Linux平台的开源手机操作系统的名称,由整个操作系统、中间件、用户界面和应用软件组成。Android系统一经推出便受到了通信业巨头和手机制造商的青睐,并且被全世界的开发者所接受。根据国际数据公司(IDC)公布的统计数据,截至2014年1月31日,手机市场上Android占有率从2013年的68.8%上升到78.9%。而iOS则从2013年的19.4%下降到15.5%,Windows Phone系统从原来的2.7%小幅上升到3.6%。从数据上看,Android平台占据了手机市场的主导地位,继续称当老大的角色。78.9%,这是一个具有明显优势的比重,足以令后面的追赶着汗颜。我们有理由相信,至少在未来一段时间内,Android依旧能够牢牢地占据着智能手机操作系统第一的这个宝座。

穿戴设备的发展历程

自从谷歌推出Google眼镜产品之后,可穿戴计算设备便成为当今科技界的火热话题之一。在CES 2013和CES 2014(国际电子展),也有不少公司推出了眼镜、腕带等各种可穿戴计算设备,从此可穿戴设备开始流行起来。

为了更好地使人们了解可穿戴设备,FierceMobileIT特意整理了可穿戴设备的发展简史,帮助人们了解这类产品的发展状况。从中不难看出,如果能够广泛普及,像谷歌眼镜这样的设备有可能在将来会改变人们的生活和工作方式。

可穿戴设备的发展最早可以追溯到1762年,当时,约翰·哈里森(John Harrison)发明了怀表。但我们还是决定将这一技术的起始时间定在1975年,也就是Hamilton Watch推出Pulsar计算器手表的那一年。那款产品一时间成为男性时尚的代名词,甚至连时任美国总统的杰拉尔德·福特(Gerald Ford)也想要一块这样的手表。

可穿戴设备简史如下。

- 1975年:Hamilton Watch推出Pulsar计算器手表。
- 1977年:CC Collins为盲人开发了一款可穿戴设备,使用头戴式摄像头将图像转换成背心上的触觉网格。
- 1979年:索尼推出Walkman卡带随身听。

.....

- 2011年:Jawbone推出UP健身腕带,可以追踪睡眠、运动、饮食状况,并与智能手机应用关联,零售价为130美元。
- 2012年:索尼推出SmartWatch,使用蓝牙与Android手机相连,零售价为150美元。Pebble发布Pebble Watch,提供健身及健康追踪、上网和语音导航等功能。可以使用蓝牙技术与智能手机应用相连,零售价为250美元。
- 2013年:谷歌向部分用户推出测试版谷歌眼镜。谷歌眼镜是一款固定在眼镜上的光学头戴显示器,可以进行语音控制,而且能够利用Wi-Fi网络上网。

全球第一大 Android 智能手机制造商三星发布 Galaxy Gear 智能手表，可以使用蓝牙与 Android 智能手机相连。日本汽车制造商日产发布 Nismo 智能手表，可以为驾驶员提供平均时速、油耗和驾驶员心率等信息，零售价为 120 美元。

- 2014 年：有关苹果 iWatch，据称，这可能是一款能够通过 Wi-Fi 上网的智能手表。

本书的内容

本书详细讲解了 Android 开发技术基础，Android 技术核心框架分析，HTTP 数据通信，使用 Socket 实现数据通信，下载远程数据，上传数据，传感器技术，人工智能技术，语音识别和手势识别，蓝牙系统应用，智能心率计设计，计步器设计等知识。内容几乎涵盖了 Android 穿戴设备应用开发方面的所有主要技术，并且全书内容言简意赅，讲解方法通俗易懂，不但适合应用高手的学习，也特别适用于初学者学习。

本书的版本

Android 系统自 2008 年 9 月发布第一个版本 1.1 以来，截至 2013 年 11 月发布最新版本 4.4，一共存在十多个版本。由此可见，Android 系统升级频率较快，一年之中最少有两个新版本诞生。如果过于追求新版本，会造成力不从心的结果；因此在此建议广大读者：“不必追求最新的版本，我们只需关注最流行的版本即可。”据官方统计，截至 2013 年 11 月 25 日，占据前三位的系统版本分别是 Android 4.3、Android 4.2 和 Android 4.1，其实这三个版本的区别并不大，只是在某些领域的细节上进行了更新。因此，本书以 Android 4.3 为基础，详细讲解了 Android 系统安全方面的基本知识。

本书特色

本书内容十分丰富，讲解细致、全面。我们的目标是通过一本图书，提供多本图书的价值，读者可以根据自己的需要进行有选择的阅读。在内容的编写上，本书具有以下特色：

（1）结构合理

从用户的实际需要出发，科学安排知识结构，内容由浅入深，叙述清楚。全书详细地讲解了和 Android 穿戴设备应用开发有关的知识，内容循序渐进，由浅入深。

（2）遵循“理论介绍—演示实例—综合演练”这一主线

为了使广大读者彻底弄清楚 Android 穿戴设备应用开发的每一个知识点，在讲解时依次剖析了基本理论、演示实例分析、综合实战演练等内容。遵循了从理论到实践，实现了实践教学这一目标。

（3）易学易懂

本书内容条理清晰、语言简洁，可以帮助读者快速掌握每个知识点，使读者既可以按照本书编排的章节顺序进行学习，也可以根据自己的需求对某一章节进行针对性的学习，并且和传统的计算机书籍相比，阅读本书会为你带来更大的乐趣。

（4）实用性强

本书彻底摒弃枯燥的理论和简单的操作，注重实用性和可操作性，详细讲解了 Android 穿戴设备应用开发各个知识点的基本知识。

（5）内容全面

本书可以称为“内容最全面的一本 Android 穿戴设备应用开发书”，无论是开发环境搭建，还是传感器开发、数据传输、蓝牙技术等，在本书中读者都能找到解决问题的答案。

读者对象

- 初学 Android 编程的自学者
- Android 应用开发人员
- Android 源码爱好者
- Android 穿戴设备开发人员
- 毕业设计的学生
- Android 编程爱好者
- 相关培训机构的老师和学员
- 从事 Android 开发的程序员

在此，特别感谢我的家人，他们在编者写作时给予了巨大支持。由于编者水平有限，再加上 Android 系统更新较快，书中如有纰漏和错误之处，恳请读者批评指正。

编 者

2014 年 9 月

	Chapter 1	Android 开发技术基础	1
1.1	智能手机系统介绍		1
1.1.1	何谓智能手机		1
1.1.2	主流智能手机系统介绍		2
1.2	Android 的巨大优势		4
1.2.1	优点一——系出名门		4
1.2.2	优点二——强大的开发团队		4
1.2.3	优点三——诱人的奖励机制		5
1.2.4	优点四——开源		5
1.3	搭建 Android 应用开发环境		6
1.3.1	安装 Android SDK 的系统要求		6
1.3.2	安装 JDK		7
1.3.3	获取并安装 Eclipse 和 Android SDK		10
1.3.4	安装 ADT		14
1.3.5	设定 Android SDK Home		16
1.3.6	验证开发环境		17
1.3.7	创建 Android 虚拟设备 (AVD)		18
1.3.8	启动 AVD 模拟器		19
1.4	穿戴设备的前世今生		22
1.4.1	发展历程		22
1.4.2	现状介绍		22
1.4.3	发展前景分析		25
1.4.4	Android 的支持		27
	Chapter 2	Android 技术核心框架分析	28
2.1	分析 Android 的系统架构		28
2.1.1	Android 体系结构介绍		28
2.1.2	Android 应用工程文件组成		31
2.2	简述五大组件		34
2.2.1	用 Activity 来表现界面		34
2.2.2	用 Intent 和 Intent Filters 实现切换		34
2.2.3	Service 为用户服务		34
2.2.4	用 BroadcastReceiver 发送广播		35

2.2.5	用 ContentProvider 存储数据	35
2.3	进程和线程	36
2.3.1	先看进程	36
2.3.2	再看线程	36
2.3.3	应用程序的生命周期	36
2.4	分析 Android 源码结构	39
2.5	Android 和 Linux 的关系	40
2.5.1	Android 继承于 Linux	40
2.5.2	Android 和 Linux 内核的区别	40
2.6	第一段 Android 程序	42

Chapter 3 HTTP 数据通信 48

3.1	HTTP 基础	48
3.1.1	HTTP 概述	48
3.1.2	HTTP 协议的功能	48
3.1.3	Android 中的 HTTP	49
3.2	使用 Apache 接口	50
3.2.1	Apache 接口基础	50
3.2.2	Apache 应用要点	51
3.2.3	Apache 应用要点	57
3.3	使用标准的 Java 接口	65
3.3.1	IP 地址	65
3.3.2	URL 地址	66
3.3.3	套接字 Socket 类	67
3.3.4	URLConnection 类	67
3.3.5	在 Android 中使用 java.net	68
3.4	使用 Android 网络接口	71
3.5	实战演练	71
3.5.1	实战演练——在手机屏幕中传递 HTTP 参数	71
3.5.2	实战演练——在 Android 手机中通过 Apache HTTP 访问 HTTP 资源	76

Chapter 4 使用 Socket 实现数据通信 79

4.1	Socket 编程初步	79
4.1.1	TCP/IP 协议基础	79
4.1.2	UDP 协议	80
4.1.3	基于 Socket 的 Java 网络编程	81
4.2	TCP 编程详解	82
4.2.1	使用 ServletSocket	82

4.2.2	使用 Socket	83
4.2.3	TCP 中的多线程	85
4.2.4	实现非阻塞 Socket 通信	88
4.3	UDP 编程	94
4.3.1	使用 DatagramSocket	94
4.3.2	使用 MulticastSocket	99
4.4	实战演练——在 Android 中使用 Socket 实现数据传输	103

Chapter 5 下载远程数据 106

5.1	下载网络中的图片数据	106
5.2	下载网络中的 JSON 数据	108
5.2.1	JSON 基础	109
5.2.2	实战演练——远程下载服务器中的 JSON 数据	109
5.3	下载某个网页的源码	115
5.4	远程获取多媒体文件	117
5.4.1	实战演练——下载并播放网络中的 MP3	117
5.4.2	实战演练——下载在线铃声	124
5.5	多线程下载	130
5.5.1	多线程下载文件的过程	131
5.5.2	实战演练——在 Android 系统中实现多线程下载	131
5.6	远程下载并安装 APK 文件	148
5.6.1	APK 基础	149
5.6.2	实战演练——在 Android 系统中下载并安装 APK 文件	152

Chapter 6 上传数据 158

6.1	Android 上传数据技术	158
6.1.1	使用 HTTP 协议上传数据	158
6.1.2	使用 TCP 协议上传数据	159
6.2	实战演练——上传文件到远程服务器	162
6.3	使用 GET 方式上传数据	165
6.4	使用 POST 方式上传数据	170
6.5	使用 HTTP 协议实现上传	175
6.5.1	一段演示代码	175
6.5.2	实战演练——HTTP 协议实现文件上传	181

Chapter 7 传感器技术 188

7.1	Android 传感器系统概述	188
7.2	使用 SensorSimulator	190

7.3	使用传感器	193
7.3.1	光线传感器	194
7.3.2	磁场传感器	194
7.3.3	加速度传感器	196
7.3.4	姿态传感器	199
7.3.5	温度传感器	201

Chapter 8 人工智能技术 204

8.1	人工智能基础	204
8.1.1	人工智能概述	204
8.1.2	两种实现人工智能的方法	205
8.2	图搜索在人工智能中的应用	205
8.2.1	深度优先搜索 (DFS)	205
8.2.2	广度优先搜索 (BFS)	208
8.2.3	戴克斯特拉算法 (Dijkstra)	209
8.2.4	A-Star 算法	211
8.3	实战演练——各种 AI 图搜索算法在 Android 游戏中的用法	219
8.3.1	搭建路径搜索框架	219
8.3.2	实现深度优先算法	227
8.3.3	实现广度优先算法	229
8.3.4	实现 Dijkstra 算法	231
8.3.5	实现广度优先 A*算法	233
8.3.6	实现 Dijkstra A*算法	235

Chapter 9 语音识别和手势识别 238

9.1	语音识别技术	238
9.1.1	Text-To-Speech 技术	238
9.1.2	谷歌的 Voice Recognition 技术	242
9.2	手势识别	245
9.2.1	类 GestureDetector 基础	245
9.2.2	使用类 GestureDetector	246
9.2.3	通过点击的方式移动图片	249

Chapter 10 蓝牙技术基础 253

10.1	蓝牙概述	253
10.1.1	蓝牙技术的发展历程	253
10.1.2	蓝牙的特点	254
10.2	低功耗蓝牙基础	254

10.2.1	低功耗蓝牙的架构	254
10.2.2	低功耗蓝牙分类	255
10.2.3	集成方式	256
10.2.4	低功耗蓝牙的特点	256
10.2.5	BLE 和传统蓝牙 BR/EDR 技术的对比	257
10.3	蓝牙规范	257
10.3.1	Bluetooth 系统中的常用规范	257
10.3.2	蓝牙协议体系结构	258
10.3.3	低功耗 (BLE) 蓝牙协议	260
10.3.4	现有的基于 GATT 的协议/服务	260
10.3.5	双模协议栈	261
10.3.6	单模协议栈	262
10.4	低功耗蓝牙协议栈详解	262
10.4.1	低功耗蓝牙协议栈基础	262
10.4.2	蓝牙协议体系中的协议	263
10.5	TI 公司的低功耗蓝牙	265
10.5.1	获取 TI 公司的低功耗蓝牙协议栈	265
10.5.2	分析 TI 公司的低功耗蓝牙协议栈	267

Chapter 11 Android 蓝牙模块详解

11.1	Android 系统中的蓝牙模块	274
11.2	分析蓝牙模块的源码	276
11.2.1	初始化蓝牙芯片	276
11.2.2	蓝牙服务	277
11.2.3	管理蓝牙电源	278
11.3	和蓝牙相关的类	278
11.3.1	BluetoothSocket 类	278
11.3.2	BluetoothServerSocket 类	279
11.3.3	BluetoothAdapter 类	280
11.3.4	BluetoothClass.Service 类	287
11.3.5	BluetoothClass.Device 类	288
11.4	在 Android 平台开发蓝牙应用程序	288
11.4.1	开发 Android 蓝牙应用程序的基本步骤	288
11.4.2	开发一个控制玩具车的蓝牙遥控器	293
11.5	在穿戴设备中开发一个蓝牙控制器	302
11.5.1	界面布局	302
11.5.2	响应单击按钮	303
11.5.3	和指定的服务器建立连接	305
11.5.4	搜索附近的蓝牙设备	306

11.5.5	建立和 OBEX 服务器的数据传输	308
11.5.6	实现蓝牙服务器端的数据处理	312

Chapter 12 蓝牙 4.0 BLE 详解 315

12.1	短距离无线通信技术概览	315
12.1.1	ZigBee——低功耗、自组网	315
12.1.2	WiFi——大带宽支持家庭互联	316
12.1.3	蓝牙——4.0 进入低功耗时代	316
12.1.4	NFC——必将逐渐远离历史舞台	316
12.2	蓝牙 4.0 BLE 基础	317
12.2.1	蓝牙 4.0 的最杰出表现是低功耗	317
12.2.2	蓝牙 4.0 的优势	318
12.2.3	Bluetooth4.0 BLE 推动了可穿戴设备的兴起	318
12.2.4	BLE 推动了 Android 可穿戴设备的发展	319
12.3	低功耗蓝牙协议栈详解	320
12.3.1	低功耗蓝牙协议栈基础	320
12.3.2	低功耗蓝牙 API 详解	321

Chapter 13 项目实战——开发智能心率计 404

13.1	什么是心率	404
13.2	什么是心率表	405
13.3	开发一个 Android 版测试心率系统	405
13.3.1	系统主界面	406
13.3.2	绘制心率表	411

Chapter 14 项目实战——开发计步器 417

14.1	系统功能模块介绍	417
14.2	系统主界面	418
14.2.1	布局文件	418
14.2.2	系统主 Activity	422
14.3	系统设置模块	430
14.3.1	系统设置 Activity	431
14.3.2	获取各个设置值	434
14.3.3	系统服务设置	437
14.3.4	获取并显示热量	444
14.3.5	显示行走距离	446
14.3.6	获取并显示步伐速率	448
14.3.7	获取并显示行走速率	451



Android 开发技术基础

Android 是一种智能手机操作系统，是建立在 Linux 开源系统基础之上的，能够迅速建立手机软件的解决方案。虽然 Android 外形比较简单，但是其功能十分强大，已经成为当前软件行业的一股新兴力量。从 2011 年开始到现在，Android 一直占据全球智能手机市场占有率第一的宝座。在本章的内容中，将简单介绍 Android 的发展历程和背景，并介绍搭建 Android 应用开发环境的基本知识，为读者学习本书后面的知识打下基础。

1.1 智能手机系统介绍



在 Android 系统诞生之前，智能手机这个新鲜事物大大丰富了人们的生活，得到了广大手机用户的青睐。各大手机厂商在利益的驱动下，纷纷建立了各种智能手机操作系统，并且大肆招兵买马来抢夺市场份额。Android 系统就是在这个风起云涌的历史背景下诞生的。

1.1.1 何谓智能手机

智能手机是指手机具有像个人电脑那样强大的功能，拥有独立的操作系统，用户可以自行安装应用软件、游戏等第三方服务商提供的程序，并且可以通过移动通信网络接入到无线网络中。在 Android 系统诞生之前已经有很多优秀的智能手机系统产品，如家喻户晓的 Symbian 系列和微软的 Windows Mobile 系列等。

对于初学者来说，可能还不知道怎样来区分什么是智能手机。某大型专业机构曾经为智能手机的问题做过一项市场调查，经过大众讨论并投票之后，总结出了智能手机所必须具备的功能标准，当时投票后得票率最高的前 5 个选项如下。

- (1) 操作系统必须支持新应用的安装。
- (2) 高速度处理芯片。
- (3) 支持播放式的手机电视。
- (4) 大存储芯片和存储扩展能力。
- (5) 支持 GPS 导航。

根据大众投票结果，手机联盟制定了一个标准，并以这个标准为基础，总结出了如下智能手机的主要特点。

PDF电子书说明：

本人可以提供各种PDF电子书资料，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 **QQ: 461573687**, 或者 **QQ: 2404062482**。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ。因PDF电子书都有版权，请不要随意传播，最近pdf也越来越难做了，希望大家尊重下个人劳动，谢谢！

(1) 具备普通手机的全部功能,例如,可以进行正常的通话和发短信等手机应用。

(2) 是一个开放性的操作系统,在系统平台上可以安装更多的应用程序,从而实现功能的无限扩充。

(3) 具备上网功能。

(4) 具备 PDA 的功能,实现个人信息管理、日程记事、任务安排、多媒体应用、浏览网页。

(5) 可以根据个人需要扩展机器的功能。

(6) 扩展性能强,并且可以支持很多第三方软件。

1.1.2 主流智能手机系统介绍

在当今市面中最主流的智能手机系统当属微软、塞班、PDA、黑莓、苹果和本书的主角 Android。

1. 微软的 Windows Mobile

Windows Mobile 是微软公司的一款杰出产品。Windows Mobile 将熟悉的 Windows 桌面扩展到了个人设备中。使用 Windows Mobile 操作系统的设备主要有 PPC 手机、PDA、随身音乐播放器等。Windows Mobile 操作系统有三种,分别是 Windows Mobile Standard、Windows Mobile Professional 和 Windows Mobile Classic。当前的最新版本是 Windows Phone 7 和 Windows Phone 8。

2. Symbian (塞班系统)

塞班系统最初是由诺基亚、索尼爱立信、摩托罗拉、西门子等几家大型移动通信设备商共同出资组建的一个合资公司开发的,该公司专门研发手机操作系统,后来被诺基亚全额收购。Symbian 有着良好的界面,采用内核与界面分离技术,对硬件的要求比较低,支持 C++、VisualBasic 和 J2ME。目前根据人机界面的不同,Symbian 体系的 UI (User Interface 用户界面) 平台分为 Series60、Series80、Series90 和 UIQ 等。其中 Series60 主要是给数字键盘手机用, Series80 是为完整键盘所设计的, Series90 则是为触控笔方式而设计的。



背景说明

(1) 2010 年 9 月,诺基亚宣布将从 2011 年 4 月起从 Symbian 基金会 (Symbian Foundation) 手中收回 Symbian 操作系统控制权。由此看来,诺基亚在 2008 年全资收购塞班公司之后希望继续扩大塞班影响力的愿望并没有实现。

(2) 在苹果和 Android 的强大市场攻势下,诺基亚在 2011 年 2 月 11 日宣布与微软达成广泛战略合作关系,并将 Windows Phone 作为其主要的智能手机操作系统。这家芬兰手机巨头试图通过结盟扭转颓势。

(3) 2011 年 8 月 15 日,谷歌和摩托罗拉移动公司共同宣布,谷歌将以每股 40.00 美元现金收购摩托罗拉移动股份,总额约 125 亿美元,相比摩托罗拉移动股份的收盘价溢价了 63%,双方董事会都已全票通过该交易。谷歌 CEO 拉里·佩奇表示,摩托罗拉移动完全专注于 Android 系统,收购摩托罗拉移动之后,将增强整个 Android 生态系统。佩奇同时表示,Android 将继续开源,收购的一个目的是为了获得专利。

(4) 2013 年 9 月 3 日, 微软公司宣布将以 37.9 亿欧元的价格收购诺基亚的设备和部门, 同时还将以 16.5 亿欧元的价格收购诺基亚的相关技术专利, 本次交易总额达到 54.4 亿欧元, 其中有 3.2 万名员工将从诺基亚转入微软, 整笔交易预计将于来年第一季度完成。

3. Palm

Palm 是流行的个人数字助理 (PDA, 又称掌上电脑) 的传统名字。从广义上讲, Palm 是 PDA 的一种, 是 Palm 公司发明的。而从狭义上讲, Palm 是 Palm 公司生产的 PDA 产品, 区别于 SONY 公司的 Clie 和 Handspring 公司的 Visor/Treo 等其他运行 Palm 操作系统的 PDA 产品。其显著特点之一是写入装置输入数据的方法, 能够点击显示器上的图标选择输入的项目。2009 年 2 月 11 日, Palm 公司 CEO Ed Colligan 宣布以后将专注于 WebOS 和 Windows Mobile 的智能设备, 而将不会再有基于 Palm OS 的智能设备推出, 除了 Palm Centro 会在以后和其他运营商合作时继续推出。

4. 黑莓 (BlackBerry)

BlackBerry 是加拿大 RIM 公司推出的一种移动电子邮件系统终端, 其特色是支持推动式电子邮件、手提电话、文字短信、互联网传真、网页浏览及其他无线资讯服务, 其最大优势在于收发邮件。在苹果和 Android 的强大市场攻势下, BlackBerry 已经考虑整体出售。

5. iOS

iOS 作为苹果移动设备 iPhone 和 iPad 的操作系统, 在 App Store 的推动之下, 成为了世界上引领潮流的操作系统之一。原本这个系统名为 iPhone OS, 直到 2010 年 6 月 7 日 WWDC 大会上宣布改名为 iOS。iOS 的用户界面的概念基础是能够使用多点触控直接操作。控制方法包括滑动、轻触开关及按键。与系统交互包括滑动 (Swiping)、轻按 (Tapping)、挤压 (Pinching, 通常用于缩小) 及反向挤压 (Reverse Pinching or unpinching, 通常用于放大)。此外通过其自带的加速器, 可以令其旋转设备改变其 y 轴以令屏幕改变方向, 这样的设计令 iPhone 更便于使用。

6. Android

Android 是本书的主角, 是于 2007 年 11 月 5 日宣布的基于 Linux 平台的开源手机操作系统的名称, 该平台由操作系统、中间件、用户界面和应用软件组成, 号称是首个为移动终端打造的真正开放和完整的移动软件。

根据国际数据公司 (IDC) 公布的新数据, 在 2013 年第一季度, Android 和 iOS 系统的装机量占有所有智能手机出货量的 92.3%。在 2013 年头三个月, 安装 Android 系统的新智能手机数量跃升至 1.621 亿部, 大大超过去年同期的 9030 万部。这意味着, 在运往世界各地的所有新智能手机中, 谷歌的移动操作系统的市场占有率已经达到 75%, 比 2012 年第一季度的 59.1% 有显著提高。

截至本书截稿时, Android 的最新版本是 Android 4.4。

1.2 Android 的巨大优势

为什么 Android 能在这么多的智能系统中脱颖而出，成为市场占有率第一的手机系统呢？要想分析其原因，需要先了解它的巨大优势，分析究竟是哪些优点吸引了厂商和消费者的青睐。在本节的内容中，将对上述问题一一进行分析。

1.2.1 优点一——系出名门

Android 出身于 Linux 世家，是一款开源的手机操作系统。Android 功成名就之后，各大手机联盟纷纷加入，这个联盟由包括中国移动、摩托罗拉、高通、HTC 和 T-Mobile 在内的 30 多家技术和无线应用的领军企业组成。通过与运营商、设备制造商、开发商和其他有关各方结成深层次的合作伙伴关系，希望借助建立标准化、开放式的移动电话软件平台，在移动产业内形成一个开放式的生态系统。

1.2.2 优点二——强大的开发团队

Android 的研发队伍阵容强大，包括摩托罗拉、Google、HTC（宏达电子）、PHILIPS、T-Mobile、高通、魅族、三星、LG 及中国移动在内的 34 家企业，这都是在手机“江湖”中享誉盛名的“大佬”。这些研发队伍都将基于该平台开发手机的新型业务，应用之间的通用性和互联性将在最大程度上得到保证。并且还成立了手机开放联盟，联盟中的成员名单如下。

1. 手机制造商

台湾宏达国际电子（HTC）（Palm 等多款智能手机的代工厂），摩托罗拉（美国最大的手机制造商），韩国三星电子，韩国 LG 电子，中国移动（全球最大的移动运营商），日本 KDDI（2 900 万用户），日本 NTT DoCoMo（5 200 万用户），美国 Sprint Nextel（美国第三大移动运营商，5 400 万用户），意大利电信（Telecom Italia）（意大利主要的移动运营商，3 400 万用户），西班牙 Telefónica（在欧洲和拉美有 1.5 亿用户），T-Mobile（德意志电信旗下公司，在美国和欧洲有 1.1 亿用户）。

2. 半导体公司

Audience Corp（声音处理器公司），Broadcom Corp（无线半导体主要提供商），英特尔（Intel），Marvell Technology Group，SiRF（GPS 技术提供商），Synaptics（手机用户界面技术），德州仪器（Texas Instruments），高通（Qualcomm），惠普 HP（Hewlett-Packard Development Company,L.P.）。

3. 软件公司

Aplix，Ascender，eBay 的 Skype，Esmertec，Living Image，NMS Communications，Noser Engineering AG，Nuance Communications，PacketVideo，SkyPop，Sonix Network，TAT-The Astonishing Tribe，Wind River Systems。

1.2.3 优点三——诱人的奖励机制

俗话说：“人为财死、鸟为食亡。”谷歌为了提高程序员的开发积极性，不但为他们提供了一流的硬件和软件服务，而且还提出了振奋人心的奖励机制，例如，定期召开开发比赛，用创意和应用夺魁的程序员将会得到重奖。

1. 开发 Android 平台的应用

在 Android 平台上，程序员可以开发出各式各样的应用。Android 应用程序是通过 Java 语言开发的，只要具备 Java 开发基础，就能很快上手并掌握其内容。作为单独的 Android 开发，其编程门槛并不高，即使没有编程经验的门外汉，也可以在突击学习 Java 之后不影响学习 Android。另外，Android 完全支持 2D、3D 和数据库，并且和浏览器实现了集成。所以通过 Android 平台，程序员可以迅速、高效地开发出绚丽多彩的应用，如常见的工具、管理、互联网和游戏等。

2. 奖金丰厚的 Android 大赛

为了吸引更多的用户使用 Android 开发，相关部门已经成功举办了奖金为 1 000 万美元的开发者竞赛。鼓励开发人员创建出创意十足、十分有用的软件。这种大赛对于开发人员来说，不但能提高自己的开发水平，还能从高额的奖金中获得学习的动力。

3. 在 Android Market 上获取收益

为了能让 Android 平台吸引更多的关注，谷歌开发了自己的 Android 软件下载店 Android Market。Android Market 地址是 <http://www.Android.com/market/>，允许开发人员将应用程序在其上面发布，也允许 Android 用户随意下载获取自己喜欢的程序。作为开发者，需要申请开发者账号，申请后才能将自己的程序上传到 Android Market，并且可以对自己的软件进行定价。所以说，只要软件程序足够吸引人，程序员就可以获得很好的金钱回报，从而实现学习、赚钱两不误。

1.2.4 优点四——开源

开源意味着对开发人员和手机厂商来说，Android 是完全无偿免费使用的。因为源代码公开，所以吸引了全世界各地无数程序员。很多手机厂商都纷纷采用 Android 作为自己产品的系统，包括很多山寨厂商。因为免费，所以降低了成本，提高了利润。而对于开发人员来说，众多厂商的采用就意味着人才需求大，所以纷纷加入到 Android 开发大军中来。有一些干得还可以的程序员禁不住高薪的诱惑，都纷纷改行做 Android 开发。至于“混”得不尽如人意的程序员，就更加坚定了“改行做 Android 手机开发”的想法，目的是想寻找自己程序员生涯的转机。而遇到发展瓶颈的程序员，也决定做 Android 开发，因为这样可以学习一门新技术，使自己的未来更加有保障。

1.3 搭建 Android 应用开发环境

“工欲善其事，必先利其器”出自《论语》，意思是要想高效地完成一件事，需要有一个合适的工具。对于 Android 开发人员来说，开发工具同样至关重要。作为一项新兴技术，在对其进行开发前首先要为其搭建一个相应的开发环境。而在搭建开发环境前，需要了解安装开发工具所需要的硬件和软件配置条件。



注 意

Android 开发包括底层开发和应用开发，底层开发大多数是指和硬件相关的开发，并且是基于 Linux 环境的，如开发驱动程序。应用开发是指开发能在 Android 系统上运行的程序，如游戏、地图等程序。本书的重点是讲解多媒体应用开发，即使讲一些底层的知识，也是为上层的应用服务的。

因为开发 Android 应用程序最合适的系统是 Windows，所以本书只介绍在 Windows 下配置 Eclipse+ADT 的过程。

1.3.1 安装 Android SDK 的系统要求

在搭建之前，一定要先确定基于 Android 应用软件所需要的开发环境，具体如表 1-1 所示。

表 1-1 基于 Android 应用软件所需要的开发环境

项 目	版 本 要 求	说 明	备 注
操作系统	Windows XP 或 Vista Mac OS X 10.4.8+Linux Ubuntu Drapper	根据自己的电脑自行选择	选择自己最熟悉的操作系统
软件开发包	Android SDK	选择最新版本的 SDK	截至目前，最新手机版本是 2.3
IDE	Eclipse IDE+ADT	Eclipse3.3 (Europa), 3.4 (Ganymede)ADT(Android Development Tools)开发插件	选择“for Java Developer”
其他	JDK Apache Ant	Java SE Development Kit 5 或 6 Linux 和 Mac 上使用 Apache Ant 1.6.5+，Windows 上使用 1.7+版本	(单独的 JRE 不可取，必须要有 JDK)，不兼容 Gnu Java 编译器(gcj)

Android 工具由多个开发包组成，具体说明如下。

- JDK：可以到网址 <http://java.sun.com/javase/downloads/index.jsp> 下载。
- Eclipse (Europa)：可以到网址 <http://www.eclipse.org/downloads/> 下载 Eclipse IDE for Java Developers。
- Android SDK：可以到网址 <http://developer.android.com> 下载。
- 还有对应的开发插件。

1.3.2 安装 JDK

JDK (Java Development Kit) 是整个 Java 的核心, 包括 Java 运行环境、Java 工具和 Java 基础的类库。JDK 是学好 Java 的第一步, 是开发和运行 Java 环境的基础, 当用户要对 Java 程序进行编译时, 必须先获得对应操作系统的 JDK, 否则将无法编译 Java 程序。在安装 JDK 之前需要先获得 JDK, 获得 JDK 的操作流程如下所示。

(1) 登录 Oracle 官方网站, 网址为 <http://developers.sun.com/downloads/>, 如图 1-1 所示。

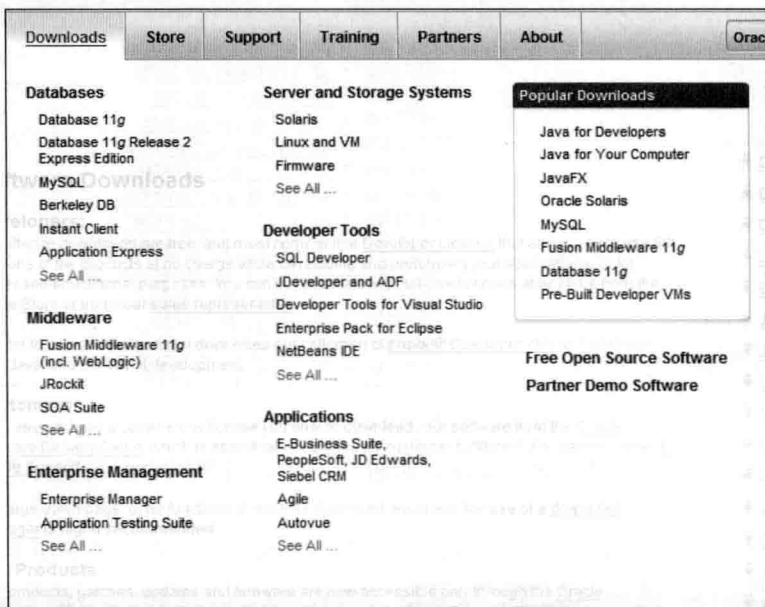


图 1-1 Oracle 官方下载页面

(2) 在图 1-1 中可以看到有很多版本, 在此选择当前最新的版本 Java 7, 下载页面如图 1-2 所示。

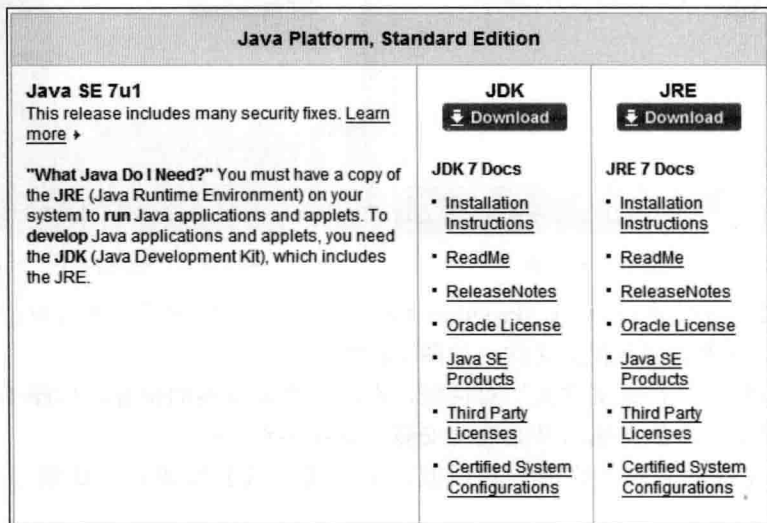


图 1-2 JDK 下载页面

(3) 在图 1-2 中单击 JDK 下方的 Download 按钮, 在弹出的新界面中选择将要下载的 JDK, 笔者在此选择的是 Windows X86 版本, 如图 1-3 所示。

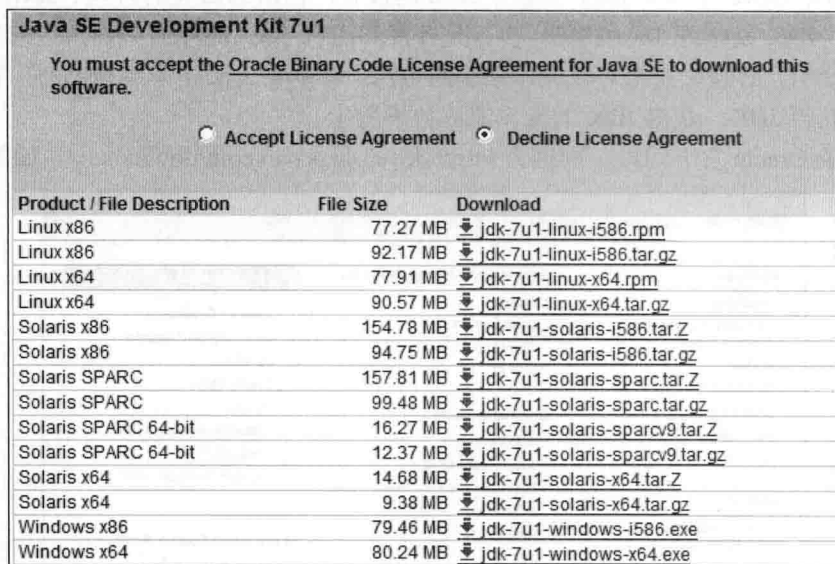


图 1-3 选择 Windows X86 版本

(4) 下载完成后双击下载的.exe 文件开始进行安装, 将弹出“安装向导”对话框, 在此单击【下一步】按钮, 如图 1-4 所示。

(6) 弹出“安装路径”对话框, 在此选择文件的安装路径, 如图 1-5 所示。

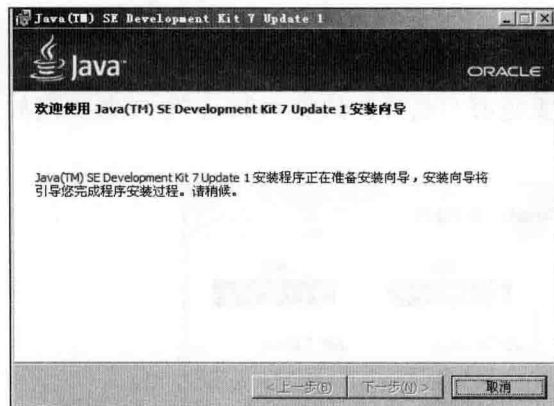


图 1-4 “安装向导”对话框

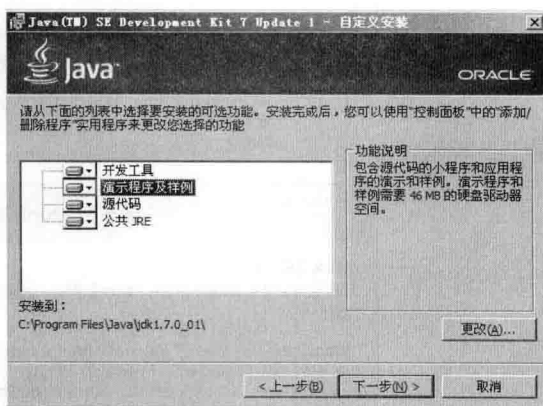


图 1-5 “安装路径”对话框

(7) 在此设置安装路径为“C:\Program Files\Java\jdk1.7.0_01”, 然后单击【下一步】按钮开始在安装路径解压缩下载的文件, 如图 1-6 所示。

(8) 完成后弹出“目标文件夹”对话框, 在此选择要安装的位置, 如图 1-7 所示。

(9) 单击【下一步】按钮后开始继续安装, 如图 1-8 所示。

(10) 安装完成后弹出“完成”对话框, 单击【完成】按钮后完成整个安装过程, 如图 1-9 所示。

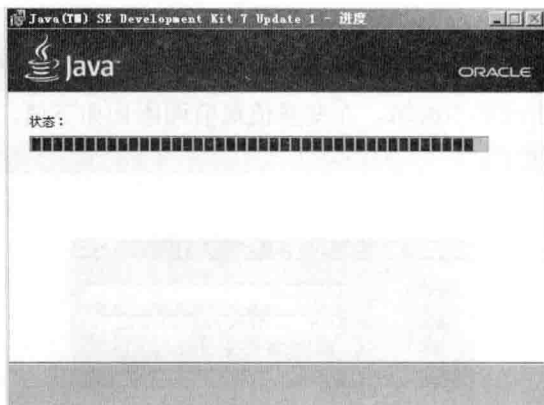


图 1-6 解压缩下载的文件

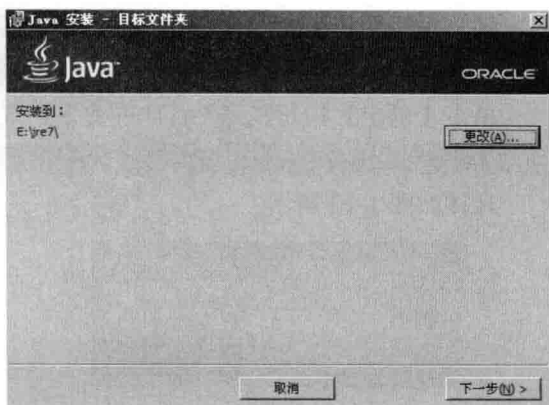


图 1-7 “目标文件夹”对话框

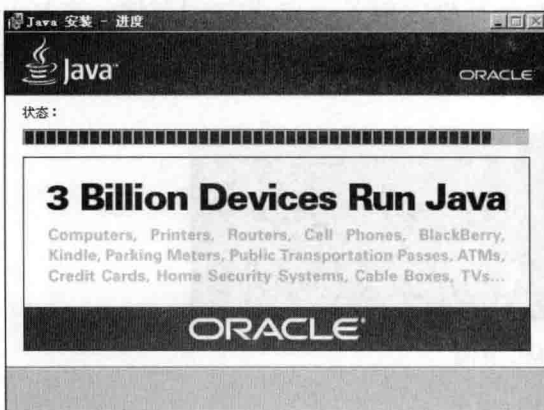


图 1-8 继续安装

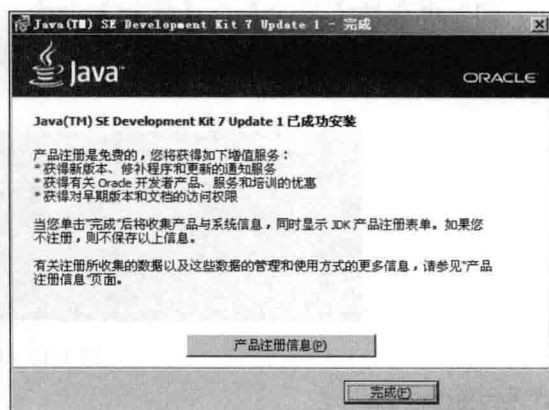


图 1-9 完成安装

完成安装后可以检测是否安装成功，检测方法是选择【开始】|【运行】命令，在运行框中输入“cmd”并按回车键，在打开的 CMD 窗口中输入 `java -version`，如果显示如图 1-10 所示的提示信息，则说明安装成功。

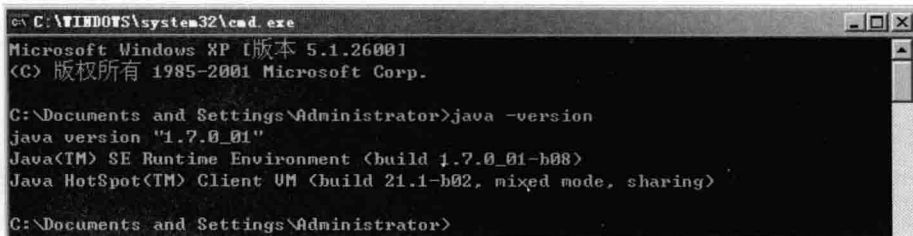


图 1-10 CMD 窗口

如果检测没有安装成功，需要将其目录的绝对路径添加到系统的 PATH 中。具体做法如下所示。

(1) 右击【我的电脑】，选择【属性】|【高级】命令，单击下面的“环境变量”，在下面的“系统变量”处选择“新建”，在变量名中输入 `JAVA_HOME`，在变量值中输入刚才的目录，比如设置为 `C:\Program Files\Java\jdk1.7.0_02`”，如图 1-11 所示。

(2) 再次新建一个变量名为 `classpath`，其变量值如下。

```
.;%JAVA_HOME%/lib/rt.jar;%JAVA_HOME%/lib/tools.jar
```

单击【确定】按钮找到 `PATH` 的变量，双击或单击编辑，在变量值最前面添加如下值。

```
%JAVA_HOME%/bin;
```

具体如图 1-12 所示。

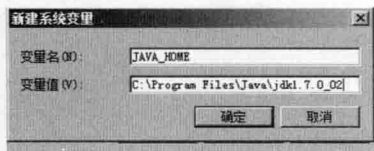


图 1-11 设置系统变量

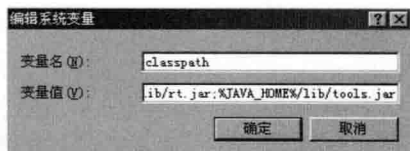


图 1-12 设置系统变量

(3) 再选择【开始】|【运行】命令，在运行框中输入 `cmd` 并按回车键，在打开的 CMD 窗口中输入 `java -version`，如果显示如图 1-13 所示的提示信息，则说明安装成功。

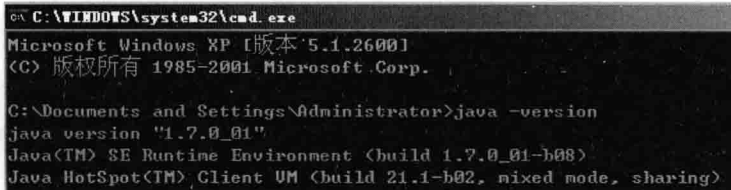


图 1-13 CMD 界面



注意

上述变量设置中，是按照笔者本人的安装路径设置的，笔者安装的 JDK 的路径是 `C:\Program Files\Java\jdk1.7.0_02`。

1.3.3 获取并安装 Eclipse 和 Android SDK

在安装好 JDK 后，接下来需要安装 Eclipse 和 Android SDK。Eclipse 是进行 Android 应用开发的一个集成工具，而 Android SDK 是开发 Android 应用程序所必须具备的框架。在 Android 官方公布的最新版本中，已经将 Eclipse 和 Android SDK 这两个工具进行了集成，一次下载即可同时获得这两个工具。获取并安装 Eclipse 和 Android SDK 的具体步骤如下。

(1) 登录 Android 的官方网站 <http://developer.android.com/index.html>，如图 1-14 所示。

(2) 单击图 1-14 左上方 Developers 右边的 ▾ 符号，在弹出的界面中单击 Get the SDK 链接，如图 1-15 所示。

(3) 在弹出的新页面中单击 Download the SDK 按钮，如图 1-16 所示。

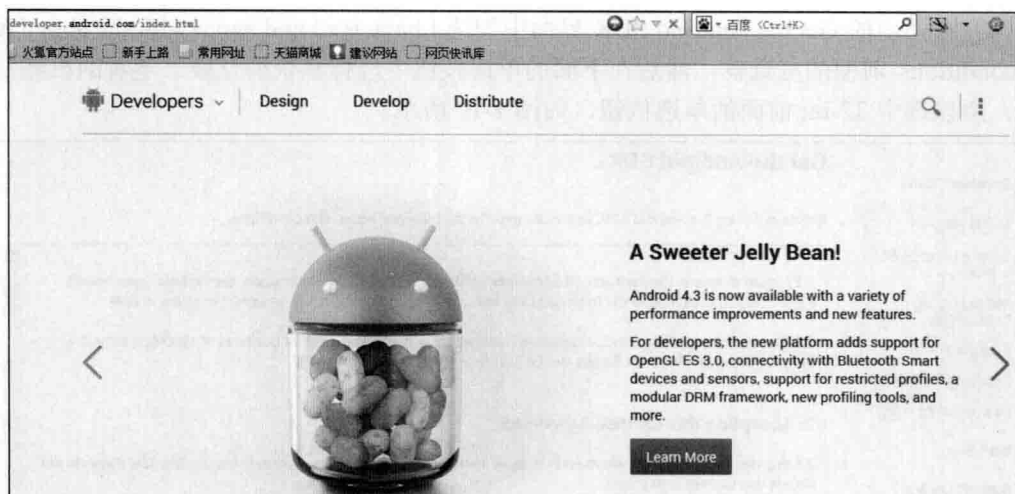


图 1-14 Android 的官方网站

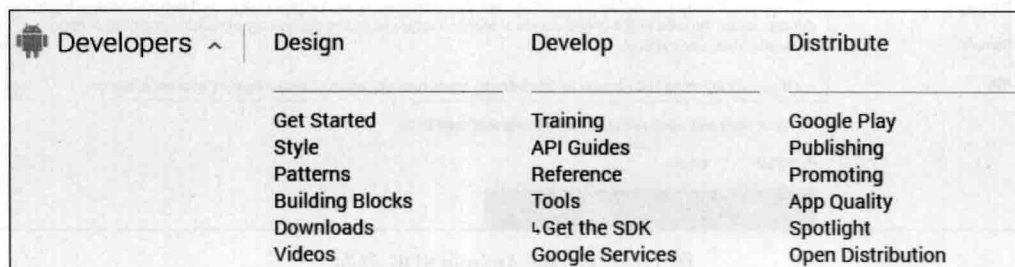


图 1-15 单击 Get the SDK 链接



图 1-16 单击 Download the SDK 按钮

(4) 在弹出的 Get the Android SDK 界面中勾选 I have read and agree with the above terms and conditions 前面的复选框, 然后在下面的单选按钮中选择系统的位数。笔者的机器是 32 位的, 所以选中 32-bit 前面的单选按钮, 如图 1-17 所示。

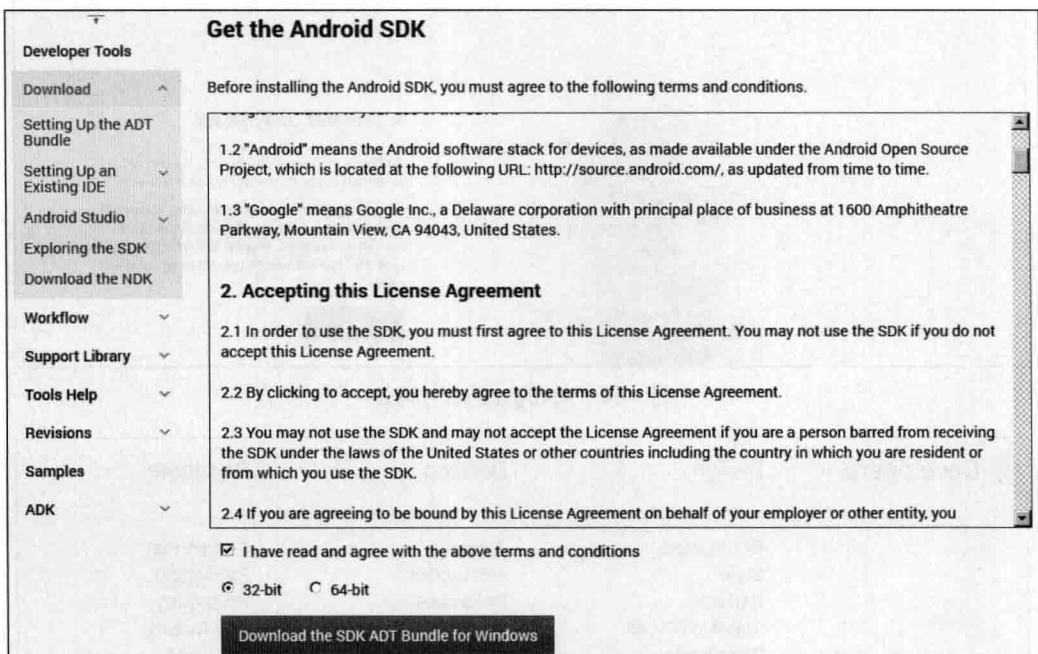


图 1-17 Get the Android SDK 界面

(5) 单击图 1-17 中的 **Download the SDK ADT Bundle for Windows** 按钮后开始下载工作, 下载的目标文件是一个压缩包, 如图 1-18 所示。

(6) 将下载得到的压缩包进行解压, 解压后的目录结构如图 1-19 所示。



图 1-18 开始下载目标文件压缩包

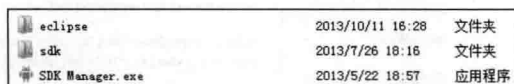


图 1-19 解压后的目录结构

由此可见, Android 官方已经将 Eclipse 和 Android SDK 实现了集成。双击 eclipse 目录中的 eclipse.exe 可以打开 Eclipse, 界面效果如图 1-20 所示。

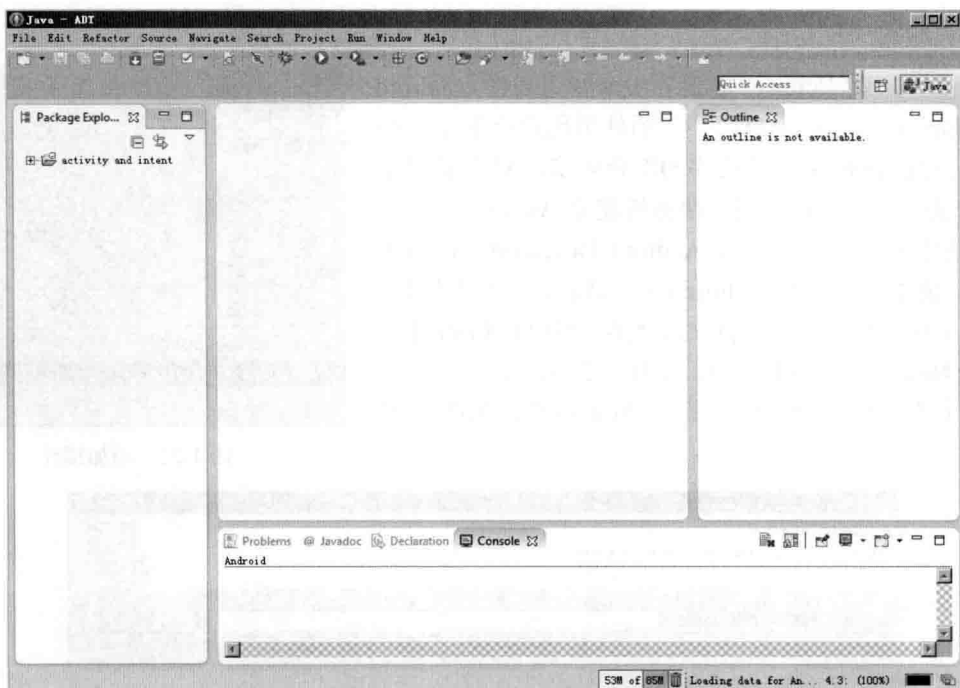



图 1-20 打开 Eclipse 后的界面效果

(7) 打开 Android SDK 的方法有两种，第一种是双击下载目录中的 SDK Manager.exe 文件，第二种是在 Eclipse 工具栏中单击  图标。打开后的效果如图 1-21 所示。

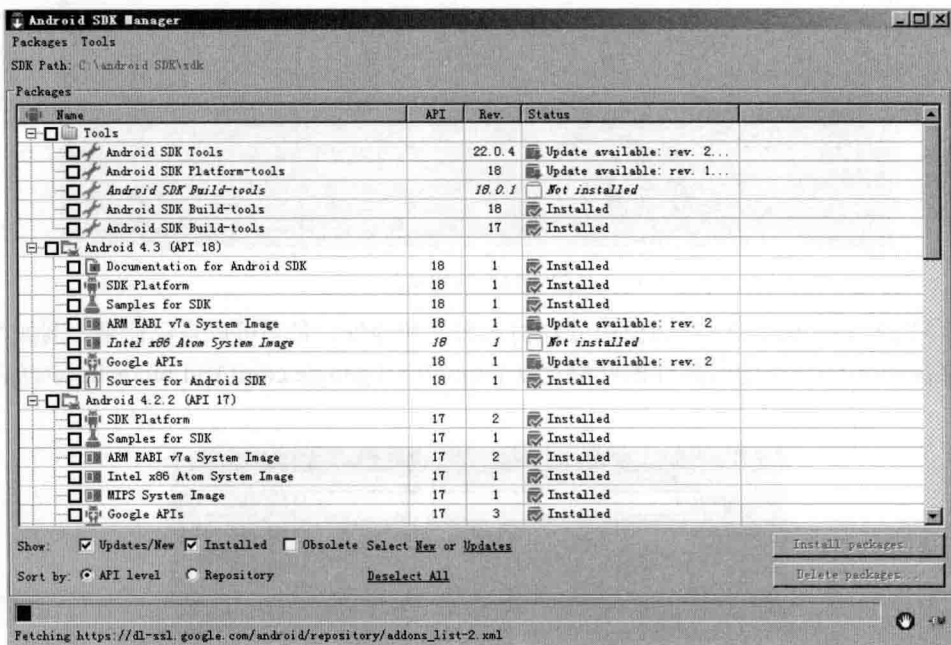


图 1-21 打开 Android SDK 后的界面效果

1.3.4 安装 ADT

Android 为 Eclipse 定制了一个专用插件 Android Development Tools (ADT)，此插件为用户提供了一个强大的开发 Android 应用程序的综合环境。ADT 扩展了 Eclipse 的功能，可以让用户快速地建立 Android 项目，创建应用程序界面。要安装 Android Development Tools plug-in，需要首先打开 Eclipse IDE。然后进行如下操作：

(1) 打开 Eclipse 后，依次选择菜单栏中的【Help】|【Install New Software】选项，如图 1-22 所示。

(2) 在弹出的对话框中单击 Add 按钮，如图 1-23 所示。

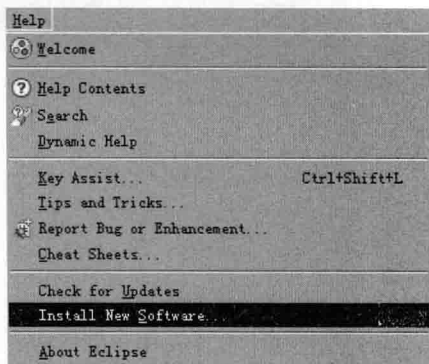


图 1-22 添加插件

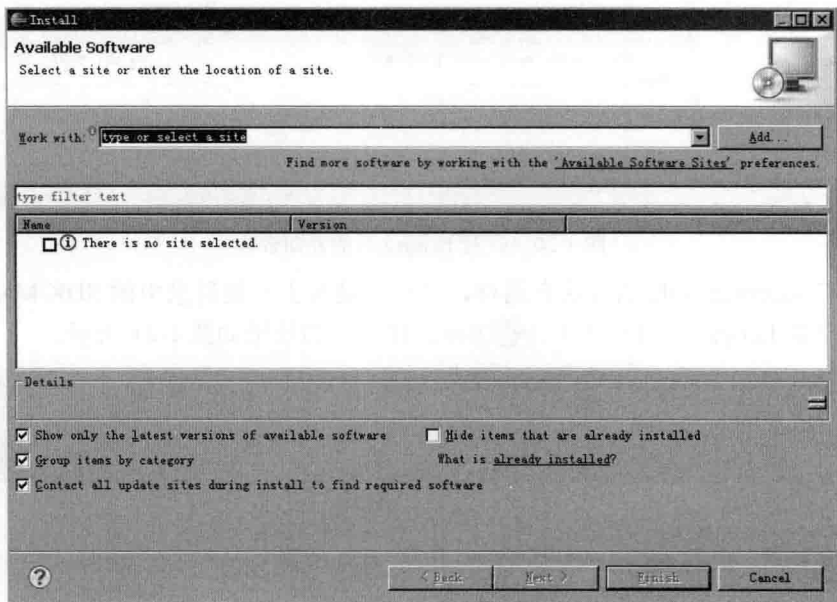


图 1-23 添加插件

(3) 在弹出的 Add Site 对话框中分别输入名字和地址，名字可以自己命名，例如“123”，但是在 Location 中必须输入插件的网络地址 <http://dl-ssl.google.com/Android/eclipse/>，如图 1-24 所示。



图 1-24 设置地址

(4) 单击 OK 按钮，此时在 Install 界面将会显示系统中可用的插件，如图 1-25 所示。

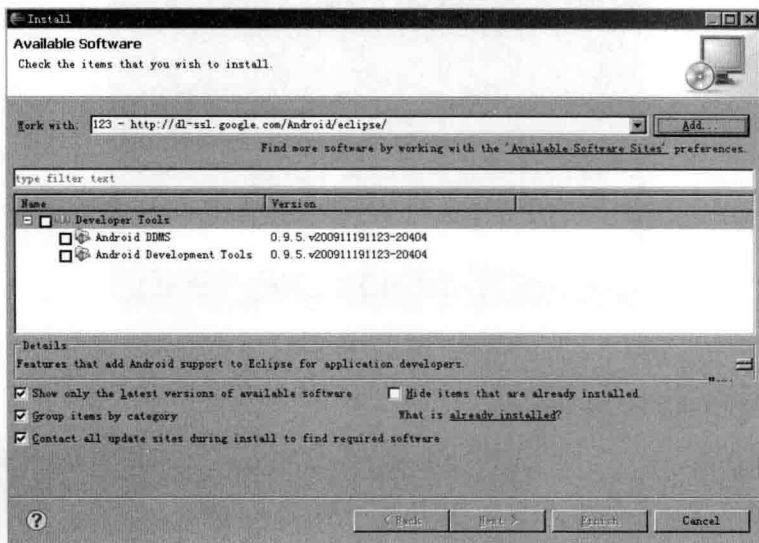


图 1-25 插件列表

(5) 选中 Android DDMS 和 Android Development Tools，然后单击 Next 按钮进入安装界面，如图 1-26 所示。

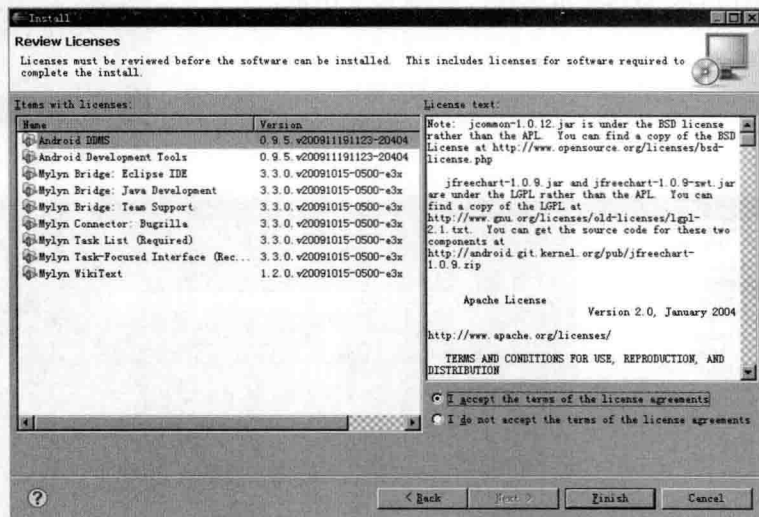


图 1-26 插件安装界面

(6) 选中 I accept the terms of the license agreements 单选按钮，单击 Finish 按钮，开始进行安装，如图 1-27 所示。



注意

在上述步骤中，可能会发生计算插件占用资源的情况，过程有点慢。完成后会提示重启 Eclipse 来加载插件，等重启后就可以用了。并且不同版本的 Eclipse 安装插件的方法和步骤是不同的，但是都大同小异，读者可以根据操作提示自行解决。

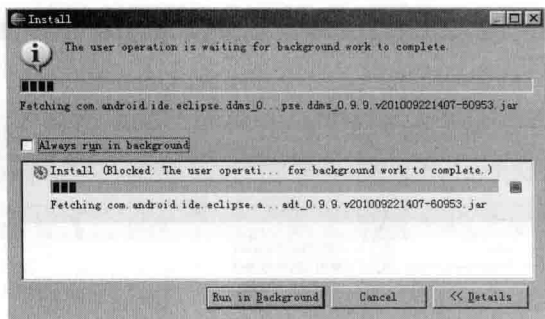


图 1-27 开始安装

1.3.5 设定 Android SDK Home

当完成上述插件安装工作后，此时还不能使用 Eclipse 创建 Android 项目，还需要在 Eclipse 中设置 Android SDK 的主目录。

(1) 打开 Eclipse，在菜单中选择【Window】|【Preferences】选项，如图 1-28 所示。

(2) 在弹出的界面左侧可以看到 Android 项，选中 Android 后，在右侧的“SDK Location”项中输入本计算机安装 Android SDK 的目录，单击 OK 按钮完成设置，如图 1-29 所示。

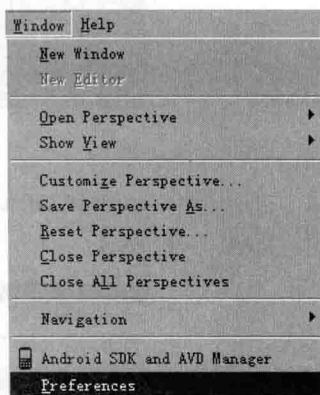


图 1-28 选择 Preferences 选项

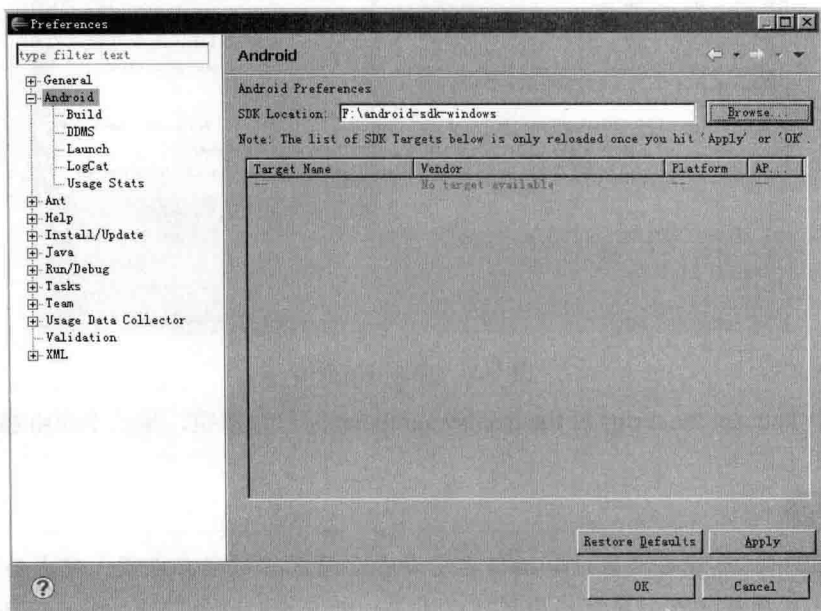


图 1-29 Preferences 项

1.3.6 验证开发环境

经过前面步骤的讲解，一个基本的 Android 开发环境就搭建完成了。下面通过新建一个项目来验证当前的环境是否可以正常工作。

(1) 打开 Eclipse，在菜单中依次选择【File】|【New】|【Project】命令，在弹出的对话框中可以看到 Android 类型的选项，如图 1-30 所示。

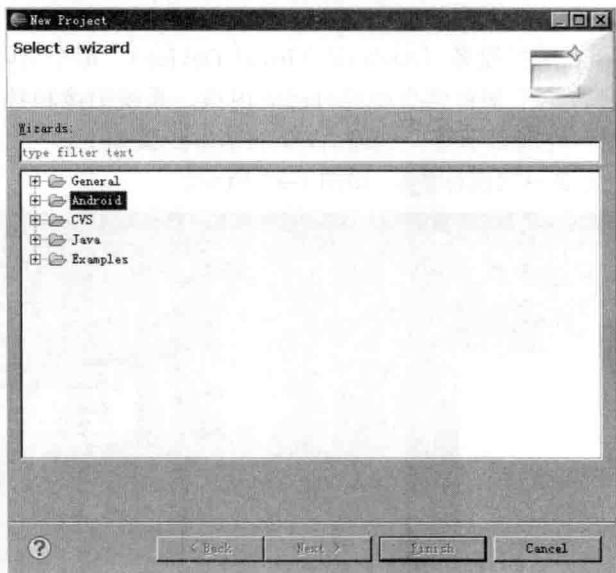


图 1-30 新建项目

(2) 在图 1-30 中选择 Android，单击 Next 按钮后打开 New Android Application 对话框，在对应的文本框中输入必要的信息，如图 1-31 所示。

(3) 单击 Finish 按钮后 Eclipse 会自动完成项目的创建工作，最后会看到如图 1-32 所示的项目结构。

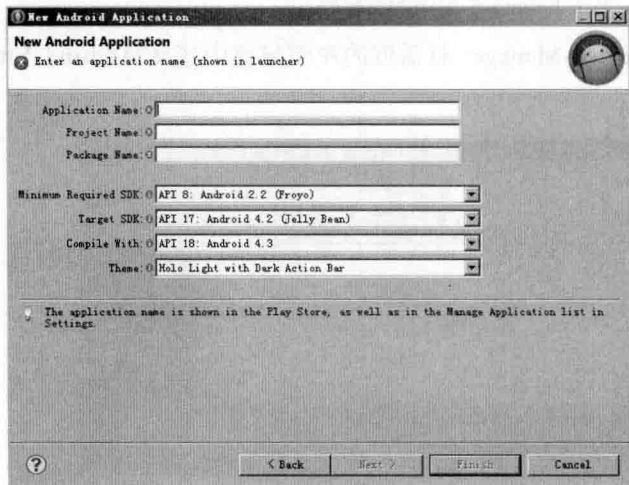


图 1-31 “New Android Application” 对话框

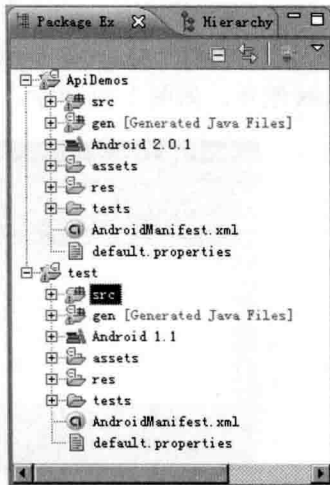



图 1-32 项目结构

1.3.7 创建 Android 虚拟设备 (AVD)

程序开发需要调试，只有经过调试之后用户才能知道程序是否正确运行。对于一款手机系统，怎样才能在电脑平台上调试 Android 程序呢？谷歌为用户提供了模拟器来解决这个问题。所谓模拟器，就是指在电脑上模拟 Android 系统。可以用这个模拟器来调试并运行开发的 Android 程序。开发人员不需要一个真实的 Android 手机，只通过电脑即可模拟运行一个手机，即可开发出应用在手机上面的程序。

AVD 全称为 Android 虚拟设备 (Android Virtual Device)，每个 AVD 模拟了一套虚拟设备来运行 Android 平台，这个平台至少要有自己的内核，系统图像和数据分区，还可以有自己的 SD 卡和用户数据及外观显示等。创建 AVD 的基本步骤如下。

(1) 单击 Eclipse 菜单中的图标，如图 1-33 所示。

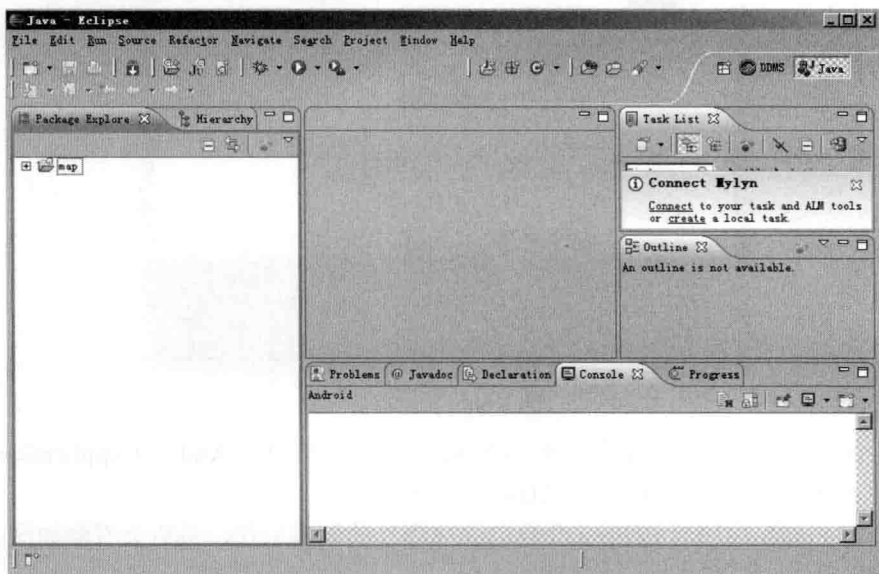


图 1-33 单击 Eclipse 菜单中的图标

(2) 在弹出的 Android Virtual Device Manager 对话框的左侧导航中选择 Android Virtual Devices 选项，如图 1-34 所示。

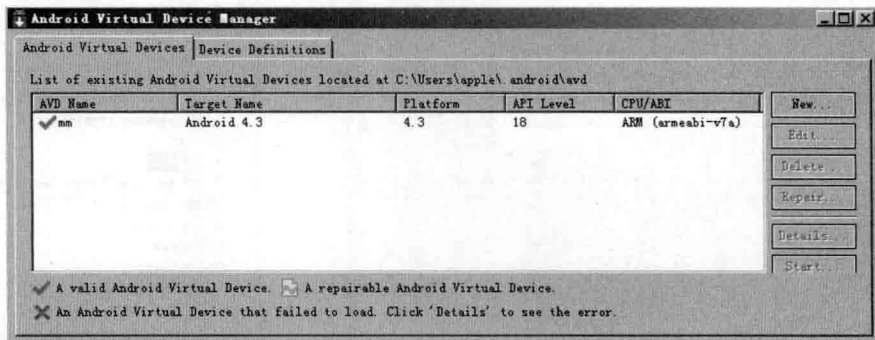


图 1-34 Android Virtual Device Manager 对话框

在 Virtual device 列表中列出了当前已经安装的 AVD 版本，可以通过右侧的按钮来创建、删除或修改 AVD。主要按钮的具体说明如下。

- **New...**：创建新的 AVD，单击此按钮在弹出的界面中可以创建一个新 AVD，如图 1-35 所示。
- **Edit...**：修改已经存在的 AVD。
- **Delete**：删除已经存在的 AVD。
- **Start**：启动一个 AVD 模拟器。

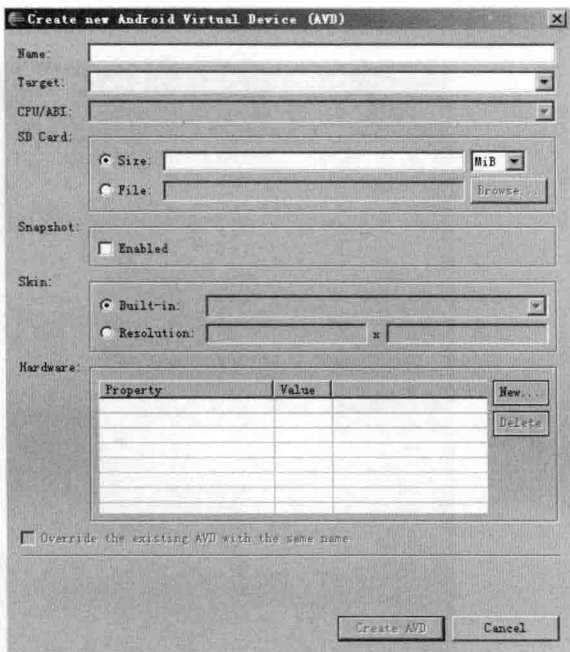


图 1-35 新建 AVD 界面

注意

可以在 CMD 中创建或删除 AVD，例如，可以按照如下 CMD 命令创建一个 AVD。

```
android create avd --name <your_avd_name> --target <targetID>
```

其中 your_avd_name 是需要创建的 AVD 的名字，在 CMD 窗口界面中，如图 1-36 所示。

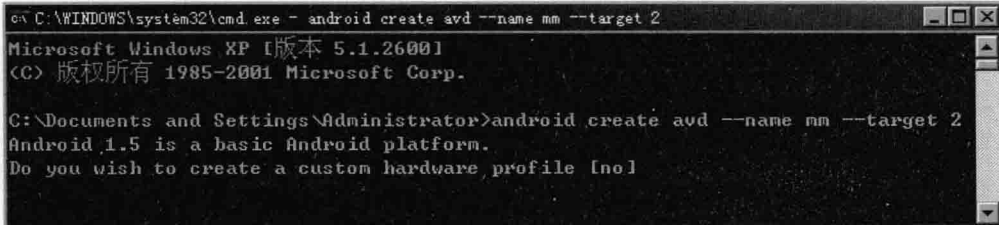


图 1-36 CMD 界面

1.3.8 启动 AVD 模拟器

对于 Android 程序的开发者来说，模拟器的推出给开发者在开发上和测试上带来了很大的便利。无论在 Windows 下还是在 Linux 下，Android 模拟器都可以顺利运行。并且官方提供了 Eclipse 插件，可以将模拟器集成到 Eclipse 的 IDE 环境。Android SDK 中包含的模拟器的功能非常齐全，电话本、通话等功能都可正常使用（当然你没办法真的从这里打电话）。甚至其内置的浏览器和 Maps 都可以联网。用户可以使用键盘输入，可以使用鼠标点击模拟器按钮输入，甚至还可以使用鼠标点击、拖动屏幕进行操纵。模拟器在电脑上模

拟运行的效果如图 1-37 所示。

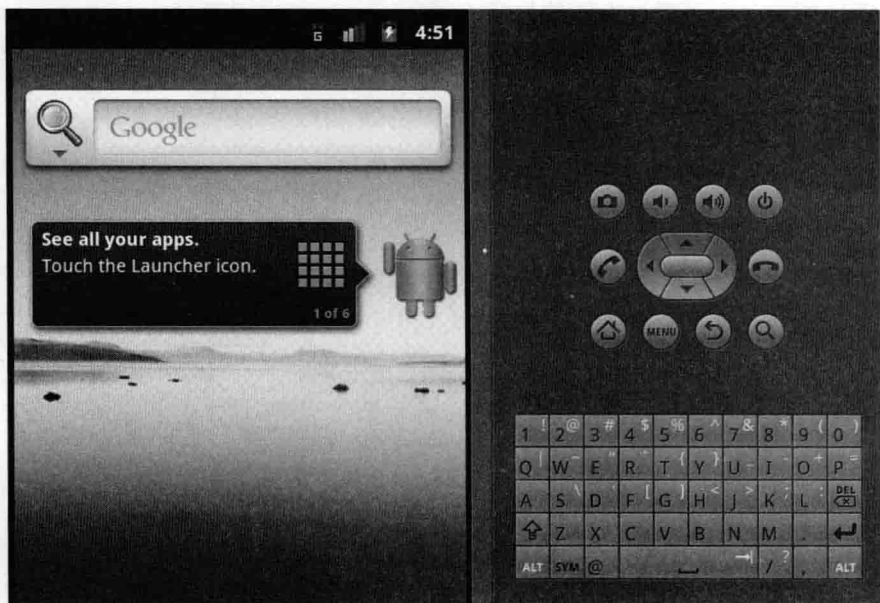


图 1-37 模拟器




注意

Android 模拟器当然不能完全替代真机，具体来说两者有如下差异。

- 模拟器不支持呼叫和接听实际来电，但可以通过控制台模拟电话呼叫（呼入和呼出）。
- 模拟器不支持 USB 连接。
- 模拟器不支持相机/视频捕捉。
- 模拟器不支持音频输入（捕捉），但支持输出（重放）。
- 模拟器不支持扩展耳机。
- 模拟器不能确定连接状态。
- 模拟器不能确定电池电量水平和交流充电状态。
- 模拟器不能确定 SD 卡的插入/弹出。
- 模拟器不支持蓝牙。

有关 Andorid 模拟器的详细知识，将在本章后面的内容中进行详细介绍。

在调试时需要启动 AVD 模拟器，启动 AVD 模拟器的基本流程如下。

(1) 选择图 1-34 列表中名称为 mm 的 AVD，单击  按钮后弹出 Launch Options 界面，如图 1-38 所示。

(2) 单击 Launch 按钮后将会运行名称为 mm 的模拟器，运行界面效果如图 1-39 所示。

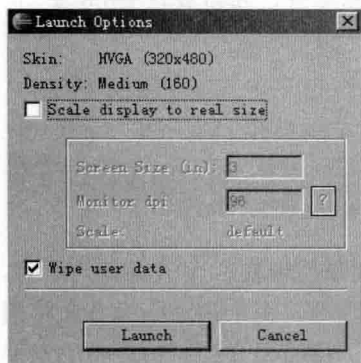


图 1-38 Launch Options 对话框

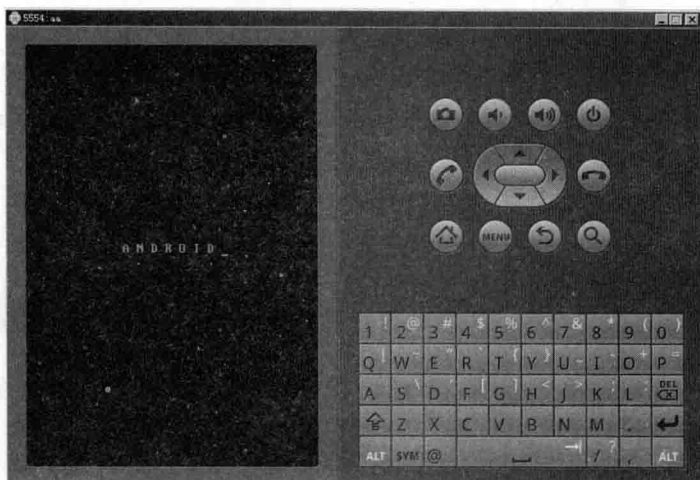


图 1-39 模拟运行成功

注意技巧

快速安装 SDK 的方法:

通过 Android SDK Manager 在线安装的速度非常慢,而且有时容易出错。其实可以先从网络中寻找 SDK 资源,用迅雷等下载工具下载后,将其放到指定目录后就可以完成安装。具体方法是先下载 android-sdk-windows,然后在 android-sdk-windows 下双击 setup.exe,在更新的过程中会发现安装 Android SDK 的速度是 1Kib/s,此时打开迅雷,分别输入下面的地址。

```
https://dl-ssl.google.com/android/repository/platform-tools_r05-windows.zip
https://dl-ssl.google.com/android/repository/docs-3.1_r01-linux.zip
https://dl-ssl.google.com/android/repository/android-2.2_r02-windows.zip
https://dl-ssl.google.com/android/repository/android-2.3.3_r01-linux.zip
https://dl-ssl.google.com/android/repository/android-2.1_r02-windows.zip
https://dl-ssl.google.com/android/repository/samples-2.3.3_r01-linux.zip
https://dl-ssl.google.com/android/repository/samples-2.2_r01-linux.zip
https://dl-ssl.google.com/android/repository/samples-2.1_r01-linux.zip
https://dl-ssl.google.com/android/repository/compatibility_r02.zip
https://dl-ssl.google.com/android/repository/tools_r11-windows.zip
https://dl-ssl.google.com/android/repository/google_apis-10_r02.zip
https://dl-ssl.google.com/android/repository/android-2.3.1_r02-linux.zip
https://dl-ssl.google.com/android/repository/usb_driver_r04-windows.zip
https://dl-ssl.google.com/android/repository/googleadmobadssdkandroid-4.1.0.zip
https://dl-ssl.google.com/android/repository/market_licensing-r01.zip
https://dl-ssl.google.com/android/repository/market_billing_r01.zip
https://dl-ssl.google.com/android/repository/google_apis-8_r02.zip
https://dl-ssl.google.com/android/repository/google_apis-7_r01.zip
https://dl-ssl.google.com/android/repository/google_apis-9_r02.zip
```


可以继续根据自己的开发要求选择不同版本的 API。

下载完后将它们复制到 android-sdk-windows/Temp 目录下，然后再运行 setup.exe，勾选需要的 API 选项，会发现其已经安装好了。一定要保留好原始文件，因为放在 temp 目录下的文件装好后立马就会没有了。

1.4 穿戴设备的前世今生

最近两年来，随着 Android 和 iOS 系统的发展，穿戴设备逐渐展现在广大消费者的面前。谷歌眼镜、苹果手表等新颖而又时尚的设备吸引了广大用户的眼球，在未来这些设备必将引领时尚的潮流，成为科技界的主流产品之一。在本节的内容中，将简要介绍穿戴设备的发展历程和发展现状，并对未来做一个大胆的预测。

1.4.1 发展历程

自从谷歌推出 Google 眼镜产品之后，可穿戴计算设备便成为了当今科技界的火热话题之一。在 CES 2013 和 CES 2014（国际电子展）上，也有不少公司推出了眼镜、腕带等各种可穿戴计算设备，可穿戴计算也越来越火热。

穿戴设备看似是一个新兴事物，但实际上其发展历史可以追溯到上世纪 80 年代。多伦多大学教授 Steve Mann 被称为“可穿戴计算之父”，是公认的第一个赛博格（Cyborg）——这是一个特殊的群体，他们很像科幻小说中的一些角色，利用机器设备来增强自己的感觉，从而加强对环境的掌控。

Steve Mann 自上世纪 80 年代就开始尝试制作类似 Google Glass 这样可以架在自己的鼻梁上，以第一人称的角度来记录周遭事物的眼镜。Mann 最初设计的设备是戴在头盔上的，而经过多年的实验和反复改进，他的头戴式智能眼镜变得越来越轻巧。后来 Mann 成功开发出令智能眼镜小型化，并与电脑和网络相连的技术 EyeTab，这比 Google 眼镜要早 13 年。

以 Google 眼镜为代表的这种穿戴设备，和智能手机最大的不同是把用户的眼睛和手从设备上解放出来了，所以不需要从兜里掏出一个东西，也不需要低下头去看它，它永远在你前面。所以我们有理由相信，可穿戴计算设备（不限于 Google 眼镜）一定可以给人们带来更大的自由，并且在将来一定会成为潮流趋势。

1.4.2 现状介绍

可穿戴计算设备将成为继智能手机、平板电脑之后的又一个潮流。当前可穿戴技术正处于一种过渡时期，一些看似疯狂的想法逐渐在摸索中变得更加成熟。在 CES 2014 电子消费展上，我们已经看到类似 Pebble Steel 这样拥有更精致设计的智能手表，还有很多其他运动腕带、智能眼镜等产品参与展出。下面一起来回顾一下这些出色的可穿戴设备。

（1）Google Project Glass

谷歌眼镜（Google Project Glass）是由谷歌公司于 2012 年 4 月发布的一款“拓展现实”眼镜，如图 1-40 所示。谷歌眼镜具有和智能手机一样的功能，可以通过声音控制拍照、视频

通话和辨别方向,以及上网冲浪、处理文字信息和电子邮件等。

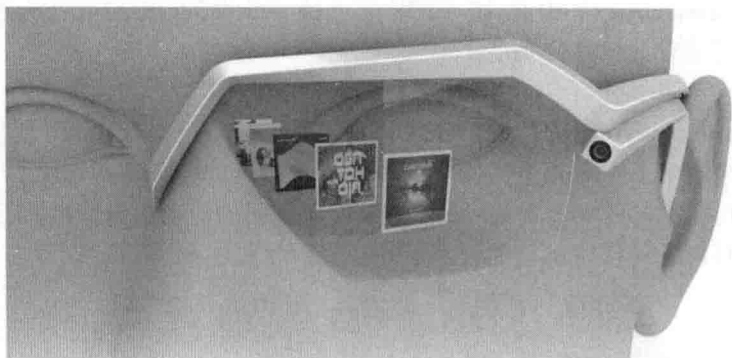


图 1-40 谷歌眼镜

2013 年 4 月 10 日,美国科技博客 Gizmodo 发布了一张图片,揭示了谷歌智能眼镜的工作原理。谷歌眼镜承载着可穿戴设备的开端,它极具想象空间,前途不可限量。但现在看来,它暂时只是一个手机伴侣。基础通信、文字输入依赖手机。

2013 年 11 月 12 日,相关部门发布谷歌眼镜的一系列新功能,包括搜索歌曲、扫描已保存播放列表,以及收听高保真音乐等。

(2) 苹果智能手表

苹果智能手表是苹果正在秘密研发的智能手表产品。苹果手表可能将采用 1.5~2 英寸显示屏,并将采用指纹识别技术。苹果成立了一支 100 人左右的开发团队,专门开发这款设备。2013 年 5 月,凯基证券分析师郭明池(Ming-Chi Kuo)在一份报告中指出,苹果手表的零部件尚未成熟,苹果手表最早 2014 年下半年投产。而苹果 CEO 蒂姆·库克(Tim Cook)曾表示,苹果期待 2013 年秋季和整个 2014 年将推出令人兴奋的新产品。苹果手表的预期效果如图 1-41 所示。



图 1-41 苹果手表

(3) MetaWatch

MetaWatch 是一款智能产品,其设计师此前设计了奢侈品手机 Vertu,所以非常擅长将时尚元素与电子产品有机结合,如图 1-42 所示。



图 1-42 MetaWatch

MetaWatch 的金属表盘充满质感，皮革腕带也显得十分高级，无论是搭配西装还是 T 恤，都十分合适。MetaWatch 支持 10 至 15 米防水，支持 iOS 设备，用户可以从 AppStore 中下载应用程序，实现更多信息的通知功能。

(4) Garmin Vivofit

健身腕带 Garmin Vivofit 是由知名 GPS 厂商 Garmin 研发的。Garmin 此前推出过运动手表产品，此次更是推出了 Vivofit 健身腕带，全面进入到运动监测设备市场。这款运动腕带的设计充满活力，其拥有多种配色款式，不仅能够实现全面的运动数据监测，还支持心率监测，与手机端的应用搭配，可实现出色的健身运动功能，如图 1-43 所示。



图 1-43 Garmin Vivofit

(5) CSR 蓝牙智能首饰

英国厂商 CSR 在 CES 上展出了一系列蓝牙智能首饰，希望通过这种更易用佩戴的产品，实现方便的通知功能。CSR 的应用形式很简单，通过与智能手机连接，闪烁不同的颜色来提醒用户。另外，珠宝公司 Cellini 也参与到设计中，有望在未来带来更精致、更漂亮的设计，如图 1-44 所示。

(6) Pebble Steel 智能手表

Pebble 是 2013 年最成功的智能手表之一，也是真正引发智能手表风潮的始作俑者。而在 2014 年，Pebble Steel 通过简单的设计变化，便获得了高度关注，有望将高人气延续下去。显然，它的最大变化是使用了全金属机身，相比此前廉价的塑料无疑更具质感，也拓宽了用户群体。其功能方面基本延续前作，逐渐丰富的应用程序将实现更好的使用体验，如图 1-45 所示。

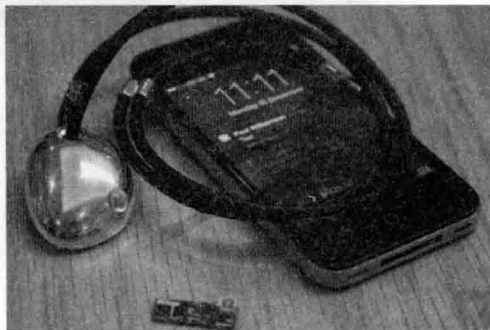


图 1-44 CSR



图 1-45 Pebble Steel 智能手表

(7) 索尼 Core 运动追踪器

索尼在 CES 上发布的 Core 运动追踪器，拥有非常小的体积，其佩戴形式也非常灵活。除了能够实现运动监测，索尼还为其加入了一种生活记录的形式，配合智能手机应用，可以

记录用户每一天的使用行为，包括天气、观看和收听的媒体文件等，这也是一种颇有新意的应用体验，如图 1-46 所示。

(8) 英特尔智能手表及手镯

芯片巨头英特尔在 CES 2014 上也宣布进入可穿戴设备领域，将陆续推出智能手表、手镯等产品。有意思的是，英特尔的智能手镯将与时尚百货 Barneys 合作推出，或许能够让可穿戴设备在时尚领域更进一步，如图 1-47 所示。



图 1-46 Core 运动追踪器

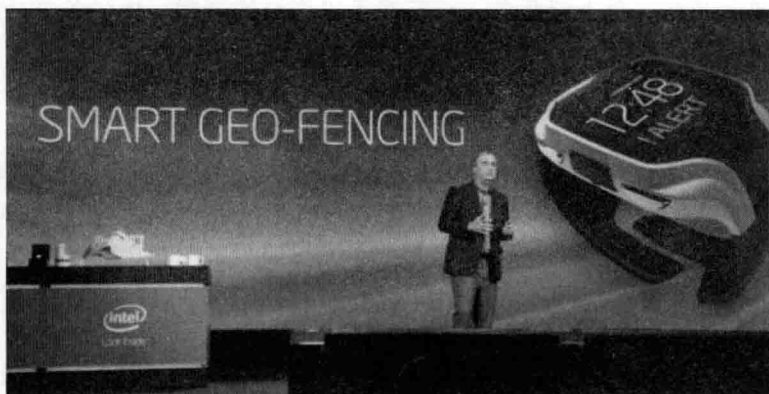


图 1-47 英特尔智能手表及手镯

(9) Razer Nabu 智能腕带

知名游戏厂商 Razer 也推出了 Nabu 智能腕带，支持 iOS 和 Android 设备，不仅拥有时尚的设计，还加入了独特的双屏设计。首先，它是一款健身腕带，内置多种先进的传感器，可以监测用户的运动、睡眠状况；同时，它也是一款通知设备，外部 OLED 屏幕能够显示简单的通知信息，而内部屏幕则可以显示详细的消息，非常方便，如图 1-48 所示。



图 1-48 Razer Nabu 智能腕带

由此可见，当前的可穿戴设备的主流应用包括信息娱乐与社交分享、医疗及健康监测、健身及运用、军用及工业应用等。目前可穿戴设备多以具备部分计算功能、可连接手机及各类终端的便携式配件形式存在，主流的产品形态包括以手腕为支撑的 Watch 类（包括手表和腕带等产品），以脚为支撑的 Shoes 类（包括鞋、袜子或者将来的其他腿上佩戴产品），以头部为支撑的 Glass 类（包括眼镜、头盔、头带等），以及智能服装、书包、拐杖、配饰等各类非主流产品形态。

1.4.3 发展前景分析

可穿戴设备是延续性地穿戴在人体上，具备先进的电路系统、无线联网及独立处理能力

的终端设备，其具备的最重要的两个特点是可长期穿戴和智能化。在智能手机和平板进入停滞期后，以智能眼镜、手表等为代表的智能可穿戴设备成为谷歌、苹果等巨头下一部竞争的主战场。

（1）智能手机推动力

智能手机是可穿戴设备爆发的核心驱动力之一，智能手机现在已经不光用于拿来打电话，或者上网，其内建的处理器与操作系统具有强大的运算能力，使它成为远程的计算机引擎，而同时智能手机具有广泛的用户基础，预计未来，手机可能作为智能控制中心和计算系统，使可穿戴设备、平板、笔记本、电视等所有终端保持互联，而每个人的身体及可穿戴设备将变成微网络，身上配戴各式与智能手机连结的装置，提供各类功能并与智能手机、云端进行数据计算和交互。根据权威统计数据证明，中国移动互联网用户已超过桌面互联网用户。

（2）跨国公司推动力

随着智能手机渗透率快速提升，便携性要求出现、硬件配置提升、传感器及电池改善，可穿戴设备的便携、云端互联等性能优势将越来越明显，预计可穿戴设备将继智能手机之后引发下一个爆发性增长点。尤其是苹果、谷歌、微软、亚马逊和 Facebook 五大平台及相应开发者都进入可穿戴设备领域，后台数据及前端检测传输更加完善时，可穿戴设备将会变成主流。按照 ShareThis 2013 年 6 月的最新数据，消费者在移动设备上点击和分享内容的行为是桌面电脑的 2 倍，随着社交网络越发重要，可供分享的数据暴增。以社交分享平台中份额最高的 Facebook 和 Google 来研究，按照 Searchmetrics 数据，目前 Facebook 的分享量以每个月 10% 的速度增长，Google 分享量目前以每月 19% 的速度增长。截至 2013 年 4 月，Facebook 和 Google 的数据分享量相比 2012 年初增长分别是 202% 及 788%，数据分享爆发时代到来。即时的数据分享和社交网络的需求将导致用户对各类移动终端需求不断提升，可穿戴设备在数据分享和社交领域具备放量基础。

（3）用户推动力

用户对健身、医疗及健康监测等需求也在持续抬升，可穿戴式设备在医疗和健康领域可加载的功能包括脉搏血氧仪、葡萄糖监测、心电图（ECG）、助听器、药物输送等。未来可穿戴设备作为新一代智能终端，将成为新的移动平台市场及生态圈，硬件终端不仅是营收增长点，也将成为粘住客户的产品形态，进而围绕消费者形成可穿戴设备、手机、平板、笔记本、电视、汽车等终端互联互通的一体化智能方案，因此原软硬件、互联网等各类厂商均参与到推出硬件终端产品的环节中来。

在可穿戴设备领域应用中，目前较受欢迎的应用是娱乐和社交，而较快进入商用的功能是健身、医疗及健康监测。娱乐和社交领域的典型产品包括 Google/百度智能眼镜、Sony/三星/果壳智能手表等。医疗领域的可穿戴式设备主要包括脉搏血氧仪、葡萄糖监测、心电图（ECG）、助听器、药物输送等类型产品。目前主要功能进行一体化整合成为趋势，如三星智能手表同时也具备健康监测功能。

据数据显示，2012 年中国可穿戴便携移动医疗设备市场销售规模达到 4.2 亿元，预计到 2015 年这一市场规模将超过 10 亿元，到 2017 年中国可穿戴便携移动医疗设备市场销售规模将接近 50 亿元，市场年复合增长达到 60%。

HIS 预计全球范围内与健康相关的可穿戴设备 App 应用装机量（或下载量）会从 2012

年的 1 亿 5600 万上升至 2017 年的 2 亿 4800 万，随着开发者加入及生态环境改善，可穿戴设备将放量增长。

第三方机构 Endpoint Technologies Associates 预计，如果未来 5 年可穿戴设备市场占比达 4000 万个，便可能为开发商带来 4 亿美元的商机，而程序内广告（in-APP advertising）可能大幅提升营收。

1.4.4 Android 的支持

到目前为止，在讨论可穿戴设备开发时都会提到一个问题：支持蓝牙还是用无线网络与智能手机相连。这是衡量可穿戴设备是否能与智能手机上的软件顺利“对话”的主要依据。其实从 Bluetooth 4.0 开始，这项技术被 Bluetooth SIG（Special Interest Group，负责推动蓝牙技术标准的开发和将其授权给制造商的非营利组织）改名为 Bluetooth Smart 或 Bluetooth Smart Ready。Bluetooth SIG 首席营销官 Suke Jawanda 对 PingWest 说，未来 Bluetooth SIG 也将继续淡化 X.0 的概念，而更加强调 Bluetooth Smart。

可穿戴设备与智能手机之间的数据传输方式对蓝牙技术的要求也与以往不同。过去谈论的蓝牙技术和数据传输，主要考虑的是类似 Spotify 这种在一个较长的时间段里输送数据的需求，现在像 Fitbit Flex 是先收集数据，再断续地在某些“时刻”里将数据传送到用户的手机上，两种数据传输方式不同。Bluetooth Smart 的低耗能技术就可以满足这一需求。现在不少设备制造商都在强调自己支持蓝牙低耗能技术（Bluetooth Low Energy），其实它只是 Bluetooth Smart 其中的一个功能，支持 Bluetooth Smart 的设备都支持蓝牙低耗能技术。

Bluetooth Smart 对可穿戴设备的支持分成硬件和软件。以 Fitbit 为例，如果 Fitbit 开发一款新的产品，支持 Bluetooth Smart，同时要求软件，也就是从 iOS 或者 Android——操作系统层面要支持 Bluetooth Smart。苹果是从 iOS 5（iPhone 4S 及以上版本的手机）开始支持 Bluetooth Smart 的，而 Google 直到 Android 4.3 才开始支持。

Google 在 Android 4.3 中添加了 Bluetooth Smart，在操作系统层面建立一个统一标准。也就是说，从 Android 4.3 开始，将完全支持新蓝牙传输技术，这样就为可穿戴设备开发铺平了道路。而在 Android 4.4 中，新增加了地磁旋转矢量、脚步探测器和计步器三个传感器类别，这些功能很可能是面向谷歌 Android 智能手表、谷歌眼镜及非谷歌出厂的设备。随着更多的厂商在产品中加入运动传感器，追踪人们运动的 Android 手机应用也将从该新功能中获益。

Android 技术核心 框架分析

学习编程不能打无把握之仗，学习 Android 开发也是如此。要想真正精通 Android 网络应用开发，不但需要学习底层和 Android 框架方面的知识，而且还需要了解一些基础的知识，如网页设计和网络传输协议等。从本章开始将简要讲解 Android 体系的具体组成，为读者学习本书后面的高级知识打下基础。

2.1 分析 Android 的系统架构

为了更加深入理解 Android 系统的精髓，初学者很有必要了解 Android 系统的整体架构，了解它的具体组成。只有这样才能知道 Android 究竟能干什么，初学者所要学的是什。

2.1.1 Android 体系结构介绍

Android 是一个移动设备的开发平台，其软件层次结构包括操作系统（OS）、中间件（MiddleWare）和应用程序（Application）。根据 Android 的软件框图，其软件层次结构自下而上分为以下 4 层。

- （1）操作系统层（OS）。
- （2）各种库（Libraries）和 Android 运行环境（RunTime）。
- （3）应用程序框架（Application Framework）。
- （4）应用程序（Application）。

上述各个层的具体结构如图 2-1 所示。

1. 操作系统层（OS）——最底层

因为 Android 源于 Linux，使用了 Linux 内核，所以 Android 使用 Linux 2.6 作为操作系统。Linux 2.6 是一种标准的技术，Linux 也是一个开放的操作系统。Android 对操作系统的使用包括核心和驱动程序两部分，Android 的 Linux 核心为标准的 Linux 2.6 内核，Android 更多的是需要一些与移动设备相关的驱动程序。其主要的驱动如下。

- 显示驱动（Display Driver）：常用基于 Linux 的帧缓冲（Frame Buffer）驱动。
- Flash 内存驱动（Flash Memory Driver）：是基于 MTD 的 Flash 驱动程序。

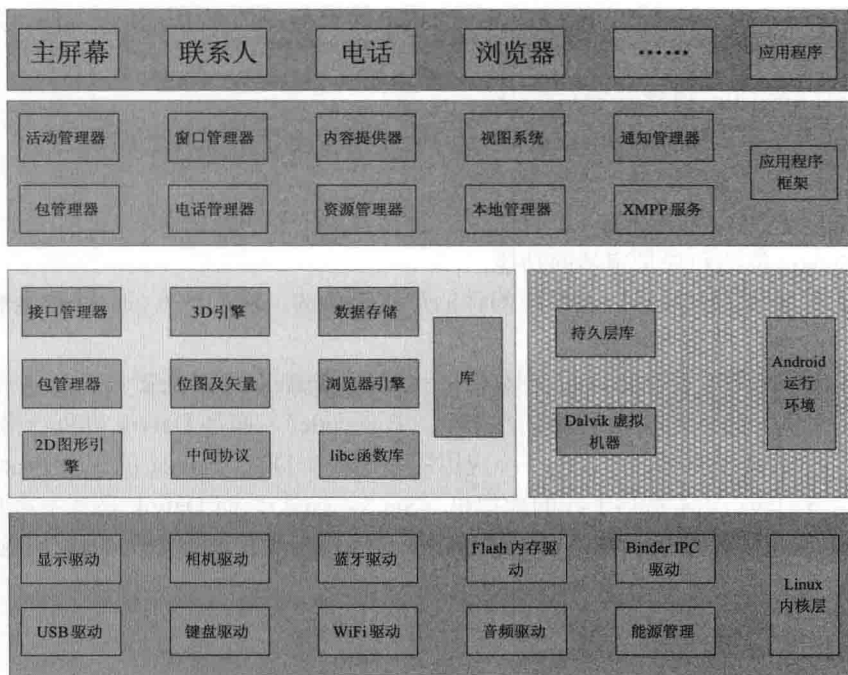


图 2-1 Android 操作系统的组件结构图

- 照相机驱动 (Camera Driver): 常用基于 Linux 的 v4l (Video for) 驱动。
- 音频驱动 (Audio Driver): 常用基于 ALSA (Advanced Linux Sound Architecture 的缩写, 意为高级 Linux 声音体系) 声音系统的驱动。
- WiFi 驱动 (Camera Driver): 基于 IEEE 802.11 标准的驱动程序。
- 键盘驱动 (KeyBoard Driver): 作为输入设备的键盘驱动。
- 蓝牙驱动 (Bluetooth Driver): 基于 IEEE 802.15.1 标准的无线传输技术。
- Binder IPC 驱动: Android 一个特殊的驱动程序, 具有单独的设备节点, 提供进程间通信的功能。
- Power Management (能源管理): 管理电池电量等信息。

2. 各种库 (Libraries) 和 Android 运行环境 (RunTime) ——中间层

本层次对应一般嵌入式系统, 相当于中间件层次。Android 的本层次分成两部分, 一部分是各种库, 另一部分是 Android 运行环境。本层的内容大多是使用 C 语言实现的。其中包含的各种库如下。

- C 库: C 语言的标准库, 也是系统中一个最为底层的库, C 库通过 Linux 的系统调用来实现。
- 多媒体框架 (MediaFramework): 这部分内容是 Android 多媒体的核心部分, 基于 PacketVideo (即 PV) 的 OpenCORE, 从功能上本库一共分为两大部分, 一部分是音频、视频的回放 (PlayBack), 另一部分则是音视频的记录 (Recorder)。
- SGL: 2D 图像引擎。

- **SSL**: 即 Secure Socket Layer, 位于 TCP/IP 协议与各种应用层协议之间, 为数据通信提供安全支持。
- **OpenGL ES 1.0**: 提供对 3D 的支持。
- **界面管理工具 (Surface Management)**: 提供对管理显示子系统等功能。
- **SQLite**: 一个通用的嵌入式数据库。
- **WebKit**: 网络浏览器的核心。
- **FreeType**: 位图和矢量字体的功能。

Android 的各种库一般是以系统中间件的形式提供的, 它们均有的一个显著特点就是与移动设备的平台的应用密切相关。

Android 运行环境主要是指虚拟机技术——Dalvik。Dalvik 虚拟机和一般 Java 虚拟机 (Java VM) 不同, 它执行的不是 Java 标准的字节码 (Bytecode), 而是 Dalvik 可执行格式 (.dex) 中的执行文件。在执行的过程中, 每一个应用程序即一个进程 (Linux 的一个 Process)。二者最大的区别在于 Java VM 是基于栈的虚拟机 (Stack-based), 而 Dalvik 是基于寄存器的虚拟机 (Register-based)。显然, 后者最大的好处在于可以根据硬件实现更大的优化, 这更适合移动设备的特点。

3. 应用程序 (Application)

Android 的应用程序主要是用户界面 (User Interface) 方面的, 通常用 Java 语言编写, 其中还可以包含各种资源文件 (放置在 res 目录中)。Java 程序和相关资源在经过编译后, 会生成一个 APK 包。Android 本身提供了主屏幕 (Home)、联系人 (Contact)、电话 (Phone)、浏览器 (Browsers) 等众多的核心应用。同时应用程序的开发者还可以使用应用程序框架层的 API 实现自己的程序。这也是 Android 开源的巨大潜力的体现。

4. 应用程序框架 (Application Framework)

Android 的应用程序框架为应用程序层的开发者提供 APIs, 它实际上是一个应用程序的框架。由于上层的应用程序是以 Java 构建的, 因此本层次为 UI 界面设计工作提供了所需要的各种控件, 例如, Views (视图组件), 其中又包括 List (列表)、Grid (栅格)、Text Box (文本框)、Button (按钮) 等, 甚至一个嵌入式的 Web 浏览器。

作为一个基本的 Android 应用程序, 可以利用应用程序框架中的以下 5 部分来构建。

- **Activity (活动)**。
- **Broadcast Intent Receiver (广播意图接收者)**。
- **Service (服务)**。
- **Content Provider (内容提供者)**。
- **Intent and Intent Filter (意图和意图过滤器)**。

本书的目的是讲解 Android 网络应用开发的知识, 这方面的内容在结构图中和应用程序 (Application) 相对应, 所以读者需要重点关注应用程序框架 (Application Framework) 的知识。这些知识都是用 Java 开发的, 当然也还需要掌握一些其他层的相关知识, 如底层的内核和驱动等知识。

2.1.2 Android 应用工程文件组成

讲解完 Android 的整体结构之后，接下来开始讲解 Android 工程文件的组成。因为学习本书的目的就是开发 Android 网络应用项目，而每个 Android 应用项目是用 Eclipse 创建的工程，所以很有必要了解一个 Android 工程文件的结构。

在 Eclipse 中，一个基本的 Android 项目的目录结构如图 2-2 所示。

1. src 目录

在 src 目录中保存了开发人员编写的程序文件。和一般的 Java 项目一样，src 目录下保存的是项目的所有包及源文件（.java），res 目录下包含了项目中的所有资源，如程序图标（drawable）、布局文件（layout）和常量（values）等。不同的是，在 Java 项目中没有 gen 目录，也没有每个 Android 项目都必须有的 AndroidManifest.xml 文件。

.java 格式文件是在建立项目时自动生成的，这个文件是只读模式，不能更改。R.java 文件是定义该项目所有资源的索引文件。某项目中 R.java 文件的代码如下。

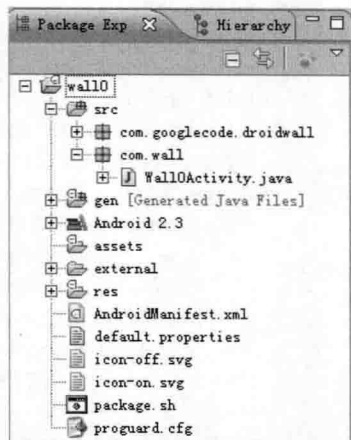


图 2-2 基本的 Android 项目的目录结构

```
package com.yarin.Android.HelloAndroid;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

在上述代码中定义了很多常量，并且这些常量的名称都与 res 文件夹中的文件名相同，这再次证明.java 文件中所存储的是该项目所有资源的索引。有了这个文件，在程序中使用资源将变得更加方便，可以很快地找到要使用的资源，由于这个文件不能被手动编辑，所以当在项目中加入了新的资源时，只需要刷新一下该项目，.java 文件便自动生成所有资源的索引。

2. 设置文件 AndroidManifest.xml

文件 AndroidManifest.xml 是一个控制文件，其中包含该项目中所使用的 Activity、Service

和 Receiver。某项目中文件 AndroidManifest.xml 的代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.yarin.Android.HelloAndroid"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
android:label="@string/app_name">
        <activity android:name=".HelloAndroid"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="9" />
</manifest>
```

在上述代码中，intent-filters 描述了 Activity 启动的位置和时间。每当一个 Activity（或者操作系统）要执行一个操作时，它将创建一个 Intent 的对象，这个 Intent 对象可以描述用户想做什么、想处理什么数据，数据的类型，以及一些其他信息。Android 会和每个 Application 所暴露的 intent-filter 的数据进行比较，找到最合适的 Activity 来处理调用者所指定的数据和操作。下面来详细分析 AndroidManifest.xml 文件，如表 2-1 所示。

表 2-1 AndroidManifest.xml 分析

参 数	说 明
manifest	根节点，描述了 package 中所有的内容
xmlns:android	包含命名空间的声明。xmlns:android=http://schemas.android.com/apk/res/android，使得 Android 中各种标准属性能在文件中使用，提供了大部分元素中的数据
Package	声明应用程序包
application	包含 package 中 application 级别组件声明的根节点。此元素也可包含 application 的一些全局和默认的属性，如标签、icon、主题、必要的权限等。一个 manifest 能包含零个或一个此元素（不能大于一个）
android:icon	应用程序图标
android:label	应用程序名字
activity	activity 是与用户交互的主要工具，是用户打开一个应用程序的初始页面，大部分被用到的其他页面也由不同的 activity 来实现，并声明在另外的 activity 标记中。注意，每一个 activity 必须有一个<activity>标记对应，无论它给外部使用或是只用于自己的 package 中。如果一个 activity 没有对应的标记，用户将不能运行它。另外，为了支持运行时查找 activity，可包含一个或多个<intent-filter>元素来描述 activity 所支持的操作
android:name	应用程序默认启动的 activity
intent-filter	声明了指定的一组组件支持的 Intent 值，从而形成了 Intent Filter。除了能在此元素下指定不同类型的值，属性也能放在这里来描述一个操作所需的唯一的标签、icon 和其他信息
action	组件支持的 Intent action
category	组件支持的 Intent Category。这里指定了应用程序默认启动的 activity
uses-sdk	该应用程序所使用的 SDK 版本

3. 常量定义文件

下面来看看在资源文件中对常量的定义，文件 `String.xml` 的代码如下。

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

上述常量定义文件的代码非常简单，只定义了两个字符串资源，不要小看上述几行代码。它们的内容很“露脸”，里面的字符直接显示在手机屏幕中，就像动态网站中的 HTML 一样。

4. 布局文件

布局 (layout) 文件一般位于 `res/layout/main.xml` 目录，通过其代码能够生成一个显示界面。例如下面的代码。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        />
</LinearLayout>
```

在上述代码中，有如下几个布局和参数。

- `<LinearLayout>`/`</LinearLayout>`：在这个标签中，所有元件都是按由上到下的排序排成的。
- `android:orientation`：表示这个介质的版面配置方式是从上到下垂直地排列其内部的视图。
- `android:layout_width`：定义当前视图在屏幕上所占的宽度，`fill_parent` 即填充整个屏幕。
- `android:layout_height`：定义当前视图在屏幕上所占的高度，`fill_parent` 即填充整个屏幕。
- `wrap_content`：随着文字栏位的不同而改变这个视图的宽度或高度。

在上述布局代码中，使用了一个 `TextView` 来配置文本标签 Widget (构件)，其中设置的属性 `android:layout_width` 为整个屏幕的宽度，`android:layout_height` 可以根据文字来改变高度，而 `android:text` 则设置了这个 `TextView` 要显示的文字内容，这里引用了 `@string` 中的 `hello` 字符串，即 `String.xml` 文件中的 `hello` 所代表的字符串资源。`hello` 字符串的内容 "Hello World, HelloAndroid!" 就是用户在 `HelloAndroid` 项目运行时看到的字符串。



注 意

上面介绍的文件只是主要文件，在项目中需要用户自行编写。在项目中还有很多其他的文件，那些文件很少需要用户来编写，所以在此就不进行讲解了。

2.2 简述五大组件

一个典型的 Android 应用程序通常由 5 个组件组成，这 5 个组件构成了 Android 的核心功能。在本节的内容中，将一一讲解这五大组件的基本知识，为读者步入本书后面知识的学习打下基础。

2.2.1 用 Activity 来表现界面

Activities 是这 5 个组件中最常用的一个组件。程序中 Activity 通常的表现形式是一个单独的界面（screen）。每个 Activity 都是一个单独的类，它扩展实现了 Activity 基础类。这个类显示为一个由 Views 组成的用户界面，并响应事件。大多数程序有多个 Activity。例如，一个文本信息程序有显示联系人列表界面、写信息界面、查看信息界面或者设置界面等。每个界面都是一个 Activity。切换到另一个界面就是载入一个新的 Activity。某些情况下，一个 Activity 可能会给前一个 Activity 返回值——例如，一个让用户选择相片的 Activity 会把选择到的相片返回给其调用者。

打开一个新界面后，前一个界面就被暂停，并放入历史栈中（界面切换历史栈）。使用者可以回溯前面已经打开的存放在历史栈中的界面，也可以从历史栈中删除没有界面价值的界面。Android 在历史栈中保留程序运行产生的所有界面：从第一个界面到最后一个界面。

2.2.2 用 Intent 和 Intent Filters 实现切换

Android 通过一个专门的 Intent 类来进行界面的切换。Intent 描述了程序想做什么（Intent 意为意图、目的、意向）。Intent 类还有一个相关类 IntentFilter。Intent 描述一个请求来做什么事情，IntentFilter 则描述一个 Activity（或下文的 IntentReceiver）能处理什么意图。显示某人联系信息的 Activity 使用一个 IntentFilter，就是说它知道如何处理应用到此人数据的 VIEW 操作。Activities 在文件 AndroidManifest.xml 中使用 IntentFilters。

通过解析，Intents 可以实现 Activity 的切换，用户可以使用 startActivity(myIntent) 启用新的 Activity。系统会考查所有安装程序的 IntentFilters，然后找到与 myIntent 匹配最好的 IntentFilters 所对应的 Activity。这个新 Activity 能够接收 Intent 传来的消息，并因此被启用。解析 Intents 的过程发生在 startActivity 被实时调用时，这样做有如下两个好处。

- （1）Activities 仅发出一个 Intent 请求，便能重用其他组件的功能。
- （2）Activities 可以随时被替换为有等价 IntentFilter 的新 Activity。

2.2.3 Service 为用户服务

Service 是一个没有 UI 且长驻系统的代码，最常见的例子是媒体播放器从播放列表中播放歌曲。在媒体播放器程序中，可能有一个或多个 Activities 让用户选择播放的歌曲。然而在后台播放歌曲时无须 Activity 干涉，因为用户希望在音乐播放的同时能够切换到其他界面。既然这样，媒体播放器 Activity 需要通过 Context.startService() 启动一个 Service，这个 Service 在后台运行以保持继续播放音乐。在媒体播放器被关闭之前，系统会保持音乐后台播放 Service 的正常运行。可以用 Context.bindService() 方法连接到一个 Service 上（如果 Service

未运行，连接后还会启动它)，连接后就可以通过一个 Service 提供的接口与 Service 进行通话。对音乐 Service 来说，提供了暂停和重放等功能。

1. 如何使用服务

在 Android 系统中有如下两种使用服务的方法。

(1) 通过调用 `Context.startService()` 启动服务，调用 `Context.stopService()` 结束服务，`startService()` 可以传递参数给 Service。

(2) 通过调用 `Context.bindService()` 启动，调用 `Context.unbindService()` 结束，还可以通过 `ServiceConnection` 访问 Service。二者可以混合使用，比如可以先 `startService()` 再 `unbindService()`。

2. Service 的生命周期

在 `startService()` 后，即使调用 `startService()` 的进程结束了，Service 还仍然存在，一直到有进程调用 `stopService()` 或者 Service 自己灭亡 (`stopSelf()`) 为止。

在 `bindService()` 后，Service 就和调用 `bindService()` 的进程“同生共死”，也就是说当调用 `bindService()` 的进程“死了”，那么它绑定的 Service 也要跟着结束，当然期间也可以调用 `unbindService()` 让 Service 结束。

当混合使用上述两种方式时，例如既 `startService()` 又 `bindService()`，那么只有 `stopService()` 并且也 `unbindService()`，这个 Service 才会结束。

3. 进程生命周期

Android 系统将会尝试保留那些启动了的或者绑定了的的服务进程，具体说明如下。

(1) 如果该服务正在进程的 `onCreate()`、`onStart()` 或者 `onDestroy()` 这些方法中执行，那么主进程将会成为一个前台进程，以确保此代码不会被停止。

(2) 如果服务已经开始，那么它的主进程的重要性会低于所有的可见进程，但是会高于不可见进程。由于只有少数几个进程是用户可见的，所以只要内存不是特别低，该服务就不会停止。

(3) 如果有多个客户端绑定了服务，只要客户端中的一个对于用户是可见的，就可以认为该服务可见。

2.2.4 用 `BroadcastIntentReceiver` 发送广播

当要执行一些与外部事件相关的代码时，比如来电响铃或者半夜时就可能用到 `IntentReceiver`。尽管 `IntentReceivers` 使用 `NotificationManager` 来通知用户一些有趣的事情发生，但是没有 UI。`IntentReceivers` 可以在文件 `AndroidManifest.xml` 中声明，也可以使用 `Context.registerReceiver()` 来声明。当一个 `IntentReceiver` 被触发时，如果需要系统自然会启动程序。程序也可以通过 `Context.broadcastIntent()` 来发送自己的 `Intent` 广播给其他程序。

2.2.5 用 `ContentProvider` 存储数据

应用程序把数据存放在一个 SQLite 数据库格式文件里，或者存放在其他有效设备里。如果其他程序能够使用程序中的数据，此时 `Content Provider` 就显得很有用。`Content Provider`

是一个实现了一系列标准方法的类，这个类使得其他程序能存储、读取某种 Content Provider 可处理的数据。

2.3 进程和线程

进程和线程很容易理解，电脑中有一个进程管理器，当打开后，会显示当前运行的所有程序。同样在 Android 中也有进程，当某个组件第一次运行时，Android 会启动一个进程。在默认情况下，所有的组件和程序运行在这个进程和线程中，也可以安排组件在其他的进程或者线程中运行。

2.3.1 先看进程

组件运行的进程由 manifest file 控制。组件的节点一般都包含一个 process 属性，如 <activity>、<service>、<receiver> 和 <provider> 节点。属性 process 可以设置组件运行的进程，可以配置组件在一个独立进程中运行，或者多个组件在同一个进程中运行，甚至可以多个程序在一个进程中运行，当然前提是这些程序共享一个 User ID 并给定同样的权限。另外 <application> 节点也包含了 process 属性，用来设置程序中所有组件的默认进程。

当更加常用的进程无法获取足够内存时，Android 会智能地关闭不常用的进程。当下次启动程序时会重新启动这些进程。当决定哪个进程需要被关闭时，Android 会考虑哪个进程对用户更加有用。例如，Android 会倾向于关闭一个长期不显示在界面的进程来支持一个经常显示在界面的进程。是否关闭一个进程取决于组件在进程中的状态。

2.3.2 再看线程

当用户界面需要很快对用户进行响应，就需要将一些费时的操作，如网络连接、下载或者非常占用服务器时间的操作等放到其他线程。也就是说，即使为组件分配了不同的进程，有时候也需要再分配线程。

线程是通过 Java 的标准对象 Thread 来创建的，在 Android 中提供了如下方便的管理线程的方法。

- (1) Looper 在线程中运行一个消息循环。
- (2) Handler 传递一个消息。
- (3) HandlerThread 创建一个带有消息循环的线程。
- (4) Android 让一个应用程序在单独的线程中，指导它创建自己的线程。
- (5) 应用程序组件 (Activity、service、broadcast receiver) 都在理想的主线程中实例化。
- (6) 当一个组件长期未被执行时，将通过阻塞操作来终止这个组件的线程。当被系统调用时，这将中断所有在该进程的其他组件。
- (7) 可以创建一个新的线程来执行长期操作。

2.3.3 应用程序的生命周期

Android 应用程序也如同自然界的生物一样，有自己的生命周期。开发一个程序的目的是为了完成一个功能，如银行计算加息的软件，每当一个用户去柜台办理取款业务时，银行

工作人员便启动了这个程序的生命，当用这个软件完成利息计算时，该软件当前的任务就完成了，此时就需要结束自己的使命。有人提出疑问：“生生死死”多么麻烦，就让这个程序一直处于“活着”的状态，一个用户办理完取款业务后，继续等着下一个用户办理取款业务，这样这个程序不就“长生不老”了吗？其实谁都想自己的程序“长生不老”，但是很不幸，用户不能这样做。原因是计算机的处理性能是一定的，一个人、两个人、三个人的计算机可以处理这个任务。但是一台安装这个软件的机器一天会处理成千上万个取款业务，如果它们都一直“活着”，一台有限配置的计算机能承受得了吗？

由此可见，应用程序的生命周期就是一个程序的存活时间，即在什么时间内有效。Android 是一个构建在 Linux 之上的开源移动开发平台，在 Android 中，多数情况下每个程序都是在各自独立的 Linux 进程中运行的。当一个程序或其某些部分被请求时，它的进程就“出生”了；当这个程序没有必要再运行下去且系统需要回收这个进程的内存用于其他程序时，这个进程就“死亡”了。可以看出，Android 程序的生命周期是由系统控制而非程序自身直接控制的。这和程序员编写桌面应用程序时的思维有一些不同，一个桌面应用程序的进程也是在其他进程或用户请求时被创建的，但是往往是在程序自身收到关闭请求后执行一个特定的动作（比如从 main 函数中返回）而导致进程结束的。要想做好某种类型的程序或者某种平台下的程序的开发，最关键的就是要弄清楚这种类型的程序或整个平台下的程序的一般工作模式并将其熟记在心。在 Android 中，程序的生命周期控制就属于这个范畴。

开发者必须理解不同的应用程序组件，尤其是 Activity、Service 和 Intent Receiver，需要了解这些组件是如何影响应用程序的生命周期的。如果不正确地使用这些组件，可能会导致系统终止正在执行重要任务的应用程序进程。

在 Android 应用中，生命周期的最常见的例子是 Intent Receiver（意图接收器），当 Intent Receiver 在 onReceive 方法中接收到一个 Intent（意图）时，它会启动一个线程，然后返回。一旦返回，系统将认为 Intent Receiver 不再处于活动状态，因而 Intent Receiver 所在的进程也就不再有用了（除非该进程中还有其他的组件处于活动状态）。因此，系统可能会在任意时刻终止该进程以回收占有的内存。这样进程中创建出的那个线程也将被终止。解决这个问题的方法是从 Intent Receiver 中启动一个服务，让系统知道进程中还有处于活动状态的工作。为了使系统能够正确决定在内存不足时应该终止哪个进程，Android 根据每个进程中运行的组件及组件的状态把进程放入一个 Importance Hierarchy（重要性分级）中。

进程的类型多种多样，按照重要的程度主要包括如下几类。

（1）前台进程（Foreground）

前台进程是看得见的，与用户当前正在做的事情密切相关，不同的应用程序组件能够通过不同的方法将它的宿主进程移到前台。在如下的任何一个条件下系统会把进程移动到前台。

- 进程正在屏幕的最前端运行一个与用户交互的活动（Activity），它的 onResume 方法被调用。
- 进程有一个正在运行的 Intent Receiver（它的 IntentReceiver.onReceive 方法正在执行）。
- 进程有一个服务（Service），并且在服务的某个回调函数（Service.onCreate、Service.onStart 或 Service.onDestroy）内有正在执行的代码。

(2) 可见进程 (Visible)

可见进程也是可见的，它有一个可以被用户从屏幕上看到的活动，但不在前台（它的 `onPause` 方法被调用）。假如前台的活动是一个对话框，以前的活动隐藏在对话框之后就会出现这种进程。可见进程非常重要，一般不允许被终止，除非是为保证前台进程的运行而不得不终止它。

(3) 服务进程 (Service)

服务进程是不可见的，拥有一个已经用 `startService()` 方法启动的服务。虽然用户无法直接看到这些进程，但它们做的事情却是用户所关心的（如后台 MP3 回放或后台网络数据的上传下载）。所以系统将一直运行这些进程，除非内存不足以维持所有的前台进程和可见进程。

(4) 后台进程 (Background)

后台进程也是不可见的，只有打开之后才能看见。例如迅雷下载，用户可以将其最小化，虽然在桌面上看不见它，但是它一直在进行下载的工作。其拥有一个当前用户看不到的活动（它的 `onStop()` 方法被调用）。这些进程对用户体验没有直接的影响。如果它们正确执行了活动生命周期，系统可以在任意时刻终止该进程以回收内存，并提供给前面三种类型的进程使用。系统中通常有很多这样的进程在运行，因此要将这些进程保存在 LRU 列表中，以确保当内存不足时用户最近看到的进程最后一个被终止。

(5) 空进程 (Empty)

空进程是指不拥有任何活动的应用程序组件的进程。保留这种进程的唯一原因是在下次应用程序的某个组件需要运行时，不需要重新创建进程，这样可以提高启动速度。系统将以进程中当前处于活动状态组件的重要程度为基础对进程进行分类。进程的优先级可能也会根据该进程与其他进程的依赖关系而增长。假如，进程 A 通过在进程 B 中设置 `Context.BIND_AUTO_CREATE` 标记或使用 `ContentProvider` 被绑定到一个服务 (Service)，那么进程 B 在分类时至少要被看成与进程 A 同等重要。

Activity 的状态转换图如图 2-3 所示。

图 2-3 所示的状态的变化是由 Android 内存管理器决定的，Android 会首先关闭那些包含 Inactive Activity 的应用程序，然后关闭 Stopped 状态的程序。只有在极端情况下才会移除 Paused 状态的程序。

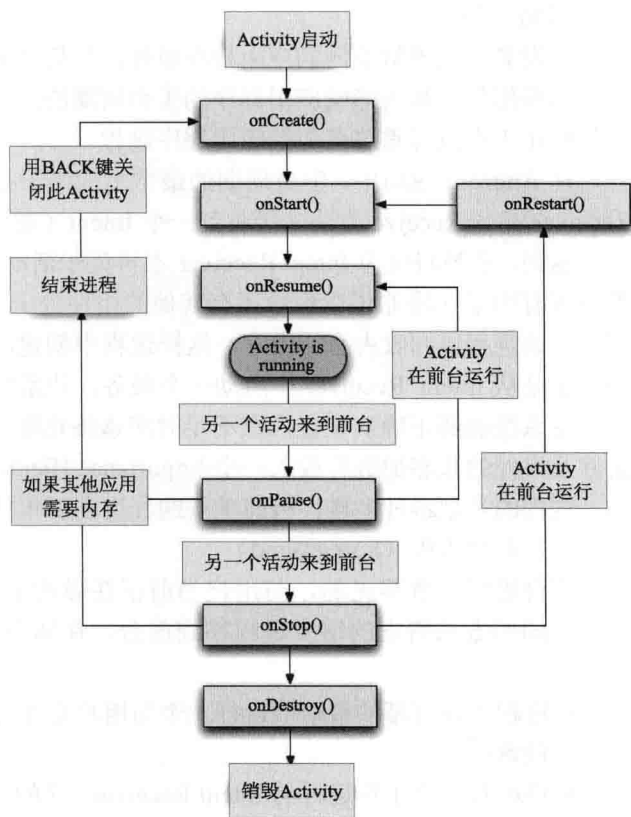


图 2-3 Activity 的状态转换图

2.4 分析 Android 源码结构

获得 Android 源码后，可将源码的全部工程分为如下三部分。

- **Core Project:** 核心工程部分，这是建立 Android 系统的基础，被保存在根目录的各个文件夹中。
- **External Project:** 扩展工程部分，可以使其他开源项目具有扩展功能，被保存在 external 文件夹中。
- **Package:** 包部分，提供了 Android 的应用程序、内容提供者、输入法和 service，被保存在 package 文件夹中。

无论是 Android 1.5 还是 Android 4.3，各个版本的源码目录基本类似。在里面包含了原始 Android 的目标机代码、主机编译工具和仿真环境。解压缩下载的 Android 4.3 源码包后，第一级别目录结构的具体说明如表 2-1 所示。

表 2-1 Android 4.3 源码的根目录

Android 源码根目录	描 述
abi	abi 相关代码，abi:application binary interface，应用程序二进制接口
bionic	bionic C 库
bootable	启动引导相关代码
build	存放系统编译规则及 generic 等基础开发配置包
cts	Android 兼容性测试套件标准
dalvik	dalvik Java 虚拟机
development	应用程序开发相关
device	设备相关代码
docs	介绍开源的相关文档
external	android 使用的一些开源的模组
frameworks	核心框架——Java 及 C++语言，是 Android 应用程序的框架
gdk	即时通信模块
hardware	主要是硬件适配层 HAL 代码
kernel	Linux 的内核文件
libcore	核心库相关
libnativehelper	Support functions for Android's class libraries 的缩写，表示动态库，是实现 JNI 库的基础。
ndk	ndk 相关代码。Android NDK (Android Native Development Kit) 是一系列的开发工具，允许程序开发人员在 Android 应用程序中嵌入 C/C++语言编写的非托管代码
out	编译完成后的代码输出在此目录
packages	应用程序包
pdk	Plug Development Kit 的缩写，是本地开发套件

续表

Android 源码根目录	描 述
prebuilts	x86 和 arm 架构下预编译的一些资源
sdk	sdk 及模拟器
system	文件系统和应用及组件，是用 C 语言实现的
tools	工具文件夹
vendor	厂商定制代码
Makefile	全局的 Makefile

2.5 Android 和 Linux 的关系

在了解 Linux 和 Android 的关系之前，首先需要明确如下 3 点。

(1) Android 采用 Linux 作为内核。

(2) Android 对 Linux 内核作了修改，以适应其在移动设备上的应用。

(3) Android 开始是作为 Linux 的一个分支，后来由于无法并入 Linux 的主开发树，曾经被 Linux 内核组从开发树中删除。2012 年 5 月 18 日，Linux kernel 3.3 发布后其又被加入 Linux。

2.5.1 Android 继承于 Linux

Android 是在 Linux 的内核基础之上运行的，提供的核心系统服务包括安全、内存管理、进程管理、网络组和驱动模型等内容。内核部分还相当于一个介于硬件层和系统中其他软件组之间的一个抽象层次。但是严格来说它不算是 Linux 操作系统。

因为 Android 内核是由标准的 Linux 内核修改而来的，所以继承了 Linux 内核的诸多优点，保留了 Linux 内核的主题架构。同时 Android 按照移动设备的需求，在文件系统、内存管理、进程间通信机制和电源管理方面进行了修改，添加了相关的驱动程序和必要的新功能。但是和其他精简的 Linux 系统相比（比如 uClinux），Android 很大程度地保留了 Linux 的基本架构，因此 Android 的应用性和扩展性更强。当前 Android 版本对应的 Linux 内核版本如下。

- Android 1.5: Linux-2.6.27。
- Android 1.6: Linux-2.6.29。
- Android 2.0,2.1: Linux-2.6.29。
- Android 2.2: Linux-2.6.32.9。
- Android 4.3: Linux-3.4。

2.5.2 Android 和 Linux 内核的区别

Android 系统的系统层面的底层是 Linux，中间加上了一个叫作 Dalvik 的 Java 虚拟机，表面层上面是 Android 运行库。每个 Android 应用都运行在自己的进程上，享有 Dalvik 虚拟机为它分配的专有实例。为了支持多个虚拟机在同一个设备上高效运行，Dalvik 被改写过。

Dalvik 虚拟机执行的是 Dalvik 格式的可执行文件（.dex）——该格式经过优化，以降低内存耗用到最低。Java 编译器将 Java 源文件转为 class 文件，class 文件又被内置的 dx 工具

转化为 dex 格式文件, 这种文件在 Dalvik 虚拟机上注册并运行。

Android 系统的应用软件都是运行在 Dalvik 之上的 Java 软件, 而 Dalvik 是运行在 Linux 中的, 在一些底层功能——比如线程和低内存管理方面, Dalvik 虚拟机依赖 Linux 内核。由此可见, Android 是运行在 Linux 之上的操作系统, 但是它本身不能算是 Linux 的某个版本。

Android 内核和 Linux 内核的差别主要体现在 11 个方面, 接下来将一一简要介绍。

(1) Android Binder

Android Binder 源代码位于:

```
drivers/staging/android/binder.c
```

Android Binder 是基于 OpenBinder 框架的一个驱动, 用于提供 Android 平台的进程间通信 (IPC, inter-process communication)。原来的 Linux 系统上层应用的进程间通信主要是 D-bus (desktop bus), 采用消息总线的方式来进行 IPC。

(2) Android 电源管理 (PM)

Android 电源管理是一个基于标准 Linux 电源管理系统的轻量级的 Android 电源管理驱动, 针对嵌入式设备做了很多优化。利用锁和定时器来切换系统状态, 控制设备在不同状态下的功耗, 以达到节能的目的。

Android 电源管理的源代码分别位于:

```
kernel/power/earlysuspend.c
kernel/power/consoleearlysuspend.c
kernel/power/fbearlysuspend.c
kernel/power/wakelock.c
kernel/power/userwakelock.c
```

(3) 低内存管理器 (Low Memory Killer)

Android 中的低内存管理器和 Linux 标准的 OOM (Out Of Memory) 相比, 其机制更加灵活, 它可以根据需要杀死的进程来释放需要的内存。Low Memory Killer 的代码很简单, 关键的一个函数是 Lowmem_shrinker。其作为一个模块在初始化时调用 register_shrinker 注册了 lowmem_shrinker, 它会被 vm 在内存紧张的情况下调用。Lowmem_shrinker 完成具体操作。简单说就是寻找一个最合适的进程将其杀死, 从而释放它占用的内存。

低内存管理器的源代码位于:

```
drivers/staging/android/lowmemorykiller.c
```

(4) 匿名共享内存 (Ashmem)

匿名共享内存为进程间提供大块共享内存, 同时为内核提供回收和管理这个内存的机制。如果一个程序尝试访问 Kernel 释放的一个共享内存块, 它将会收到一个错误提示, 然后重新分配内存并重载数据。

匿名共享内存的源代码位于:

```
mm/ashmem.c
```

(5) Android PMEM (Physical)

PMEM 用于向用户空间提供连续的物理内存区域, DSP 和某些设备只能工作在连续的物理内存上。驱动中提供了 mmap、open、release 和 ioctl 等接口。

Android PMEM 的源代码位于:

```
drivers/misc/pmem.c
```


(6) Android Logger

Android Logger 是一个轻量级的日志设备，用于抓取 Android 系统的各种日志，是 Linux 中所没有的。

Android Logger 的源代码位于：

```
drivers/staging/android/logger.c
```

(7) Android Alarm

Android Alarm 提供了一个定时器用于把设备从睡眠状态唤醒，同时它也提供了一个即使在设备睡眠时也会运行的时钟基准。

Android Alarm 的源代码位于：

```
drivers rtc/alarm.c  
drivers rtc/alarm-dev.c
```

(8) USB Gadget 驱动

USB Gadget 驱动是一个基于标准 Linux USB gadget 驱动框架的设备驱动，Android 的 USB 驱动基于 gadget 框架。

USB Gadget 驱动的源代码位于：

```
drivers/usb/gadget/android.c  
drivers/usb/gadget/f_adb.c  
drivers/usb/gadget/f_mass_storage.c
```

(9) Android Ram Console

为了提供调试功能，Android 允许将调试日志信息写入一个被称为 RAM Console 的设备中，它是一个基于 RAM 的 Buffer。

Android Ram Console 的源代码位于：

```
drivers/staging/android/ram_console.c
```

(10) Android timed device

Android timed device 提供了对设备进行定时控制的功能，目前仅仅支持 vibrator 和 LED 设备。

Android timed device 的源代码位于：

```
drivers/staging/android/timed_output.c(timed_gpio.c)
```

(11) Yaffs2 文件系统

在 Android 系统中，采用 Yaffs2 作为 MTD nand flash 文件系统。Yaffs2 是一个快速稳定的应用于 NAND 和 NOR Flash 的跨平台的嵌入式设备文件系统，同其他 Flash 文件系统相比，Yaffs2 使用更小的内存来保存其运行状态，因此它占用内存小；Yaffs2 的垃圾回收非常简单而且快速，因此能达到更好的性能；Yaffs2 在大容量的 NAND Flash 上性能表现尤为明显，非常适合大容量的 Flash 存储。

Yaffs2 文件系统源代码位于 fs/yaffs2/目录下。

Android 是在 Linux 的内核基础之上运行的，提供的核心系统服务包括安全、内存管理、进程管理、网络组和驱动模型等内容。内核部分还相当于一个介于硬件层和系统中其他软件组之间的一个抽象层次。但是严格来说它不算是 Linux 操作系统。

2.6 第一段 Android 程序

本实例的功能是在手机屏幕中显示问候语“你好我的朋友！”，在具体开始之前先做一个简单的流程规划，如图 2-4 所示。

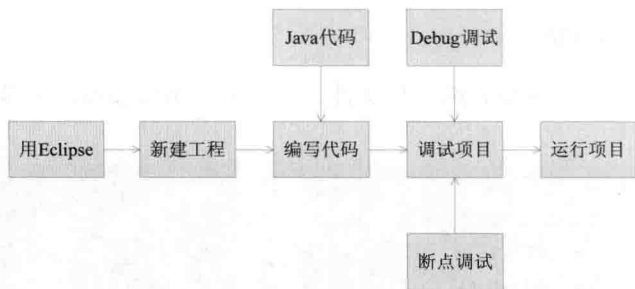


图 2-4 规划流程图

题 目	目 的	源码路径
实例 2-1	在手机屏幕中显示问候语	光盘\daima\2\first

本实例的具体实现流程如下。

1. 新建 Android 工程

(1) 在 eclipse 中选择【File】|【New】|【Project】命令，新建一个工程文件，如图 2-5 所示。

(2) 选择 Android Project 选项，单击 Next 按钮。

(3) 在弹出的 New Android Project 对话框中，设置工程信息，如图 2-6 所示。

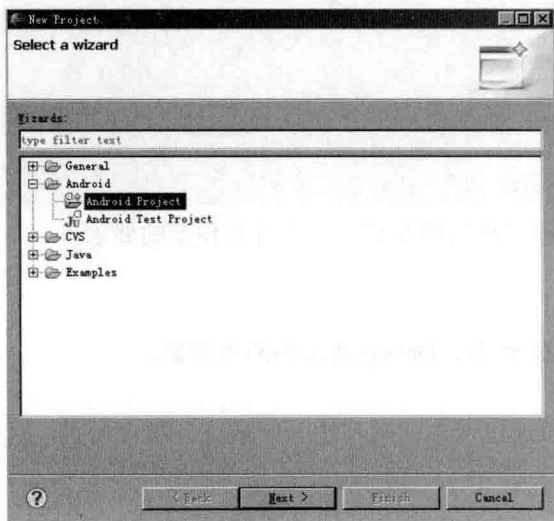


图 2-5 新建工程文件

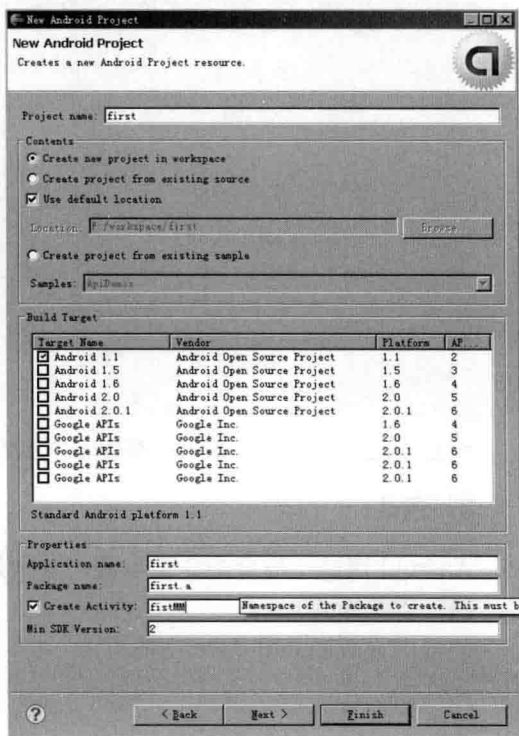


图 2-6 设置工程信息

在图 2-6 所示的界面中依次设置工程名字、包名字、Activity 名字和应用名字。

2. 编写代码和代码分析

现在已经创建了一个名为 `first` 的工程文件，打开文件 `first.java`，会显示自动生成的如下代码。

```
package first.a;
import android.app.Activity;
import android.os.Bundle;
public class fistMM extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

如果此时运行程序，将不会显示任何信息。此时可以对上述代码进行稍微的修改，让程序输出 `HelloWorld`。具体代码如下。

```
package first.a;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class fistMM extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView tv = new TextView(this);
        tv.setText("你好我的朋友!");
        setContentView(tv);
    }
}
```

经过上述代码改写后，可以在屏幕中输出“你好我的朋友！”，完全符合预期的要求。

3. 调试

Android 调试一般分为 3 个步骤，分别是设置断点、Debug 调试和断点调试。

(1) 设置断点

此处的设置断点和 Java 中的方法一样，可以通过双击代码左边的区域进行断点设置，如图 2-7 所示。



图 2-7 设置断点

为了调试方便，可以设置显示代码的行数。只需在代码左侧的空白部分右击，在弹出的快捷菜单中选择 Show Line Numbers，如图 2-8 所示。

(2) Debug 调试

Debug Android 调试项目的方法和普通 Debug Java 调试项目的方法类似，唯一的不同的是在选择调试项目时选择 Android Application 命令。具体方法是右击项目名，在弹出的快捷菜单中选择【Debug As】|【Android Application】命令，如图 2-9 所示。

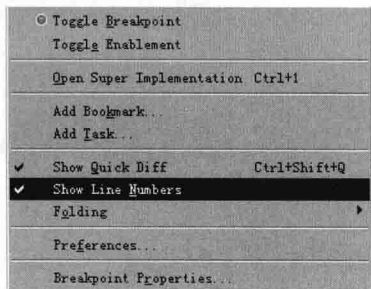


图 2-8 显示行数

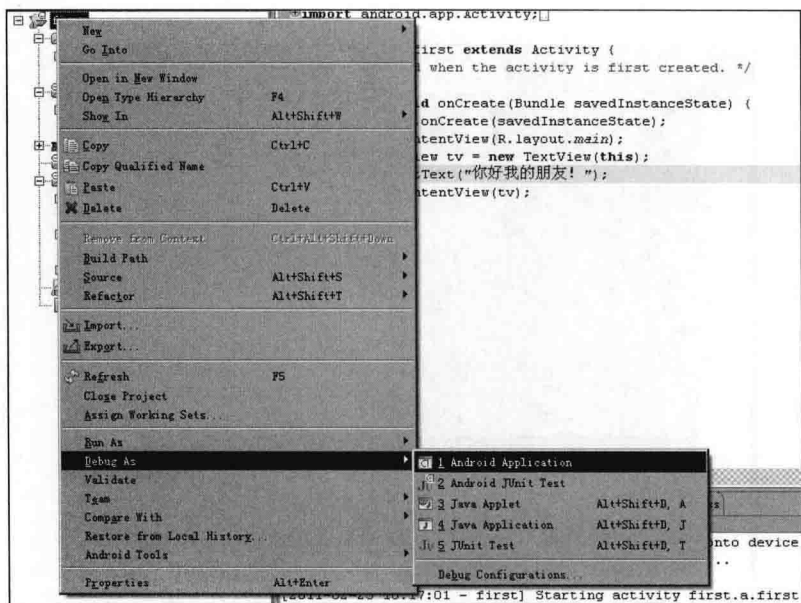


图 2-9 Debug 项目

(3) 断点调试

可以进行单步调试，具体调试方法和调试普通 Java 程序的方法类似，调试界面如图 2-10 所示。

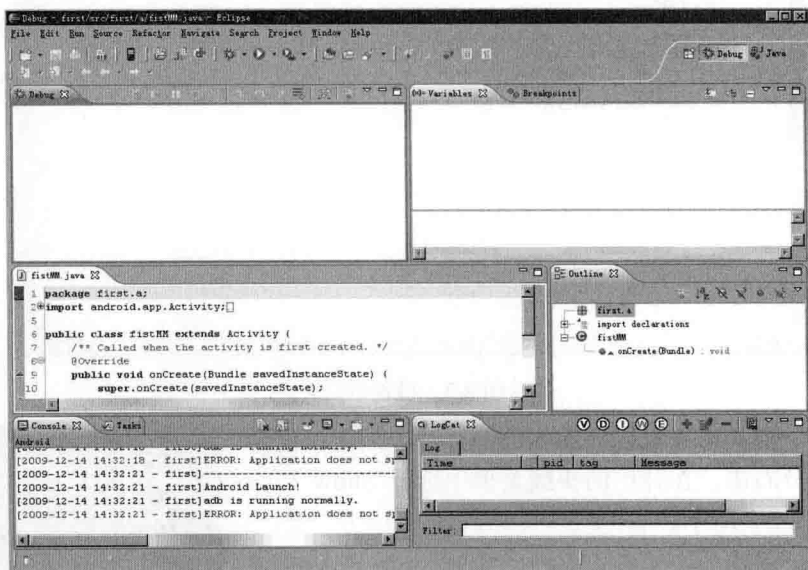


图 2-10 调试界面

4. 运行项目

将上述代码保存后即可运行这段程序，具体过程如下。

(1) 右击项目名，在弹出的快捷菜单中选择【Run As】|【Android Application】命令，如图 2-11 所示。

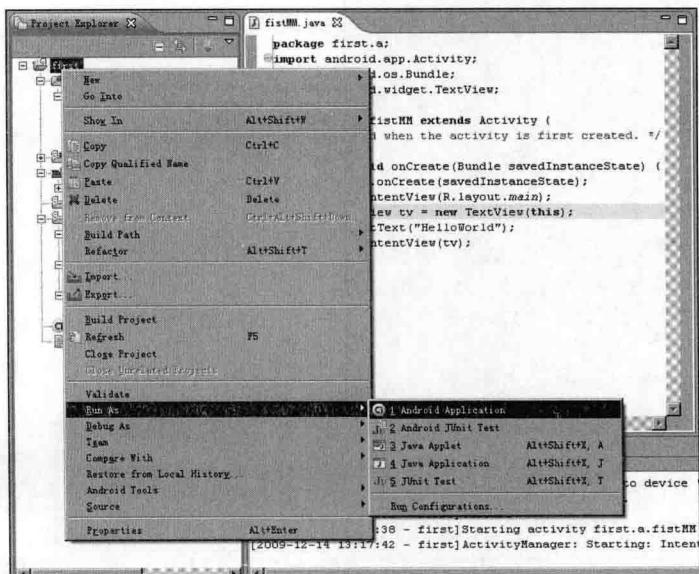


图 2-11 开始调试

(2) 此时工程开始运行，运行完成后在屏幕中输出“你好我的朋友！”这段文字，如图 2-12 所示。

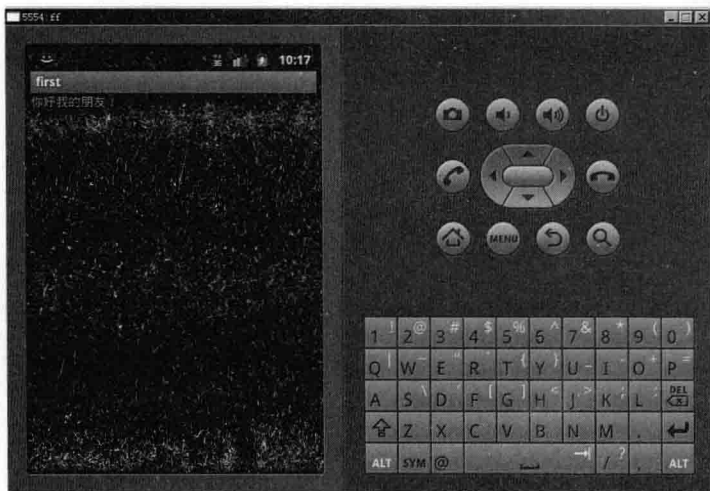


图 2-12 运行结果

HTTP 数据通信

超文本传输协议（HTTP，HyperText Transfer Protocol）是互联网上应用最为广泛的一种网络协议。所有的 WWW 文件都必须遵守这个标准。设计 HTTP 最初的目的是为了提供一种发布和接收 HTML 页面的方法。在本章的内容中，简要介绍在 Android 系统中使用 HTTP 传输数据的方法。

3.1 HTTP 基础



在本节的内容中，首先简要介绍 HTTP 技术的相关基本理论知识，为读者学习本书后面的知识打下基础。

3.1.1 HTTP 概述

HTTP 是一个客户端和服务端请求和应答的标准（TCP）。客户端是终端用户，服务器端是网站。通过使用 Web 浏览器、网络爬虫或者其他工具，客户端发起一个到服务器上指定端口（默认端口为 80）的 HTTP 请求。这个客户端被称为用户代理（user agent）。应答的服务器上存储着一些资源，如 HTML 文件和图像。这个应答服务器被称为源服务器（origin server）。在用户代理和源服务器中间可能存在多个中间层，如代理、网关或者隧道（tunnels）。尽管 TCP/IP 协议是互联网上最流行的应用，HTTP 协议并没有规定必须使用它和（基于）其支持的层。事实上，HTTP 可以在任何其他互联网协议上，或者在其他网络上实现。HTTP 只假定（其下层协议提供）可靠的传输，任何能够提供这种保证的协议都可以被其使用。

通常，由 HTTP 客户端发起一个请求，建立一个到服务器指定端口（默认是 80 端口）的 TCP 连接。HTTP 服务器则在那个端口监听客户端发送过来的请求。一旦收到请求，服务器（向客户端）发回一个状态行，比如 HTTP/1.1 200 OK 和（响应的）消息，消息的消息体可能是请求的文件、错误消息或者其他一些信息。

HTTP 使用 TCP 而不是 UDP 的原因在于打开一个网页必须传送很多数据，而 TCP 协议提供传输控制，按顺序组织数据和进行错误纠正。

3.1.2 HTTP 协议的功能

HTTP 是超文本传输协议，是客户端浏览器或其他程序与 Web 服务器之间的应用层通信

协议。在 Internet 上的 Web 服务器上存放的都是超文本信息，客户机需要通过 HTTP 协议传输所要访问的超文本信息。HTTP 包含命令和传输信息，不仅可用于 Web 访问，也可以用于其他因特网/内联网应用系统之间的通信，从而实现各类应用资源超媒体访问的集成。

当需要浏览一个网站时，只要在浏览器的地址栏里输入网站的地址就可以了，如 `www.*****.com`，但是在浏览器的地址栏中出现的却是：`http://www.*****.com`，为什么会多出一个 `http` 呢？

在浏览器的地址栏里输入的网站地址叫作 URL (Uniform Resource Locator，统一资源定位符)。就像每家每户都有一个门牌地址一样，每个网页也都有一个 Internet 地址。在浏览器的地址框中输入一个 URL 或单击一个超链接时，URL 就确定了要浏览的地址。浏览器通过超文本传输协议 (HTTP)，将 Web 服务器上站点的网页代码提取出来，并翻译成漂亮的网页。因此，在认识 HTTP 之前，有必要先弄清楚 URL 的组成。例如，`http://www.*****.com/china/index.htm`，它的含义如下。

- (1) `http://`：代表超文本转移协议，通知 `*****.com` 服务器显示 Web 页，通常不用输入。
- (2) `www`：代表一个 Web (万维网) 服务器。
- (3) `*****.com/`：这是装有网页的服务器的域名，或站点服务器的名称。
- (4) `china/`：为该服务器上的子目录，就好像用户的文件夹。
- (5) `index.htm`：`index.htm` 是文件夹中的一个 HTML 文件 (网页)。

众所周知，Internet 的基本协议是 TCP/IP 协议，然而在 TCP/IP 模型最上层的是应用层 (Application layer)，它包含所有高层的协议。高层协议有：文件传输协议 FTP、电子邮件传输协议 SMTP、域名系统服务 DNS、网络新闻传输协议 NNTP 和 HTTP 协议等。

HTTP 协议 (HyperText Transfer Protocol，超文本传输协议) 是用于从 WWW 服务器传输超文本到本地浏览器的传输协议。它可以使浏览器更加高效，使网络传输减少。它不仅保证计算机正确快速地传输超文本文档，还确定传输文档中的哪一部分，以及哪部分内容首先显示 (如文本先于图形) 等。这就是为什么在浏览器中看到的网页地址都是以 “`http://`” 开头的原因。

3.1.3 Android 中的 HTTP

在 Android 系统中，提供了如下 3 种通信接口。

- 标准 Java 接口：`java.net`。
- Apache 接口：`org.apache.http`。
- Android 网络接口：`android.net.http`。

网络编程在无线应用程序开发过程中起到了重要的作用。在 Android 系统中包括 Apache HttpClient 库，此库为执行 Android 中的网络操作之首选方法。除此之外，Android 还可允许通过标准的 Java 联网 API (`java.net` 包) 来访问网络。即便使用 `Java.net` 包，也是在内部使用该 Apache 库。

为了访问互联网，需要设置应用程序获取 `android.permission.INTERNET` 权限的许可。

在 Android 系统中，存在如下与网络连接相关的包。

(1) `java.net`

提供联网相关的类，包括流和数据报套接字、互联网协议及通用的 HTTP 处理。此为多

用途的联网资源。经验丰富的 Java 开发人员可立即使用此惯用的包来创建应用程序。

(2) java.io

尽管未明确联网，但其仍然非常重要。此包中的各种类通过其他 Java 包中提供的套接字和链接来使用。它们也可用来与本地文件进行交互（与网络进行交互时经常发生）。

(3) java.nio

包含表示具体数据类型的缓冲的各种类。便于基于 Java 语言的两个端点之间的网络通信。

(4) org.apache.*

表示可为进行 HTTP 通信提供精细控制和功能的各种包。可将 Apache 识别为普通的开源 Web 服务器。

(5) android.net

包括核心 java.net.* 类之外的各种附加的网络接入套接字。此包包括 URL 类，其通常在传统联网之外的 Android 应用程序开发中使用。

(6) android.net.http

包含可操作 SSL 证书的各种类。

(7) android.net.wifi

包含可管理 Android 平台中 WiFi (802.11 无线以太网) 所有方面的各种类。并非所有的设备均配备有 WiFi 能力，尤其随着 Android 在对制造商（如诺基亚和 LG）手机的翻盖手机研发方面取得了进展。

(8) android.telephony.gsm

包含管理和发送短信（文本）消息所要求的各种类。随着时间的推移，可能将引入一种附加的包，以提供有关非 GSM 网络（如 CDMA 或类似 android.telephony.cdma）的类似功能。

3.2 使用 Apache 接口

在 Android 平台中，使用的最多的是 Apache 接口。因此在本节的内容中，将详细介绍使用 Apache 接口（org.apache.http）实现和网络连接的基本知识。读者可结合演示代码来理解每一个知识点，为学习本书后面的知识打下基础。

3.2.1 Apache 接口基础

在 Apache HttpClient 库中，如下内容为对网络连接有用的各种包。

- org.apache.http.HttpResponse。
- org.apache.http.client.HttpClient。
- org.apache.http.client.methods.HttpGet。
- org.apache.http.impl.client.DefaultHttpClient。
- HttpClient httpClient=new DefaultHttpClient()。

如果要从服务器检索此信息，则需要使用 HttpGet 类的构造器，代码如下。

```
HttpGet request=new HttpGet("http://innovator.samsungmobile.com");
```

然后用 `HttpClient` 类的 `execute()` 方法中的 `HttpGet` 对象来检索 `HttpResponse` 对象,代码如下。

```
HttpResponse response = client.execute(request);
```

接着读取已检索的响应,代码如下。

```
BufferedReader rd = new BufferedReader
    (new InputStreamReader(response.getEntity().
        getContent()));

String line = "";
while ((line = rd.readLine()) != null) {
    Log.d("output: ",line);
}
```

3.2.2 Apache 应用要点

1. 连网流程

在 Android 系统中,可以采用 `HttpPost` 和 `HttpGet` 来封装 `post` 请求和 `get` 请求,再使用 `HttpClient` 的 `excute` 方法发送 `post` 或者 `get` 请求并返回服务器的响应数据。使用 Apache 连网的基本流程如下。

(1) 设置连接和读取超时时间,并新建 `HttpClient` 对象。代码如下。

```
// 设置连接超时时间和数据读取超时时间
HttpParams httpParams = new BasicHttpParams();
HttpConnectionParams.setConnectionTimeout(httpParams,
    KeySource.CONNECTION_TIMEOUT_INT);
HttpConnectionParams.setSoTimeout(httpParams,
    KeySource.SO_TIMEOUT_INT);
//新建 HttpClient 对象
HttpClient httpClient = new DefaultHttpClient(httpParams)
```

(2) 实现 `Get` 请求,代码如下。

```
// 获取请求
HttpGet get = new HttpGet(url);
// set HTTP head parameters
//Map<String, String> headers
if (headers != null)
{
    Set<String> setHead = headers.keySet();
    Iterator<String> iteratorHead = setHead.iterator();
    while (iteratorHead.hasNext())
    {
        String headerName = iteratorHead.next();
        String headerValue = (String) headers.get(headerName);
        MyLog.d(headerName, headerValue);
        get.setHeader(headerName, headerValue);
    }
}

// connect
//need try catch
response = httpClient.execute(get);
```

(3) 实现 Post 发送请求处理，代码如下。

```

HttpPost post = new HttpPost(KeySource.HOST_URL_STR);
// set HTTP head parameters
Map<String, String> headers = heads;
Set<String> setHead = headers.keySet();
Iterator<String> iteratorHead = setHead.iterator();
while (iteratorHead.hasNext())
{
    String headName = iteratorHead.next();
    String headValue = (String) headers.get(headName);
    post.setHeader(headName, headValue);
}
/**
 * 通常的 HTTP 实体需要在执行上下文时动态生成
 * HttpClient 的提供使用 EntityTemplate 实体类和 ContentProducer 接口支持动态实体
 * 内容制作是通过将需求的内容写到一个输出流来实现的，每次请求时都会产生
 * 因此，通过 EntityTemplate 创建实体通常是独立的，重复性好
 */
ContentProducer cp = new ContentProducer()
{
    public void writeTo(OutputStream outstream)
        throws IOException
    {
        Writer writer = new OutputStreamWriter(outstream,
            "UTF-8");
        writer.write(requestBody);
        writer.flush();
        writer.close();
    }
};
HttpEntity entity = new EntityTemplate(cp);
post.setEntity(entity);
}
//connect , need try catch
response = httpClient.execute(post);

```

(4) 通过 Response 响应请求，代码如下。

```

if (response.getStatusLine().getStatusCode() == 200)
{
    /**
     * 因为直接调用 toString 可能会导致某些中文字符出现乱码的情况。所以此处使用
     * toByteArray
     * 如果需要转成 String 对象，可以先调用 EntityUtils.toByteArray() 方法将消
     * 息实体转成 byte 数组
     * 再由 new String(byte[] bArray) 转换成字符串
     */
    byte[] bResultXml = EntityUtils.toByteArray(response
        .getEntity());
    if (bResultXml != null)
    {

```

```
String strXml = new String(bResultXml, "utf-8");
```

这样使用 Apache 实现连网处理数据交互的过程就完成了，无论多么复杂的项目，都需要遵循上述流程。

2. HttpClient 网络通信

Apache 的核心功能是 HttpClient，和网络有关的功能几乎都需要用 HttpClient 来实现。在 Android 开发中经常会用到网络连接功能与服务器进行数据的交互，为此 Android 的 SDK 提供了 Apache 的 HttpClient 来方便用户使用各种 Http 服务。可以把 HttpClient 想象成一个浏览器，通过它的 API 可以很方便地发出 GET 请求和 POST 请求。

例如，只需要如下几行代码就能发出一个简单的 GET 请求并打印响应结果。

```
try {
    // 创建一个默认的 HttpClient
    HttpClient httpClient = new DefaultHttpClient();
    // 创建一个 GET 请求
    HttpGet request = new HttpGet("www.google.com");
    // 发送 GET 请求，并将响应内容转换成字符串
    String response = httpClient.execute(request, new BasicResponseHandler());
    Log.v("response text", response);
} catch (ClientProtocolException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

肯定有读者禁不住要问为什么上述代码要使用单例 HttpClient 呢？这只是一段演示代码，实际的项目中的请求与响应处理会复杂一些，并且还要考虑到代码的容错性，但是这并不是本篇的重点。重点注意代码的第三行：

```
HttpClient httpClient = new DefaultHttpClient();
```

在发出 HTTP 请求前先创建了一个 HttpClient 对象，而在实际项目中，很可能在多处需要进行 HTTP 通信，这时候不需要为每个请求都创建一个新的 HttpClient。因为之前已经提到，HttpClient 就像一个小型的浏览器，对于整个应用，只需要一个 HttpClient 就够了。由此可以得出，使用简单的单例就可以实现，代码如下。

```
public class CustomerHttpClient {
    private static HttpClient customerHttpClient;
    private CustomerHttpClient() {
    }

    public static HttpClient getHttpClient() {
        if(null == customerHttpClient) {
            customerHttpClient = new DefaultHttpClient();
        }
        return customerHttpClient;
    }
}
```

但是如果同时有多个请求需要处理呢？答案是使用多线程。假如现在应用程序使用同一个 `HttpClient` 来管理所有的 `Http` 请求，一旦出现并发请求，那么一定会出现多线程的问题。这就好像我们的浏览器只有一个标签页却有多个用户，A 要上 google，B 要上 baidu，这时浏览器就会忙不过来了。幸运的是，`HttpClient` 提供了创建线程安全对象的 API，帮助用户能很快地得到线程安全的“浏览器”。如下代码很好地解决了多线程问题。

```
public class CustomerHttpClient {
    private static final String CHARSET = HTTP.UTF_8;
    private static HttpClient customerHttpClient;
    private CustomerHttpClient() {
    }
    public static synchronized HttpClient getHttpClient() {
        if (null == customerHttpClient) {
            HttpParams params = new BasicHttpParams();
            // 设置一些基本参数
            HttpProtocolParams.setVersion(params, HttpVersion.HTTP_1_1);
            HttpProtocolParams.setContentCharset(params,
                CHARSET);
            HttpProtocolParams.setUseExpectContinue(params, true);
            HttpProtocolParams
                .setUserAgent(
                    params,
                    "Mozilla/5.0 (Linux;U;Android
2.2.1;en-us;Nexus One Build.FRG83) "
                        + "AppleWebKit/553.1 (KHTML,like Gecko) Version/
4.0 Mobile Safari/533.1");

            // 超时设置
            /* 从连接池中取连接的超时时间 */
            ConnManagerParams.setTimeout(params, 1000);
            /* 连接超时 */
            HttpConnectionParams.setConnectionTimeout(params, 2000);
            /* 请求超时 */
            HttpConnectionParams.setSoTimeout(params, 4000);
            // 设置我们的 HttpClient 支持 HTTP 和 HTTPS 两种模式
            SchemeRegistry schReg = new SchemeRegistry();
            schReg.register(new Scheme("http", PlainSocketFactory
                .getSocketFactory(), 80));
            schReg.register(new Scheme("https", SSLSocketFactory
                .getSocketFactory(), 443));
            // 使用线程安全的连接管理来创建 HttpClient
            ClientConnectionManager conMgr = new ThreadSafeClientConnManager(
                params, schReg);
            customerHttpClient = new DefaultHttpClient(conMgr, params);
        }
        return customerHttpClient;
    }
}
```

在上述代码中，通过 `getHttpClient()` 方法为 `HttpClient` 配置了一些基本参数和超时设置，然后使用 `ThreadSafeClientConnManager` 来创建线程安全的 `HttpClient`。

3. HttpClient 网络编程

接下来将介绍使用 Apache HttpClient 库进行网络连接的过程,通过一段具体代码的实现过程来讲解。

(1) 在文件 AndroidManifest.xml 中添加 android.permission.INTERNET 许可,这样就可以允许用户的应用程序访问网络。具体代码如下。

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

(2) 使用“Apache Connection”界面来创建项目 com.apache。将布局文件 main.xml 更改为如下代码。

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:text="Enter URL"
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </TextView>
    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:text="http://innovator.samsungmobile.com"
        android:layout_height="wrap_content">
    </EditText>
    <Button
        android:text="Click Here"
        android:id="@+id/button1"
        android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content">
    </Button>
    <EditText
        android:id="@+id/editText2"
        android:layout_width="match_parent"
        android:layout_height="fill_parent">
    </EditText>
</LinearLayout>

```

(3) 编写主程序文件 `ApacheConnection.java`，此代码将可允许查看 HTML 代码，具体实现代码如下。

```

package com.apache;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class ApacheConnection extends Activity {
    Button bt;
    TextView textView1;
    TextView textView2;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        bt = (Button) findViewById(R.id.button1);
        textView1 = (TextView) findViewById(R.id.editText1);
        textView2 = (TextView) findViewById(R.id.editText2);
        bt.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                textView2.setText("");
                try {
                    /*Apache HttpClient Library*/
                    HttpClient client = new DefaultHttpClient();
                    HttpGet request = new HttpGet(textView1.getText().toString());
                    HttpResponse response = client.execute(request);
                    /* response code*/

```



```

BufferedReader rd = new BufferedReader(
    new InputStreamReader(response.getEntity().
        getContent()));
String line = "";
while ((line = rd.readLine()) != null) {
    textView2.append(line);
}
} catch (Exception exe) {
    exe.printStackTrace();
}
});
}
}

```

执行上述代码后，可以在手机浏览器中查看输入网址网页的 HTML 代码，如图 3-1 所示。



图 3-1 执行效果

3.2.3 Apache 应用要点

Apache 中的 HttpClient 是一个完善的 HTTP 客户端，它提供了对 HTTP 协议的全面支持，可以使用 HTTP GET 和 POST 进行访问。下面结合实例来介绍 HttpClient 的使用方法。

(1) 新建一个 http 项目，项目结构如图 3-2 所示。

在这个项目中不需要任何的 Activity，所有的操作都在单元测试类 HttpTest.java 中完成。

(2) 因为用到了单元测试，所以在这里先介绍一下如何配置 Android 中的单元测试。所有配置信息均在 AndroidManifest.xml 中完成，具体代码如下。



图 3-2 项目结构

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.scott.http"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <!-- 配置测试要使用的类库 -->
        <uses-library android:name="android.test.runner"/>
    </application>
    <!-- 配置测试设备的主类和目标包 -->
    <instrumentation android:name="android.test.InstrumentationTestRunner"
        android:targetPackage="com.scott.http"/>
    <!-- 访问 HTTP 服务所需的网络权限 -->
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-sdk android:minSdkVersion="8" />
</manifest>

```

这里的单元测试类需要继承于 `android.test.AndroidTestCase` 类，此类继承于 `junit.framework.TestCase`，并提供了 `getContext()`方法来获取 Android 上下文环境。

(3) 编写测试文件 `HttpTest.java`，具体代码如下。

```
package com.scot.http.test;

import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

import junit.framework.Assert;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.mime.MultipartEntity;
import org.apache.http.entity.mime.content.InputStreamBody;
import org.apache.http.entity.mime.content.StringBody;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;

import android.test.AndroidTestCase;

public class HttpTest extends AndroidTestCase {

    private static final String PATH = "http://192.168.1.57:8080/web";

    public void testGet() throws Exception {
        HttpClient client = new DefaultHttpClient();
        HttpGet get = new HttpGet(PATH + "/TestServlet?id=1001&name=john&age=60");
        HttpResponse response = client.execute(get);
        if (response.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
            InputStream is = response.getEntity().getContent();
            String result = inStream2String(is);
            Assert.assertEquals(result, "GET_SUCCESS");
        }
    }

    public void testPost() throws Exception {
        HttpClient client = new DefaultHttpClient();
        HttpPost post = new HttpPost(PATH + "/TestServlet");
    }
}
```

```

List<NameValuePair> params = new ArrayList<NameValuePair>();
params.add(new BasicNameValuePair("id", "1001"));
params.add(new BasicNameValuePair("name", "john"));
params.add(new BasicNameValuePair("age", "60"));
HttpEntity formEntity = new UrlEncodedFormEntity(params);
post.setEntity(formEntity);
HttpResponse response = client.execute(post);
if (response.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
    InputStream is = response.getEntity().getContent();
    String result = inStream2String(is);
    Assert.assertEquals(result, "POST_SUCCESS");
}
}

public void testUpload() throws Exception {
    InputStream is = getContext().getAssets().open("books.xml");
    HttpClient client = new DefaultHttpClient();
    HttpPost post = new HttpPost(PATH + "/UploadServlet");
    InputStreamBody isb = new InputStreamBody(is, "books.xml");
    MultipartEntity multipartEntity = new MultipartEntity();
    multipartEntity.addPart("file", isb);
    multipartEntity.addPart("desc", new StringBody("this is description."));
    post.setEntity(multipartEntity);
    HttpResponse response = client.execute(post);
    if (response.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
        is = response.getEntity().getContent();
        String result = inStream2String(is);
        Assert.assertEquals(result, "UPLOAD_SUCCESS");
    }
}

//将输入流转换成字符串
private String inStream2String(InputStream is) throws Exception {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    byte[] buf = new byte[1024];
    int len = -1;
    while ((len = is.read(buf)) != -1) {
        baos.write(buf, 0, len);
    }
    return new String(baos.toByteArray());
}
}

```

在上述代码中包含了3个测试用例。首先，在定位服务器地址时用到了IP，因为这里不能用localhost，服务器端是在windows上运行的，而本单元测试运行在Android平台，如果使用localhost就意味着在Android内部去访问服务，可能是访问不到的，所以必须用IP来定位服务。

(1) testGet 测试

使用 `HttpGet` 将请求参数直接附在 URL 后面, 然后由 `HttpClient` 执行 GET 请求, 如果响应成功则取得响应内如输入流, 并转换成字符串, 最后判断是否为 `GET_SUCCESS`。`testGet` 测试对应的服务器端 `Servlet` 代码如下。

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    System.out.println("doGet method is called.");
    String id = request.getParameter("id");
    String name = request.getParameter("name");
    String age = request.getParameter("age");
    System.out.println("id:" + id + ", name:" + name + ", age:" + age);
    response.getWriter().write("GET_SUCCESS");
}
```

(2) testPost 测试

在此使用 `HttpPost`, URL 后面并没有附带参数信息, 参数信息被包装成一个由 `NameValuePair` 类型组成的集合的形式, 然后经过 `UrlEncodedFormEntity` 处理后调用 `HttpPost` 的 `setEntity` 方法进行参数设置, 最后由 `HttpClient` 执行。`testPost` 测试对应的服务器端代码如下。

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    System.out.println("doPost method is called.");
    String id = request.getParameter("id");
    String name = request.getParameter("name");
    String age = request.getParameter("age");
    System.out.println("id:" + id + ", name:" + name + ", age:" + age);
    response.getWriter().write("POST_SUCCESS");
}
```

上述的两端代码是最基本的 GET 请求和 POST 请求, 参数都是文本数据类型, 能满足普通的需求, 不过在有的场合, 例如要用到上传文件时, 就不能使用基本的 GET 请求和 POST 请求了, 要使用多部件的 POST 请求。下面介绍如何使用多部件 POST 操作上传一个文件到服务器端。

因为 Android 附带的 `HttpClient` 版本暂不支持多部件 POST 请求, 所以需要用到一个 `HttpMime` 开源项目, 该组件是专门处理与 MIME 类型有关的操作。因为 `HttpMime` 包含在 `HttpComponents` 项目中, 所以用户需要去 Apache 官方网站下载 `HttpComponents`, 然后把其中的 `HttpMime.jar` 包放到项目中去, 如图 3-3 所示。

然后看一下 `testUpload` 中的测试用例, 用 `HttpMime` 提供的 `InputStreamBody` 处理文件流参数, 用 `StringBody` 处理普通文本参数, 最后把所有类型参数都加入到一个

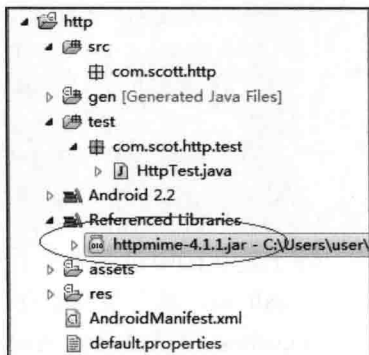


图 3-3 添加 `HttpMime.jar` 包

MultipartEntity 的实例中, 并将这个 MultipartEntity 设置为此次 POST 请求的参数实体, 然后执行 POST 请求。服务器端 Servlet 代码如下。

```
package com.scott.web.servlet;

import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileItemFactory;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;

@SuppressWarnings("serial")
public class UploadServlet extends HttpServlet {

    @Override
    @SuppressWarnings("rawtypes")
    protected void doPost(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
        boolean isMultipart = ServletFileUpload.isMultipartContent(request);
        if (isMultipart) {
            FileItemFactory factory = new DiskFileItemFactory();
            ServletFileUpload upload = new ServletFileUpload(factory);
            try {
                List items = upload.parseRequest(request);
                Iterator iter = items.iterator();
                while (iter.hasNext()) {
                    FileItem item = (FileItem) iter.next();
                    if (item.isFormField()) {
                        //普通文本信息处理
                        String paramName = item.getFieldName();
                        String paramValue = item.getString();
                        System.out.println(paramName + ":" + paramValue);
                    } else {
                        //上传文件信息处理
                        String fileName = item.getName();
                        byte[] data = item.get();
                        String filePath = getServletContext().getRealPath("/files")
                            + "/" + fileName;
                        FileOutputStream fos = new FileOutputStream(filePath);
                        fos.write(data);
                    }
                }
            } catch (FileUploadException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

```

        fos.close();
    }
}
} catch (FileUploadException e) {
    e.printStackTrace();
}
}
response.getWriter().write("UPLOAD_SUCCESS");
}
}

```

这样在服务器端成功地使用 Apache 开源项目 FileUpload 实现了文件上传处理，在使用时一定不要忘记附加 commons-fileupload 和 commons-io 这两个项目的 jar 包，对服务器端开发不太熟悉的读者可以到网上查找一下相关资料。

介绍完上述三种不同的情况之后还需要考虑一个问题，在实际项目中不可能每次都新建 HttpClient，而是应该只为整个应用创建一个 HttpClient，这样就可以将其用于所有 HTTP 通信。另外还需要注意在通过一个 HttpClient 的同时发出多个请求可能会引发多线程问题。针对上述两个问题，需要优化处理上述项目，优化处理过程如下。

(1) 扩展系统默认的 Application，并将其应用在项目中。

(2) 使用 HttpClient 类库提供的 ThreadSafeClientManager 来创建和管理 HttpClient。优化处理后的工程文件结构如图 3-4 所示。

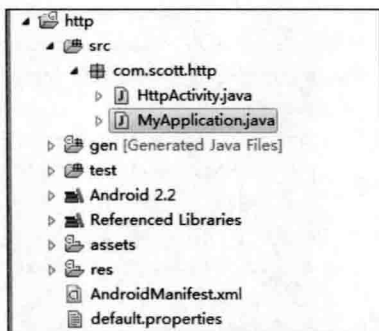


图 3-4 工程文件结构

(3) 在文件 MyApplication.java 中扩展了系统的 Application，具体代码如下。

```

package com.scott.http;

import org.apache.http.HttpVersion;
import org.apache.http.client.HttpClient;
import org.apache.http.conn.ClientConnectionManager;
import org.apache.http.conn.scheme.PlainSocketFactory;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.scheme.SchemeRegistry;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.impl.conn.tsccm.ThreadSafeClientConnManager;
import org.apache.http.params.BasicHttpParams;
import org.apache.http.params.HttpParams;
import org.apache.http.params.HttpProtocolParams;
import org.apache.http.protocol.HTTP;

import android.app.Application;

public class MyApplication extends Application {

    private HttpClient httpClient;

```



```
@Override
public void onCreate() {
    super.onCreate();
    httpClient = this.createHttpClient();
}

@Override
public void onLowMemory() {
    super.onLowMemory();
    this.shutdownHttpClient();
}

@Override
public void onTerminate() {
    super.onTerminate();
    this.shutdownHttpClient();
}

//创建 HttpClient 实例
private HttpClient createHttpClient() {
    HttpParams params = new BasicHttpParams();
    HttpProtocolParams.setVersion(params, HttpVersion.HTTP_1_1);
    HttpProtocolParams.setContentCharset(params, HTTP.DEFAULT_CONTENT_CHARSET);
    HttpProtocolParams.setUseExpectContinue(params, true);

    SchemeRegistry schReg = new SchemeRegistry();
    schReg.register(new Scheme("http", PlainSocketFactory.getSocketFactory(),
    80));
    schReg.register(new Scheme("https", SSLSocketFactory.getSocketFactory(),
    443));

    ClientConnectionManager connMgr = new ThreadSafeClientConnManager
    (params, schReg);

    return new DefaultHttpClient(connMgr, params);
}

//关闭连接管理器并释放资源
private void shutdownHttpClient() {
    if (httpClient != null && httpClient.getConnectionManager() != null) {
        httpClient.getConnectionManager().shutdown();
    }
}

//对外提供 HttpClient 实例
public HttpClient getHttpClient() {
    return httpClient;
}
```

在上述代码中重写了 `onCreate()` 方法，在系统启动时就创建一个 `HttpClient`；重写了

onLowMemory()和 onTerminate()方法,在内存不足和应用结束时关闭连接,释放资源。需要注意的是,当实例化 DefaultHttpClient 时,传入一个由 ThreadSafeClientConnManager 创建的一个 ClientConnectionManager 实例,负责管理 HttpClient 的 HTTP 连接。

(4) 在文件 AndroidManifest.xml 中进行如下配置,目的是让“优化”版的 Application 生效。

```
<application android:name=".MyApplication" ...>
...
</application>
```

如果不进行上述配置,系统依旧会默认使用 android.app.Application。在添加上述配置后,系统就会使用前面编写的 com.scott.http.MyApplication,然后就可以在 context 中调用 getApplication()来获取 MyApplication 实例。

(5) 经过上面的“优化”处理配置,接下来就可以在活动中应用了。编写的文件 HttpActivity.java 的实现代码如下。

```
package com.scott.http;

import java.io.ByteArrayOutputStream;
import java.io.InputStream;

import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class HttpActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button btn = (Button) findViewById(R.id.btn);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                execute();
            }
        });
    }

    private void execute() {
        try {
            MyApplication app = (MyApplication) this.getApplication();
```

```

//获取 MyApplication 实例
HttpClient client = app.getHttpClient(); //获取 HttpClient 实例
HttpGet get = new HttpGet("http://192.168.1.57:8080/web/TestServlet?id=1001&name=john&age=60");
HttpResponse response = client.execute(get);
if (response.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
    InputStream is = response.getEntity().getContent();
    String result = inStream2String(is);
    Toast.makeText(this, result, Toast.LENGTH_LONG).show();
}
} catch (Exception e) {
    e.printStackTrace();
}
}

//将输入流转换成字符串
private String inStream2String(InputStream is) throws Exception {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    byte[] buf = new byte[1024];
    int len = -1;
    while ((len = is.read(buf)) != -1) {
        baos.write(buf, 0, len);
    }
    return new String(baos.toByteArray());
}
}

```

执行后在手机屏幕中单击 `execute` 按钮, 会显示 `GET_SUCCESS` 的提示, 如图 3-5 所示。

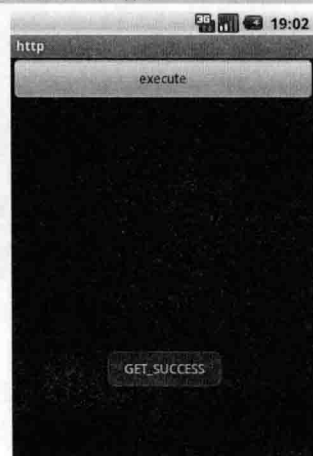


图 3-5 执行效果

3.3 使用标准的 Java 接口

在本节的内容中, 将带领广大读者漫游 `java.net` 包, 按照网络方面的知识来逐步学习 Android 中的 Java 网络编程, 并给大家介绍一些小例子, 以加深读者对各个知识点的理解。

3.3.1 IP 地址

所谓 IP 地址就是给每个连接在 Internet 上的主机分配的一个 32bit 地址。 `java.net` 中处理 IP 地址的类是 `InetAddress`, 其结构如图 3-6 所示。

如下一段代码演示了 `InetAddress` 的具体用法。

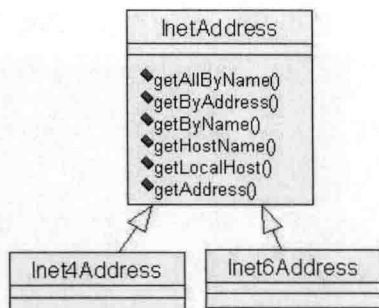


图 3-6 InetAddress 结构

```
String GetHostAddress (String strHostName)
{
    InetAddress address = null;
    try
    {
        address = InetAddress.getByName (strHostName);
    }
    catch(UnknownHostException e)
    {
        System.out.println(e.getMessage());
    }
    return InetAddress.getHostAddress () ;
}

void GetAllIP (String strHostName)
{
    InetAddress[] add = null;
    try
    {
        add = InetAddress.getAllByName (strHostName);
        for(int i=0;i<addr.lenth;i++)
        System.out.println(addr[i]);
    }
    catch(UnknownHostException e)
    {
        System.out.println(e.getMessage());
    }
}
```

上述代码非常简单，但是有一点必须说明的是，在写网络编程方面的程序时，必须注意异常的捕获，网络异常是比较正常的现象，比如当前网络繁忙，网络连接超时更是“家常便饭”。因此在写网络编程时，必须养成捕获异常情况的好习惯，查看完函数说明后，必须要注意网络异常的说明。特别注意，在使用 `getByAddress()` 函数的时候就必须捕获 `UnknownHostException` 这个异常。

3.3.2 URL 地址

在 Java 中直接提供了类 `URL` 来处理和 `URL` 相关的知识，如图 3-7 所示。

使用 `URL` 类的演示代码如下。

```
Void EasyURL (String strURL)
{
    URL url = new URL(strURL);
    try
    {
        InputStream html = url.openStream ();
        int c;
        do{
            c= html.read();
            cf(c!=-1) System.out.println((char)c);
        }
    }
}
```

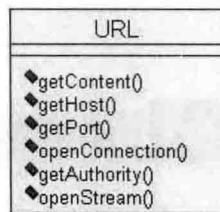


图 3-7 URL 结构

```

}while(c!=-1)
}
catch(IOException e)
{
System.out.println(e.getMessage());
}
}

```

3.3.3 套接字 Socket 类

套接字 Socket 类的基本结构如图 3-8 所示。

套接字通信的基本思想比较简单，客户端建立一个到服务器的连接，一旦连接建立了，客户端就可以往套接字里写入数据，并向服务器发送数据；反过来，服务器读取客户端写入套接字里的数据。也许细节会复杂些，但是基本思想就这么简单。使用 Socket 类的代码如下。

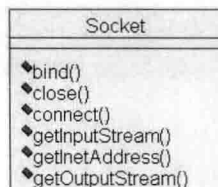


图 3-8 Socket 结构

```

void WebPing (String strURL)
{
try
{
InetAddress addr;
Socket sock = new Socket(strURL,80);
Addr = sock.getInetAddress ();
System.out.println("Conncted to"+addr);
Sock.close();
}
catch(IOException e)
{
System.out.println(e.getMessage());
}
}

```

如果使用本地主机（localhost）来测试这个程序，则输出如下结果。

```
Conncted to localhost/127.0.0.1
```

其中 `InetAddress.toString()` 的隐含调用（`println` 调用）自动输出主机名和 IP 地址。另外还有其他套接字，如 `DatagramSocket`（通过 UDP 通信的套接字）、`MulticastSocket`（一种用于多点传送的套接字）及 `ServerSocket`（一种用于监听来自客户端的连接套接字），在这里不再一一说明。

3.3.4 URLConnction 类

在一般情况下，URL 类就可以满足项目需求，但是在一些特殊情况下，比如 HTTP 数据头的传递，这时候就得使用 `URLConnection`，其结构如图 3-9 所示。

使用 `URLConnection` 后，用户对网络的控制就增加了很多，代码如下。

```

void SendRequest (String strURL)
{
URL url = URL(strURL);
HttpURLConnection conn = (HttpURLConnection)url.openConnection ();
conn.setDoInput (true);

```



```

conn.setDoOutput (true);
conn.setRequestProperty ("Content-type","application/xxx");
conn.connect ();
System.out.println(Conn.getResponseMessage ());
InputMessage is = Conn.getIputStream();
int c;
do{
c = is.read();
if(c!=-1) System.out.println((char)c);
}while(c!=-1)
}

```

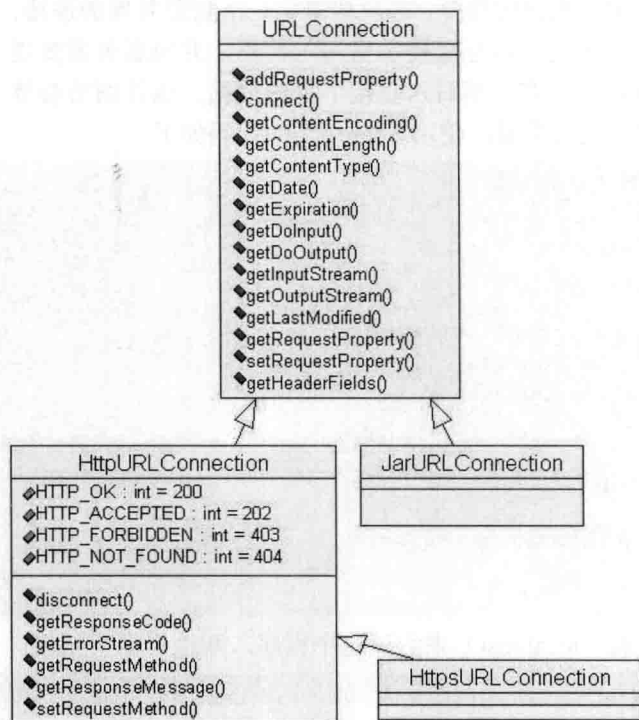


图 3-9 URLConnection 结构

3.3.5 在 Android 中使用 java.net

在接下来的内容中，将通过具体代码来演示在 Android 中使用 java.net 的基本流程。

(1) 在文件 AndroidManifest.xml 中添加 android.permission.INTERNET 许可，这样才能允许应用程序访问网络。具体代码如下。

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.net"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <application android:icon="@drawable/icon" android:label="@string/app_name">

```



```

<activity android:name=".NetworkingProject" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
</application>
</manifest>

```

(2) 编写布局文件 main.xml, 主要代码如下。

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:text="Enter URL"
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </TextView>
    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:text="http://innovator.samsungmobile.com"
        android:layout_height="wrap_content">
    </EditText>
    <Button
        android:text="Click Here"
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </Button>
    <EditText
        android:id="@+id/editText2"
        android:layout_width="match_parent"
        android:layout_height="fill_parent">
    </EditText>
</LinearLayout>

```

(3) 编写主程序文件 NetworkingProject.java, 功能也是创建一个可以查看网页 HTML 代码的 Java 程序。具体代码如下。

```

package com.net;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;

```

```

import android.widget.Button;
import android.widget.TextView;
public class NetworkingProject extends Activity {
    Button bt;
    TextView textView1;
    TextView textView2;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        bt = (Button) findViewById(R.id.button1);
        textView1 = (TextView) findViewById(R.id.editText1);
        textView2 = (TextView) findViewById(R.id.editText2);
        bt.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                textView2.setText("");
                try {
                    /*Java Networking API*/
                    URL url = new URL(textView1.getText().toString());
                    URLConnection conn = url.openConnection();
                    /*Read the Response*/
                    BufferedReader rd = new BufferedReader(new
                        InputStreamReader(conn.getInputStream()));
                    String line = "";
                    while ((line = rd.readLine()) != null) {
                        textView2.append(line);
                    }
                } catch (Exception exe) {
                    exe.printStackTrace();
                }
            }
        });
    }
}

```

执行上述代码后，可以在手机浏览器中查看输入网址网页的 HTML 代码，如图 3-10 所示。



注意

因为本书前面已经简要介绍了 java.net 的基本知识，所以在本节只是对相关知识进行了简单讲解。并且在本书后面的内容中，还将详细介绍 URL 处理的相关知识，因此本节的篇幅较短。

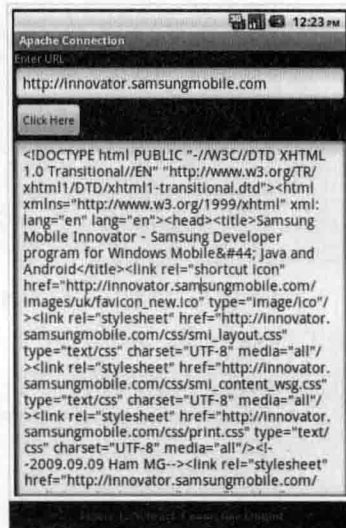


图 3-10 执行效果

3.4 使用 Android 网络接口

在 Android 平台中, 可以使用 Android 网络接口 `android.net.http` 来处理 HTTP 请求。`android.net.http` 是 `android.net` 中的一个包, 在里面主要包含处理 SSL 证书的类。在 `android.net.http` 中存在如下 4 个类。

- `AndroidHttpClient`。
- `SslCertificate`。
- `SslCertificate.DName`。
- `SslError`。

其中 `AndroidHttpClient` 就是用来处理 HTTP 请求的。

`android.net.*` 实际上是通过封装 Apache 的 `HttpClient` 来实现的一个 HTTP 编程接口, 同时还提供了 HTTP 请求队列管理, 以及 HTTP 连接池管理, 以提高并发请求情况下 (如转载网页时) 的处理效率, 除此之外还有网络状态监视等接口。

下面是一个通过 `AndroidHttpClient` 访问服务器的最简单的例子。

```
import import android.net.http.AndroidHttpClient;
try {
    AndroidHttpClient client = AndroidHttpClient.newInstance("your_user_agent");
    // 创建 HttpGet 方法, 该方法会自动处理 URL 地址的重定向
    HttpGet httpGet = new HttpGet ("http://www.test_test.com/");
    HttpResponse response = client.execute(httpGet);
    if (response.getStatusLine().getStatusCode() != HttpStatus.SC_OK) {
        // 错误处理
    }
    // 关闭连接
    client.close();
} catch (Exception ee) {
}
```

另外当用户的应用需要同时从不同的主机获取数目不等的数, 并且仅关心数据的完整性而不关心其先后顺序时, 也可以使用这部分的接口。典型用例就是 `android.webkit` 在转载网页和下载网页资源时, 具体可参考 `android.webkit.*` 中的相关类来实现。

3.5 实战演练

经过前面的学习, 了解到 HTTP 是一种网络传输协议, 现实中的大多数网页都是通过 `HTTP://WWW` 的形式实现显示的。在具体应用时, 一些需要的数据都是通过其参数传递的。在本节的内容中, 将通过具体实例来讲解在 Android 手机中使用 HTTP 的具体方法。

3.5.1 实战演练——在手机屏幕中传递 HTTP 参数

和网络 HTTP 有关的是 HTTP protocol, 在 Android SDK 中, 集成了 Apache 的 `HttpClient` 模块。通过这些模块, 可以方便地编写出和 HTTP 有关的程序。在 Android SDK 中通常使用 `HttpClient 4.0`。

题 目	目 的	源码路径
实例 3-1	在手机屏幕中传递 HTTP 参数	光盘\daima\3\httpSHI

1. 设计思路

在本实例中插入了两个按钮，一个用于以 POST 方式获取网站数据，另外一个用于以 GET 方式获取数据，并在 TextView 文本框中显示由服务器端返回网页的内容结果。当然首先得建立和 HTTP 的连接，连接之后才能获取 Web Server 返回的结果。

2. 具体实现

(1) 编写布局文件 main.xml，主要代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:background="@drawable/white"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:id="@+id/myTextView1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/title"/>
    <Button
        android:id="@+id/myButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/str_button1" />
    <Button
        android:id="@+id/myButton2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/str_button2" />
</LinearLayout>
```

(2) 编写文件 httpSHI.java，其具体实现流程如下。

① 引用 apache.http 相关类实现 HTTP 联机，然后引用 java.io 与 java.util 相关类来读写档案。具体代码如下。

```
/*引用 apache.http 相关类来建立 HTTP 联机*/
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
```

```
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.protocol.HTTP;
import org.apache.http.util.EntityUtils;
/*必须引用 java.io 与 java.util 相关类来读写档案*/
import irdc.httpSHI.R;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
```

② 使用 `OnClickListener` 来聆听单击第一个按钮事件，声明网址字符串并使用建立 `Post` 方式联机，最后通过 `mTextView1.setText` 输出提示字符。具体代码如下。

```
/*设定 OnClickListener 来聆听 onClick 事件*/
mButton1.setOnClickListener(new Button.OnClickListener()
{
    /*覆写 onClick 事件*/
    @Override
    public void onClick(View v)
    {
        /*声明网址字符串*/
        String uriAPI = "http://www.dubblogs.cc:8751/Android/Test/API/Post/index.php";
        /*建立 HTTP Post 联机*/
        HttpPost httpRequest = new HttpPost(uriAPI);
        /*
         * Post 运行传送变量必须用 NameValuePair[] 数组存储
         */
        List <NameValuePair> params = new ArrayList <NameValuePair>();
        params.add(new BasicNameValuePair("str", "I am Post String"));
        try
        {
            httpRequest.setEntity(new UrlEncodedFormEntity(params, HTTP.UTF_8));
            /*取得 HTTP 输出*/
            HttpResponse httpResponse = new DefaultHttpClient().execute(httpRequest);
            /*如果状态码为 200 */
            if(httpResponse.getStatusLine().getStatusCode() == 200)
            {
                /*获取应答字符串*/
                String strResult = EntityUtils.toString(httpResponse.getEntity());
                mTextView1.setText(strResult);
            }
            else
            {
                mTextView1.setText("Error Response: "+httpResponse.getStatusLine().
```



```

        toString());
    }
}
catch (ClientProtocolException e)
{
    mTextView1.setText(e.getMessage().toString());
    e.printStackTrace();
}
catch (IOException e)
{
    mTextView1.setText(e.getMessage().toString());
    e.printStackTrace();
}
catch (Exception e)
{
    mTextView1.setText(e.getMessage().toString());
    e.printStackTrace();
}
}
});

```

③ 使用 `OnClickListener` 来聆听单击第二个按钮的事件，声明网址字符串并建立 `Get` 方式的联机功能，分别实现发出 `HTTP` 获取请求、获取应答字符串和删除冗余字符操作，最后通过 `mTextView1.setText` 输出提示字符。具体代码如下。

```

mButton2.setOnClickListener(new Button.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        // TODO Auto-generated method stub
        /*声明网址字符串*/
        String uriAPI = "http://www.XXXX.cc:8751/index.php?str=I+am+Get+String";
        /*建立 HTTP Get 联机*/
        HttpGet httpRequest = new HttpGet(uriAPI);
        try
        {
            /*发出 HTTP 获取请求*/
            HttpResponse httpResponse = new DefaultHttpClient().execute(httpRequest);
            /*若状态码为 200 ok*/
            if (httpResponse.getStatusLine().getStatusCode() == 200)
            {
                /*获取应答字符串*/
                String strResult = EntityUtils.toString(httpResponse.getEntity());
                /*删除冗余字符*/
                strResult = eregi_replace("\\r\\n\\r\\n\\n\\r", "", strResult);
                mTextView1.setText(strResult);
            }
            else
            {

```



```

        mTextView1.setText("Error Response: "+httpResponse.getStatusLine().
            toString());
    }
}
catch (ClientProtocolException e)
{
    mTextView1.setText(e.getMessage().toString());
    e.printStackTrace();
}
catch (IOException e)
{
    mTextView1.setText(e.getMessage().toString());
    e.printStackTrace();
}
catch (Exception e)
{
    mTextView1.setText(e.getMessage().toString());
    e.printStackTrace();
}
}
});
}
.....

```

④ 定义替换字符串函数 `eregi_replace` 来替换掉一些非法字符，具体代码如下。

```

/* 字符串替换函数 */
public String eregi_replace(String strFrom, String strTo, String strTarget)
{
    String strPattern = "(?i)"+strFrom;
    Pattern p = Pattern.compile(strPattern);
    Matcher m = p.matcher(strTarget);
    if(m.find())
    {
        return strTarget.replaceAll(strFrom, strTo);
    }
    else
    {
        return strTarget;
    }
}
}

```

(3) 在文件 `AndroidManifest.xml` 中声明网络连接权限，具体代码如下。

```

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

```

执行后的效果如图 3-11 所示，单击图中的按钮能够以不同方式获取 HTTP 参数。

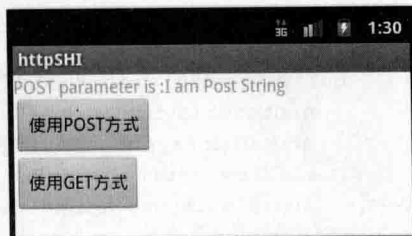


图 3-11 单击【使用 PSST 方式】按钮后的效果

3.5.2 实战演练——在 Android 手机中通过 Apache HTTP 访问 HTTP 资源

在本实例中首先创建了 `HttpGet` 和 `HttpPost` 对象，并将要请求的 URL 对象构造方法传入 `HttpGet`、`HttpPost` 对象中。然后通过 `HttpClient` 接口的实现类 `DefaultClient` 的 `execute(HttpUriRequest request)` 方法实现连接处理。因为已经知道 `HttpGet` 和 `HttpPost` 类都实现了 `HttpUriRequest` 接口，所以可以将前面创建好的 `HttpGet` 或者 `HttpPost` 对象传入以得到 `HttpResponse` 对象。最后通过 `HttpResponse` 获取返回的 HTTP 资源信息，然后再做提取工作。

题 目	目 的	源码路径
实例 3-2	通过 Apache HTTP 访问 HTTP 资源	光盘\daima\3\http

本实例的具体实现流程如下。

(1) 编写布局文件 `main.xml`，在界面中分别插入 3 个 `Button` 按钮和两个 `EditText` 控件，主要代码如下。

```
<LinearLayout android:orientation="horizontal"
    android:layout_width="fill_parent" android:layout_height="wrap_content">
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="url:" />
    <EditText android:id="@+id/urlText" android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="" />
</LinearLayout>
<LinearLayout android:orientation="horizontal"
    android:layout_width="fill_parent" android:layout_height="wrap_content"
    android:gravity="right">
    <Button android:id="@+id/getBtn" android:text="GET 请求"
        android:layout_width="wrap_content" android:layout_height="wrap_content" />
    <Button android:id="@+id/postBtn" android:text="POST 请求"
        android:layout_width="wrap_content" android:layout_height="wrap_content" />
</LinearLayout>
<TextView android:id="@+id/resultView" android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
<LinearLayout android:orientation="horizontal"
    android:layout_width="fill_parent" android:layout_height="wrap_content">
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="图片 url:" />
    <EditText android:id="@+id/imageurlText" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="" />
</LinearLayout>
<Button android:id="@+id/imgBtn" android:text="获取图片"
    android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:layout_gravity="right" />
<ImageView android:id="@+id/imgView01"
    android:layout_height="wrap_content" android:layout_width="fill_parent" />
</LinearLayout>
```

(2) 编写核心文件 `HTTPDemoActivity.java`，根据 `EditText` 控件中的输入的数据来访问远

程 HTTP 资源, 并将得到的信息转换成一个输出流并返回。在整个实现过程中需要通过 url 创建 HttpGet 对象, 并通过 DefaultClient 的 excute 方法返回一个 HttpResponse 对象。文件 HTTPDemoActivity.java 的主要实现代码如下。

```
private String request(String method, String url) {
    HttpResponse httpResponse = null;
    StringBuffer result = new StringBuffer();
    try {
        if (method.equals("GET")) {
            // 1.通过url 创建 HttpGet 对象
            HttpGet httpGet = new HttpGet(url);
            // 2.通过 DefaultClient 的 excute 方法执行返回一个 HttpResponse 对象
            HttpClient httpClient = new DefaultHttpClient();
            httpResponse = httpClient.execute(httpGet);
            // 3.取得相关信息
            // 取得 HttpEntiy
            HttpEntity httpEntity = httpResponse.getEntity();
            // 得到一些数据
            // 通过 EntityUtils 并指定编码方式取到返回的数据
            result.append(EntityUtils.toString(httpEntity, "utf-8"));
            //得到 StatusLine 接口对象
            StatusLine statusLine = httpResponse.getStatusLine();

            //得到协议
            ;
            result.append("协议:" + statusLine.getProtocolVersion() + "\r\n");
            int statusCode = statusLine.getStatusCode();

            result.append("状态码:" + statusCode + "\r\n");

        } else if (method.equals("POST")) {

            // 1.通过url 创建 HttpPost 对象
            HttpPost httpPost = new HttpPost(url);
            // 2.通过 DefaultClient 的 excute 方法执行返回一个 HttpResponse 对象
            HttpClient httpClient = new DefaultHttpClient();
            httpResponse = httpClient.execute(httpPost);
            // 3.取得相关信息
            // 取得 HttpEntiy
            HttpEntity httpEntity = httpResponse.getEntity();
            // 得到一些数据
            // 通过 EntityUtils 并指定编码方式取到返回的数据
            result.append(EntityUtils.toString(httpEntity, "utf-8"));
            StatusLine statusLine = httpResponse.getStatusLine();
            statusLine.getProtocolVersion();
            int statusCode = statusLine.getStatusCode();

            result.append("状态码:" + statusCode + "\r\n");

        }
    }
}
```

```

    } catch (Exception e) {
        Toast.makeText(HTTPDemoActivity.this, "网络连接异常", Toast.LENGTH_LONG)
            .show();
    }
    return result.toString();
}

public void getImage(String url) {
    try {
        // 1.通过url 创建HttpGet 对象
        HttpGet httpGet = new HttpGet(url);
        // 2.通过DefaultClient 的 excute 方法执行返回一个 HttpResponse 对象
        HttpClient httpClient = new DefaultHttpClient();
        HttpResponse httpResponse = httpClient.execute(httpGet);
        // 3.取得相关信息
        // 取得 HttpEntiy
        HttpEntity httpEntity = httpResponse.getEntity();
        // 4.通过 HttpEntiy.getContent 得到一个输入流
        InputStream inputStream = httpEntity.getContent();
        System.out.println(inputStream.available());

        //通过传入的流再通过 Bitmap 工厂创建一个 Bitmap
        Bitmap bitmap = BitmapFactory.decodeStream(inputStream);
        //设置 imageView
        imageView.setImageBitmap(bitmap);
    } catch (Exception e) {
        Toast.makeText(HTTPDemoActivity.this, "网络连接异常", Toast.LENGTH_LONG)
            .show();
    }
}

```

(3) 在设置文件 AndroidManifest.xml 中添加访问网络资源的权限，具体代码如下。

```

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

```

(4) 设置一个 Java 服务器环境，在其中添加服务器资源供前面的 Android 客户端来访问。将光盘中的源码的 Servers 部分复制到本地 Java 服务器的 Tomcat 中。

最终客户端的执行效果如图 3-12 所示。



图 3-12 执行效果



4

使用 Socket 实现数据通信

在现实网络传输应用中，通常使用 TCP、IP 或 UDP 这三种协议实现数据传输。在传输数据的过程中，需要通过一个双向的通信连接实现数据的交互。在这个传输过程中，通常将这个双向链路的一端称为 Socket，一个 Socket 通常由一个 IP 地址和一个端口号来确定。由此可见，在整个数据传输过程中，Socket 的作用是巨大的。在 Java 编程应用中，Socket 是 Java 网络编程的核心。因为 Java 是 Android 应用开发的主流语言，所以在本章的内容中，将详细讲解在 Android 系统中使用 Socket 实现通信的基本知识，为读者学习本书后面的知识打下基础。

4.1 Socket 编程初步



在网络编程中有两个主要的问题，一个是如何准确地定位网络上一台或多台主机，另一个就是找到主机后如何可靠高效地进行数据传输。在 TCP/IP 协议中 IP 层主要负责网络主机的定位，数据传输的路由，由 IP 地址可以唯一地确定 Internet 上的一台主机。而 TCP 层则提供面向应用的可靠（TCP）的或非可靠（UDP）的数据传输机制，这是网络编程的主要对象，一般不需要关心 IP 层是如何处理数据的。目前较为流行的网络编程模型是客户机/服务器（C/S）结构，即通信双方一方作为服务器等待客户提出请求并予以响应。客户则在需要服务时向服务器提出申请。服务器一般作为守护进程始终运行，监听网络端口，一旦有客户请求，就会启动一个服务进程来响应该客户，同时自己继续监听服务端口，使后来的客户也能及时得到服务。在接下来的内容中，将简要讲解 TCP/IP 和 UDP 协议的基本知识。

4.1.1 TCP/IP 协议基础

TCP/IP 是 Transmission Control Protocol/Internet Protocol 的简写，中文名为传输控制协议/因特网互联协议，又名网络通信协议，是 Internet 最基本的协议、Internet 国际互联网络的基础，由网络层的 IP 协议和传输层的 TCP 协议组成。TCP/IP 定义了电子设备如何连入因特网，以及数据如何在它们之间传输的标准。TCP/IP 协议采用了 4 层的层级结构，每一层都呼叫它的下一层所提供的协议来完成自己的需求。也就是说，TCP 负责发现传输的问题，一旦发现问题便发出信号要求重新传输，直到所有数据安全、正确地传输到目的地。而 IP 的功能是给因特网的每一台电脑规定一个地址。

TCP/IP 协议不是 TCP 和 IP 这两个协议的合称，而是指因特网整个 TCP/IP 协议簇。从协议分层模型方面来讲，TCP/IP 由 4 个层次组成，分别是网络接口层、网络层、传输层和应用层。

其实 TCP/IP 协议并不完全符合 OSI 的 7 层参考模型，OSI (Open System Interconnect) 是传统的开放式系统互连参考模型，是一种通信协议的 7 层抽象的参考模型，其中每一层执行某一特定任务。该模型的目的是使各种硬件在相同的层次上相互通信。这 7 层是：物理层、数据链路层（网络接口层）、网络层、传输层、会话层、表示层和应用层。而 TCP/IP 通信协议采用了 4 层的层级结构，每一层都呼叫它的下一层所提供的网络来完成自己的需求。由于 ARPANET 的设计者注重的是网络互联，允许通信子网（网络接口层）采用已有的或将来的各种协议，所以这个层次中没有提供专门的协议。实际上，TCP/IP 协议可以通过网络接口层连接到任何网络上，如 X.25 交换网或 IEEE802 局域网。

4.1.2 UDP 协议

UDP 是 User Datagram Protocol 的简称，是一种无连接的协议，每个数据报都是一个独立的信息，包括完整的源地址或目的地址，它在网络上以任何可能的路径传往目的地，因此能否到达目的地、到达目的地的时间，以及内容的正确性都是不能被保证的。

在现实网络数据传输过程中，大多数功能是由 TCP 协议和 UDP 协议实现的，在接下来的内容中，将列出上述两种协议的主要特点，以便读者可以区分这两种数据传输协议。

(1) TCP 协议

TCP 协议的主要特点如下。

- 面向连接的协议，在 Socket 之间进行数据传输之前必然要建立连接，所以在 TCP 中需要连接时间。
- TCP 传输数据有大小限制，一旦连接建立起来，双方的 Socket 就可以按统一的格式传输大的数据。
- TCP 是一个可靠的协议，它确保接收方完全正确地获取发送方所发送的全部数据。

(2) UDP 协议

UDP 协议的主要特点如下。

- 每个数据包中都给出了完整的地址信息，因此无须建立发送方和接收方的连接。
- UDP 传输数据时是有大小限制的，每个被传输的数据包必须限定在 64kB 之内。
- UDP 是一个不可靠的协议，发送方所发送的数据包并不一定以相同的次序到达接收方。

在日常应用中，可以根据如下两点来选择使用哪一种传输协议。

(1) TCP 在网络通信上有极强的生命力，如远程连接 (Telnet) 和文件传输 (FTP) 都需要不定长度的数据被可靠地传输。但是可靠地传输是要付出代价的，对数据内容正确性的检验必然占用计算机的处理时间和网络的带宽，因此 TCP 传输的效率不如 UDP 高。

(2) UDP 操作简单，而且仅需要较少的监护，因此通常用于局域网高可靠性的分散系统中 Client/Server 应用程序。例如，视频会议系统并不要求音频、视频数据绝对的正确，只要保证连贯性就可以了，这种情况下显然使用 UDP 会更合理一些。

4.1.3 基于 Socket 的 Java 网络编程

网络上的两个程序通过一个双向的通信连接实现数据的交换, 这个双向链路的一端称为一个 Socket。Socket 通常用来实现客户方和服务方的连接。Socket 是 TCP/IP 协议的一个十分流行的编程界面, 一个 Socket 由一个 IP 地址和一个端口号唯一确定。但是, Socket 所支持的协议种类也不光是 TCP/IP 一种, 因此两者之间没有必然联系。在 Java 环境下, Socket 编程主要是指基于 TCP/IP 协议的网络编程。

1. Socket 通信的过程

Server 端 Listen (监听) 某个端口是否有连接请求, Client 端向 Server 端发出 Connect (连接) 请求, Server 端向 Client 端发回 Accept (接受) 消息。一个连接就建立起来了。Server 端和 Client 端都可以通过 Send、Write 等方法与对方通信。

在 Java 网络编程应用中, 对于一个功能齐全的 Socket 来说, 其工作过程包含如下的基本步骤。

- (1) 创建 Socket。
- (2) 打开连接到 Socket 的输入/输出流。
- (3) 按照一定的协议对 Socket 进行读/写操作。
- (4) 关闭 Socket (在实际应用中, 并未用到显示的 Close, 虽然很多文章都推荐如此, 不过在笔者的程序中, 可能因为程序本身比较简单, 要求不高, 所以并未造成影响)。

2. 创建 Socket

在 Java 网络编程应用中, 在包 java.net 中提供了两个类 Socket 和 ServerSocket, 分别用来表示双向连接的客户端和服务端。这是两个封装得非常好的类, 其中包含了如下所示的构造方法。

- Socket(InetAddress address, int port)。
- Socket(InetAddress address, int port, boolean stream)。
- Socket(String host, int port)。
- Socket(String host, int port, boolean stream)。
- Socket(SocketImpl impl)。
- Socket(String host, int port, InetAddress localAddr, int localPort)。
- Socket(InetAddress address, int port, InetAddress localAddr, int localPort)。
- ServerSocket(int port)。
- ServerSocket(int port, int backlog)。
- ServerSocket(int port, int backlog, InetAddress bindAddr)。

在上述构造方法中, 参数 address、host 和 port 分别是双向连接中另一方的 IP 地址、主机名和端口号, stream 指明 socket 是流 socket 还是数据报 socket, localPort 表示本地主机的端口号, localAddr 和 bindAddr 是本地机器的地址 (ServerSocket 的主机地址), impl 是 socket 的父类, 既可以用来创建 serverSocket 又可以用来创建 Socket。count 则表示服务端所能支持的最大连接数。例如:

```
Socket client = new Socket("127.0.0.1", 80);
ServerSocket server = new ServerSocket(80);
```



注意

必须小心地选择端口，每一个端口提供一种特定的服务，只有给出正确的端口，才能获得相应的服务。0~1 023 的端口号为系统所保留，例如，http 服务的端口号为 80，telnet 服务的端口号为 21，ftp 服务的端口号为 23，所以在选择端口号时，最好选择一个大于 1 023 的数以防止发生冲突。另外，在创建 Socket 时如果发生错误，将产生 IOException，在程序中必须对其作出处理。所以在创建 Socket 或 ServerSocket 时必须捕获或抛出例外。

4.2 TCP 编程详解

TCP/IP 通信协议是一种可靠的网络协议，能够在通信的两端各建立一个 Socket，从而在通信的两端之间形成网络虚拟链路。一旦建立了虚拟的网络链路，两端的程序就可以通过虚拟链路进行通信。Java 语言对 TCP 网络通信提供了良好的封装，通过 Socket 对象代表两端的通信端口，并通过 Socket 产生的 IO 流进行网络通信。在本章的内容中，将首先详细讲解 Java 应用中 TCP 编程的基本知识，为读者学习本章后面的 Android 编程打下基础。

4.2.1 使用 ServletSocket

在 Java 程序中，使用类 ServerSocket 接受其他通信实体的连接请求。对象 ServerSocket 的功能是监听来自客户端的 Socket 连接，如果没有连接则会一直处于等待状态。在类 ServerSocket 中包含了如下监听客户端连接请求的方法。

- Socket accept(): 如果接收到一个客户端 Socket 的连接请求，该方法将返回一个与客户端 Socket 对应的 Socket，否则该方法将一直处于等待状态，线程也被阻塞。

为了创建 ServerSocket 对象，ServerSocket 类提供了如下构造器。

- ServerSocket(int port): 用指定的端口 port 创建一个 ServerSocket，该端口应该有一个有效的端口整数值：0~65 535。
- ServerSocket(int port,int backlog): 增加一个用来改变连接队列长度的参数 backlog。
- ServerSocket(int port,int backlog,InetAddress localAddr): 在机器存在多个 IP 地址的情况下，允许通过 localAddr 这个参数来指定将 ServerSocket 绑定到指定的 IP 地址。

当使用 ServerSocket 后，需要使用 ServerSocket 中的方法 close()关闭该 ServerSocket。在通常情况下，因为服务器不会只接受一个客户端请求，而是会不断地接受来自客户端的所有请求，所以可以通过循环来不断地调用 ServerSocket 中的方法 accept()。例如下面的代码。

```
//创建一个 ServerSocket，用于监听客户端 Socket 的连接请求
ServerSocket ss = new ServerSocket(30000);
//采用循环不断接受来自客户端的请求
while (true)
{
```

```
//每当接受到客户端 Socket 的请求, 服务器端也对应产生一个 Socket
Socket s = ss.accept();
//下面即可使用 Socket 进行通信
...
}
```

在上述代码中, 创建的 `ServerSocket` 没有指定 IP 地址, 该 `ServerSocket` 会绑定到本机默认的 IP 地址。在代码中使用 40 000 作为该 `ServerSocket` 的端口号, 通常推荐使用 10 000 以上的端口, 主要是为了避免与其他应用程序的通用端口冲突。

4.2.2 使用 Socket

在客户端可以使用 `Socket` 的构造器实现和指定服务器的连接, 在 `Socket` 中可以使用如下两个构造器。

- `Socket(InetAddress/String remoteAddress, int port)`: 创建连接到指定远程主机、远程端口的 `Socket`, 该构造器没有指定本地地址、本地端口, 默认使用本地主机的默认 IP 地址, 默认使用系统动态指定的 IP 地址。
- `Socket(InetAddress/String remoteAddress, int port, InetAddress localAddr, int localPort)`: 创建连接到指定远程主机、远程端口的 `Socket`, 并指定本地 IP 地址和本地端口号, 适用于本地主机有多个 IP 地址的情形。

在使用上述构造器指定远程主机时, 既可以使用 `InetAddress` 来指定, 也可以使用 `String` 对象指定, 在 Java 中通常使用 `String` 对象指定远程 IP, 如 192.168.2.23。当本地主机只有一个 IP 地址时, 建议使用第一个方法, 因为这样更简单。例如下面的代码。

```
//创建连接到本机、30000 端口的 Socket
Socket s = new Socket("127.0.0.1", 30000);
```

当程序执行上述代码后会连接到指定服务器, 让服务器端的 `ServerSocket` 的方法 `accept()` 向下执行, 于是服务器端和客户端就产生一对互相连接的 `Socket`。上述代码连接到“远程主机”的 IP 地址是 127.0.0.1, 此 IP 地址总是代表本级的 IP 地址。因为笔者示例程序的服务器端、客户端都是在本机运行, 所以 `Socket` 连接到远程主机的 IP 地址使用 127.0.0.1。

当客户端、服务器端产生对应的 `Socket` 之后, 程序无须再区分服务器端和客户端, 而是通过各自的 `Socket` 进行通信。在 `Socket` 中提供如下两个方法获取输入流和输出流。

- `InputStream getInputStream()`: 返回该 `Socket` 对象对应的输入流, 让程序通过该输入流从 `Socket` 中取出数据。
- `OutputStream getOutputStream()`: 返回该 `Socket` 对象对应的输出流, 让程序通过该输出流向 `Socket` 中输出数据。

下面是一段 TCP 协议的服务器端程序。

源码路径: 光盘:\daima\4\tcpudp\src\Server.java

```
import java.net.*;
import java.io.*;
public class Server
{
    public static void main(String[] args)
        throws IOException
```

```

{
    //创建一个 ServerSocket, 用于监听客户端 Socket 的连接请求
    ServerSocket ss = new ServerSocket(30000);
    //采用循环不断接收来自客户端的请求
    while (true)
    {
        //每当接收到客户端 Socket 的请求, 服务器端也对应产生一个 Socket
        Socket s = ss.accept();
        //将 Socket 对应的输出流包装成 PrintStream
        PrintStream ps = new PrintStream(s.getOutputStream());
        //进行普通 IO 操作
        ps.println("圣诞快乐!");
        //关闭输出流, 关闭 Socket
        ps.close();
        s.close();
    }
}
}

```

通过上述代码建立了 `ServerSocket` 监听, 并且使用 `Socket` 获取了输出流, 所以执行后不会显示任何信息。

而下面是一段 TCP 协议的客户端程序。

源码路径: 光盘\daima\4\tcpudp\src\Client.java

```

import java.net.*;
import java.io.*;
public class Client
{
    public static void main(String[] args)
        throws IOException
    {
        Socket socket = new Socket("127.0.0.1", 30000);
        //将 Socket 对应的输入流包装成 BufferedReader
        BufferedReader br = new BufferedReader(
            new InputStreamReader(socket.getInputStream()));
        //进行普通 IO 操作
        String line = br.readLine();
        System.out.println("来自服务器的数据: " + line);
        //关闭输入流、socket
        br.close();
        socket.close();
    }
}

```

上述代码使用 `Socket` 建立了与指定 IP、指定端口的连接, 并使用 `Socket` 获取输入流读取数据。执行后的效果如图 4-1 所示。

由此可见, 一旦使用 `ServerSocket` 和 `Socket` 建立网络连接之后, 程序通过网络通信与普通 IO 并没有太大的区别。如果先运行上面程序中的 `Server` 类, 将看到服务器一直处于等待状态, 因为服务器使用了死循环来接受来自客户端的请求; 再运行 `Client` 类, 将可看到程序输出: “来自服务器的数据: 圣诞快乐!”



图 4-1 执行效果

这表明客户端和服务端通信成功。上述代码为了突出通过 `ServerSocket` 和 `Socket` 建立连接、并通过底层 IO 流进行通信的主题，程序没有进行异常处理，也没有使用 `finally` 块来关闭资源。

4.2.3 TCP 中的多线程

在本章 4.2.2 的实例中，`Server` 和 `Client` 只是进行了简单的通信操作，当服务器接收到客户端连接之后，服务器向客户端输出一个字符串，而客户端也只是读取服务器的字符串后就退出了。在实际应用中，客户端可能需要和服务端保持长时间通信，即服务器需要不断地读取客户端数据，并向客户端写入数据，客户端也需要不断地读取服务器数据，并向服务器写入数据。

当使用 `readLine()` 方法读取数据时，如果在该方法成功返回之前线程被阻塞，则程序无法继续执行。所以此服务器很有必要为每个 `Socket` 单独启动一条线程，每条线程负责与一个客户端进行通信。另外，因为客户端读取服务器数据的线程同样会被阻塞，所以系统应该单独启动一条线程，该线程专门负责读取服务器数据。

假设要开发一个聊天室程序，在服务器端应该包含多条线程，其中每个 `Socket` 对应一条线程，该线程负责读取 `Socket` 对应输入流的数据（从客户端发送过来的数据），并将读到的数据向每个 `Socket` 输出流发送一遍（将一个客户端发送的数据“广播”给其他客户端），因此需要在服务器端使用 `List` 来保存所有的 `Socket`。在具体实现时，为服务器提供了如下两个类。

- 创建 `ServerSocket` 监听的主类。
- 处理每个 `Socket` 通信的线程类。

接下来介绍具体实现流程，首先看下面的一段代码。

源码路径：光盘\ldaima\4\tcpudp\src\liao\server\IServer.java

```
package liao.server;
import java.net.*;
import java.io.*;
import java.util.*;

public class IServer
{
    //定义保存所有 Socket 的 ArrayList
    public static ArrayList<Socket> socketList = new ArrayList<Socket>();
    public static void main(String[] args)
        throws IOException
    {
        ServerSocket ss = new ServerSocket(30000);
        while(true)
        {
            //此行代码会阻塞，将一直等待别人的连接
            Socket s = ss.accept();
            socketList.add(s);
            //每当客户端连接后启动一条 ServerThread 线程为该客户端服务
            new Thread(new Serverxian(s)).start();
        }
    }
}
```



```

    }
}

```

在上述代码中，服务器端只负责接受客户端 Socket 的连接请求，每当客户端 Socket 连接到该 ServerSocket 之后，程序将对应 Socket 加入 socketList 集合中保存，并为该 Socket 启动一条线程，该线程负责处理该 Socket 所有的通信任务。

然后看服务器端线程类文件的主要代码。

源码路径：光盘\daima\4\tcpudp\src\liao\server\Serverxian.java

```

//负责处理每个线程通信的线程类
public class Serverxian implements Runnable
{
    //定义当前线程所处理的 Socket
    Socket s = null;
    //该线程所处理的 Socket 所对应的输入流
    BufferedReader br = null;
    public Serverxian(Socket s)
        throws IOException
    {
        this.s = s;
        //初始化该 Socket 对应的输入流
        br = new BufferedReader(new InputStreamReader(s.getInputStream()));
    }
    public void run()
    {
        try
        {
            String content = null;
            //采用循环不断从 Socket 中读取客户端发送过来的数据
            while ((content = readFromClient()) != null)
            {
                //遍历 socketList 中的每个 Socket，将读到的内容向每个 Socket 发送一次
                for (Socket s : IServer.socketList)
                {
                    PrintStream ps = new PrintStream(s.getOutputStream());
                    ps.println(content);
                }
            }
        }
        catch (IOException e)
        {
            //e.printStackTrace();
        }
    }
    //定义读取客户端数据的方法
    private String readFromClient()
    {
        try
        {
            return br.readLine();

```



```

    }
    //如果捕捉到异常,表明该 Socket 对应的客户端已经关闭
    catch (IOException e)
    {
        //删除该 Socket
        IServer.socketList.remove(s);
    }
    return null;
}
}

```

在上述代码中,服务器端线程类会不断读取客户端数据,在获取时使用方法 `readFromClient()` 来读取客户端数据。如果读取数据过程中捕获到 `IOException` 异常,则说明此 `Socket` 对应的客户端 `Socket` 出现了问题,程序就会将此 `Socket` 从 `socketList` 中删除。当服务器线程读到客户端数据之后会遍历整个 `socketList` 集合,并将该数据向 `socketList` 集合中的每个 `Socket` 发送一次,该服务器线程将把从 `Socket` 中读到的数据向 `socketList` 中的每个 `Socket` 转发一次。

接下来开始客户端的编码工作,在实例的每个客户端应该包含如下 2 条线程。

- 第一条: 功能是读取用户的键盘输入,并将用户输入的数据写入 `Socket` 对应的输出流中。
- 第二条: 功能是读取 `Socket` 对应输入流中的数据(从服务器发送过来的数据),并将这些数据打印输出。其中负责读取用户键盘输入的线程由 `Myclient` 负责,也就是由程序的主线程负责。

客户端主程序文件的主要代码如下。

源码路径: 光盘\daima\4\tcpudp\src\liao\server\Iclient.java

```

public class IClient
{
    public static void main(String[] args)
        throws IOException
    {
        Socket s = s = new Socket("127.0.0.1", 30000);
        //客户端启动 ClientThread 线程不断读取来自服务器的数据
        new Thread(new ClientThread(s)).start();
        //获取该 Socket 对应的输出流
        PrintStream ps = new PrintStream(s.getOutputStream());
        String line = null;
        //不断读取键盘输入
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        while ((line = br.readLine()) != null)
        {
            //将用户的键盘输入内容写入 Socket 对应的输出流
            ps.println(line);
        }
    }
}

```

在上述代码中,当线程读到用户键盘输入的内容后,会将用户键盘输入的内容写入该 `Socket` 对应的输出流。当主线程使用 `Socket` 连接到服务器之后,会启动 `ClientThread` 来处理

该线程的 Socket 通信。

最后编写客户端的线程处理文件，此线程负责读取 Socket 输入流中的内容，并将这些内容在控制台打印出来。具体代码（光盘\daima\4\tcpudp\src\liao\server\Clientxian.java）如下。

```
public class Clientxian implements Runnable
{
    //该线程负责处理的 Socket
    private Sockets;
    //该线程所处理的 Socket 所对应的输入流
    BufferedReader br = null;
    public Clientxian(Socket s)
        throws IOException
    {
        this.s = s;
        br = new BufferedReader(
            new InputStreamReader(s.getInputStream()));
    }
    public void run()
    {
        try
        {
            String content = null;
            //不断读取 Socket 输入流中的内容，并将这些内容打印输出
            while ((content = br.readLine()) != null)
            {
                System.out.println(content);
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

上述代码能够不断获取 Socket 输入流中的内容，当获取 Socket 输入流中的内容后，直接将这些内容打印在控制台。先运行上面程序中的类 IServer，该类运行后作为本实例的服务器，不会看到任何输出。接着可以运行多个 IClient——相当于启动多个聊天室客户端登录该服务器，此时可以看到在任何一个客户端通过键盘输入一些内容后单击“回车”键，将可看到所有客户端（包括自己）都会在控制台收到他刚刚输入的内容，这就简单实现了一个聊天室的功能。

4.2.4 实现非阻塞 Socket 通信

在 Java 应用程序中，可以使用 NIO API 来开发高性能网络服务器。当程序执行输入、输出操作后，在这些操作返回之前会一直阻塞该线程，服务器必须为每个客户端都提供一条独立线程进行处理。这说明前面的程序是基于阻塞式 API 的，当服务器需要同时处理大量客户端时，这种做法会降低性能。

在 Java 应用程序中可以用 NIO API 让服务器使用一个或有限几个线程来同时处理连接到服务器上的所有客户端。在 Java 的 NIO 中, 为非阻塞式的 Socket 通信提供了下面的特殊类。

- **Selector:** `SelectableChannel` 对象的多路复用器, 所有希望采用非阻塞方式进行通信的 `Channel` 都应该注册到 `Selector` 对象。可通过调用此类的静态 `open()` 方法来创建 `Selector` 实例, 该方法将使用系统默认的 `Selector` 来返回新的 `Selector`。`Selector` 可以同时监控多个 `SelectableChannel` 的 IO 状况, 是非阻塞 IO 的核心。一个 `Selector` 实例有如下 3 个 `SelectionKey` 的集合。
 - 所有 `SelectionKey` 集合: 代表了注册在该 `Selector` 上的 `Channel`, 这个集合可以通过 `keys()` 方法返回。
 - 被选择的 `SelectionKey` 集合: 代表了所有可通过 `select()` 方法监测到、需要进行 IO 处理的 `Channel`, 这个集合可以通过 `selectedKeys()` 返回。
 - 被取消的 `SelectionKey` 集合: 代表了所有被取消注册关系的 `Channel`, 在下次执行 `select()` 方法时, 这些 `Channel` 对应的 `SelectionKey` 会被彻底删除, 程序通常无须直接访问该集合。

除此之外, `Selector` 还提供了如下和 `select()` 相关的方法。

- `int select()`: 监控所有注册的 `Channel`, 当它们中间有需要处理的 IO 操作时, 该方法返回, 并将对应的 `SelectionKey` 加入被选择的 `SelectionKey` 集合中, 该方法返回这些 `Channel` 的数量。
- `int select(long timeout)`: 可以设置超时时长的 `select()` 操作。
- `int selectNow()`: 执行一个立即返回的 `select()` 操作, 相对于无参数的 `select()` 方法而言, 该方法不会阻塞线程。
- `Selector wakeup()`: 使一个还未返回的 `select()` 方法立刻返回。
- `SelectableChannel`: 它代表可以支持非阻塞 IO 操作的 `Channel` 对象, 可以将其注册到 `Selector` 上, 这种注册的关系由 `SelectionKey` 实例表示。在 `Selector` 对象中, 可以使用 `select()` 方法设置允许应用程序同时监控多个 IO `Channel`。Java 程序可调用 `SelectableChannel` 中的 `register()` 方法将其注册到指定 `Selector` 上, 当该 `Selector` 上某些 `SelectableChannel` 上有需要处理的 IO 操作时, 程序可以调用 `Selector` 实例的 `select()` 方法获取它们的数量, 并通过 `selectedKeys()` 方法返回它们对应的 `SelectionKey` 集合。这个集合的作用巨大, 因为通过该集合就可以获取所有需要处理 IO 操作的 `SelectableChannel` 集。

对象 `SelectableChannel` 支持阻塞和非阻塞两种模式, 其中所有 `Channel` 默认都是阻塞模式, 必须使用非阻塞式模式才可以利用非阻塞 IO 操作。

在 `SelectableChannel` 中提供了如下两个方法来设置和返回该 `Channel` 的模式状态。

- `SelectableChannel configureBlocking(boolean block)`: 设置是否采用阻塞模式。
- `boolean isBlocking()`: 返回该 `Channel` 是否是阻塞模式。

不同的 `SelectableChannel` 所支持的操作不一样, 例如 `ServerSocketChannel` 代表一个 `ServerSocket`, 它就只支持 `OP_ACCEPT` 操作。在 `SelectableChannel` 中提供了如下方法来返回它支持的所有操作。

`int validOps()`: 返回一个 bit mask, 表示这个 Channel 上支持的 IO 操作。

除此之外, `SelectableChannel` 还提供了如下方法获取它的注册状态。

- `boolean isRegistered()`: 返回该 Channel 是否已注册在一个或多个 `Selector` 上。
- `SelectionKey keyFor(Selector sel)`: 返回该 Channel 和 `sel` `Selector` 之间的注册关系, 如果不存在注册关系, 则返回 `null`。
- `SelectionKey`: 该对象代表 `SelectableChannel` 和 `Selector` 之间的注册关系。
- `ServerSocketChannel`: 支持非阻塞操作, 对应于 `java.net.ServerSocket` 这个类, 提供了 TCP 协议 IO 接口, 只支持 `OP_ACCEPT` 操作。该类也提供了 `accept()` 方法, 功能相当于 `ServerSocket` 提供的 `accept()` 方法。
- `SocketChannel`: 支持非阻塞操作, 对应于 `java.net.Socket` 这个类, 提供了 TCP 协议 IO 接口, 支持 `OP_CONNECT`、`OP_READ` 和 `OP_WRITE` 操作。这个类还实现了 `ByteChannel` 接口、`ScatteringByteChannel` 接口和 `GatheringByteChannel` 接口, 所以可以直接通过 `SocketChannel` 来读写 `ByteBuffer` 对象。

服务器上所有 Channel 都需要向 `Selector` 注册, 包括 `ServerSocketChannel` 和 `SocketChannel`。该 `Selector` 则负责监视这些 Socket 的 IO 状态, 当其中任意一个或多个 Channel 具有可用的 IO 操作时, 该 `Selector` 的 `select()` 方法将会返回大于 0 的整数, 该整数值就表示该 `Selector` 上有多少个 Channel 具有可用的 IO 操作, 并提供了 `selectedKeys()` 方法来返回这些 Channel 对应的 `SelectionKey` 集合。正是通过 `Selector` 才使得服务器端只需不断地调用 `Selector` 实例的 `select()` 方法, 这样就可以知道当前所有 Channel 是否有需要处理的 IO 操作。当 `Selector` 上注册的所有 Channel 都没有需要处理的 IO 操作时, 将会阻塞 `select()` 方法, 此时调用该方法的线程被阻塞。

继续以聊天室为例, 讲解非阻塞 Socket 通信在 Java 应用项目中的实现过程。我们的目标是, 在服务器端使用循环不断获取 `Selector` 的 `select()` 方法返回值, 当该返回值大于 0 时就处理该 `Selector` 上被选择的 `SelectionKey` 所对应的 Channel。在具体实现时, 服务器端使用 `ServerSocketChannel` 来监听客户端的连接请求, 程序先调用它的 `socket()` 方法获得关联 `ServerSocket` 对象, 再用该 `ServerSocket` 对象的绑定值来指定需要监听对象的 IP 地址和具体端口。最后在服务器端调用 `Selector` 的 `select()` 方法来监听所有 Channel 上的 IO 操作。

接下来开始具体编码, 其中服务器端的主要代码如下。

源码路径: 光盘:\daima\4\tcpudp\src\feizu\feizuServer.java

```
public class feizuServer
{
    //用于检测所有 Channel 状态的 Selector
    private Selector selector = null;
    //定义实现编码、解码的字符集对象
    private Charset charset = Charset.forName("UTF-8");
    public void init() throws IOException
    {
        selector = Selector.open();
        //通过 open 方法来打开一个未绑定的 ServerSocketChannel 实例
        ServerSocketChannel server = ServerSocketChannel.open();
        InetSocketAddress isa = new InetSocketAddress(
```



```
"127.0.0.1", 30000);
//将该 ServerSocketChannel 绑定到指定 IP 地址
server.socket().bind(isa);
//设置 ServerSocket 以非阻塞方式工作
server.configureBlocking(false);
//将 server 注册到指定 Selector 对象
server.register(selector, SelectionKey.OP_ACCEPT);
while (selector.select() > 0)
{
    //依次处理 selector 上的每个已选择的 SelectionKey
    for (SelectionKey sk : selector.selectedKeys())
    {
        //从 selector 上的已选择 Key 集中删除正在处理的 SelectionKey
        selector.selectedKeys().remove(sk);
        //如果 sk 对应的通道包含客户端的连接请求
        if (sk.isAcceptable())
        {
            //调用 accept 方法接受连接, 产生服务器端对应的 SocketChannel
            SocketChannel sc = server.accept();
            //设置采用非阻塞模式
            sc.configureBlocking(false);
            //将该 SocketChannel 也注册到 selector
            sc.register(selector, SelectionKey.OP_READ);
            //将 sk 对应的 Channel 设置成准备接受其他请求
            sk.interestOps(SelectionKey.OP_ACCEPT);
        }
        //如果 sk 对应的通道有数据需要读取
        if (sk.isReadable())
        {
            //获取该 SelectionKey 对应的 Channel, 该 Channel 中有可读的数据
            SocketChannel sc = (SocketChannel)sk.channel();
            //定义准备执行读取数据的 ByteBuffer
            ByteBuffer buff = ByteBuffer.allocate(1024);
            String content = "";
            //开始读取数据
            try
            {
                while(sc.read(buff) > 0)
                {
                    buff.flip();
                    content += charset.decode(buff);
                }
                //打印从该 sk 对应的 Channel 里读取到的数据
                System.out.println("====" + content);
                //将 sk 对应的 Channel 设置成准备下一次读取
                sk.interestOps(SelectionKey.OP_READ);
            }
            //如果捕捉到该 sk 对应的 Channel 出现了异常, 即表明该 Channel
            //对应的 Client 出现了问题, 所以从 Selector 中取消 sk 的注册
            catch (IOException ex)
```

```

        {
            //从 Selector 中删除指定的 SelectionKey
            sk.cancel();
            if (sk.channel() != null)
            {
                sk.channel().close();
            }
        }
        //如果 content 的长度大于 0，即聊天信息不为空
        if (content.length() > 0)
        {
            //遍历该 selector 里注册的所有 SelectKey
            for (SelectionKey key : selector.keys())
            {
                //获取该 key 对应的 Channel
                Channel targetChannel = key.channel();
                //如果该 channel 是 SocketChannel 对象
                if (targetChannel instanceof SocketChannel)
                {
                    //将读到的内容写入该 Channel 中
                    SocketChannel dest = (SocketChannel)targetChannel;
                    dest.write(charset.encode(content));
                }
            }
        }
    }
}

public static void main(String[] args)
    throws IOException
{
    new feizuServer().init();
}
}

```

通过上述代码，在启动时马上建立一个可监听连接请求的 `ServerSocketChannel`，并将该 `Channel` 注册到指定的 `Selector`，接着程序直接采用循环不断监控 `Selector` 对象的 `select()` 方法返回值，当该返回值大于 0 时处理该 `Selector` 上所有被选择的 `SelectionKey`。在处理指定 `SelectionKey` 之后立即从该 `Selector` 中的被选择的 `SelectionKey` 集合中删除该 `SelectionKey`。服务器端的 `Selector` 仅需要监听连接和读数据这两种操作，在处理连接操作时只需将接受连接后产生的 `SocketChannel` 注册到指定 `Selector` 对象即可。当处理读数据操作后，系统先从该 `Socket` 中读取数据，再将数据写入 `Selector` 上注册的所有 `Channel`。

接下来开始编写客户端的代码，本应用的客户端程序需要如下两个线程。

- 负责读取用户的键盘输入，并将输入的内容写入 `SocketChannel` 中。
- 不断查询 `Selector` 对象的 `select()` 方法的返回值。

客户端的主要代码如下。

源码路径：光盘:\daima\4\tcpudp\src\feizu\feizuClient.java

```
public class feizuClient{
    //定义检测 SocketChannel 的 Selector 对象
    private Selector selector = null;
    //定义处理编码和解码的字符集
    private Charset charset = Charset.forName("UTF-8");
    //客户端 SocketChannel
    private SocketChannel sc = null;
    public void init()throws IOException
    {
        selector = Selector.open();
        InetSocketAddress isa = new InetSocketAddress("127.0.0.1", 30000);
        //调用 open 静态方法创建连接到指定主机的 SocketChannel
        sc = SocketChannel.open(isa);
        //设置该 sc 以非阻塞方式工作
        sc.configureBlocking(false);
        //将 SocketChannel 对象注册到指定 Selector
        sc.register(selector, SelectionKey.OP_READ);
        //启动读取服务器端数据的线程
        new ClientThread().start();
        //创建键盘输入流
        Scanner scan = new Scanner(System.in);
        while (scan.hasNextLine())
        {
            //读取键盘输入
            String line = scan.nextLine();
            //将键盘输入的内容输出到 SocketChannel 中
            sc.write(charset.encode(line));
        }
    }
    //定义读取服务器数据的线程
    private class ClientThread extends Thread
    {
        public void run()
        {
            try
            {
                while (selector.select() > 0)
                {
                    //遍历每个有可用 IO 操作 Channel 对应的 SelectionKey
                    for (SelectionKey sk : selector.selectedKeys())
                    {
                        //删除正在处理的 SelectionKey
                        selector.selectedKeys().remove(sk);
                        //如果该 SelectionKey 对应的 Channel 中有可读的数据
                        if (sk.isReadable())
                        {
                            //使用 NIO 读取 Channel 中的数据
                        }
                    }
                }
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

```

        SocketChannel sc = (SocketChannel)sk.channel();
        ByteBuffer buff = ByteBuffer.allocate(1024);
        String content = "";
        while(sc.read(buff) > 0)
        {
            sc.read(buff);
            buff.flip();
            content += charset.decode(buff);
        }
        //打印输出读取的内容
        System.out.println("聊天信息: " + content);
        //为下一次读取作准备
        sk.register(sk.channel(), SelectionKey.OP_READ);
    }
}

catch (IOException ex)
{
    ex.printStackTrace();
}

}

public static void main(String[] args)
    throws IOException
{
    new feizuClient().init();
}
}

```

上述客户端代码只有一条 `SocketChannel`，当此 `SocketChannel` 注册到指定的 `Selector` 后，程序会启动另一条线程来监测该 `Selector`。

在使用 `NIO` 来实现服务器时，甚至无须使用 `ArrayList` 来保存服务器中所有的 `SocketChannel`，因为所有的 `SocketChannel` 都需要注册到指定的 `Selector` 对象。除此之外，当客户端关闭时会导致服务器对应的 `Channel` 也抛出异常，而且本程序只有一条线程，如果该异常得不到处理将会导致整个服务器退出，所以程序捕捉了这种异常，并在处理异常时从 `Selector` 删除异常 `Channel` 的注册。

4.3 UDP 编程

Java 提供了 `DatagramSocket` 对象作为基于 `UDP` 协议的 `Socket`，可以使用 `DatagramPacket` 代表 `DatagramSocket` 发送或接收的数据报。

4.3.1 使用 `DatagramSocket`

`DatagramSocket` 本身只是码头，不维护状态，不能产生 `IO` 流，其唯一的功能是接收和发送数据报。Java 语言使用 `DatagramPacket` 代表数据报，`DatagramSocket` 的接收和发送数据

功能都是通过 DatagramPacket 对象实现的。

在 DatagramSocket 中有如下 3 个构造器。

- DatagramSocket(): 负责创建一个 DatagramSocket 实例, 并将该对象绑定到本机默认 IP 地址、本机所有可用端口中随机选择的某个端口。
- DatagramSocket(int prot): 负责创建一个 DatagramSocket 实例, 并将该对象绑定到本机默认 IP 地址、指定端口。
- DatagramSocket(int port, InetAddress laddr): 负责创建一个 DatagramSocket 实例, 并将该对象绑定到指定 IP 地址、指定端口。

在 Java 程序中, 通过上述任意一个构造器即可创建一个 DatagramSocket 实例。在创建服务器时必须创建指定端口的 DatagramSocket 实例, 目的是保证其他客户端可以将数据发送到该服务器。一旦得到了 DatagramSocket 实例, 就可以通过下面的两个方法接收和发送数据。

- receive(DatagramPacket p): 从该 DatagramSocket 中接收数据报。
- send(DatagramPacket p): 以该 DatagramSocket 对象向外发送数据报。

在使用 DatagramSocket 发送数据报时, DatagramSocket 并不知道将该数据报发送到哪里, 而是由 DatagramPacket 自身决定数据报的目的地。就像码头并不知道每个集装箱的目的地, 码头只是将这些集装箱发送出去, 而集装箱本身包含了该集装箱的目的地。

当 Client/Server 程序使用 UDP 协议时, 实际上并没有明显的服务器和客户端, 因为两方都需要先建立一个 DatagramSocket 对象, 用来接收或发送数据报, 然后使用 DatagramPacket 对象作为传输数据的载体。通常固定 IP、固定端口的 DatagramSocket 对象所在的程序被称为服务器, 因为该 DatagramSocket 可以主动接收客户端数据。

在 DatagramPacket 中包含了如下常用的构造器。

- DatagramPacket(byte buf[], int length): 以一个空数组来创建 DatagramPacket 对象, 该对象的作用是接收 DatagramSocket 中的数据。
- DatagramPacket(byte buf[], int length, InetAddress addr, int port): 以一个包含数据的数组来创建 DatagramPacket 对象, 创建该 DatagramPacket 时还指定了 IP 地址和端口——这就决定了该数据报的目的。
- DatagramPacket(byte[] buf, int offset, int length): 以一个空数组来创建 DatagramPacket 对象, 并指定接收到的数据放入 buf 数组中时从 offset 开始, 最多放 length 个字节。
- DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port): 创建一个用于发送的 DatagramPacket 对象, 也多指定了一个 offset 参数。

在接收数据前, 应该采用上面的第一个或第三个构造器生成一个 DatagramPacket 对象, 给出接收数据的字节数组及其长度。然后调用 DatagramSocket 中的 receive() 方法等待数据报的到来, 此方法将一直等待 (也就是说会阻塞调用该方法的线程), 直到收到一个数据报为止。例如下面的代码。

```
//创建接收数据的 DatagramPacket 对象
DatagramPacket packet=new DatagramPacket(buf, 256);
//接收数据
socket.receive(packet);
```

在发送数据之前, 调用第二个或第四个构造器创建 DatagramPacket 对象, 此时的字节数

组里存放了想发送的数据。除此之外，还要给出完整的目的地址，包括 IP 地址和端口号。发送数据是通过 `DatagramSocket` 的方法 `send()` 实现的，方法 `send()` 根据数据报的目的地址来寻径以传递数据报。例如下面的代码。

```
//创建一个发送数据的 DatagramPacket 对象
DatagramPacket packet = new DatagramPacket(buf, length, address, port);
//发送数据报
socket.send(packet);
```

接着 `DatagramPacket` 提供了方法 `getData()`，此方法可以返回 `DatagramPacket` 对象中封装的字节数组。

当服务器（也可以是客户端）接收到一个 `DatagramPacket` 对象后，如果想向该数据报的发送者“反馈”一些信息，但由于 UDP 是面向非连接的，所以接收者并不知道每个数据报由谁发送过来，但程序可以调用 `DatagramPacket` 的如下 3 个方法来获取发送者的 IP 和端口信息。

- `InetAddress getAddress()`：返回某台机器的 IP 地址，当程序准备发送此数据报时，该方法返回此数据报的目标机器的 IP 地址；当程序刚刚接收到一个数据报时，该方法返回该数据报的发送主机的 IP 地址。
- `int getPort()`：返回某台机器的端口，当程序准备发送此数据报时，该方法返回此数据报的目标机器的端口；当程序刚刚接收到一个数据报时，该方法返回该数据报的发送主机的端口。
- `SocketAddress getSocketAddress()`：返回完整的 `SocketAddress`，通常由 IP 地址和端口组成。当程序准备发送此数据报时，该方法返回此数据报的目标 `SocketAddress`；当程序刚刚接收到一个数据报时，该方法的返回结果是数据报源地址 `SocketAddress`。

上述 `getSocketAddress` 方法的返回值是一个 `SocketAddress` 对象，该对象实际上就是一个 IP 地址和一个端口号，也就是说 `SocketAddress` 对象封装了一个 `InetAddress` 对象和一个代表端口的整数，所以使用 `SocketAddress` 对象可以同时代表 IP 地址和端口。

下面是一段实现 UDP 协议的服务器端代码。

源码路径：光盘\daima\4\tcpudp\src\UdpServer.java

```
public class UdpServer
{
    public static final int PORT = 30000;
    //定义每个数据报的大小最大为 4K
    private static final int DATA_LEN = 4096;
    //定义该服务器使用的 DatagramSocket
    private DatagramSocket socket = null;
    //定义接收网络数据的字节数组
    byte[] inBuff = new byte[DATA_LEN];
    //以指定字节数组创建准备接收数据的 DatagramPacket 对象
    private DatagramPacket inPacket =
        new DatagramPacket(inBuff, inBuff.length);
    //定义一个用于发送的 DatagramPacket 对象
    private DatagramPacket outPacket;
    //定义一个字符串数组，服务器发送该数组的元素
    String[] books = new String[]
```

```
{
    "AAA",
    "BBB",
    "CCC",
    "DDD"
};
public void init() throws IOException
{
    try
    {
        //创建 DatagramSocket 对象
        socket = new DatagramSocket(PORT);
        //采用循环接收数据
        for (int i = 0; i < 1000 ; i++ )
        {
            //读取 Socket 中的数据, 读到的数据放在 inPacket 所封装的字节数组中
            socket.receive(inPacket);
            //判断 inPacket.getData() 和 inBuff 是否是同一个数组
            System.out.println(inBuff == inPacket.getData());
            //将接收到的内容转成字符串后输出
            System.out.println(new String(inBuff ,
                0 , inPacket.getLength()));
            //从字符串数组中取出一个元素作为发送的数据
            byte[] sendData = books[i % 4].getBytes();
            //以指定字节数组作为发送数据、以刚接收到的 DatagramPacket 的
            //源 SocketAddress 作为目标 SocketAddress 创建 DatagramPacket.
            outPacket = new DatagramPacket(sendData ,
                sendData.length , inPacket.getSocketAddress());
            //发送数据
            socket.send(outPacket);
        }
    }
    //使用 finally 块保证关闭资源
    finally
    {
        if (socket != null)
        {
            socket.close();
        }
    }
}
public static void main(String[] args)
    throws IOException
{
    new UdpServer().init();
}
```

上述代码使用 DatagramSocket 实现了 Server/Client 结构的网络通信程序, 其中服务器端使用循环 1 000 次来读取 DatagramSocket 中的数据报, 每当读到内容之后便向该数据报的发

送者送回一条信息。

接下来看客户端的实现代码，客户端代码与服务器端类似，也是采用循环不断地读取用户键盘输入，每当读到用户输入内容后就将该内容封装成 `DatagramPacket` 数据报，再将该数据报发送出去。然后把 `DatagramSocket` 中的数据读入接收用的 `DatagramPacket` 中（实际上是读入该 `DatagramPacket` 所封装的字节数组中）。下面是一段实现 UDP 协议的客户端代码。

源码路径：光盘\daima\4\tcpudp\src\UdpClient.java

```
public class UdpClient{
    //定义发送数据报的目的地
    public static final int DEST_PORT = 30000;
    public static final String DEST_IP = "127.0.0.1";
    //定义每个数据报的大小最大为 4K
    private static final int DATA_LEN = 4096;
    //定义该客户端使用的 DatagramSocket
    private DatagramSocket socket = null;
    //定义接收网络数据的字节数组
    byte[] inBuff = new byte[DATA_LEN];
    //以指定字节数组创建准备接收数据的 DatagramPacket 对象
    private DatagramPacket inPacket =
        new DatagramPacket(inBuff , inBuff.length);
    //定义一个用于发送的 DatagramPacket 对象
    private DatagramPacket outPacket = null;
    public void init()throws IOException{
        try
        {
            //创建一个客户端 DatagramSocket，使用随机端口
            socket = new DatagramSocket();
            //初始化发送用的 DatagramSocket，它包含一个长度为 0 的字节数组
            outPacket = new DatagramPacket(new byte[0] , 0 ,
                InetAddress.getByName(DEST_IP) , DEST_PORT);
            //创建键盘输入流
            Scanner scan = new Scanner(System.in);
            //不断读取键盘输入
            while(scan.hasNextLine())
            {
                //将键盘输入的一行字符串转换字节数组
                byte[] buff = scan.nextLine().getBytes();
                //设置发送用的 DatagramPacket 里的字节数据
                outPacket.setData(buff);
                //发送数据报
                socket.send(outPacket);
                //读取 Socket 中的数据，读到的数据放在 inPacket 所封装的字节数组中
                socket.receive(inPacket);
                System.out.println(new String(inBuff , 0 ,
                    inPacket.getLength()));
            }
        }
        //使用 finally 块保证关闭资源
        finally
```



```
{
    if (socket != null)
    {
        socket.close();
    }
}

public static void main(String[] args)
    throws IOException
{
    new UdpClient().init();
}
}
```

上述代码通过 `DatagramSocket` 实现了发送并接收 `DatagramPacket` 的功能，具体实现与服务器的实现代码基本相似。而客户端与服务器端的唯一区别是服务器所在 IP 地址和端口是固定的，所以客户端可以直接将该数据报发送给服务器，而服务器需要根据接收到的数据报决定将“反馈”数据报的目的地。

4.3.2 使用 MulticastSocket

`DatagramSocket` 只允许将数据报发送给指定的目标地址，而 `MulticastSocket` 可以将数据报以广播的方式发送到数量不等的多个客户端。如果要使用多点广播，需要让一个数据报标有一组目标主机地址，当发出数据报后，整个组的所有主机都能收到该数据报。IP 多点广播（或多点发送）实现可以将单一信息发送到多个接收者，功能是设置一组特殊网络地址作为多点广播地址，每一个多点广播地址都被看作一个组，当客户端需要发送、接收广播信息时，只需加入到该组即可。

IP 协议为多点广播提供了这批特殊的 IP 地址，这些 IP 地址的范围是 224.0.0.0 至 234.255.255.255。

类 `MulticastSocket` 既可以将数据报发送到多点广播地址，也可以接收其他主机的广播信息。类 `MulticastSocket` 是 `DatagramSocket` 类的一个子类，当要发送一个数据报时，可使用随机端口创建 `MulticastSocket`，也可以在指定端口来创建 `MulticastSocket`。

在类 `MulticastSocket` 中提供了如下 3 个构造器。

- `public MulticastSocket()`: 使用本机默认地址、随机端口来创建一个 `MulticastSocket` 对象。
- `public MulticastSocket(int portNumber)`: 使用本机默认地址、指定端口来创建一个 `MulticastSocket` 对象。
- `public MulticastSocket(SocketAddress bindaddr)`: 使用本机指定 IP 地址、指定端口来创建一个 `MulticastSocket` 对象。

在创建一个 `MulticastSocket` 对象后，需要将该 `MulticastSocket` 加入到指定的多点广播地址。在 `MulticastSocket` 中使用方法 `joinGroup()` 加入到一个指定的组，使用方法 `leaveGroup()` 从一个组中脱离出去。这两个方法的具体说明如下。

- `joinGroup(InetAddress multicastAddr)`: 将该 `MulticastSocket` 加入指定的多点广播地址。

- `leaveGroup(InetAddress multicastAddr)`: 让该 `MulticastSocket` 离开指定的多点广播地址。

在某些系统中可能有多个网络接口，这可能会对多点广播带来问题，此时程序需要在一个指定的网络接口上监听，通过调用 `setInterface` 可选择 `MulticastSocket` 所使用的网络接口，也可以使用 `getInterface` 方法查询 `MulticastSocket` 监听的网络接口。

如果创建只发送数据报的 `MulticastSocket` 对象，只需使用默认地址和随机端口即可。如果创建接收用的 `MulticastSocket` 对象，则该 `MulticastSocket` 对象必须具有指定端口，否则发送方无法确定发送数据报的目标端口。

虽然 `MulticastSocket` 实现发送/接收数据报的方法与 `DatagramSocket` 的完全一样，但是 `MulticastSocket` 比 `DatagramSocket` 多了下面的方法。

```
setTimeToLive(int ttl)
```

参数 `ttl` 设置数据报最多可以跨过多少个网络，具体说明如下。

- 为 0 时：指定数据报应停留在本地主机。
- 为 1 时：指定数据报发送到本地局域网。
- 为 32 时：只能发送到本站点的网络上。
- 为 64 时：数据报应保留在本地区。
- 为 128 时：数据报应保留在本大洲。
- 为 255 时：数据报可发送到所有地方。
- 为 1 时：是默认值。

在使用 `MulticastSocket` 实现多点广播时，所有通信实体都是平等的，都将自己的数据报发送到多点广播 IP 地址，并使用 `MulticastSocket` 接收其他人发送的广播数据报。例如，在下面的代码中，使用 `MulticastSocket` 实现了一个基于广播的多人聊天室，程序只需要一个 `MulticastSocket`，两条线程，其中 `MulticastSocket` 既用于发送，也用于接收，其中一条线程分别负责接受用户键盘输入，并向 `MulticastSocket` 发送数据，另一条线程则负责从 `MulticastSocket` 中读取数据。

源码路径（光盘：\daima\4\tcpudp\src\manySocket.java）：

```
import java.awt.*;
import java.net.*;
import java.io.*;
import java.util.*;

//让该类实现 Runnable 接口，该类的实例可作为线程的 target
public class manySocket implements Runnable
{
    //使用常量作为本程序的多点广播 IP 地址
    private static final String IP
        = "230.0.0.1";
    //使用常量作为本程序的多点广播目的的端口
    public static final int PORT = 30000;
    //定义每个数据报的大小最大为 4K
    private static final int LEN = 2048;
```

```
//定义本程序的 MulticastSocket 实例
private MulticastSocket socket = null;
private InetAddress bAddress = null;
private Scanner scan = null;
//定义接收网络数据的字节数组
byte[] inBuff = new byte[LEN];
//以指定字节数组创建准备接收数据的 DatagramPacket 对象
private DatagramPacket inPacket =
    new DatagramPacket(inBuff , inBuff.length);
//定义一个用于发送的 DatagramPacket 对象
private DatagramPacket oPacket = null;
public void init()throws IOException
{
    try
    {
        //创建用于发送、接收数据的 MulticastSocket 对象
        //因为该 MulticastSocket 对象需要接收，所以有指定端口
        socket = new MulticastSocket(PORT);
        bAddress = InetAddress.getByName(IP);
        //将该 socket 加入指定的多点广播地址
        socket.joinGroup(bAddress);
        //设置本 MulticastSocket 发送的数据报被回送到自身
        socket.setLoopbackMode(false);
        //初始化发送用的 DatagramSocket，它包含一个长度为 0 的字节数组
        oPacket = new DatagramPacket(new byte[0] , 0 ,
            bAddress , PORT);
        //启动以本实例的 run() 方法作为线程体的线程
        new Thread(this).start();
        //创建键盘输入流
        scan = new Scanner(System.in);
        //不断读取键盘输入
        while(scan.hasNextLine())
        {
            //将键盘输入的一行字符串转换字节数组
            byte[] buff = scan.nextLine().getBytes();
            //设置发送用的 DatagramPacket 里的字节数据
            oPacket.setData(buff);
            //发送数据报
            socket.send(oPacket);
        }
    }
    finally
    {
        socket.close();
    }
}

public void run()
{
    try
```

```

    {
        while(true)
        {
            //读取 Socket 中的数据, 读到的数据放在 inPacket 所封装的字节数组里
            socket.receive(inPacket);
            //打印输出从 socket 中读取的内容
            System.out.println("聊天信息: " + new String(inBuff , 0 ,
                inPacket.getLength()));
        }
    }
    //捕捉异常
    catch (IOException ex)
    {
        ex.printStackTrace();
        try
        {
            if (socket != null)
            {
                //让该 Socket 离开该多点 IP 广播地址
                socket.leaveGroup(bAddress);
                //关闭该 Socket 对象
                socket.close();
            }
            System.exit(1);
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}

public static void main(String[] args)
    throws IOException
{
    new manySocket().init();
}
}

```

上述代码的实现流程如下。

- 在方法 init()中创建一个 MulticastSocket 对象, 因为需要使用该对象接收数据报, 所以为此 Socket 对象设置使用固定端口。
- 将该 Socket 对象添加到指定的多点广播 IP 地址。
- 设置该 Socket 发送的数据报会被回送到自身, 即该 Socket 可以接收到自己发送的数据报。
- 使用 MulticastSocket 发送并接收数据报的代码, 与使用 DatagramSocket 实现的方法并没有区别。

4.4 实战演练——在 Android 中使用 Socket 实现数据传输

通过本章前面内容的学习，读者已经了解了 Java 应用中 Socket 网络编程的基本知识。在 Android 平台中，可以使用相同的方法用 Socket 实现数据传输功能。在本节的内容中，将通过一个具体实例的实现过程，来讲解在 Android 中使用 Socket 实现数据传输的基本方法。

题 目	目 的	源码路径
实例 4-1	使用 Socket 实现数据传输	光盘\daima\4\socket

本实例的具体实现流程如下。

(1) 首先实现服务器端，使用 Eclipse 新建一个名为 android_server 的 Java 工程，然后编写服务器端的实现文件 AndroidServer.java，功能是创建 Socket 对象 client 以接受客户端请求，并创建 BufferedReader 对象 in 向服务器发送消息。文件 AndroidServer.java 的具体实现代码如下。

```
public class AndroidServer implements Runnable{
    public void run() {
        try {
            ServerSocket serverSocket=new ServerSocket(54321);
            while(true)
            {
                System.out.println("等待接收用户连接: ");
                //接受客户端请求
                Socket client=serverSocket.accept();
                try
                {
                    //接受客户端信息
                    BufferedReader in=new BufferedReader(new InputStreamReader
                        (client.getInputStream()));
                    String str=in.readLine();
                    System.out.println("read: "+str);
                    //向服务器发送消息
                    PrintWriter out=new PrintWriter(new BufferedWriter(new
                        OutputStreamWriter(client.getOutputStream())),true);
                    out.println("return "+str);
                    in.close();
                    out.close();
                }catch(Exception ex)
                {
                    System.out.println(ex.getMessage());
                    ex.printStackTrace();
                }
            }
            finally
            {
                client.close();
                System.out.println("close");
            }
        }
    }
}
```



```

        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
    public static void main(String [] args)
    {
        Thread desktopServerThread=new Thread(new AndroidServer());
        desktopServerThread.start();
    }
}

```

(2) 开始实现客户端的测试程序，使用 Eclipse 新建一个名称为 testSocket 的 Android 工程，编写布局文件 main.xml，在主界面中插入一个信息输入文本框和一个“发送”按钮。文件 main.xml 的具体实现代码如下。

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <EditText android:id="@+id/edit" android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
    <Button android:id="@+id/but1" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="发送" />
    <TextView android:id="@+id/text1" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="@string/hello" />
</LinearLayout>

```

(3) 编写测试文件 TestSocket.java，功能是获取输入框的文本信息，并将信息发送到“192.168.2.113”。文件 TestSocket.java 的具体实现代码如下。

```

//客户端的实现
public class TestSocket extends Activity {
    private TextView text1;
    private Button but1;
    private EditText edit1;
    private final String DEBUG_TAG="mySocketAct";

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        text1=(TextView) findViewById(R.id.text1);
        but1=(Button) findViewById(R.id.but1);
        edit1=(EditText) findViewById(R.id.edit);

        but1.setOnClickListener(new Button.OnClickListener()
        {
            @Override
            public void onClick(View v) {
                Socket socket=null;
                String mesg=edit1.getText().toString()+"\r\n";
                edit1.setText("");
            }
        });
    }
}

```

```

Log.e("dddd", "sent id");

try {
    socket=new Socket("192.168.2.113",54321);
    //向服务器发送信息
    PrintWriter out=new PrintWriter(new BufferedWriter(new
        OutputStreamWriter(socket.getOutputStream()),true);
    out.println(mesg);

    //接受服务器的信息
    BufferedReader br=new BufferedReader(new InputStreamReader
        (socket.getInputStream()));
    String mstr=br.readLine();
    if(mstr!=null)
    {
        text1.setText(mstr);
    }else
    {
        text1.setText("数据错误");
    }
    out.close();
    br.close();
    socket.close();
} catch (UnknownHostException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} catch (Exception e)
{
    Log.e(DEBUG_TAG,e.toString());
}

});
}
}

```

(4) 在文件 AndroidManifest.xml 中添加访问网络的权限，具体代码如下。

```
<!-- 添加可以通信协议 -->
```

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

至此，整个实例介绍完毕，执行后的效果如图 4-2 所示。

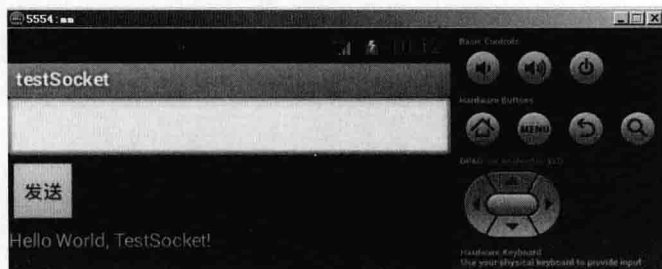


图 4-2 执行效果

下载远程数据

下载是指通过网络进行传输文件，将互联网或其他电子计算机中的信息保存到本地电脑中的一种网络活动。下载可以显式或隐式地进行，只要是获得本地电脑中所没有的信息的活动，都可以认为是下载。在 Android 网络开发应用中，下载功能是十分常见的一个应用。在本书的内容中，将详细讲解在 Android 手机中实现远程下载数据的基本知识，为读者学习本书后面的知识打下基础。

5.1 下载网络中的图片数据

在 Android 系统应用中，获取网络中的图片工作是一件耗时的操作，如果直接获取有可能会弹出一个应用程序无响应（ANR:Application Not Responding）对话框。对于这种情况，一般的方法就是使用线程来实现比较耗时操作。在 Android 网络应用中，有如下 3 种获取网络图片的方法。

（1）直接获取，演示代码如下。

```
mImageView = (ImageView)this.findViewById(R.id.imageThreadConcept) ;
Drawable drawable = loadImageFromNetwork(IMAGE_URL);
mImageView.setImageDrawable(drawable) ;
```

对应的公用方法的实现代码如下。

```
private Drawable loadImageFromNetwork(String imageUrl)
{
    Drawable drawable = null;
    try {
        // 可以在这里通过文件名来判断，是否本地有此图片
        drawable = Drawable.createFromStream(
            new URL(imageUrl).openStream(), "image.jpg");
    } catch (IOException e) {
        Log.d("test", e.getMessage());
    }
    if (drawable == null) {
        Log.d("test", "null drawable");
    } else {
        Log.d("test", "not null drawable");
    }
}
```

```
        return drawable ;  
    }  
}
```

(2) 后台线程获取 url 图片，演示代码如下。

```
mImageView = (ImageView)this.findViewById(R.id.imageThreadConcept) ;  
new Thread(new Runnable(){  
    Drawable drawable = loadImageFromNetwork(IMAGE_URL);  
    @Override  
    public void run() {  
  
        // post() 用于在 UI 主线程中更新图片  
        mImageView.post(new Runnable(){  
            @Override  
            public void run() {  
                // TODO Auto-generated method stub  
                mImageView.setImageDrawable(drawable) ;  
            }  
        }) ;  
    }  
}).start() ;
```

(3) AsyncTask 获取 url 图片，演示代码如下。

```
mImageView = (ImageView)this.findViewById(R.id.imageThreadConcept) ;  
new DownloadImageTask().execute(IMAGE_URL) ;  
private class DownloadImageTask extends AsyncTask<String, Void, Drawable>  
{  
  
    protected Drawable doInBackground(String... urls) {  
        return loadImageFromNetwork(urls[0]);  
    }  
  
    protected void onPostExecute(Drawable result) {  
        mImageView.setImageDrawable(result);  
    }  
}
```

在接下来的内容中，将通过一个具体实例的实现过程，来讲解在 Android 手机中下载远程网络图片的方法。

题 目	目 的	源码路径
实例 5-1	在 Android 手机中下载网络中的图片	光盘\codes\5\GetAPicture

本实例的具体实现流程如下。

(1) 在布局文件 main.xml 中设置一个网址文本框，主要代码如下。

```
<EditText  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="http://xxxx.jpg"  
    android:id="@+id/path"  
>
```

在上述代码中，<http://xxxx.jpg> 是网络中一副图片的地址。

(2) 编写主程序文件 `GetAPictureFromInternetActivity.java`，主要实现代码如下。

```
public class GetAPictureFromInternetActivity extends Activity {
    private EditText pathText;
    private ImageView imageView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        pathText = (EditText) this.findViewById(R.id.path);
        imageView = (ImageView) this.findViewById(R.id.imageView);
    }

    public void showimage(View v){
        String path = pathText.getText().toString();
        try {
            Bitmap bitmap = ImageService.getImage(path);
            imageView.setImageBitmap(bitmap);
        } catch (Exception e) {
            e.printStackTrace();
            Toast.makeText(getApplicationContext(), R.string.error, 1).show();
        }
    }
}
```

执行后的效果如图 5-1 所示。

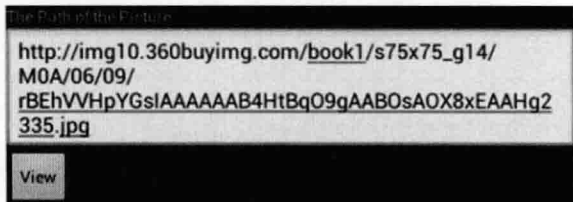


图 5-1 执行效果

5.2 下载网络中的 JSON 数据

JSON 是 JavaScript Object Notation 的缩写，是一种轻量级的数据交换格式。JSON 是基于 JavaScript (Standard ECMA-262 3rd Edition - December 1999) 的一个子集，采用完全独立于语言的文本格式，但是也使用了类似 C 语言家族的习惯(包括 C、C++、C#、Java、JavaScript、Perl、Python 等)。这些特性使 JSON 成为理想的数据交换语言。易于阅读和编写，同时也易于机器解析和生成。在本节的内容中，将详细讲解在 Android 系统中下载获取 JSON 数据的基本知识。

5.2.1 JSON 基础

简单来说,JSON 是 JavaScript 中的对象和数组,所以其结构就是对象和数组两种结构,通过这两种结构可以表示各种复杂的结构。

(1) 对象

对象在 js 中表示为“{}”括起来的内容,数据结构为 {key: value,key: value,...}的键值对的结构,在面向对象的语言中, key 为对象的属性, value 为对应的属性值,所以很容易理解,取值方法为对象.key 获取属性值,这个属性值的类型可以是数字、字符串、数组、对象几种。

(2) 数组

数组在 js 中为“[]”括起来的内容,数据结构为 ["java","javascript","vb",...], 取值方式和所有语言中的一样,使用索引获取,字段值的类型可以是数字、字符串、数组、对象几种。

通过对象、数组这两种结构就可以组合成复杂的数据结构。

和 XML 一样,JSON 也是基于纯文本的数据格式。由于 JSON 天生是为 JavaScript 准备的,因此,JSON 的数据格式非常简单,可以用 JSON 传输一个简单的 String、Number、Boolean,也可以传输一个数组,或者一个复杂的 Object 对象。

用 JSON 表示 String、Number 和 Boolean 的方法非常简单,例如,用 JSON 表示一个简单的 String 数据“abc”,其表示格式为:

```
"abc"
```

除了字符“\”、“/”和一些控制符(\b, \f, \n, \r, \t)需要编码外,其他 Unicode 字符可以直接输出。图 5-2 所示为一个 String 的完整表示结构。

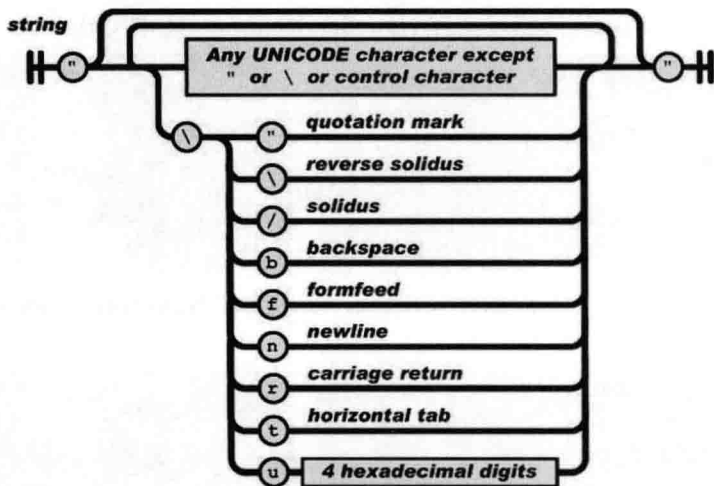


图 5-2 String 的完整表示结构

5.2.2 实战演练——远程下载服务器中的 JSON 数据

在接下来的内容中,将通过一个具体实例的实现过程,来详细讲解在 Android 系统中远程下载服务器中的 JSON 数据的方法。

题 目	目 的	源码路径
实例 5-2	在手机屏幕中显示 QQ 空间中的照片	光盘:\codes\5\json

本实例的具体实现流程如下。

(1) 使用 Eclipse 新建一个 JavaEE 工程作为服务器端，设置工程名为 ServerForJSON。自动生成工程文件后，打开文件 web.xml 进行配置，配置后的代码如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>ServerForJSON</display-name>
    <servlet>
        <display-name>NewsListServlet</display-name>
        <servlet-name>NewsListServlet</servlet-name>
        <servlet-class>com.guan.server.xml.NewsListServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>NewsListServlet</servlet-name>
        <url-pattern>/NewsListServlet</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

(2) 编写业务接口 Bean 的实现文件 NewsService.java，具体代码如下。

```
public interface NewsService {
    /**
     * 获取最新的视频资讯
     * @return
     */
    public List<News> getLastNews();
}
```

设置业务 Bean 的名称为 NewsServiceBean，实现文件 NewsServiceBean.java 的具体代码如下。

```
package com.guan.server.service.implement;

import java.util.ArrayList;
import java.util.List;

import com.guan.server.domain.News;
import com.guan.server.service.NewsService;

public class NewsServiceBean implements NewsService {
    /**
```

```

    * 获取最新的视频资讯
    * @return
    */
    public List<News> getLastNews(){
        List<News> newes = new ArrayList<News>();
        newes.add(new News(10, "aaa", 20));
        newes.add(new News(45, "bbb", 10));
        newes.add(new News(89, "Android is good", 50));
        return newes;
    }
}

```

(3) 创建一个名为 News 的实现类，实现文件 News.java 的具体代码如下。

```

package com.guan.server.domain;
public class News {
    private Integer id;
    private String title;
    private Integer timelength;
    public News(Integer id, String title, Integer timelength) {
        this.id = id;
        this.title = title;
        this.timelength = timelength;
    }
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public Integer getTimelength() {
        return timelength;
    }
    public void setTimelength(Integer timelength) {
        this.timelength = timelength;
    }
}

```

(4) 编写文件 NewsListServlet，具体实现代码如下。

```

public class NewsListServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private NewsService newsService = new NewsServiceBean();
    protected void doGet(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
        doPost(request, response);
    }
}

```

```
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    List<News> neues = newsService.getLastNews();//获取最新的视频资讯
    // [{id:20,title:"xxx",timelength:90},{id:10,title:"xbx",timelength:20}]
    StringBuilder json = new StringBuilder();
    json.append('[');
    for(News news : neues){
        json.append('{');
        json.append("id:").append(news.getId()).append(",");
        json.append("title:\"").append(news.getTitle()).append("\",");
        json.append("timelength:").append(news.getTimelength());
        json.append("},");
    }
    json.deleteCharAt(json.length() - 1);
    json.append(']');
    request.setAttribute("json", json.toString());
    request.getRequestDispatcher("/WEB-INF/page/jsonnewslist.jsp").
    forward(request, response);
}
}
```

(5) 新建一个 JSP 文件 jsonnewslist.jsp, 在里面引入 JSON 功能, 具体实现代码如下。

```
<%@ page language="java" contentType="text/plain; charset=UTF-8" pageEncoding
="UTF-8"%>${json}
```

(6) 使用 Eclipse 新建一个名称为 GetNewsInJSONFromInternet 的 Android 工程文件, 在文件 AndroidManifest.xml 申明对网络权限的应用, 具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.guan.internet.json"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name="com.guan.internet.json.MainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="8" />
    <!-- 访问 internet 权限 -->
    <uses-permission android:name="android.permission.INTERNET"/>
</manifest>
```

(7) 编写主界面布局文件 mian.xml, 具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```

        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
    >
    <ListView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/listView"
    />
</LinearLayout>

```

在上述代码中，通过 ListView 控件列表显示获取的 JSON 数据。其中 ListView 的 Item 显示的数据为 item.xml，具体实现代码如下。

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">

    <TextView
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:id="@+id/title"
    />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/timelength"
    />
</LinearLayout>

```

(8) 编写文件 MainActivity.java，功能是获取 JSON 数据并显示数据，具体实现代码如下。

```

public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        ListView listView = (ListView) this.findViewById(R.id.listView);

        String length = this.getResources().getString(R.string.length);
        try {
            List<News> newes = NewsService.getJSONLastNews();
            List<HashMap<String, Object>> data = new ArrayList<HashMap<String, Object>>();
            for(News news : newes){
                HashMap<String, Object> item = new HashMap<String, Object>();
                item.put("id", news.getId());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```



```

        item.put("title", news.getTitle());
        item.put("timelength", length+ news.getTimelength());
        data.add(item);
    }
    SimpleAdapter adapter = new SimpleAdapter(this, data, R.layout.item,
        new String[]{"title", "timelength"}, new int[]{R.id.title,
            R.id.timelength});
    listView.setAdapter(adapter);
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

(9) 编写文件 NewsService.java, 定义方法 getJSONLastNews()请求前面搭建的 JavaEE 服务器, 当获取 JSON 输入流后解析 JSON 的数据, 并返回集合中的数据。文件 NewsService.java 的具体实现代码如下。

```

public class NewsService {
    /**
     * 获取最新视频资讯
     * @return
     * @throws Exception
     */
    public static List<News> getJSONLastNews() throws Exception{
        String path = "http://192.168.1.100:8080/ServerForJSON/NewsListServlet";
        HttpURLConnection conn = (HttpURLConnection) new URL(path).openConnection();
        conn.setConnectTimeout(5000);
        conn.setRequestMethod("GET");
        if(conn.getResponseCode() == 200){
            InputStream json = conn.getInputStream();
            return parseJSON(json);
        }
        return null;
    }
    private static List<News> parseJSON(InputStream jsonStream) throws Exception{
        List<News> list = new ArrayList<News>();
        byte[] data = StreamTool.read(jsonStream);
        String json = new String(data);
        JSONArray jsonArray = new JSONArray(json);
        for(int i = 0; i < jsonArray.length(); i++){
            JSONObject jsonObject = jsonArray.getJSONObject(i);
            int id = jsonObject.getInt("id");
            String title = jsonObject.getString("title");
            int timelength = jsonObject.getInt("timelength");
            list.add(new News(id, title, timelength));
        }
        return list;
    }
}

```

至此, 整个实例介绍完毕, 执行后将成功获取服务器端 JSON 的数据。

5.3 下载某个网页的源码

当在 Android 系统中加载非本地的 HTML 文件时,会对此 HTML 进行缓存操作,同时会建一个数据库,用以保存 url 地址对应的缓存文件名称、打开时间等信息。可以将 WebView 加载的 url 地址作为查询条件获取对应的缓存文件名称,匹配出的缓存文件就是完整的 HTML 代码。

题 目	目 的	源码路径
实例 5-3	在手机屏幕上显示 QQ 空间中的照片	光盘:\codes\5\WebCodeViewer

本实例的具体实现流程如下。

(1) 编写界面布局文件 main.xml, 具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/path"
    />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="http://www.baidu.com"
        android:id="@+id/path"
    />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button"
        android:onClick="showhtml"
    />
    <ScrollView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
    >
        <TextView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:id="@+id/textView"
        />
    </ScrollView>
</LinearLayout>
```

(2) 编写文件 HtmlService.java, 定义一个获取网页代码的业务类 HtmlService, 主要代码如下。

```

public class HtmlService {
    /**
     * 获取网页源码
     * @param path 网页路径
     * @return
     */
    public static String getHtml(String path) throws Exception {
        HttpURLConnection conn = (HttpURLConnection)new
        URL(path).openConnection();
        conn.setConnectTimeout(5000);
        conn.setRequestMethod("GET");
        if(conn.getResponseCode() == 200){
            InputStream inStream = conn.getInputStream();
            byte[] data = StreamTool.read(inStream);
            return new String(data);
        }
        return null;
    }
}

```

(3) 编写文件 StreamTool.java，功能是将流转换为字节数组，主要代码如下。

```

public static byte[] read(InputStream inStream) throws Exception{
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    byte[] buffer = new byte[1024];
    int len = 0;
    while( (len = inStream.read(buffer)) != -1){
        outputStream.write(buffer, 0, len);
    }
    inStream.close();
    return outputStream.toByteArray();
}

```

(4) 编写文件 WebCodeViewerActivity.java，当单击屏幕中的 GetWebCode 按钮时会获取指定网页的源码。文件 WebCodeViewerActivity.java 的具体实现代码如下。

```

public class WebCodeViewerActivity extends Activity {
    private EditText pathText;
    private TextView textView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        pathText = (EditText) this.findViewById(R.id.path);
        textView = (TextView) this.findViewById(R.id.textView);
    }
    public void showhtml(View v) {
        String path = pathText.getText().toString();
        try {
            String html = HtmlService.getHtml(path);
            textView.setText(html);
        } catch (Exception e) {

```

```
e.printStackTrace();
Toast.makeText(getApplicationContext(), R.string.error, Toast.LENGTH_
LONG).show();
}
}
```

执行后的效果如图 5-3 所示。



图 5-3 执行效果

5.4 远程获取多媒体文件

在 Android 日常应用中，经常会涉及和多媒体有关的网络应用，如下载手机铃声和下载远程音乐文件等。在本节的内容中，将详细讲解在 Android 系统中远程获取多媒体文件的基本知识。

5.4.1 实战演练——下载并播放网络中的 MP3

为了节约手机的存储空间，在听音乐时可以用从网络中下载的方式播放 MP3。在本实例中，首先插入 4 个按钮，分别用于播放、暂停、重新播放和停止处理。执行后，通过 Runnable 发起运行线程，在线程中远程下载指定的 MP3 文件，是通过网络传输方式下载的。下载完毕后，临时保存到 SD 卡中，这样可以通过 4 个按钮对其进行控制。当关闭程序后，会自动删除 SD 卡中的临时性文件。

题 目	目 的	源码路径
实例 5-4	播放网络中的 MP3	光盘:\codes\5\mp

本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，在里面插入 4 个图片按钮，主要代码如下。

```
<TextView
    android:id="@+id/myTextView1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="@drawable/blue"
    android:text="@string/hello"
/>
<LinearLayout
    android:orientation="horizontal"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
```

```

        android:padding="10dip"
    >
    <ImageButton android:id="@+id/play"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:src="@drawable/play"
    />
    <ImageButton android:id="@+id/pause"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:src="@drawable/pause"
    />
    <ImageButton android:id="@+id/reset"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:src="@drawable/reset"
    />
    <ImageButton android:id="@+id/stop"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:src="@drawable/stop"
    />
</LinearLayout>

```

(2) 编写主程序文件 `mp.java`，其具体实现流程如下。

① 定义 `currentFilePath` 用于记录当前正在播放 MP3 的 URL 地址，定义 `currentTempFilePath` 表示当前播放 MP3 的路径。具体代码如下。

```

/*记录当前正在播放 MP3 的地址 URL*/
private String currentFilePath = "";
/*当前播放 MP3 的路径*/
private String currentTempFilePath = "";
private String strVideoURL = "";

```

② 使用 `strVideoURL` 设置要播放 mp3 文件的网址，并设置透明度。具体代码如下。

```

public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    /* mp3 文件不会被下载到 local */
    strVideoURL = "http://www.lrn.cn/zywh/xyxy/yyxs/200805/W020080505536315331315.mp3";
    mTextView01 = (TextView) findViewById(R.id.myTextView1);
    /*设置透明度*/
    getWindow().setFormat(PixelFormat.TRANSPARENT);
    mPlay = (ImageButton) findViewById(R.id.play);
    mReset = (ImageButton) findViewById(R.id.reset);
    mPause = (ImageButton) findViewById(R.id.pause);
    mStop = (ImageButton) findViewById(R.id.stop);
    .....
}

```

③ 编写单击【播放】按钮所触发的处理事件，具体代码如下。


```
/* 播放按钮 */
mPlay.setOnClickListener(new ImageButton.OnClickListener()
{
    public void onClick(View view)
    {
        /* 调用播放影片 Function */
        playVideo(strVideoURL);
        mTextView01.setText
        (
            getResources().getText(R.string.str_play).toString()+
            "\n"+ strVideoURL
        );
    }
});
```

④ 编写单击【重播】按钮所触发的处理事件，具体代码如下。

```
/* 重新播放 */
mReset.setOnClickListener(new ImageButton.OnClickListener()
{
    public void onClick(View view)
    {
        if(bIsReleased == false)
        {
            if (mMediaPlayer01 != null)
            {
                mMediaPlayer01.seekTo(0);
                mTextView01.setText(R.string.str_play);
            }
        }
    }
});
```

⑤ 编写单击【暂停】按钮所触发的处理事件，具体代码如下。

```
/* 暂停播放 */
mPause.setOnClickListener(new ImageButton.OnClickListener()
{
    public void onClick(View view)
    {
        if (mMediaPlayer01 != null)
        {
            if(bIsReleased == false)
            {
                if(bIsPaused==false)
                {
                    mMediaPlayer01.pause();
                    bIsPaused = true;
                    mTextView01.setText(R.string.str_pause);
                }
                else if(bIsPaused==true)
                {
                    mMediaPlayer01.start();
                }
            }
        }
    }
});
```

```

        bIsPaused = false;
        mTextView01.setText(R.string.str_play);
    }
}
});

```

⑥ 编写单击【停止】按钮所触发的处理事件，具体代码如下。

```

/*停止*/
mStop.setOnClickListener(new ImageButton.OnClickListener()
{
    public void onClick(View view)
    {
        try
        {
            if (mMediaPlayer01 != null)
            {
                if(bIsReleased==false)
                {
                    mMediaPlayer01.seekTo(0);
                    mMediaPlayer01.pause();
                    //mMediaPlayer01.stop();
                    //mMediaPlayer01.release();
                    //bIsReleased = true;
                    mTextView01.setText(R.string.str_stop);
                }
            }
        }
        catch(Exception e)
        {
            mTextView01.setText(e.toString());
            Log.e(TAG, e.toString());
            e.printStackTrace();
        }
    }
});
.....

```

⑦ 定义方法 `playVideo(final String strPath)` 来播放指定的 MP3，其播放的是存储卡中暂时保存的 MP3 文件，具体代码如下。

```

private void playVideo(final String strPath)
{
    try
    {
        if (strPath.equals(currentFilePath)&& mMediaPlayer01 != null)
        {
            mMediaPlayer01.start();
            return;
        }
    }
}

```

```

    }
    currentFilePath = strPath;
    mMediaPlayer01 = new MediaPlayer();
    mMediaPlayer01.setAudioStreamType(2);
    .....

```

⑧ 编写 `setOnErrorListener` 来监听错误处理，具体代码如下。

```

/*错误事件 */
mMediaPlayer01.setOnErrorListener(new MediaPlayer.OnErrorListener()
{
    @Override
    public boolean onError(MediaPlayer mp, int what, int extra)
    {
        //TODO Auto-generated method stub
        Log.i(TAG, "Error on Listener, what: " + what + "extra: " + extra);
        return false;
    }
});

```

⑨ 编写 `setOnBufferingUpdateListener` 来监听 `MediaPlayer` 缓冲区的更新，具体代码如下。

```

/* 捕捉使用 MediaPlayer 缓冲区的更新事件 */
mMediaPlayer01.setOnBufferingUpdateListener(new MediaPlayer.
OnBufferingUpdateListener()
{
    @Override
    public void onBufferingUpdate(MediaPlayer mp, int percent)
    {
        //TODO Auto-generated method stub
        Log.i(TAG, "Update buffer: " + Integer.toString(percent)+ "%");
    }
});

```

⑩ 编写 `setOnCompletionListener` 来监听播放完毕所触发的事件，具体代码如下。

```

/* 播放完毕所触发的事件 */
mMediaPlayer01.setOnCompletionListener(new MediaPlayer.OnCompletionListener()
{
    @Override
    public void onCompletion(MediaPlayer mp)
    {
        //TODO Auto-generated method stub
        //delFile(currentTempFilePath);
        Log.i(TAG, "mMediaPlayer01 Listener Completed");
    }
});

```

⑪ 编写 `setOnPreparedListener` 来监听开始阶段的事件，具体代码如下。

```

/* 开始阶段的监听 Listener */
mMediaPlayer01.setOnPreparedListener(new MediaPlayer.OnPreparedListener()
{
    @Override
    public void onPrepared(MediaPlayer mp)
    {

```

```
//TODO Auto-generated method stub
Log.i(TAG, "Prepared Listener");
}
});
```

⑫ 将文件存到 SD 卡后，通过方法 `mMediaPlayer01.start()` 播放 MP3。具体代码如下。

```
/* 用 Runnable 来确保文件在存储完后才开始 start() */
Runnable r = new Runnable()
{
    public void run()
    {
        try
        {
            /* setDataSource 将文件存到 SD 卡 */
            setDataSource(strPath);
            /* 因为线程顺利进行，所以在 setDataSource 后运行 prepare() */
            mMediaPlayer01.prepare();
            Log.i(TAG, "Duration: " + mMediaPlayer01.getDuration());

            /* 开始播放 mp3 */
            mMediaPlayer01.start();
            bIsReleased = false;
        }
        catch (Exception e)
        {
            Log.e(TAG, e.getMessage(), e);
        }
    }
};
new Thread(r).start();
}
.....
```

⑬ 如果有异常则输出提示，具体代码如下。

```
catch(Exception e)
{
    if (mMediaPlayer01 != null)
    {
        /* 线程发生异常则停止播放 */
        mMediaPlayer01.stop();
        mMediaPlayer01.release();
    }
    e.printStackTrace();
}
```

⑭ 定义函数 `setDataSource` 用于存储 URL 的 MP3 文件到存储卡。首先判断传入的地址是否为 URL，然后创建 URL 对象和临时文件。具体代码如下。

```
/* 定义函数用于将 URL 的 mp3 文件存储到存储卡 */
private void setDataSource(String strPath) throws Exception
{
    /* 判断传入的地址是否为 URL */
```



```

if (!URLUtil.isNetworkUrl(strPath))
{
    mMediaPlayer01.setDataSource(strPath);
}
else
{
    if(bIsReleased == false)
    {
        /* 创建 URL 对象 */
        URL myURL = new URL(strPath);
        URLConnection conn = myURL.openConnection();
        conn.connect();

        /* 获取 URLConnection 的 InputStream */
        InputStream is = conn.getInputStream();
        if (is == null)
        {
            throw new RuntimeException("stream is null");
        }
        /* 创建临时文件 */
        File myTempFile = File.createTempFile("yinyue", "."+getFileExtension(strPath));
        currentTempFilePath = myTempFile.getAbsolutePath();
        FileOutputStream fos = new FileOutputStream(myTempFile);
        byte buf[] = new byte[128];
        do
        {
            int numread = is.read(buf);
            if (numread <= 0)
            {
                break;
            }
            fos.write(buf, 0, numread);
        }while (true);

        /*直到 fos 存储完毕, 调用 MediaPlayer.setDataSource */
        mMediaPlayer01.setDataSource(currentTempFilePath);
        try
        {
            is.close();
        }
        catch (Exception ex)
        {
            Log.e(TAG, "error: " + ex.getMessage(), ex);
        }
    }
}
}
}

```

⑮ 定义方法 `getFileExtension(String strFileName)` 来获取音乐文件的扩展名, 如果无法顺利获取扩展名则默认为 “.dat”。具体代码如下。

```
/* 获取音乐文件扩展名自定义函数 */
```



```
private String getFileExtension(String strFileName)
{
    File myFile = new File(strFileName);
    String strFileExtension=myFile.getName();
    strFileExtension=(strFileExtension.substring(strFileExtension.lastIndexOf(
    ".")+1)).toLowerCase();
    if(strFileExtension=="")
    {
        /* 如果无法顺利获取扩展名则默认为.dat */
        strFileExtension = "dat";
    }
    return strFileExtension;
}
```

⑩ 定义方法 `delFile(String strFileName)`来设置当离开程序时删除临时音乐文件，具体代码如下。

```
/* 离开程序时需要调用自定义函数删除临时音乐文件*/
private void delFile(String strFileName)
{
    File myFile = new File(strFileName);
    if(myFile.exists())
    {
        myFile.delete();
    }
}

@Override
protected void onPause()
{
    //TODO Auto-generated method stub

    /* 删除临时文件 */
    try
    {
        delFile(currentTempFilePath);
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
    super.onPause();
}
}
```

执行后可以通过播放、暂停、重新播放和停止 4 个按钮拉控制指定的 MP3 音乐，如图 5-4 所示。

5.4.2 实战演练——下载在线铃声

在日常的手机应用中，我们经常从网络中下载一个 MP3 文件作为手机铃声。在本实



图 5-4 执行效果

例中, 可以在 EditText 中输入一个 MP3 的网址, 当下载完成此网址的 MP3 后打开 RingtoneManager.ACTION_RINGTONE_PICKER 这个 Intent, 在打开 Intent 的同时传入一个参数, 这个 ACTION_RINGTONE_PICKER 的 Intent 会带入刚才下载文件让用户选择。

题 目	目 的	源码路径
实例 5-5	下载在线铃声	光盘:\codes\5\ling

在本实例的具体实现过程中, 会首先判断下载文件是否完整, 并判断用户是否已经设置铃声, 会以 SD 卡中的铃声文件作为存储网络下载音乐文件的路径, 打开 RingtoneManager 的 ACTION_RINGTONE_PICKER 的 Intent 让用户找到下载的音乐, 并作为铃声。

本实例的主程序文件是文件 ling.java, 具体实现流程如下。

(1) 用 private 私有声明系统中需要的对象, 具体代码如下。

```
protected static final String APP_TAG = "DOWNLOAD_RINGTONE";
private Button mButton1;
private TextView mTextView1;
private EditText mEditText1;
private String strURL = "";
public static final int RINGTONE_PICKED = 0x108;
private String currentFilePath = "";
private String currentTempFilePath = "";
private String fileEx="";
private String fileNa="";
private String strRingtoneFolder = "/sdcard/music/ling";
```

(2) 判断是否包含文件夹/sdcard/music/ringtones, 如果不存在则输出提示。具体代码如下。

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mButton1 =(Button) findViewById(R.id.myButton1);
    mTextView1 = (TextView) findViewById(R.id.myTextView1);
    mEditText1 = (EditText) findViewById(R.id.myEditText1);
    /*判断是否有/sdcard/music/ringtones 文件夹*/
    if(bIfExistRingtoneFolder(strRingtoneFolder))
    {
        Log.i(APP_TAG, "Ringtone Folder exists.");
    }
    .....
}
```

(3) 使用 fileEx 和 getFile 取得远程 MP3 文件的名称, 具体代码如下。

```
mButton1.setOnClickListener(new Button.OnClickListener()
{
    @Override
    public void onClick(View arg0)
    {
```

```

        strURL = mEditText1.getText().toString();
        Toast.makeText(ling.this, getString(R.string.str_msg)
            , Toast.LENGTH_SHORT).show();
        /*取得文件名称*/
        fileEx = strURL.substring(strURL.lastIndexOf(".") + 1, strURL.
            length()).toLowerCase();
        fileNa = strURL.substring(strURL.lastIndexOf("/") + 1, strURL.
            lastIndexOf("."));
        getFile(strURL);
    }
});
}
.....

```

(4) 定义方法 `getMimeType(File f)` 来判断文件 `MimeType` 的打开格式，具体代码如下。

```

/* 判断文件 MimeType 的 method */
private String getMimeType(File f)
{
    String type="";
    String fName=f.getName();
    /* 取得扩展名 */
    String end=fName.substring(fName.lastIndexOf(".") + 1,
        fName.length()).toLowerCase();
    /* 依扩展名的类型决定 MimeType */
    if(end.equals("m4a") || end.equals("mp3") || end.equals("mid") ||
        end.equals("xmf") || end.equals("ogg") || end.equals("wav"))
    {
        type = "audio";
    }
    else if(end.equals("3gp") || end.equals("mp4"))
    {
        type = "video";
    }
    else if(end.equals("jpg") || end.equals("gif") ||
        end.equals("png") || end.equals("jpeg") ||
        end.equals("bmp"))
    {
        type = "image";
    }
    else
    {
        type="*";
    }
    /*如果无法直接打开，就跳出软件列表给用户选择 */
    if(end.equals("image"))
    {
    }
    else
    {
        type += "/*";
    }
}

```

```

    }
    return type;
}

```

(5) 定义方法 `getFile(final String strPath)` 来获取 MP3 文件，如果地址和当前地址一样则直接使用 `getDataSource` 数据，如果有异常则输出异常信息。具体代码如下。

```

private void getFile(final String strPath)
{
    try
    {
        if (strPath.equals(currentFilePath) )
        {
            getDataSource(strPath);
        }
        currentFilePath = strPath;
        Runnable r = new Runnable()
        {
            public void run()
            {
                try
                {
                    getDataSource(strPath);
                }
                catch (Exception e)
                {
                    Log.e(APP_TAG, e.getMessage(), e);
                }
            }
        };
        new Thread(r).start();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

```

(6) 定义方法 `getDataSource(String strPath)` 来获取远程文件，如果地址错误则输出错误信息。具体代码如下。

```

/*取得远程文件*/
private void getDataSource(String strPath) throws Exception
{
    if (!URLUtil.isNetworkUrl(strPath))

```



```

{
    mTextView1.setText("错误的 URL");
}
else
{
    /*取得 URL*/
    URL myURL = new URL(strPath);
    /*创建连接*/
    URLConnection conn = myURL.openConnection();
    conn.connect();
    /*InputStream 下载文件*/
    InputStream is = conn.getInputStream();
    if (is == null)
    {
        throw new RuntimeException("stream is null");
    }

    /*创建文件地址*/
    File myTempFile = new File("/sdcard/music/ling/",
        fileName+"."+fileEx);
    /*取得在暂存盘的路径*/
    currentTempFilePath = myTempFile.getAbsolutePath();
    /*将文件写入暂存盘*/
    FileOutputStream fos = new FileOutputStream(myTempFile);
    byte buf[] = new byte[128];
    do
    {
        int numread = is.read(buf);
        if (numread <= 0)
        {
            break;
        }
        fos.write(buf, 0, numread);
    }while (true);
}
.....

```

(7) 打开 RingtoneManager 以选择铃声，通过 Intent 对象 intent 来设置铃声，然后设置显示铃声的文件夹和显示铃声开头。如果有异常则输出异常。具体代码如下。

```

/* 打开 RingtoneManager 进行铃声选择 */
String uri = null;
if (bIfExistRingtoneFolder(strRingtoneFolder))
{
    /*设置铃声*/
    Intent intent = new Intent(RingtoneManager.
        ACTION_RINGTONE_PICKER);
    /*设置显示铃声的文件夹*/
    intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TYPE,
        RingtoneManager.TYPE_RINGTONE);
    /*设置显示铃声开头*/
    intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TITLE,
        "设置铃声");
}

```



```

        if( uri != null)
        {
            intent.putExtra( RingtoneManager.
                EXTRA_RINGTONE_EXISTING_URI, Uri.parse( uri));
        }
        else
        {
            intent.putExtra( RingtoneManager.
                EXTRA_RINGTONE_EXISTING_URI, (Uri)null);
        }
        startActivityForResult(intent, RINGTONE_PICKED);
    }

    try
    {
        is.close();
    }
    catch (Exception ex)
    {
        Log.e(APP_TAG, "error: " + ex.getMessage(), ex);
    }
}
}
.....

```

(8)定义方法 `onActivityResult` 根据用户选择的铃声设置保存对应的信息。当选择完毕后,再次返回选择 Activity 界面。具体代码如下。

```

protected void onActivityResult(int requestCode,
                                int resultCode, Intent data)
{
    if (resultCode != RESULT_OK)
    {
        return;
    }
    switch (requestCode)
    {
        case (RINGTONE_PICKED):
            try
            {
                Uri pickedUri = data.getParcelableExtra
                    (RingtoneManager.EXTRA_RINGTONE_PICKED_URI);
                if(pickedUri!=null)
                {
                    RingtoneManager.setActualDefaultRingtoneUri
                        (ling.this,RingtoneManager.TYPE_RINGTONE,
                            pickedUri);
                }
            }
            catch(Exception e)
            {
                e.printStackTrace();
            }
        }
    }
}

```

```

    }
    break;
default:
    break;
}
super.onActivityResult(requestCode, resultCode, data);
}

```

(9) 定义 `bIfExistRingtoneFolder` 来判断是否包含文件夹 “/sdcard/music/ringtones”，具体代码如下。

```

/*判断是否包含 “/sdcard/music/ringtones 文件夹” */
private boolean bIfExistRingtoneFolder(String strFolder)
{
    boolean bReturn = false;

    File f = new File(strFolder);
    if(!f.exists())
    {
        /*创建/sdcard/music/ringtones 文件夹*/
        if(f.mkdirs())
        {
            bReturn = true;
        }
        else
        {
            bReturn = false;
        }
    }
    else
    {
        bReturn = true;
    }
    return bReturn;
}
}

```

执行后会先显示一个下载界面，单击【点击下载】按钮后开始下载指定的 MP3 文件，下载完成后会弹出一个界面，如图 5-5 所示。选择一种选项，并单击 OK 按钮后，完成铃声设置。

<http://www.lrn.cn/zywh/xyyy/yyxs/200805/W020080505536315331317.mp3>

点击下载

图 5-5 执行效果

5.5 多线程下载

线程可以理解为下载的通道，一个线程就是一个文件的下载通道，多线程也就是同时开启好几个下载通道。当服务器提供下载服务时，使用下载者可共享带宽，在优先级相同的情

况下，总服务器会对总下载线程进行平均分配。线程越多，下载的速度就会越快。现在流行的下载软件都支持多线程。在本节的内容中，将详细讲解在 Android 系统中实现多线程下载的过程。

5.5.1 多线程下载文件的过程

在 Android 系统中，实现多线程下载的基本过程如下。

(1) 获得下载文件的长度，然后设置本地文件的长度。

```
URLConnection.getContentLength(); //获取下载文件的长度
RandomAccessFile file = new RandomAccessFile("QQWubiSetup.exe", "rwd");
file.setLength(filesize); //设置本地文件的长度
```

(2) 根据文件长度和线程数计算每条线程下载的数据长度和下载位置。例如，文件的长度为 6M，线程数为 3，那么每条线程下载的数据长度为 2M，每条线程开始下载的位置如图 5-6 所示。



图 5-6 每条线程开始下载的位置

文件的长度为 10M，则使用 3 个线程来下载，具体说明如下。

- 线程下载的数据长度： $(10\%3 = 0 ? 10/3:10/3+1)$ ，第 1、2 个线程下载长度是 4M，第三个线程下载长度为 2M
- 下载开始位置：线程 id * 每条线程下载的数据长度 = ?
- 下载结束位置： $(\text{线程 id} + 1) * \text{每条线程下载的数据长度} - 1 = ?$

(3) 使用 Http 的 Range 头字段指定每条线程从文件的什么位置开始下载，下载到什么位置为止，例如，指定从文件的 2M 位置开始下载，下载到位置(4M-1byte)为止，代码如下：

```
URLConnection.setRequestProperty("Range", "bytes=2097152-4194303");
```

(4) 保存文件，使用类 RandomAccessFile 指定每条线程从本地文件的什么位置开始写入数据。

```
RandomAccessFile threadfile = new RandomAccessFile("QQWubiSetup.exe", "rwd");
threadfile.seek(2097152); //从文件的什么位置开始写入数据
```

5.5.2 实战演练——在 Android 系统中实现多线程下载

本实例介绍了在 Android 平台下通过 HTTP 协议实现断点续传下载的方法。本实例是一个 HTTP 协议多线程断点下载应用程序，直接使用单线程下载 HTTP 文件对初学者来说是一件非常简单的事。多线程断点需要具备如下功能。

- 多线程下载。
- 支持断点。

在接下来的内容中，将通过一个具体实例的实现过程，来讲解在 Android 手机中下载在

线铃声的方法。

题 目	目 的	源码路径
实例 5-6	下载在线铃声	光盘:\codes\5\Multiple

本实例的具体实现流程如下。

(1) 打开 Eclipse, 新建一个名称为 MultipleThreadDownload 的动态 Web 工程。然后准备一个 MP3 文件保存在 WebContent 目录下, 最后发布服务器端的 Web 工程程序。

(2) 打开 Eclipse, 新建一个名称为 MultipleThreadDownloadrAndroid 的 Android 工程。然后编写主程序文件 main.xml, 具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <!-- 下载路径提示文字 -->
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/path"
    />
    <!-- 下载路径输入框, 此处为了方便测试, 我们设置了默认的路径, 可以根据需要在用户界面处修改 -->
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="http://192.168.1.100:8080/ServerForMultipleThreadDownloader/
        CNNRecordingFromWangjialin.mp3"
        android:id="@+id/path"
    />
    <!-- 水平 LinearLayout 布局, 包括下载按钮和暂停按钮 -->
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
    >
        <!-- 下载按钮, 用于触发下载事件 -->
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/button"
            android:id="@+id/downloadbutton"
        />
        <!-- 暂停按钮, 在初始状态下为不可用 -->
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/stopbutton"
```

```

        android:enabled="false"
        android:id="@+id/stopbutton"
    />
</LinearLayout>
<!-- 水平进度条，用图形化的方式实时显示进步信息 -->
<ProgressBar
    android:layout_width="fill_parent"
    android:layout_height="18dp"
    style="?android:attr/progressBarStyleHorizontal"
    android:id="@+id/progressBar"
/>
<!-- 文本框，用于显示实时下载的百分比 -->
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:id="@+id/resultView"
/>
</LinearLayout>

```

(3) 创建数据库管理类 DBOpenHelper，实现文件 DBOpenHelper.java 的具体代码如下。

```

/**
 * SQLite 管理器，实现创建数据库和表，但版本变化时实现对表的数据库表的操作
 *
 */
public class DBOpenHelper extends SQLiteOpenHelper {
    private static final String DBNAME = "eric.db"; //设置数据库的名称
    private static final int VERSION = 1; //设置数据库的版本

    /**
     * 通过构造方法
     * @param context 应用程序的上下文对象
     */
    public DBOpenHelper(Context context) {
        super(context, DBNAME, null, VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) { //建立数据表
        db.execSQL("CREATE TABLE IF NOT EXISTS filednlog (id integer primary
            key autoincrement, downpath varchar(100), threadid INTEGER, downlength
            INTEGER)");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    { //当版本变化时系统会调用该回调方法
        db.execSQL("DROP TABLE IF EXISTS filednlog");
        //此处是删除数据表，在实际的业务中一般需要数据备份
    }
}

```



```
onCreate(db);
```

```
//调用 onCreate 方法重新创建数据表,也可以自己根据业务需要创建新的数据表
```

```
}
```

(4) 建立数据库业务操作类 FileService, 此类的实现文件是 FileService.java, 具体代码如下。

```
/**
 * 业务 Bean, 实现对数据的操作
 *
 */
public class FileService {
    private DBOpenHelper openHelper;           //声明数据库管理器

    public FileService(Context context) {
        openHelper = new DBOpenHelper(context); //根据上下文对象实例化数据库管理器
    }
    /**
     * 获取特定 URI 的每条线程已经下载的文件长度
     * @param path
     * @return
     */
    public Map<Integer, Integer> getData(String path) {
        SQLiteDatabase db = openHelper.getReadableDatabase();
        //获取可读的数据库句柄, 一般情况下在该操作的内部实现中其返回的其实是可写的数据库句柄
        Cursor cursor = db.rawQuery("select threadid, downlength from filedownlog where downpath=?", new String[]{path});
        //根据下载路径查询所有线程下载数据, 返回的 Cursor 指向第一条记录之前
        Map<Integer, Integer> data = new HashMap<Integer, Integer>();
        //建立一个哈希表用于存放每条线程的已经下载的文件长度
        while(cursor.moveToNext()){ //从第一条记录开始遍历 Cursor 对象
            data.put(cursor.getInt(0), cursor.getInt(1));
            //把线程 id 和该线程已下载的长度设置进 data 哈希表中
            data.put(cursor.getInt(cursor.getColumnIndexOrThrow("threadid")),
                cursor.getInt(cursor.getColumnIndexOrThrow("downlength")));
        }
        cursor.close();           //关闭 cursor, 释放资源
        db.close();               //关闭数据库
        return data;              //返回获得的每条线程和每条线程的下载长度
    }
    /**
     * 保存每条线程已经下载的文件长度
     * @param path    下载的路径
     * @param map 现在的 id 和已经下载的长度的集合
     */
    public void save(String path, Map<Integer, Integer> map){
        SQLiteDatabase db = openHelper.getWritableDatabase();
        //获取可写的数据库句柄
        db.beginTransaction();    //开始事务, 因为此处要插入多批数据
        try{
```

```

        for(Map.Entry<Integer, Integer> entry : map.entrySet()){
            //采用 For-Each 的方式遍历数据集
            db.execSQL("insert into filedownload(downpath, threadid, downlength)
            values(?,?,?)",
                new Object[]{path, entry.getKey(), entry.getValue()});
            //插入特定下载路径下特定线程 ID 已经下载的数据
        }
        db.setTransactionSuccessful(); //设置事务执行的标志为成功
    }finally{
        //如果不杀死虚拟机,此部分的代码肯定会被执行
        db.endTransaction();
        //结束一个事务,如果事务设立了成功标志,则提交事务,否则会滚事务
    }
    db.close(); //关闭数据库,释放相关资源
}
/**
 * 实时更新每条线程已经下载的文件长度
 * @param path
 * @param map
 */
public void update(String path, int threadId, int pos){
    SQLiteDatabase db = openHelper.getWritableDatabase();
    //获取可写的数据库句柄
    db.execSQL("update filedownload set downlength=? where downpath=? and
    threadid=?",
        new Object[]{pos, path, threadId});
    //更新特定下载路径下特定线程已经下载的文件长度
    db.close(); //关闭数据库,释放相关的资源
}
/**
 * 当文件下载完成后,删除对应的下载记录
 * @param path
 */
public void delete(String path){
    SQLiteDatabase db = openHelper.getWritableDatabase();
    //获取可写的数据库句柄
    db.execSQL("delete from filedownload where downpath=?", new
    Object[]{path}); //删除特定下载路径的所有线程记录
    db.close(); //关闭数据库,释放资源
}
}

```

(5)编写文件下载类 FileDownloader, 此类调用类 DownloadThread 实现具体的下载功能。类 FileDownloader 在文件 FileDownloader.java 中定义, 具体实现代码如下。

```

public class FileDownloader {
    private static final String TAG = "FileDownloader";
    //设置标签, 方便 Logcat 日志记录
    private static final int RESPONSEOK = 200; //响应码为 200, 即访问成功
    private Context context; //应用程序的上下文对象
    private FileService fileService; //获取本地数据库的业务 Bean
    private boolean exited; //停止下载标志
}

```

```

private int downloadedSize = 0;           //已下载文件长度
private int fileSize = 0;                //原始文件长度
private DownloadThread[] threads;        //根据线程数设置下载线程池
private File saveFile;                   //数据保存到的本地文件
private Map<Integer, Integer> data = new ConcurrentHashMap<Integer,
Integer>();                               //缓存各线程下载的长度
private int block;                        //每条线程下载的长度
private String downloadUrl;              //下载路径

/**
 * 获取线程数
 */
public int getThreadSize() {
    return threads.length;               //根据数组长度返回线程数
}

/**
 * 退出下载
 */
public void exit(){
    this.exited = true;                  //设置退出标志为 true
}
public boolean getExited(){
    return this.exited;
}
/**
 * 获取文件大小
 * @return
 */
public int getFileSize() {
    return fileSize;                     //从类成员变量中获取下载文件的大小
}

/**
 * 累计已下载大小
 * @param size
 */
protected synchronized void append(int size) { //使用同步关键字解决并发访问问题
    downloadedSize += size;              //把实时下载的长度加入到总下载长度中
}

/**
 * 更新指定线程最后下载的位置
 * @param threadId 线程 id
 * @param pos 最后下载的位置
 */
protected synchronized void update(int threadId, int pos) {
    this.data.put(threadId, pos);
    //把制定线程 id 的线程赋予最新的下载长度，以前的值会被覆盖掉
    this.fileService.update(this.downloadUrl, threadId, pos);
}

```

```

//更新数据库中指定线程的下载长度
}
/**
 * 构建文件下载器
 * @param downloadUrl 下载路径
 * @param fileSaveDir 文件保存目录
 * @param threadNum 下载线程数
 */
public FileDownloader(Context context, String downloadUrl, File fileSaveDir,
int threadNum) {
    try {
        this.context = context;           //对上下文对象赋值
        this.downloadUrl = downloadUrl;    //对下载的路径赋值
        fileService = new FileService(this.context);
        //实例化数据操作业务Bean, 此处需要使用 Context, 因为此处的数据是引用程序私有数据
        URL url = new URL(this.downloadUrl); //根据下载路径实例化 URL
        if(!fileSaveDir.exists()) fileSaveDir.mkdirs();
        //如果指定的文件不存在, 则创建目录, 此处可以创建多层目录
        this.threads = new DownloadThread[threadNum];
        //根据下载的线程数创建下载线程池
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        //建立一个远程连接句柄, 此时尚未真正连接
        conn.setConnectTimeout(5*1000);      //设置连接超时时间为 5 秒
        conn.setRequestMethod("GET");        //设置请求方式为 GET
        conn.setRequestProperty("Accept", "image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/x-shockwave-flash, application/xaml+xml, application/vnd.ms-xpsdocument, application/x-ms-xbap, application/x-ms-application, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*"); //设置客户端可以接受的媒体类型
        conn.setRequestProperty("Accept-Language", "zh-CN");
        //设置客户端语言
        conn.setRequestProperty("Referer", downloadUrl);
        //设置请求的来源页面, 便于服务端进行来源统计
        conn.setRequestProperty("Charset", "UTF-8"); //设置客户端编码
        conn.setRequestProperty("User-Agent", "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.2; Trident/4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)"); //设置用户代理
        conn.setRequestProperty("Connection", "Keep-Alive");
        //设置 Connection 的方式
        conn.connect(); //和远程资源建立真正的连接, 但尚无返回的数据流
        printResponseHeader(conn); //答应返回的 HTTP 头字段集合
        if (conn.getResponseCode() == RESPONSEOK) {
            //此处的请求会打开回流并获取返回的状态码, 用于检查是否请求成功, 当返回码为
            200 时执行下面的代码
            this.fileSize = conn.getContentLength(); //根据响应获取文件大小
            if (this.fileSize <= 0) throw new RuntimeException("Unkown file size ");
            //当文件大小为小于等于零时抛出运行时异常

            String filename = getFileName(conn); //获取文件名称

```



```

this.saveFile = new File(fileSaveDir, filename);
//根据文件保存目录和文件名构建保存文件
Map<Integer, Integer> logdata = fileService.getData(downloadUrl);
//获取下载记录

if(logdata.size()>0){ //如果存在下载记录
    for(Map.Entry<Integer, Integer> entry : logdata.entrySet())
        //遍历集合中的数据
        data.put(entry.getKey(), entry.getValue());
        //把各条线程已经下载的数据长度放入 data 中
}

if(this.data.size()==this.threads.length){
//如果已经下载的数据的线程数和现在设置的线程数相同则计算所有线程已经下
载的数据总长度
    for (int i = 0; i < this.threads.length; i++) {
        //遍历每条线程已经下载的数据
        this.downloadedSize += this.data.get(i+1);
        //计算已经下载的数据之和
    }
    print("已经下载的长度"+ this.downloadedSize + "个字节");
    //打印出已经下载的数据总和
}

this.block = (this.fileSize % this.threads.length)==0? this.
fileSize / this.threads.length : this.fileSize / this.threads.
length + 1; //计算每条线程下载的数据长度
}else{
    print("服务器响应错误:" + conn.getResponseCode() + conn.
getResponseMessage()); //打印错误
    throw new RuntimeException("server response error ");
    //抛出运行时服务器返回异常
}
} catch (Exception e) {
    print(e.toString()); //打印错误
    throw new RuntimeException("Can't connection this url");
    //抛出运行时无法连接的异常
}
}
/**
 * 获取文件名
 */
private String getFileName(URLConnection conn) {
    String filename = this.downloadUrl.substring(this.downloadUrl.
lastIndexOf('/') + 1); //从下载路径的字符串中获取文件名称

    if(filename==null || "".equals(filename.trim())){
        //如果获取不到文件名称
        for (int i = 0;; i++) { //无限循环遍历
            String mine = conn.getHeaderField(i);

```



```

//从返回的流中获取特定索引的头字段值
    if (mine == null) break; //如果遍历到末尾则退出循环
    if ("content-disposition".equals(conn.getHeaderFieldKey(i).
        toLowerCase())){
        //获取 content-disposition 返回头字段, 里面可能会包含文件名
        Matcher m = Pattern.compile(".*filename=(.*)").matcher
            (mine.toLowerCase()); //使用正则表达式查询文件名
        if(m.find()) return m.group(1); //如果有符合正则表达规则的字符串
    }
}

filename = UUID.randomUUID()+ ".tmp";
//由网卡上的标识数字(每个网卡都有唯一的标识号)以及 CPU 时钟的唯一数字生成的
//的一个 16 字节的二进制作为文件名
}

return filename;
}

/**
 * 开始下载文件
 * @param listener 监听下载数量的变化, 如果不需要了解实时下载的数量, 可以设置为 null
 * @return 已下载文件大小
 * @throws Exception
 */
public int download(DownloadProgressListener listener) throws Exception{
//进行下载, 并抛出异常给调用者, 如果有异常
    try {
        RandomAccessFile randOut = new RandomAccessFile(this.saveFile, "rwd");
        //The file is opened for reading and writing. Every change of the
        //file's content must be written synchronously to the target device.
        if(this.fileSize>0) randOut.setLength(this.fileSize);
        //设置文件的大小
        randOut.close(); //关闭该文件, 使设置生效
        URL url = new URL(this.downloadUrl);
        //A URL instance specifies the location of a resource on the internet
        //as specified by RFC 1738
        if(this.data.size() != this.threads.length){
            //如果原先未曾下载或者原先的下载线程数与现在的线程数不一致
            this.data.clear();
            //Removes all elements from this Map, leaving it empty.
            for (int i = 0; i < this.threads.length; i++) { //遍历线程池
                this.data.put(i+1, 0); //初始化每条线程已经下载的数据长度为 0
            }
            this.downloadedSize = 0; //设置已经下载的长度为 0
        }
        for (int i = 0; i < this.threads.length; i++) { //开启线程进行下载
            int downloadedLength = this.data.get(i+1);
            //通过特定的线程 ID 获取该线程已经下载的数据长度
            if(downloadedLength < this.block && this.downloadedSize <
                this.fileSize){ //判断线程是否已经完成下载, 否则继续下载
                this.threads[i] = new DownloadThread(this, url, this.

```

```

        saveFile, this.block, this.data.get(i+1), i+1);
        //初始化特定 id 的线程
        this.threads[i].setPriority(7);
        //设置线程的优先级, Thread.NORM_PRIORITY = 5 Thread.MIN_
        PRIORITY = 1 Thread.MAX_PRIORITY = 10
        this.threads[i].start(); //启动线程
    }else{
        this.threads[i] = null; //表明在线程已经完成下载任务
    }
}
fileService.delete(this.downloadUrl);
//如果存在下载记录, 删除它们, 然后重新添加
fileService.save(this.downloadUrl, this.data);
//把已经下载的实时数据写入数据库
boolean notFinished = true; //下载未完成
while (notFinished) { //循环判断所有线程是否完成下载
    Thread.sleep(900);
    notFinished = false; //假定全部线程下载完成
    for (int i = 0; i < this.threads.length; i++){
        if (this.threads[i] != null && !this.threads[i].isFinished()) {
            //如果发现线程未完成下载
            notFinished = true; //设置标志为下载没有完成
            if(this.threads[i].getDownloadedLength() == -1){
                //如果下载失败, 再重新在已经下载的数据长度的基础上下载
                this.threads[i] = new DownloadThread(this, url,
                this.saveFile, this.block, this.data.get(i+1), i+1);
                //重新开辟下载线程
                this.threads[i].setPriority(7); //设置下载的优先级
                this.threads[i].start(); //开始下载线程
            }
        }
    }
    if(listener!=null) listener.onDownloadSize(this.downloadedSize);
    //通知目前已经下载完成的数据长度
}
if(downloadedSize == this.fileSize) fileService.delete(this.
downloadUrl); //下载完成删除记录
} catch (Exception e) {
    print(e.toString()); //打印错误
    throw new Exception("File downloads error"); //抛出文件下载异常
}
return this.downloadedSize;
}
/**
 * 获取 Http 响应头字段
 * @param http HttpURLConnection 对象
 * @return 返回头字段的 LinkedHashMap
 */
public static Map<String, String> getHttpResponseHeader(HttpURLConnection http) {
    Map<String, String> header = new LinkedHashMap<String, String>();

```

```

//使用 LinkedHashMap 保证写入和遍历时的顺序相同, 而且允许空值存在
for (int i = 0;; i++) { //此处为无限循环, 因为不知道头字段的数量
    String fieldValue = http.getHeaderField(i);
    //getHeaderField(int n) 用于返回第 n 个头字段的值

    if (fieldValue == null) break;
    //如果第 i 个字段没有值了, 则表明头字段部分已经循环完毕, 此处使用 Break 退出循环
    header.put(http.getHeaderFieldKey(i), fieldValue);
    //getHeaderFieldKey(int n) 用于返回第 n 个头字段的键
}
return header;
}
/**
 * 打印 Http 头字段
 * @param http HttpURLConnection 对象
 */
public static void printResponseHeader(HttpURLConnection http) {
    Map<String, String> header = getHttpResponseHeader(http);
    //获取 Http 响应头字段
    for(Map.Entry<String, String> entry : header.entrySet()){
        //使用 For-Each 循环的方式遍历获取的头字段的值, 此时遍历的循序和输入的顺序相同
        String key = entry.getKey() != null ? entry.getKey() + ":" : "";
        //当有键时则获取键, 如果没有则为空字符串
        print(key + entry.getValue()); //答应键和值的组合
    }
}

/**
 * 打印信息
 * @param msg 信息字符串
 */
private static void print(String msg) {
    Log.i(TAG, msg); //使用 LogCat 的 Information 方式打印信息
}
}

```

(6) 类 DownloadThread 在文件 DownloadThread.java 中定义, 具体实现代码如下。

```

/**
 * 下载线程, 根据具体下载地址、保存到的文件、下载块的大小、已经下载的数据大小等信息进行下载
 */
public class DownloadThread extends Thread {
    private static final String TAG = "DownloadThread";
    //定义 TAG, 方便日子的打印输出
    private File saveFile; //下载的数据保存到的文件
    private URL downUrl; //下载的 URL
    private int block; //每条线程下载的大小
    private int threadId = -1; //初始化线程 id 设置
    private int downloadedLength; //该线程已经下载的数据长度
    private boolean finished = false; //该线程是否完成下载的标志
}

```

```

private FileDownloader downloader; //文件下载器

public DownloadThread(FileDownloader downloader, URL downUrl, File
saveFile, int block, int downloadedLength, int threadId) {
    this.downUrl = downUrl;
    this.saveFile = saveFile;
    this.block = block;
    this.downloader = downloader;
    this.threadId = threadId;
    this.downloadedLength = downloadedLength;
}

@Override
public void run() {
    if(downloadedLength < block){ //未下载完成
        try {
            HttpURLConnection http = (HttpURLConnection) downUrl.
openConnection(); //开启 HttpURLConnection 连接
            http.setConnectTimeout(5 * 1000); //设置连接超时时间为 5 秒钟
            http.setRequestMethod("GET"); //设置请求的方法为 GET
            http.setRequestProperty("Accept", "image/gif, image/jpeg,
image/pjpeg, image/pjpeg, application/x-shockwave-flash, application
/xaml+xml, application/vnd.ms-xpsdocument, application/x-ms-xbap,
application/x-ms-application, application/vnd.ms-excel,
application/vnd.ms-powerpoint, application/msword, */*");
            //设置客户端可以接受的返回数据类型
            http.setRequestProperty("Accept-Language", "zh-CN");
            //设置客户端使用的语言为中文
            http.setRequestProperty("Referer", downUrl.toString());
            //设置请求的来源, 便于对访问来源进行统计
            http.setRequestProperty("Charset", "UTF-8");
            //设置通信编码为 UTF-8
            int startPos = block * (threadId - 1) + downloadedLength;
            //开始位置
            int endPos = block * threadId - 1; //结束位置
            http.setRequestProperty("Range", "bytes=" + startPos + "-" +
endPos); //设置获取实体数据的范围, 如果超过了实体数据的大小会自动返回
            实际的数据大小
            http.setRequestProperty("User-Agent", "Mozilla/4.0 (compatible;
MSIE 8.0; Windows NT 5.2; Trident/4.0; .NET CLR 1.1.4322; .NET
CLR 2.0.50727; .NET CLR 3.0.04506.30; .NET CLR 3.0.4506.
2152; .NET CLR 3.5.30729)"); //客户端用户代理
            http.setRequestProperty("Connection", "Keep-Alive");
            //使用长连接

            InputStream inStream = http.getInputStream();
            //获取远程连接的输入流
            byte[] buffer = new byte[1024]; //设置本地数据缓存的大小为 1M
            int offset = 0; //设置每次读取的数据量
            print("Thread " + this.threadId + " starts to download from

```

```

position "+ startPos); //打印该线程开始下载的位置
RandomAccessFile threadFile = new RandomAccessFile(this.
saveFile, "rwd");
//If the file does not already exist then an attempt will be made
to create it and it require that every update to the file's
content be written synchronously to the underlying storage device.
threadFile.seek(startPos); //文件指针指向开始下载的位置
while (!downloader.getExited() && (offset = inStream.read
(buffer, 0, 1024)) != -1) {
//但用户没有要求停止下载, 同时没有到达请求数据的末尾会一直循环读取数据
threadFile.write(buffer, 0, offset); //直接把数据写到文件中
downloadedLength += offset;
//把新下载的已经写到文件中的数据加入到下载长度中
downloader.update(this.threadId, downloadedLength);
//将该线程已经下载的数据长度更新到数据库和内存哈希表中
downloader.append(offset);
//把新下载的数据长度加入到已经下载的数据总长度中
} //该线程下载数据完毕或者下载被用户停止
threadFile.close();
//Closes this random access file stream and releases any system
resources associated with the stream.
inStream.close();
//Concrete implementations of this class should free any resources
during close
if(downloader.getExited())
{
    print("Thread " + this.threadId + " has been paused");
}
else
{
    print("Thread " + this.threadId + " download finish");
}

this.finished = true;
//设置完成标志为 true, 无论是下载完成还是用户主动中断下载
} catch (Exception e) { //出现异常
    this.downloadedLength = -1; //设置该线程已经下载的长度为-1
    print("Thread " + this.threadId+ " :"+ e); //打印出异常信息
}
}

}
/**
 * 打印信息
 * @param msg 信息
 */
private static void print(String msg){
    Log.i(TAG, msg); //使用 Logcat 的 Information 方式打印信息
}

/**

```



```

        * 下载是否完成
        * @return
        */
    public boolean isFinished() {
        return finished;
    }

    /**
     * 已经下载的内容大小
     * @return 如果返回值为-1,代表下载失败
     */
    public long getDownloadedLength() {
        return downloadedLength;
    }
}

```

(7) 在类 `FileDownloader` 中调用了类 `DownloadProgressListener` 来监听下载进度, 类 `DownloadProgressListener` 在文件 `DownloadProgressListener.java` 中定义, 具体实现代码如下。

```

public interface DownloadProgressListener {
    /**
     * 下载进度监听方法获取和处理下载点数据的大小
     * @param size 数据大小
     */
    public void onDownloadSize(int size);
}

```

(8) 编写主界面文件 `MultipleThreadDownloadAndroid.java`, 具体实现代码如下。

```

/**
 * 主界面, 负责下载界面的显示、与用户交互、响应用户事件等
 */
public class MultipleThreadDownloadAndroid
    extends Activity {
    private static final int PROCESSING = 1;    //正在下载实时数据传输 Message 标志
    private static final int FAILURE = -1;     //下载失败时的 Message 标志

    private EditText pathText;                //下载输入文本框
    private TextView resultView;              //现在进度显示百分比文本框
    private Button downloadButton;             //下载按钮, 可以触发下载事件
    private Button stopbutton;                //停止按钮, 可以停止下载
    private ProgressBar progressBar;          //下载进度条, 实时图形化的显示进度信息
    //handler 对象的作用是向创建 Handler 对象所在的线程所绑定的消息队列发送消息并处理消息
    private Handler handler = new UIHandler();

    private final class UIHandler extends Handler{
        /**
         * 系统会自动调用回调方法处理消息事件
         * Message 一般会包含消息的标志和消息的内容以及消息的处理器 Handler

```

```

*/
public void handleMessage(Message msg) {
    switch (msg.what) {
        case PROCESSING: //下载时
            int size = msg.getData().getInt("size");
            //从消息中获取已经下载的数据长度
            progressBar.setProgress(size); //设置进度条的进度
            float num = (float)progressBar.getProgress() / (float)
            progressBar.getMax(); //计算已经下载的百分比,此处需要转换为浮点数计算
            int result = (int)(num * 100); //把获取的浮点数计算结构转化为整数
            resultView.setText(result+ "%"); //把下载的百分比显示在界面显示控件上
            if(progressBar.getProgress() == progressBar.getMax()){
                //当下载完成时
                Toast.makeText(getApplicationContext(), R.string.
                success, Toast.LENGTH_LONG).show(); //使用 Toast 技术提示用户下载完成
            }
            break;

        case -1: //下载失败时
            Toast.makeText(getApplicationContext(), R.string.error,
            Toast.LENGTH_LONG).show(); //提示用户下载失败
            break;
    }
}

@Override
public void onCreate(Bundle savedInstanceState) {
    //应用程序启动时会首先调用且在应用程序整个生命周期中只会调用一次,适合于初始化工作
    super.onCreate(savedInstanceState);
    //使用父类的 onCreate 用作屏幕主界面的底层和基本绘制工作
    setContentView(R.layout.main); //根据 XML 界面文件设置主界面
    pathText = (EditText) this.findViewById(R.id.path);
    //获取下载 URL 的文本输入框对象
    resultView = (TextView) this.findViewById(R.id.resultView);
    //获取显示下载百分比文本控件对象
    downloadButton = (Button) this.findViewById(R.id.downloadbutton);
    //获取下载按钮对象
    stopbutton = (Button) this.findViewById(R.id.stopbutton);
    //获取停止下载按钮对象
    progressBar = (ProgressBar) this.findViewById(R.id.progressBar);
    //获取进度条对象
    ButtonClickListener listener = new ButtonClickListener();
    //声明并定义按钮监听器对象
    downloadButton.setOnClickListener(listener); //设置下载按钮的监听器对象
    stopbutton.setOnClickListener(listener); //设置停止下载按钮的监听器对象
}
/**
 * 按钮监听器实现类
 */

```

```

*/
private final class ButtonClickListener implements View.OnClickListener{
    public void onClick(View v) {
        //该方法在注册了该按钮监听器的对象被单击时会自动调用，用于响应单击事件
        switch (v.getId()) {
            //获取单击对象的 ID
            case R.id.downloadbutton:
                //当单击下载按钮时
                String path = pathText.getText().toString(); //获取下载路径
                if(Environment.getExternalStorageState().equals
                    (Environment.MEDIA_MOUNTED)){ //获取 SDCard 是否存在，当 SDCard 存在时
                    File saveDir = Environment.getExternalStorageDirectory();
                    //获取 SDCard 根目录文件
                    File saveDir1 = Environment.getExternalStoragePublicDirectory
                        (Environment.DIRECTORY_MOVIES);

                    File saveDir11 = getApplicationContext().getExternalFilesDir
                        (Environment.DIRECTORY_MOVIES);
                    download(path, saveDir11); //下载文件
                }else{ //当 SDCard 不存在时
                    Toast.makeText(getApplicationContext(), R.string.
                        sdcarderror, Toast.LENGTH_LONG).show(); //提示用户 SDCard 不存在
                }
                downloadButton.setEnabled(false); //设置下载按钮不可用
                stopbutton.setEnabled(true); //设置停止下载按钮可用
                break;
            case R.id.stopbutton:
                //当单击停止下载按钮时
                exit(); //停止下载
                downloadButton.setEnabled(true); //设置下载按钮可用
                stopbutton.setEnabled(false); //设置停止按钮不可用
                break;
        }
    }
}

//////////////////////////////////////
//由于用户的输入事件(单击button,触摸屏幕...)是由主线程负责处理的，如果主线程处于工作状态
//此时用户产生的输入事件如果没能在5秒内得到处理，系统就会报“应用无响应”错误
//所以在主线程里不能执行一件比较耗时的工作，否则会因主线程阻塞而无法处理用户的输入事件
//导致“应用无响应”错误的出现，耗时的工作应该在子线程里执行
//////////////////////////////////////

private DownloadTask task; //声明下载执行者
/**
 * 退出下载
 */
public void exit(){
    if(task!=null) task.exit(); //如果有下载对象时，退出下载
}
/**
 * 下载资源函数，声明下载执行者并马上开辟线程
 * @param path 下载的路径
 * @param saveDir 保存文件

```

```

*/
private void download(String path, File saveDir){ //此方法运行在主线程
    task = new DownloadTask(path, saveDir); //实例化下载任务
    new Thread(task).start(); //开始下载
}
/*
 * UI 控件画面的重绘(更新)是由主线程负责处理的,如果在子线程中更新 UI 控件的
值,更新后的值不会重绘到屏幕上
 * 一定要在主线程里更新 UI 控件的值,这样才能在屏幕上显示出来,不能在子线程中
更新 UI 控件的值
 */
private final class DownloadTask implements Runnable{
    private String path; //下载路径
    private File saveDir; //下载到保存到的文件
    private FileDownloader loader; //文件下载器(下载线程的容器)
    /**
     * 构造方法,实现变量初始化
     * @param path 下载路径
     * @param saveDir 下载要保存到的文件
     */
    public DownloadTask(String path, File saveDir) {
        this.path = path;
        this.saveDir = saveDir;
    }

    /**
     * 退出下载
     */
    public void exit(){
        if(loader!=null) loader.exit(); //如果下载器存在则退出下载
    }

    DownloadProgressListener downloadProgressListener = new
    DownloadProgressListener() { //开始下载,并设置下载的监听器

        /**
         * 下载的文件长度会不断的被传入该回调方法
         */
        public void onDownloadSize(int size) {
            Message msg = new Message(); //新建一个 Message 对象
            msg.what = PROCESSING; //设置 id 为 1
            msg.getData().putInt("size", size);
            //把文件下载的 size 设置进 Message 对象
            handler.sendMessage(msg); //通过 handler 发送消息到消息队列
        }
    };

    /**
     * 下载线程的执行方法,会被系统自动调用
     */
    public void run() {
        try {

```

```

        loader = new FileDownloader(getApplicationContext(),
            path, saveDir, 3); //初始化下载
        progressBar.setMax(loader.getFileSize()); //设置进度条的最大刻度
        loader.download(downloadProgressListener);

    } catch (Exception e) {
        e.printStackTrace();
        handler.sendMessage(handler.obtainMessage(FAILURE));
        //下载失败时向消息队列发送消息
        /*Message message = handler.obtainMessage();
        message.what = FAILURE;*/
    }
}
}
}
}
}

```

(9) 在文件 `AndroidManifest.xml` 中申明使用网络的权限和操作 `SDCard` 的权限，具体实现代码如下。

```

<!-- 访问 internet 权限 -->
<uses-permission android:name="android.permission.INTERNET"/>
<!-- 在 SDCard 中创建与删除文件权限 -->
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
<!-- 往 SDCard 写入数据权限 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

```

至此，整个实例介绍完毕，执行后的效果如图 5-7 所示。



图 5-7 执行效果

5.6 远程下载并安装 APK 文件

APK 是 Android Package 的缩写，即 Android 安装包。APK 是类似 Symbian Sis 或 Sixx 的文件格式。通过将 APK 文件直接传到 Android 模拟器或 Android 手机中执行即可安装。在本节的内容中，将详细讲解在 Android 系统中下载并安装 APK 文件的基本方法。

5.6.1 APK 基础

APK 文件和 SIS 一样最终把 Android SDK 编译的工程打包成一个安装程序文件格式为 APK。APK 文件其实是 zip 格式，但扩展名被修改为 APK，通过 UnZip 解压后，可以看到 Dex 文件，Dex 是 Dalvik VM executes 的缩写，即 Android Dalvik 执行程序并非 Java ME 的字节码而是 Dalvik 字节码。一个 APK 文件结构为：META-INF\Jar，此文件结构的具体说明如下。

- “res\”：存放资源文件的目录。
- AndroidManifest.xml：程序全局配置文件。
- classes.dex：Dalvik 字节码。
- resources.arsc：编译后的二进制资源文件。

Android 在运行一个程序时，首先需要 UnZip 解压缩，这一点和 Symbian 比较相似，而和 Windows Mobile 中的 PE 文件有所区别。这样做使程序的保密性和可靠性不高，通过 dexdump 命令可以反编译，但这样做符合发展规律，微软的 Windows Gadgets 或者 WPF 也采用了这种构架方式。在 Android 平台中 dalvik vm 的执行文件被打包为 APK 格式，最终运行时加载器会解压然后获取编译后的 androidmanifest.xml 文件中的 permission 分支相关的安全访问，但仍然存在很多安全限制，如果将 APK 文件传到 “/system/app” 文件夹下会发现执行是不受限制的。最终平时安装的文件可能不在这个文件夹，而在 Android ROOM 中系统的 APK 文件默认会放入这个文件夹，它们拥有 ROOT 权限。

(1) 下载 APK 应用程序

用户可以从哪里取得好用的 Android APK 应用程序，并安装到 Android 手机上呢？对拥有 G1 实体手机的使用者而言，Android Market 就是最佳的地方，只要使用手机内应用程序列表的 Market 程序，就可以直接连接到 Android Market，而点选喜爱的应用程序后，就会直接下载并安装到 G1 手机上。不过对使用 Android 仿真器的使用者而言，就没有如此方便了，Android 仿真器并没有 Android Market 这个应用程序，只能使用内附的浏览器浏览 Android Market，为何说是浏览呢？因为 Android Market 不是采用通用网页浏览方式来下载文件的，虽然可以使用常见的浏览器看到 Android Market 上的应用程序，但是没有办法下载到 Android 仿真器或一般的计算机上，原因是 Android Market 采用特有的网页 API，使用 native UI 的方式来访问，唯有通过内建在 G1 手机内的 Market 应用程序，才能下载 Android Market 网页中的应用程序，并自动安装到 G1 手机上。

所以 Android 仿真器的使用者，只好浏览该网页上的应用程序，然后通过搜索引擎去寻找，看是否有开发人员将应用程序放到 Android Market 之后，还另外将 APK 文件放置在一般网页上。至此，使用 Android 仿真器的用户，也不需要灰心，因为有太多的人遇到同样的问题，也就生成非常多的 Android 应用程序网页。用户可以浏览这些网页并把上面的 APK 文件下载到一般计算机上，再安装到 Android 仿真器上。

常用的 APK 应用程序下载网站如下：

- <http://andappstore.com/>
- <http://www.getjar.com/software>
- <http://www.phoload.com/android>

- <http://slideme.org/>
- <http://androidforums.com/market/>
- <http://www.cyrket.com/>
- <http://www.androidfreeware.org/>
- <http://androidsoftwaredownload.com/>
- <http://www.freeandroidsoft.com/>
- <http://code.google.com/p/apps-for-android/>
- <http://code.google.com/p/openintents/downloads/list>

(2) 安装 APK 应用程序

所有的 APK 应用程序要安装到 Android 仿真器上，应使用如下 adb install 指令来开启一个命令字符的终端机窗口，并运行 APK 安装指令。

```
adb install filename.apk
```

这样 adb 指令就会自动将 filename.apk 应用程序安装到 Android 仿真器上，而仿真器上的应用程序列表也会立即出现刚刚安装的应用程序图标，如果应用程序没有安装成功，或安装不完善，也可以重复运行 adb install -r filename.apk 指令重新安装一次，这样会保留已经设置的信息，而仅需重新安装应用程序本身。

不过在运行 adb 安装 APK 应用程序组件时，不可以同时运行多个 Android 仿真器，因为 adb 会不知要将 APK 应用程序安装到哪一个仿真器，最好的方法就是仅运行一个 Android 仿真器。如果有同时运行多个仿真器的需要，就要在安装 APK 组件时，使用 adb 先指定某一个仿真器。用户可以从 Android 仿真器的窗口上，看到类似 Android Emulator (5554) 的字样，而 5554 就是仿真器的运行序号，每一个仿真器有其独特的运行序号，只要将 adb 加上-s <serialNumber> 参数，就可以指定 adb 将 APK 应用程序安装在仿真器上。

```
adb -s emulator-5554 install filename.apk
(指定安装 APK 组件在 5554 的 Android 仿真器中)
```

(3) 移除 APK 应用程序

如果已经安装了很多 Android 应用程序，要删除一些应用程序图标也非常简单，使用一行指令即可，adb uninstall 指令可以将 APK 应用程序移除。

```
adb uninstall package
```

代码如下：

```
adb uninstall com.android.email (把 email 程序移除)
```

Android 使用的 package 名称类似用户浏览网页时常用的域名方式，所以上面的示范是将 com.android.email 这个 email package 移除，记住，package 名称不是安装 APK 组件时的文件名或显示在 Android 仿真器中的应用程序名称。另外 Package 名称也并非一定都是 com.android 这样的形式，它可以是各式各样的域名方式来命名，如 org.iii.ro.iiivpa 或 com.deafcode.android.Cinema。APK 文件的 Package 名称完全是由当初的开发人员制定的，所以并没有统一的命名方式，唯一相同的就是它一定是类似 Domain 域名的命名格式。

另外，在移除该 APK 应用程序时，如果想要保留信息与 Cache 目录，则加上 -k 参数即可。

```
adb uninstall -k package
```

(移除程序时, 保留信息)

不过麻烦的是, 用户可能不知道这个想要移除的应用程序的 Package 名称, 所以必须先运行 adb shell 进入 Android 操作系统的指令列模式, 然后到 /data/data 或 /data/app 目录下, 得知欲移除的 Package 名称, 然后使用 adb uninstall 指令删除 APK 应用程序, 这样就可以简单地从 Android 仿真器将不想使用的 APK 应用程序移除。

```
adb shell
```

```
ls /data/data 或 /data/app
```

(查询 Package 名称)

```
exit
```

```
adb uninstall package
```

(移除查询到的 Package)

幸运的是, 从 Android SDK 1.5 版起, 已经内建应用程序管理系统, 不需要再辛苦地使用 adb uninstall 指令移除 APK 应用程序组件, 只要在 Android 手机主画面点选 MENU 按键, 然后依序点选“Settings→Applications→Manage applications”, 就可以启动应用程序管理系统。当前 Android 系统已经安装的所有应用程序都会条列出来, 只要点选想要移除的应用程序然后选择 Uninstall 就可以移除该程序, 这样就不需要使用 adb uninstall 指令来移除 Android 应用程序。

下载文件与打开网页是一样的, 打开网页是将内容显示出来, 保存文件就是保存到文件中。例如, 可以通过下面的代码将内容保存到 SD 卡等设备上。

```
public void downFile(String url, String path, String fileName)
    throws IOException {
    if (fileName == null || fileName == "")
        this.FileName = url.substring(url.lastIndexOf("/") + 1);
    else
        this.FileName = fileName; // 取得文件名, 如果输入新文件名, 则使用新文件名
    URL Url = new URL(url);
    URLConnection conn = Url.openConnection();
    conn.connect();
    InputStream is = conn.getInputStream();
    this.fileSize = conn.getContentLength(); // 根据响应获取文件大小
    if (this.fileSize <= 0) {                // 获取内容长度为 0
        throw new RuntimeException("无法获知文件大小 ");
    }
    if (is == null) {                        // 没有下载流
        sendMsg(Down_ERROR);
        throw new RuntimeException("无法获取文件");
    }
    FileOutputStream FOS = new FileOutputStream(path + this.FileName);
    // 创建写入文件内存流, 通过此流向目标写文件
    byte buf[] = new byte[1024];
    downloadFilePosition = 0;
    int numread;
    while ((numread = is.read(buf)) != -1) {
        FOS.write(buf, 0, numread);
        downloadFilePosition += numread
    }
    try {
        is.close();
```

```

    } catch (Exception ex) {
        ;
    }
}

```

5.6.2 实战演练——在 Android 系统中下载并安装 APK 文件

本实例的功能是远程下载指定网址的 Android 应用程序，下载到手机后打开 application installer 软件来安装这个软件。在具体实现上，先设置一个 EditText 来获取远程程序的 URL，按后通过自定义按钮打开下载程序（使用 java.net 的 URLConnection 对象来创建连接，通过 InputStream 将下载文件写入到存储卡的缓存）。下载后通过自定义方法 openFile() 打开文件，并根据文件扩展名，判断是否为 APK 格式，是则启动内置的 Install 程序，开始安装。安装完成后，在离开 Install 时通过方法 delFile() 将存储卡中的临时文件删除。

题 目	目 的	源码路径
实例 5-7	在 Android 系统中下载并安装 APK 文件	光盘\codes\5\xia

本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，主要代码如下。

```

<TextView
    android:id="@+id/myTextView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/str_text"
>
</TextView>
<EditText
    android:id="@+id/myEditText1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/str_url"
    android:textSize="18sp"
>
</EditText>
<Button
    android:id="@+id/myButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/str_button"
>
</Button>

```

(2) 编写主程序文件 xia.java，其具体实现流程如下。

① 单击按钮后设置将文件下载到 local 本地，获取要安装程序的文件名称。具体代码如下。

```

mButton01.setOnClickListener(new Button.OnClickListener()
{
    public void onClick(View v)

```

```

{
    /* 文件会下载至 local 端 */
    mTextView01.setText("下载中...");
    strURL = mEditText01.getText().toString();
    /*取得欲安装程序的文件名称*/
    fileEx = strURL.substring(strURL.lastIndexOf(".")
+1,strURL.length()).toLowerCase();
    fileNa = strURL.substring(strURL.lastIndexOf("/")
+1,strURL.lastIndexOf("."));
    getFile(strURL);
}
};

```

② 如果框中的远程地址为空，则输出“请输入 URL”的提示。具体代码如下。

```

mEditText01.setOnClickListener(new EditText.OnClickListener()
{
    @Override
    public void onClick(View arg0)
    {
        // TODO Auto-generated method stub
        mEditText01.setText("");
        mTextView01.setText("远程安装程序 (请输入 URL)");
    }
});

```

③ 定义方法 `getFile(final String strPath)` 来获取下载的 URL 文件，如果有异常则输出提示。具体代码如下。

```

/* 处理下载 URL 文件自定义函数 */
private void getFile(final String strPath) {
    try
    {
        if (strPath.equals(currentFilePath) )
        {
            getDataSource(strPath);
        }
        currentFilePath = strPath;
        Runnable r = new Runnable()
        {
            public void run()
            {
                try
                {
                    getDataSource(strPath);
                }
                catch (Exception e)
                {
                    Log.e(TAG, e.getMessage(), e);
                }
            }
        }
    }
}

```



```

    };
    new Thread(r).start();
}
catch(Exception e)
{
    e.printStackTrace();
}
}

```

④ 定义方法 `getDataSource` 来获取远程文件，主要代码如下。

```

/*取得远程文件*/
private void getDataSource(String strPath) throws Exception
{
    if (!URLUtil.isNetworkUrl(strPath))
    {
        mTextView01.setText("错误的 URL");
    }
    else
    {
        /*取得 URL*/
        URL myURL = new URL(strPath);
        /*创建连接*/
        URLConnection conn = myURL.openConnection();
        conn.connect();
        /*InputStream 下载文件*/
        InputStream is = conn.getInputStream();
        if (is == null)
        {
            throw new RuntimeException("stream is null");
        }
        /*创建临时文件*/
        File myTempFile = File.createTempFile(fileNa, "."+fileEx);
        /*取得暂存盘案路径*/
        currentTempFilePath = myTempFile.getAbsolutePath();
        /*将文件写入暂存盘*/
        FileOutputStream fos = new FileOutputStream(myTempFile);
        byte buf[] = new byte[128];
        do
        {
            int numread = is.read(buf);
            if (numread <= 0)
            {
                break;
            }
            fos.write(buf, 0, numread);
        }while (true);

        /*打开文件进行安装*/
        openFile(myTempFile);
        try

```

```

    {
        is.close();
    }
    catch (Exception ex)
    {
        Log.e(TAG, "error: " + ex.getMessage(), ex);
    }
}
}

```

⑤ 定义方法 `openFile(File f)`来设置在手机上打开文件，主要代码如下。

```

/* 在手机上打开文件的 method */
private void openFile(File f)
{
    Intent intent = new Intent();
    intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    intent.setAction(android.content.Intent.ACTION_VIEW);

    /* 调用 getMimeType() 来取得 MimeType */
    String type = getMimeType(f);
    /* 设置 intent 的 file 与 MimeType */
    intent.setDataAndType(Uri.fromFile(f), type);
    startActivity(intent);
}

/* 判断文件 MimeType 的 method */
private String getMimeType(File f)
{
    String type="";
    String fName=f.getName();
    /* 取得扩展名 */
    String end=fName.substring(fName.lastIndexOf(".")
    +1,fName.length()).toLowerCase();

    /* 依扩展名的类型决定 MimeType */
    if(end.equals("m4a")||end.equals("mp3")||end.equals("mid")||
    end.equals("xmf")||end.equals("ogg")||end.equals("wav"))
    {
        type = "audio";
    }
    else if(end.equals("3gp")||end.equals("mp4"))
    {
        type = "video";
    }
    else if(end.equals("jpg")||end.equals("gif")||end.equals("png")||
    end.equals("jpeg")||end.equals("bmp"))
    {
        type = "image";
    }
    else if(end.equals("apk"))
    {

```

```

        /* android.permission.INSTALL_PACKAGES */
        type = "application/vnd.android.package-archive";
    }
    else
    {
        type="*";
    }
    /*如果无法直接打开，就跳出软件列表给用户选择 */
    if(end.equals("apk"))
    {
    }
    else
    {
        type += "/*";
    }
    return type;
}

```

⑥ 定义方法 `delFile` 来删除 SD 卡上的临时文件，主要代码如下。

```

/*自定义删除文件方法*/
private void delFile(String strFileName)
{
    File myFile = new File(strFileName);
    if(myFile.exists())
    {
        myFile.delete();
    }
}

```

⑦ 定义方法 `onPause()` 和 `onResume()` 分别设置 `onPause` 暂停和 `onResume` 重新开始的状态。具体代码如下。

```

/*当 Activity 处于 onPause 状态时,更改 TextView 文字状态*/
@Override
protected void onPause()
{
    mTextView01 = (TextView)findViewById(R.id.myTextView1);
    mTextView01.setText("下载成功");
    super.onPause();
}
/*当 Activity 处于 onResume 状态时,删除临时文件*/
@Override
protected void onResume()
{
    // TODO Auto-generated method stub
    /* 删除临时文件 */
    delFile(currentTempFilePath);
    super.onResume();
}

```

执行后将在文本框中显示目标安装程序的路径，如图 5-8 所示。实例中的默认路径是 http://mz.ruan8.com/soft/2/sougoushoujishurufa_7786.apk，这是一个 sogou 输入法程序。单击

“安装”按钮后，开始下载目标文件，如图 5-9 所示。下载完成后弹出安装界面，单击 Install 按钮后开始安装，安装完成后输出提示。



图 5-8 下载目标文件



图 5-9 下载界面

上传数据

“上传”的反义词是“下载”，上传就是将信息从个人计算机（本地计算机）传递到中央计算机（远程计算机）系统上，让网络中的用户都能看到。将制作好的网页、文字、图片等发布到互联网上，以便让其他人浏览、欣赏。这一过程称为上传。本章的内容将详细讲解在 Android 系统中上传数据的基本知识，为读者学习本书后面的知识打下基础。

6.1 Android 上传数据技术

在 Android 系统中，实现上传文件功能的方法有两种，第一种是基于 HTTP 协议的 `HttpURLConnection`，第二种是基于 TCP 协议的 `Socket`。这两种方式的区别是使用 `HttpURLConnection` 上传时内部有缓存机制，如果上传较大文件会导致内存溢出。如果用 TCP 协议 `Socket` 方式上传就会解决这种弊端。

6.1.1 使用 HTTP 协议上传数据

在 Android 系统中，使用 HTTP 协议 `HttpURLConnection` 上传数据的具体流程如下所示。

- (1) 通过 URL 封装路径打开一个 `HttpURLConnection`。
- (2) 设置请求方式以及头字段：Content-Type、Content-Length、Host。
- (3) 拼接数据发送。

例如如下所示的演示代码。

```
private static final String BOUNDARY = "-----7db1c523809b2";//数据分割线

public boolean uploadHttpURLConnection(String username, String password,
String path) throws Exception {
    //找到 sdcard 上的文件
    File file = new File(Environment.getExternalStorageDirectory(), path);
    //仿 Http 协议发送数据方式进行拼接
    StringBuilder sb = new StringBuilder();
    sb.append("--" + BOUNDARY + "\r\n");
    sb.append("Content-Disposition: form-data; name=\"username\" + "\r\n");
    sb.append("\r\n");
    sb.append(username + "\r\n");

    sb.append("--" + BOUNDARY + "\r\n");
```



```

sb.append("Content-Disposition: form-data; name=\"password\"\" + "\r\n");
sb.append("\r\n");
sb.append(password + "\r\n");

sb.append("--" + BOUNDARY + "\r\n");
sb.append("Content-Disposition: form-data; name=\"file\"; filename=\""
+ path + "\"\" + "\r\n");
sb.append("Content-Type: image/jpeg\" + "\r\n");
sb.append("\r\n");

byte[] before = sb.toString().getBytes("UTF-8");
byte[] after = ("\r\n--" + BOUNDARY + "--\r\n").getBytes("UTF-8");

URL url = new URL("http://192.168.1.16:8080/14_Web/servlet/LoginServlet");
URLConnection conn = (URLConnection) url.openConnection();
conn.setRequestMethod("POST");
conn.setRequestProperty("Content-Type", "multipart/form-data; boundary="
+ BOUNDARY);
conn.setRequestProperty("Content-Length", String.valueOf(before.length
+ file.length() + after.length));
conn.setRequestProperty("HOST", "192.168.1.16:8080");
conn.setDoOutput(true);

OutputStream out = conn.getOutputStream();
InputStream in = new FileInputStream(file);

out.write(before);

byte[] buf = new byte[1024];
int len;
while ((len = in.read(buf)) != -1)
    out.write(buf, 0, len);

out.write(after);

in.close();
out.close();
return conn.getResponseCode() == 200;
}

```

6.1.2 使用 TCP 协议上传数据

在 Android 系统中，可以使用 Socket 发送 TCP 请求的方式将分段发送上传数据。在 Android 系统中，使用 HTTP 协议是不能实现断点上传的，根据文件大小与实际需求可以使用 Socket 断点上传。Android 客户端发送上传文件头字段给服务器，服务器判断文件是否在服务器上，文件是否有上传的记录。如果文件不存在，则服务器返回一个 id（断点数据）通知客户端从什么位置开始上传，客户端开始从获得的位置开始上传文件。例如如下所示的上传数据代码。

```

public boolean uploadBySocket(String username, String password, String path)
throws Exception {
    // 根据 path 找到 SDCard 中的文件
    File file = new File(Environment.getExternalStorageDirectory(), path);
    // 组装表单字段和文件之前的数据
    StringBuilder sb = new StringBuilder();

    sb.append("--" + BOUNDARY + "\r\n");
    sb.append("Content-Disposition: form-data; name=\"username\""+ "\r\n");
    sb.append("\r\n");
    sb.append(username + "\r\n");

    sb.append("--" + BOUNDARY + "\r\n");
    sb.append("Content-Disposition: form-data; name=\"password\""+ "\r\n");
    sb.append("\r\n");
    sb.append(password + "\r\n");

    sb.append("--" + BOUNDARY + "\r\n");
    sb.append("Content-Disposition: form-data; name=\"file\"; filename=\""
+ path + "\"" + "\r\n");
    sb.append("Content-Type: image/jpeg" + "\r\n");
    sb.append("\r\n");

    // 文件之前的数据
    byte[] before = sb.toString().getBytes("UTF-8");
    // 文件之后的数据
    byte[] after = ("\r\n--" + BOUNDARY + "--\r\n").getBytes("UTF-8");

    URL url = new URL("http://192.168.1.199:8080/14_Web/servlet/LoginServlet");

    // 由于 HttpURLConnection 中会缓存数据，上传较大文件时会导致内存溢出，所以我们使用 Socket 传输
    Socket socket = new Socket(url.getHost(), url.getPort());
    OutputStream out = socket.getOutputStream();
    PrintStream ps = new PrintStream(out, true, "UTF-8");

    // 写出请求头
    ps.println("POST /14_Web/servlet/LoginServlet HTTP/1.1");
    ps.println("Content-Type: multipart/form-data; boundary=" + BOUNDARY);
    ps.println("Content-Length: " + String.valueOf(before.length + file.length()
+ after.length));
    ps.println("Host: 192.168.1.199:8080");

    InputStream in = new FileInputStream(file);

    // 写出数据
    out.write(before);

    byte[] buf = new byte[1024];
    int len;

```

```

        while ((len = in.read(buf)) != -1)
            out.write(buf, 0, len);

        out.write(after);

        in.close();
        out.close();

        return true;
    }

```

然后搭建如下所示的服务器代码，以便实现数据上传功能。

```

package cn.test.web.servlet;

import java.io.File;
import java.io.IOException;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileItemFactory;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;

public class LoginServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doPost(request, response);
    }

    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        boolean isMultipart =
ServletFileUpload.isMultipartContent(request);
        if (isMultipart)
            try {
                FileItemFactory factory = new DiskFileItemFactory();
                ServletFileUpload upload = new ServletFileUpload(factory);
                List<FileItem> items = upload.parseRequest(request);
                File dir = new File(request.getSession().getServletContext().
getRealPath("/WEB-INF/upload"));

```

```

        //创建目录
        dir.mkdir();
        for (FileItem item : items)
            if (item.isFormField())
                System.out.println(item.getFieldName() + ": " + item.
getString());
            else{
                item.write(new File(dir,item.getName().substring
(item.getName().lastIndexOf("\\")+1)));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    } else {
        System.out.println(request.getMethod());
        System.out.println(request.getParameter("username"));
        System.out.println(request.getParameter("password"));
    }
}
}

```

6.2 实战演练——上传文件到远程服务器

在本节的内容中，将通过一个具体实例的实现过程，介绍在 Android 系统中实现文件上传的基本方法。

题 目	目 的	源码路径
实例 6-1	上传文件到远程服务器	光盘\daima\6\chuan

本实例的具体实现流程如下。

(1) 编写界面布局文件 main.xml，主要代码如下。

```

<TextView
    android:id="@+id/myText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/str_title"
    android:textSize="20sp"
    android:textColor="@drawable/black"
    android:layout_x="10px"
    android:layout_y="12px"
>
</TextView>
<TextView
    android:id="@+id/myText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="16sp"
    android:textColor="@drawable/black"

```

```

        android:layout_x="10px"
        android:layout_y="52px"
    >
</TextView>
<TextView
    android:id="@+id/myText3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="16sp"
    android:textColor="@drawable/black"
    android:layout_x="10px"
    android:layout_y="102px"
>
</TextView>
<Button
    android:id="@+id/myButton"
    android:layout_width="92px"
    android:layout_height="49px"
    android:text="@string/str_button"
    android:textSize="15sp"
    android:layout_x="90px"
    android:layout_y="170px"
>
</Button>

```

(2) 编写主程序文件 `chuan.java`，其具体实现流程如下。

① 分别声明变量 `newName`、`uploadFile` 和 `actionUrl`，具体代码如下。

```

public class chuan extends Activity
{
    /* 变量声明
    * newName: 上传后在服务器上的文件名称
    * uploadFile: 要上传的文件路径
    * actionUrl: 服务器上对应的程序路径 */
    private String newName="image.jpg";
    private String uploadFile="/data/data/irdc.example9/image.jpg";
    private String actionUrl="http://127.127.0.1/upload/upload.jsp";
    private TextView mText1;
    private TextView mText2;
    private Button mButton;
    .....
}

```

② 通过 `mText1` 对象获取文件路径，根据 `mText2` 设置上传网址，单击按钮后调用上传方法 `uploadFile()`。具体代码如下。

```

public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    mText1 = (TextView) findViewById(R.id.myText2);
    mText1.setText("文件路径: \n"+uploadFile);
    mText2 = (TextView) findViewById(R.id.myText3);
}

```



```

mText2.setText("上传网址: \n"+actionUrl);
/* 设置mButton的 onClick 事件处理 */
mButton = (Button) findViewById(R.id.myButton);
mButton.setOnClickListener(new View.OnClickListener()
{
    public void onClick(View v)
    {
        uploadFile();
    }
});
}

```

③ 定义方法 `uploadFile()` 将文件上传至 Server，具体代码如下。

```

/* 上传文件至 Server 的方法 */
private void uploadFile()
{
    String end = "\r\n";
    String twoHyphens = "--";
    String boundary = "*****";
    try
    {
        URL url = new URL(actionUrl);
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        /* 允许 Input、Output，不使用 Cache */
        con.setDoInput(true);
        con.setDoOutput(true);
        con.setUseCaches(false);
        /* 设置传送的 method=POST */
        con.setRequestMethod("POST");
        /* setRequestProperty */
        con.setRequestProperty("Connection", "Keep-Alive");
        con.setRequestProperty("Charset", "UTF-8");
        con.setRequestProperty("Content-Type",
            "multipart/form-data;boundary="+boundary);
        /* 设置 DataOutputStream */
        DataOutputStream ds =
            new DataOutputStream(con.getOutputStream());
        ds.writeBytes(twoHyphens + boundary + end);
        ds.writeBytes("Content-Disposition: form-data; " +
            "name=\"file1\";filename=\"" +
            newName + "\"" + end);
        ds.writeBytes(end);
        /* 取得文件的 FileInputStream */
        FileInputStream fStream = new FileInputStream(uploadFile);
        /* 设置每次写入 1024bytes */
        int bufferSize = 1024;
        byte[] buffer = new byte[bufferSize];
        int length = -1;
        /* 从文件读取数据至缓冲区 */
        while((length = fStream.read(buffer)) != -1)
        {
            /* 将资料写入 DataOutputStream 中 */

```

```

        ds.write(buffer, 0, length);
    }
    ds.writeBytes(end);
    ds.writeBytes(twoHyphens + boundary + twoHyphens + end);
    fStream.close();
    ds.flush();
    /* 取得 Response 内容 */
    InputStream is = con.getInputStream();
    int ch;
    StringBuffer b =new StringBuffer();
    while( ( ch = is.read() ) != -1 )
    {
        b.append( (char)ch );
    }
    /* 将 Response 显示在 Dialog 对话框中 */
    showDialog(b.toString().trim());
    /* 关闭 DataOutputStream */
    ds.close();
}
catch(Exception e)
{
    showDialog(""+e);
}
}

```

④ 定义方法 `showDialog(String mess)`来显示提示对话框，具体代码如下。

```

/* 显示 Dialog 的方法*/
private void showDialog(String mess)
{
    new AlertDialog.Builder(example9.this).setTitle("Message")
        .setMessage(mess)
        .setNegativeButton("确定",new DialogInterface.OnClickListener()
        {
            public void onClick(DialogInterface dialog, int which)
            {
            }
        })
        .show();
}
}

```

执行后单击“上传”按钮可以将指定的文件上传到服务器，如图 6-1 所示。



图 6-1 执行效果

6.3 使用 GET 方式上传数据

在 Andorid 系统中可以通过 GET 方式或 POST 方式上传数据，两者的具体区别如下。

- GET 上传的数据一般是很小的并且安全性能不高的数据。
- POST 上传的数据适用于数据量大、数据类型复杂、数据安全性能要求高的地方

在 Android 网络开发应用中，采用 GET 方式向服务器传递数据的基本步骤如下。

(1) 利用 Map 集合获取数据并进行数据处理，例如：

```
if (params!=null&&!params.isEmpty()) {
    for (Map.Entry<String, String> entry:params.entrySet()) {
        sb.append(entry.getKey()).append("=");
        sb.append(URLEncoder.encode(entry.getValue(),encoding));
        sb.append("&");
    }
    sb.deleteCharAt(sb.length()-1);
}
```

(2) 新建一个 StringBuilder 对象，例如：

```
sb=new StringBuilder()
```

(3) 新建一个 HttpURLConnection 的 URL 对象，打开连接并传递服务器的 path，例如：

```
connection=(HttpURLConnection) new URL(path).openConnection();
```

(4) 设置超时和连接的方式，例如：

```
connection.setConnectTimeout(5000);
connection.setRequestMethod("GET");
```

在本节的内容中，将通过一个具体实例的实现过程，介绍在 Android 系统中采用 GET 方式向服务器传递数据的基本方法。

题 目	目 的	源码路径
实例 6-2	在 Android 系统中采用 GET 方式向服务器传递数据	光盘\daima\6\get

本实例的具体实现流程如下所示。

(1) 打开 Eclipse，新建一个名为“ServerForGETMethod”的 Web 工程，并自动生成配置文件 web.xml。

(2) 创建一个名为 ServletForGETMethod 的 Servlet，功能是接收并处理通过 GET 方式上传的数据。实现文件 ServletForGETMethod.java 的具体代码如下。

```
@WebServlet("/ServletForGETMethod")
public class ServletForGETMethod extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String name= request.getParameter("name");
        //String name= new String(request.getParameter("name").getBytes("ISO8859-1"),"UTF-8");
        String age= request.getParameter("age");
        System.out.println("name: " + name );
        System.out.println("age: " + age );

    }
}
```

在上述代码中，为了避免出现中文乱码的问题，特意实现了 ISO8859-1 和 UTF-8 转换处理。下面的代码就很好地解决了乱码问题。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%
String zh_value=new String(request.getParameter("zh_value").getBytes("ISO-
8859-1"),"UTF-8")
%>
```

由此可见,在使用 get 方式传递数据时,需要使用如下代码声明当前页的字符集。

```
pageEncoding="UTF-8" //声明当前页的字符集
```

(3) 在配置文件 web.xml 中配置 ServletForGETMethod, 具体实现代码如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <display-name>ServerForGETMethod</display-name>
  <servlet>
    <display-name>ServletForGETMethod</display-name>
    <servlet-name>ServletForGETMethod</servlet-name>
    <servlet-class>com.guan.internet.servlet.ServletForGETMethod</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ServletForGETMethod</servlet-name>
    <url-pattern>/ServletForGETMethod</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

(4) 打开 Eclipse, 新建一个名为 “UserInformation” 的 Android 工程。然后编写界面布局文件 main.xml, 具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/title"
  />
  <EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/title"
```

```
    />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/length"
    />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:numeric="integer"
        android:id="@+id/length"
    />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button"
        android:onClick="save"
    />
</LinearLayout>
```

(5) 编写文件 `UserInfoInformationActivity.java`, 具体实现代码如下。

```
public class UserInfoInformationActivity extends Activity {
    private EditText titleText;
    private EditText lengthText;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        titleText = (EditText) this.findViewById(R.id.title);
        lengthText = (EditText) this.findViewById(R.id.length);
    }

    public void save(View v){
        String title = titleText.getText().toString();
        String length = lengthText.getText().toString();
        try {
            boolean result = false;

            result = UserInfoInformationService.save(title, length);

            if(result){
                Toast.makeText(this, R.string.success, 1).show();
            }else{
                Toast.makeText(this, R.string.fail, 1).show();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



```
Toast.makeText(this, R.string.fail, 1).show();
```

(6) 编写业务类的实现文件 `UserInfoenService.java`, 主要实现代码如下。

```
public class UserInfoenService {
    public static boolean save(String title, String length) throws Exception{
        String path =
"http://192.166.1.100:8080/ServerForGETMethod/ServletForGETMethod";
        Map<String, String> params = new HashMap<String, String>();
        params.put("name", title);
        params.put("age", length);
        return sendGETRequest(path, params, "UTF-8");
    }
    /**
     * 发送 GET 请求
     * @param path 请求路径
     * @param params 请求参数
     * @return
     */
    private static boolean sendGETRequest(String path, Map<String, String>
params, String encoding) throws Exception{
        // http://192.176.1.100:8080/ServerForGETMethod/ServletForGETMethod?title=
xxxx&length=90
        StringBuilder sb = new StringBuilder(path);
        if(params!=null && !params.isEmpty()){
            sb.append("?");
            for(Map.Entry<String, String> entry : params.entrySet()){
                sb.append(entry.getKey()).append("=");
                sb.append(URLEncoder.encode(entry.getValue(), encoding));
                sb.append("&");
            }
            sb.deleteCharAt(sb.length() - 1);
        }
        HttpURLConnection conn = (HttpURLConnection) new URL(sb.toString()).
openConnection();
        conn.setConnectTimeout(5000);
        conn.setRequestMethod("GET");
        if(conn.getResponseCode() == 200){
            return true;
        }
        return false;
    }
}
```

(7) 编写配置文件 `AndroidManifest.xml`, 申明网络访问权限, 主要代码如下。

```
<uses-sdk android:minSdkVersion="18" />
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name" >
```

```

<activity
    android:label="@string/app_name"
    android:name="com.guan.internet.userInfo.get.
    UserInfoActivity" >
    <intent-filter >
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
</application>
<uses-permission android:name="android.permission.INTERNET"/>
</manifest>

```

至此，整个实例讲解完毕，执行后的效果如图 6-2 所示。输入用户名和年龄后单击 save 按钮，会将输入的数据上传至服务器。

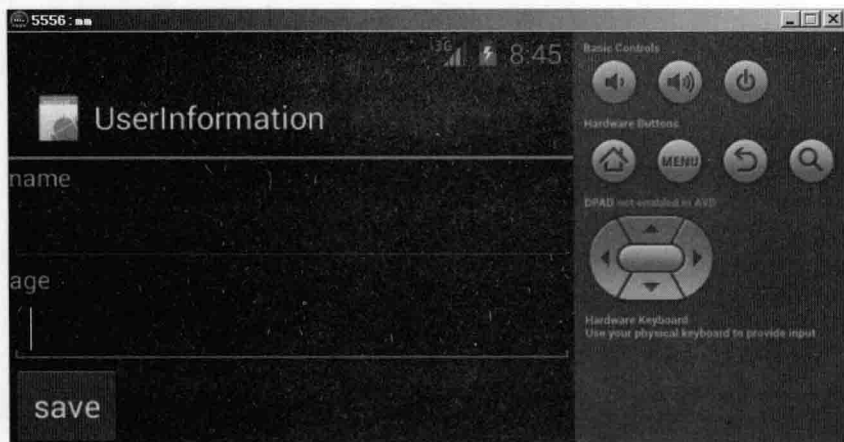


图 6-2 执行效果

6.4 使用 POST 方式上传数据

在 Android 网络应用中，采用 POST 方式向服务器传递数据的基本步骤如下。

(1) 利用 Map 集合获取数据并进行数据处理，例如：

```

if (params!=null&&!params.isEmpty()) {
    for (Map.Entry<String, String> entry:params.entrySet()) {
        sb.append(entry.getKey()).append("=");
        sb.append(URLEncoder.encode(entry.getValue(),encoding));
        sb.append("&");
    }
    sb.deleteCharAt(sb.length()-1);
}

```

(2) 新建一个 StringBuilder 对象，得到 POST 传给服务器的数据，例如：

```

sb=new StringBuilder()
byte[] data=sb.toString().getBytes();

```

(3) 新建一个 `URLConnection` 的 `URL` 对象, 打开连接并传递服务器的 `path`, 例如:

```
connection=(URLConnection) new URL(path).openConnection();
```

(4) 设置超时和允许对外连接数据, 例如:

```
connection.setDoOutput(true);
```

(5) 设置连接的 `setRequestProperty` 属性, 例如:

```
connection.setRequestProperty("Content-Type","application/x-www-form-urlencoded");
connection.setRequestProperty("Content-Length", data.length+"");
```

(6) 得到连接输出流, 例如:

```
outputStream =connection.getOutputStream();
```

(7) 把得到的数据写入输出流中并刷新, 例如:

```
outputStream.write(data);
outputStream.flush();
```

在接下来的内容中, 将通过一个具体实例的实现过程, 介绍在 `Android` 系统中采用 `POST` 方式向服务器传递数据的基本方法。

题 目	目 的	源码路径
实例 6-3	在 <code>Android</code> 系统中采用 <code>POST</code> 方式向服务器传递数据	光盘:\daima\6\post

本实例的具体实现流程如下。

(1) 打开 `Eclipse`, 新建一个名为 `ServerForPOSTMethod` 的 `Web` 工程, 并自动生成配置文件 `web.xml`。

(2) 创建一个名为 `ServletForPOSTMethod` 的 `Servlet`, 功能是接收并处理通过 `POST` 方式上传的数据。实现文件 `ServletForPOSTMethod.java` 的具体代码如下。

```
@WebServlet("/ServletForPOSTMethod")
public class ServletForPOSTMethod extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        String name= request.getParameter("name");
        String age= request.getParameter("age");
        System.out.println("name from POST method: " + name );
        System.out.println("age from POST method: " + age );
    }
}
```

(3) 在配置文件 `web.xml` 中配置 `ServletForGETMethod`, 具体实现代码如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
//java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/
web-app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <display-name>ServerForPOSTMethod</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
```

```

<welcome-file>index.htm</welcome-file>
<welcome-file>index.jsp</welcome-file>
<welcome-file>default.html</welcome-file>
<welcome-file>default.htm</welcome-file>
<welcome-file>default.jsp</welcome-file>
</welcome-file-list>
</web-app>

```

(4) 打开 Eclipse, 新建一个名称为 POST 的 Android 工程。然后编写界面布局文件 main.xml, 具体实现代码如下。

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/title"
    />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/title"
    />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/length"
    />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:numeric="integer"
        android:id="@+id/length"
    />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button"
        android:onClick="save"
    />
</LinearLayout>

```

(5) 编写文件 UploadUserInformationByPOSTActivity.java, 具体实现代码如下。

```

public class UploadUserInformationByPOSTActivity extends Activity {
    private EditText titleText;
    private EditText lengthText;
    @Override
    public void onCreate(Bundle savedInstanceState) {

```

```

super.onCreate(savedInstanceState);
setContentView(R.layout.main);

titleText = (EditText) this.findViewById(R.id.title);
lengthText = (EditText) this.findViewById(R.id.length);
}

public void save(View v){
    String title = titleText.getText().toString();
    String length = lengthText.getText().toString();
    try {
        boolean result = false;

        result = UploadUserInformationByPostService.save(title, length);

        if(result){
            Toast.makeText(this, R.string.success, 1).show();
        }else{
            Toast.makeText(this, R.string.fail, 1).show();
        }
    } catch (Exception e) {
        e.printStackTrace();
        Toast.makeText(this, R.string.fail, 1).show();
    }
}
}

```

(6) 编写业务类的实现文件 UploadUserInformationByPostService.java, 主要实现代码如下。

```

public class UploadUserInformationByPostService {
    public static boolean save(String title, String length) throws Exception{
        String path = "http://192.166.1.100:8080/ServerForPOSTMethod/ServletForPOSTMethod";
        Map<String, String> params = new HashMap<String, String>();
        params.put("name", title);
        params.put("age", length);
        return sendPOSTRequest(path, params, "UTF-8");
    }

    /**
     * 发送 POST 请求
     * @param path 请求路径
     * @param params 请求参数
     * @return
     */
    private static boolean sendPOSTRequest(String path, Map<String, String> params, String encoding) throws Exception{
        // title=liming&length=30
        StringBuilder sb = new StringBuilder();
        if(params!=null && !params.isEmpty()){

```



```

        for(Map.Entry<String, String> entry : params.entrySet()){
            sb.append(entry.getKey()).append("=");
            sb.append(URLEncoder.encode(entry.getValue(), encoding));
            sb.append("&");
        }
        sb.deleteCharAt(sb.length() - 1);
    }
    byte[] data = sb.toString().getBytes();

    HttpURLConnection conn = (HttpURLConnection) new URL(path).openConnection();
    conn.setConnectTimeout(5000);
    conn.setRequestMethod("POST");
    conn.setDoOutput(true); //允许对外传输数据
    conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
    conn.setRequestProperty("Content-Length", data.length+"");
    OutputStream outputStream = conn.getOutputStream();
    outputStream.write(data);
    outputStream.flush();
    if(conn.getResponseCode() == 200){
        return true;
    }
    return false;
}
}

```

(7) 编写配置文件 AndroidManifest.xml, 申明网络访问权限, 主要代码如下。

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.guan.internet.userInfo.post"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="8" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name="com.guan.internet.userInfo.post.
            UploadUserInfoByPOSTActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.INTERNET"/>
</manifest>

```

至此, 整个实例讲解完毕, 执行后的效果如图 6-3 所示。输入用户名和年龄后单击 save 按钮, 会将输入的数据上传至服务器。

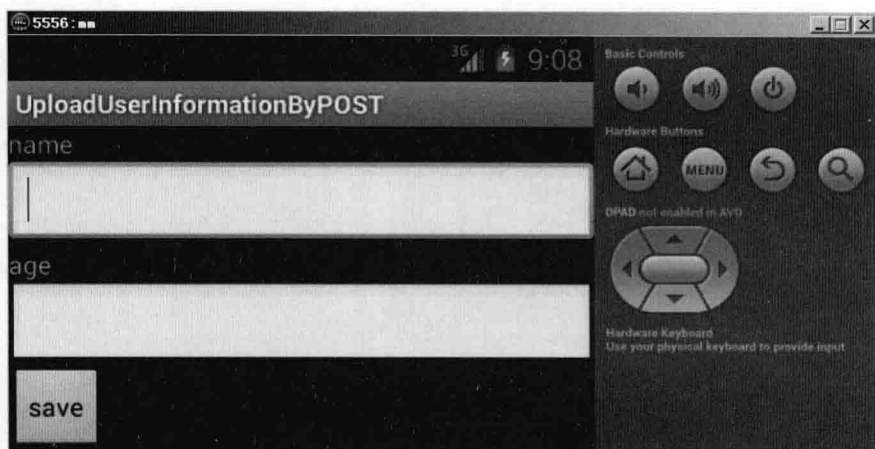


图 6-3 执行效果

6.5 使用 HTTP 协议实现上传

在现实中的网络应用中,http 协议上传的文件一般最大为 2MB,比较适合上传小于 2MB 的文件。在本节的内容中,将详细讲解在 Android 系统中使用 http 协议实现文件上传功能的方法。

6.5.1 一段演示代码

在 Android 系统中使用 http 协议实现文件上传的通用代码,内容如下。

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;

/**
 * 上传的文件
 */
public class FormFile {
    /** 上传文件的数据 */
    private byte[] data;
    private InputStream inStream;
    private File file;
    /** 文件名称 */
    private String filename;
    /** 请求参数名称*/
    private String parameterName;
    /** 内容类型 */
    private String contentType = "application/octet-stream";
    /**
     *
     * @param filename 文件名称
     */
}
```

```
* @param data 上传的文件数据
* @param parameterName 参数
* @param contentType 内容类型
*/
public FormFile(String filename, byte[] data, String parameterName, String
contentType) {
    this.data = data;
    this.filename = filename;
    this.parameterName = parameterName;
    if(contentType!=null) this.contentType = contentType;
}
/**
 *
 * @param filename 文件名
 * @param file 上传的文件
 * @param parameterName 参数
 * @param contentType 内容类型
 */
public FormFile(String filename, File file, String parameterName, String contentType) {
    this.filename = filename;
    this.parameterName = parameterName;
    this.file = file;
    try {
        this.inStream = new FileInputStream(file);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    if(contentType!=null) this.contentType = contentType;
}

public File getFile() {
    return file;
}

public InputStream getInStream() {
    return inStream;
}

public byte[] getData() {
    return data;
}

public String getFilename() {
    return filename;
}

public void setFilename(String filename) {
    this.filename = filename;
}
```

```

public String getParameterName() {
    return parameterName;
}

public void setParameterName(String parameterName) {
    this.parameterName = parameterName;
}

public String getContentType() {
    return contentType;
}

public void setContentType(String contentType) {
    this.contentType = contentType;
}
}

/**
 * 直接通过 HTTP 协议提交数据到服务器, 实现如下表单提交功能:
 * <FORM METHOD=POST ACTION="http://192.166.0.200:8080/ssi/fileload/test.do"
 *   enctype="multipart/form-data">
 *   <INPUT TYPE="text" NAME="name">
 *   <INPUT TYPE="text" NAME="id">
 *   <input type="file" name="imagefile"/>
 *   <input type="file" name="zip"/>
 * </FORM>
 * @param path 上传路径(注: 避免使用 localhost 或 127.0.0.1 这样的路径测试, 因为它会
 *   指向手机模拟器, 可以使用 http://www.xxx.cn 或 http://192.166.1.10:8080 这样的路径测试)
 * @param params 请求参数 key 为参数名, value 为参数值
 * @param file 上传文件
 */
public static boolean post(String path, Map<String, String> params, FormFile[]
files) throws Exception{
    final String BOUNDARY = "-----7da2137580612";
    //数据分隔线
    final String endlne = "--" + BOUNDARY + "--\r\n"; //数据结束标志

    int fileDataLength = 0;
    for(FormFile uploadFile : files){ //得到文件类型数据的总长度
        StringBuilder fileExplain = new StringBuilder();
        fileExplain.append("--");
        fileExplain.append(BOUNDARY);
        fileExplain.append("\r\n");
        fileExplain.append("Content-Disposition: form-data; name=\"" + uploadFile.
getParameterName()+"\"; filename=\"" + uploadFile.getFilename() + "\"\r\n");
        fileExplain.append("Content-Type: " + uploadFile.getContentType()+"\r\n\r\n");
        fileExplain.append("\r\n");
        fileDataLength += fileExplain.length();
        if(uploadFile.getInStream()!=null){

```

```

        fileDataLength += uploadFile.getFile().length();
    }else{
        fileDataLength += uploadFile.getData().length;
    }
}
StringBuilder textEntity = new StringBuilder();
for (Map.Entry<String, String> entry : params.entrySet()) {
    //构造文本类型参数的实体数据
    textEntity.append("--");
    textEntity.append(BOUNDARY);
    textEntity.append("\r\n");
    textEntity.append("Content-Disposition: form-data; name=\"" + entry.getKey() +
        "\"\r\n\r\n");
    textEntity.append(entry.getValue());
    textEntity.append("\r\n");
}
//计算传输给服务器的实体数据总长度
int dataLength = textEntity.toString().getBytes().length + fileDataLength
    + endlene.getBytes().length;

URL url = new URL(path);
int port = url.getPort() == -1 ? 80 : url.getPort();
Socket socket = new Socket(InetAddress.getByName(url.getHost()), port);
OutputStream outStream = socket.getOutputStream();
//下面完成 HTTP 请求头的发送
String requestmethod = "POST " + url.getPath() + " HTTP/1.1\r\n";
outStream.write(requestmethod.getBytes());
String accept = "Accept: image/gif, image/jpeg, image/pjpeg, image/pjpeg,
application/x-shockwave-flash, application/xaml+xml, application/vnd.ms-
xpsdocument, application/x-ms-xbap, application/x-ms-application, application/
vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*\r\n";
outStream.write(accept.getBytes());
String language = "Accept-Language: zh-CN\r\n";
outStream.write(language.getBytes());
String contenttype = "Content-Type: multipart/form-data; boundary="+
    BOUNDARY+ "\r\n";
outStream.write(contenttype.getBytes());
String contentlength = "Content-Length: " + dataLength + "\r\n";
outStream.write(contentlength.getBytes());
String alive = "Connection: Keep-Alive\r\n";
outStream.write(alive.getBytes());
String host = "Host: " + url.getHost() + ":" + port + "\r\n";
outStream.write(host.getBytes());
//写完 HTTP 请求头后根据 HTTP 协议再写一个回车换行
outStream.write("\r\n".getBytes());
//把所有文本类型的实体数据发送出来
outStream.write(textEntity.toString().getBytes());
//把所有文件类型的实体数据发送出来
for (FormFile uploadFile : files){
    StringBuilder fileEntity = new StringBuilder();

```



```

fileEntity.append("--");
fileEntity.append(BOUNDARY);
fileEntity.append("\r\n");
fileEntity.append("Content-Disposition: form-data; name=\"" + uploadFile.
getParameterName() + "\"; filename=\"" + uploadFile.getFilename() + "\"\r\n");
fileEntity.append("Content-Type: " + uploadFile.getContentType() + "\r\n\r\n");
outStream.write(fileEntity.toString().getBytes());
if(uploadFile.getInStream()!=null){
    byte[] buffer = new byte[1024];
    int len = 0;
    while((len = uploadFile.getInStream().read(buffer, 0, 1024))!=-1){
        outStream.write(buffer, 0, len);
    }
    uploadFile.getInStream().close();
}else{
    outStream.write(uploadFile.getData(), 0, uploadFile.getData().length);
}
outStream.write("\r\n".getBytes());
}
//下面发送数据结束标志, 表示数据已经结束
outStream.write(endline.getBytes());

BufferedReader reader = new BufferedReader(new InputStreamReader
(socket.getInputStream()));
if(reader.readLine().indexOf("200")==-1){
    //读取 Web 服务器返回的数据, 判断请求码是否为 200, 如果不是 200, 代表请求失败
    return false;
}
outStream.flush();
outStream.close();
reader.close();
socket.close();
return true;
}

/**
 * 将数据提交到服务器
 * @param path 上传路径(注: 避免使用 localhost 或 127.0.0.1 这样的路径测试, 因为它会
指向手机模拟器, 可以使用 http://www.xxx.cn 或 http://192.166.1.10:8080 这样的路径测试)
 * @param params 请求参数 key 为参数名, value 为参数值
 * @param file 上传文件
 */
public static boolean post(String path, Map<String, String> params, FormFile
file) throws Exception{
    return post(path, params, new FormFile[]{file});
}

```

再看如下演示代码。

```

/**
 * 通过拼接的方式构造请求内容, 实现参数传输及文件传输

```

```

* @param actionUrl
* @param params
* @param files
* @return
* @throws IOException
*/
public static String post(String actionUrl, Map<String, String> params,
    Map<String, File> files) throws IOException {

    String BOUNDARY = java.util.UUID.randomUUID().toString();
    String PREFIX = "--" , LINEND = "\r\n";
    String MULTIPART_FROM_DATA = "multipart/form-data";
    String CHARSET = "UTF-8";

    URL uri = new URL(actionUrl);
    HttpURLConnection conn = (HttpURLConnection) uri.openConnection();
    conn.setReadTimeout(5 * 1000);    // 缓存的最长时间
    conn.setDoInput(true);            // 允许输入
    conn.setDoOutput(true);           // 允许输出
    conn.setUseCaches(false);         // 不允许使用缓存
    conn.setRequestMethod("POST");
    conn.setRequestProperty("connection", "keep-alive");
    conn.setRequestProperty("Charset", "UTF-8");
    conn.setRequestProperty("Content-Type", MULTIPART_FROM_DATA + ";boundary="
+ BOUNDARY);

    // 首先组拼文本类型的参数
    StringBuilder sb = new StringBuilder();
    for (Map.Entry<String, String> entry : params.entrySet()) {
        sb.append(PREFIX);
        sb.append(BOUNDARY);
        sb.append(LINEND);
        sb.append("Content-Disposition: form-data; name=\"" + entry.getKey() +
            "\"" + LINEND);
        sb.append("Content-Type: text/plain; charset=" + CHARSET+LINEND);
        sb.append("Content-Transfer-Encoding: 8bit" + LINEND);
        sb.append(LINEND);
        sb.append(entry.getValue());
        sb.append(LINEND);
    }

    DataOutputStream outputStream = new DataOutputStream(conn.getOutputStream());
    outputStream.write(sb.toString().getBytes());
    // 发送文件数据
    if (files!=null){
        int i = 0;
        for (Map.Entry<String, File> file: files.entrySet()) {
            StringBuilder sb1 = new StringBuilder();
            sb1.append(PREFIX);
            sb1.append(BOUNDARY);

```

```

        sb1.append(LINEND);
        sb1.append("Content-Disposition: form-data; name=\"file\"+(i++)+\"\"";
        filename=\"\"+file.getKey()+"\""+LINEND);
        sb1.append("Content-Type: application/octet-stream; charset="+CHARSET+
        LINEND);
        sb1.append(LINEND);
        outputStream.write(sb1.toString().getBytes());

        InputStream is = new FileInputStream(file.getValue());
        byte[] buffer = new byte[1024];
        int len = 0;
        while ((len = is.read(buffer)) != -1) {
            outputStream.write(buffer, 0, len);
        }

        is.close();
        outputStream.write(LINEND.getBytes());
    }
}

//请求结束标志
byte[] end_data = (PREFIX + BOUNDARY + PREFIX + LINEND).getBytes();
outStream.write(end_data);
outStream.flush();

//得到响应码
int res = conn.getResponseCode();
InputStream in = null;
if (res == 200) {
    in = conn.getInputStream();
    int ch;
    StringBuilder sb2 = new StringBuilder();
    while ((ch = in.read()) != -1) {
        sb2.append((char) ch);
    }
}
return in == null ? null : in.toString();
}
}

```

在上述代码中, 通过使用 http 协议在 Android 系统中实现了文件上传功能, 可以用如下 PHP 代码来测试上述代码。

```

if($_FILES){
    foreach($_FILES as $v){
        copy($v[tmp_name], $v[name]);
    }
}
}

```

6.5.2 实战演练——HTTP 协议实现文件上传

在接下来的内容中, 将详细讲解使用 TTP 协议在 Android 系统中实现文件上传的具体流程。

(1) 编写服务器端的测试程序，使用 PHP 语言实现，具体代码如下。

```
<?php
$base_path = "./uploads/"; //接收文件目录
$target_path = $base_path . basename ( $_FILES ['uploadfile'] ['name'] );
if (move_uploaded_file ( $_FILES ['uploadfile'] ['tmp_name'], $target_path )) {
    $array = array ("code" => "1", "message" => $_FILES ['uploadfile'] ['name'] );
    echo json_encode ( $array );
} else {
    $array = array ("code" => "0", "message" => "There was an error uploading
the file, please try again!" . $_FILES ['uploadfile'] ['error'] );
    echo json_encode ( $array );
}
?>
```

(2) 编写客户端的上传处理程序，分别采用了人造 POST 请求、httpclient4（需要 httpmime-4.1.3.jar）和 AsyncHttpClient（对 apache 的 HttpClient 进行了进一步封装）三种处理方式。为了让整个程序在对比上显得比较方便，所以特意在主线程中实现了前两种上传功能。具体代码如下。

```
package com.example.fileupload;
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.HttpVersion;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.mime.MultipartEntity;
import org.apache.http.entity.mime.content.FileBody;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.params.CoreProtocolPNames;
import org.apache.http.util.EntityUtils;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
```

```

import com.loopj.android.http.AsyncHttpClient;
import com.loopj.android.http.AsyncHttpResponseHandler;
import com.loopj.android.http.RequestParams;

/**
 *
 * ClassName:UploadActivity Function: TODO 测试上传<strong>文件</strong>, PHP
 服务器端<strong>接收</strong> Reason: TODO ADD
 */
public class UploadActivity extends Activity implements OnClickListener {
    private final String TAG = "UploadActivity";

    private static final String path = "/mnt/sdcard/Desert.jpg";
    private String uploadUrl = "http://192.166.1.102:8080/Android/testupload.php";
    private Button btnAsync, btnHttpClient, btnCommonPost;
    private AsyncHttpClient client;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_upload);
        initView();
        client = new AsyncHttpClient();
    }

    private void initView() {
        btnCommonPost = (Button) findViewById(R.id.button1);
        btnHttpClient = (Button) findViewById(R.id.button2);
        btnAsync = (Button) findViewById(R.id.button3);
        btnCommonPost.setOnClickListener(this);
        btnHttpClient.setOnClickListener(this);
        btnAsync.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        long startTime = System.currentTimeMillis();
        String tag = null;
        try {
            switch (v.getId()) {
                case R.id.button1:
                    uploadByCommonPost();
                    tag = "CommonPost====>";
                    break;
                case R.id.button2:
                    uploadByHttpClient4();
                    tag = "HttpClient====>";
                    break;
                case R.id.button3:
                    uploadByAsyncHttpClient();

```



```

        tag = "AsyncHttpClient====>";
        break;
    default:
        break;
    }
} catch (Exception e) {
    e.printStackTrace();
}

Log.i(TAG, tag + "wasteTime = "
    + (System.currentTimeMillis() - startTime));
}

/**
 * 造 post 上传
 * @return void
 * @throws IOException
 * @throws
 */
private void upLoadByCommonPost() throws IOException {
    String end = "\r\n";
    String twoHyphens = "--";
    String boundary = "*****";
    URL url = new URL(uploadUrl);
    HttpURLConnection httpURLConnection = (HttpURLConnection) url
        .openConnection();
    httpURLConnection.setChunkedStreamingMode(128 * 1024); // 128K
    // 允许输入、输出流
    httpURLConnection.setDoInput(true);
    httpURLConnection.setDoOutput(true);
    httpURLConnection.setUseCaches(false);
    // 使用 POST 方法
    httpURLConnection.setRequestMethod("POST");
    httpURLConnection.setRequestProperty("Connection", "Keep-Alive");
    httpURLConnection.setRequestProperty("Charset", "UTF-8");
    httpURLConnection.setRequestProperty("Content-Type",
        "multipart/form-data;boundary=" + boundary);

    DataOutputStream dos = new DataOutputStream(
        httpURLConnection.getOutputStream());
    dos.writeBytes(twoHyphens + boundary + end);
    dos.writeBytes("Content-Disposition: form-data; name=\"uploadfile\"; filename=\"\"
        + path.substring(path.lastIndexOf("/") + 1) + "\"" + end);
    dos.writeBytes(end);

    FileInputStream fis = new FileInputStream(path);
    byte[] buffer = new byte[8192]; // 8k
    int count = 0;
    // 读取文件

```

```

while ((count = fis.read(buffer)) != -1) {
    dos.write(buffer, 0, count);
}
fis.close();
dos.writeBytes(end);
dos.writeBytes(twoHyphens + boundary + twoHyphens + end);
dos.flush();
InputStream is = httpURLConnection.getInputStream();
InputStreamReader isr = new InputStreamReader(is, "utf-8");
BufferedReader br = new BufferedReader(isr);
String result = br.readLine();
Log.i(TAG, result);
dos.close();
is.close();
}

/**
 * upLoadByAsyncHttpClient:由 HttpClient4 上传
 *
 * @return void
 * @throws IOException
 * @throws ClientProtocolException
 */
private void upLoadByHttpClient4() throws ClientProtocolException,
    IOException {
    HttpClient httpclient = new DefaultHttpClient();
    httpclient.getParams().setParameter(
        CoreProtocolPNames.PROTOCOL_VERSION, HttpVersion.HTTP_1_1);
    HttpPost httppost = new HttpPost(uploadUrl);
    File file = new File(path);
    MultipartEntity entity = new MultipartEntity();
    FileBody fileBody = new FileBody(file);
    entity.addPart("uploadfile", fileBody);
    httppost.setEntity(entity);
    HttpResponse response = httpclient.execute(httppost);
    HttpEntity resEntity = response.getEntity();
    if (resEntity != null) {
        Log.i(TAG, EntityUtils.toString(resEntity));
    }
    if (resEntity != null) {
        resEntity.consumeContent();
    }
    httpclient.getConnectionManager().shutdown();
}

/**
 * upLoadByAsyncHttpClient:由 AsyncHttpClient 框架上传
 *
 * @return void
 * @throws FileNotFoundException
 * @throws

```

```

        * @since CodingExample Ver 1.1
        */
        private void upLoadByAsyncHttpClient() throws FileNotFoundException {
            RequestParams params = new RequestParams();
            params.put("uploadfile", new File(path));
            client.post(uploadUrl, params, new AsyncHttpResponseHandler() {
                @Override
                public void onSuccess(int arg0, String arg1) {
                    super.onSuccess(arg0, arg1);
                    Log.i(TAG, arg1);
                }
            });
        }
    }
}

```

其实在 Andriod 系统中，用户最为熟知的上传是由 apache 提供给 httpclient4 实现的，例如，腾讯微博的 SDK 就是基于此实现的，演示代码如下。

```

/**
 * Post 方法传送<strong>文件</strong>和消息
 * @param url 连接的 URL
 * @param queryString 请求参数串
 * @param files 上传的<strong>文件</strong>列表
 * @return 服务器返回的信息
 */
public String httpPostWithFile(String url, String queryString, List
<NameValuePair> files) throws Exception {
    String responseData = null;
    URI tmpUri=new URI(url);
    URI uri = URIUtils.createURI(tmpUri.getScheme(), tmpUri.getHost(),
    tmpUri.getPort(), tmpUri.getPath(),
        queryString, null);
    Log.i(TAG, "QHHttpClient httpPostWithFile [1] uri = "+uri.toURL());
    MultipartEntity mpEntity = new MultipartEntity();
    HttpPost httpPost = new HttpPost(uri);
    StringBody stringBody;
    FileBody fileBody;
    File targetFile;
    String filePath;
    FormBodyPart fbp;

    List<NameValuePair> queryParamList=QStrOperate.getQueryParamsList
    (queryString);
    for(NameValuePair queryParam:queryParamList){
        stringBody=new StringBody(queryParam.getValue(),Charset.forName
        ("UTF-8"));
        fbp= new FormBodyPart(queryParam.getName(), stringBody);
        mpEntity.addPart(fbp);
    }
    // Log.i(TAG, "----- "+queryParam.getName()+" = "+queryParam.getValue());
}

```

```

for (NameValuePair param : files) {
    filePath = param.getValue();
    targetFile= new File(filePath);
    fileBody = new FileBody(targetFile,"application/octet-stream");
    fbp= new FormBodyPart(param.getName(), fileBody);
    mpEntity.addPart(fbp);
}

// Log.i(TAG, "----- Entity Content Type = "+mpEntity.getContentType());

httpPost.setEntity(mpEntity);

try {
    HttpResponse response=httpClient.execute(httpPost);
    Log.i(TAG, "QHttpClient httpPostWithFile [2] StatusLine = "+response.
        getStatusLine());
    responseData =EntityUtils.toString(response.getEntity());
} catch (Exception e) {
    e.printStackTrace();
}finally{
    httpPost.abort();
}
Log.i(TAG, "QHttpClient httpPostWithFile [3] responseData = "+responseData);
return responseData;
}

```

通过上述代码的分析测试可知，使用开源框架上传方式是最为简单有效的实现方法，而且是异步的，能够提供 Handler 来返回上传结果。而普通人造 post 方式的实现方式需要编写很多代码，并且上传速度一般，例如，上传 850KB 大小图片的测试结果如图 6-4 所示。而使用 asyncHttpClient 上传的方式的速度尚可，并且实现方法比较简单。

PID	TID	Application	Tag	Text
544	544	com.example.fileu...	UploadActivity	{"code":"1","message":"Desert.jpg"}
544	544	com.example.fileu...	UploadActivity	CommonPost====>wasteTime = 273
544	544	com.example.fileu...	UploadActivity	{"code":"1","message":"Desert.jpg"}
544	544	com.example.fileu...	UploadActivity	HttpClient====>wasteTime = 332
544	544	com.example.fileu...	dalvikvm-heap	Grow heap (frag case) to 9.200MB for 1040
544	544	com.example.fileu...	UploadActivity	AsyncHttpClient====>wasteTime = 348
544	544	com.example.fileu...	UploadActivity	{"code":"1","message":"Desert.jpg"}
544	544	com.example.fileu...	UploadActivity	{"code":"1","message":"Desert.jpg"}
544	544	com.example.fileu...	UploadActivity	CommonPost====>wasteTime = 186
544	544	com.example.fileu...	UploadActivity	{"code":"1","message":"Desert.jpg"}
544	544	com.example.fileu...	UploadActivity	HttpClient====>wasteTime = 300
544	544	com.example.fileu...	UploadActivity	AsyncHttpClient====>wasteTime = 218
544	544	com.example.fileu...	UploadActivity	{"code":"1","message":"Desert.jpg"}
544	544	com.example.fileu...	UploadActivity	{"code":"1","message":"Desert.jpg"}
544	544	com.example.fileu...	UploadActivity	CommonPost====>wasteTime = 263
544	544	com.example.fileu...	UploadActivity	{"code":"1","message":"Desert.jpg"}
544	544	com.example.fileu...	UploadActivity	HttpClient====>wasteTime = 436
544	544	com.example.fileu...	UploadActivity	AsyncHttpClient====>wasteTime = 236
544	555	com.example.fileu...	dalvikvm-heap	Grow heap (frag case) to 10.314MB for 840
544	544	com.example.fileu...	UploadActivity	{"code":"1","message":"Desert.jpg"}

图 6-4 测试截图

传感器技术

传感器是近年来随着物联网这一概念的流行而推出的，现在人们已经逐渐了解了传感器的概念。其实传感器在人们的日常生活中很常见甚至经常用到，比如楼宇的声控楼梯灯和马路上的路灯等。本章将详细讲解在 Android 系统中开发传感器应用的基本知识，为读者学习本书后面的知识打下基础。

7.1 Android 传感器系统概述

在 Android 系统中提供的主要传感器有：加速度传感器、磁场传感器、方向传感器、陀螺仪传感器、光线传感器、压力传感器、温度传感器和接近传感器等。传感器系统会主动向上层报告传感器精度和数据的变化，并且提供了设置传感器精度的接口，这些接口可以在 Java 应用和 Java 框架中使用。

Android 传感器系统的基本层次结构如图 7-1 所示。

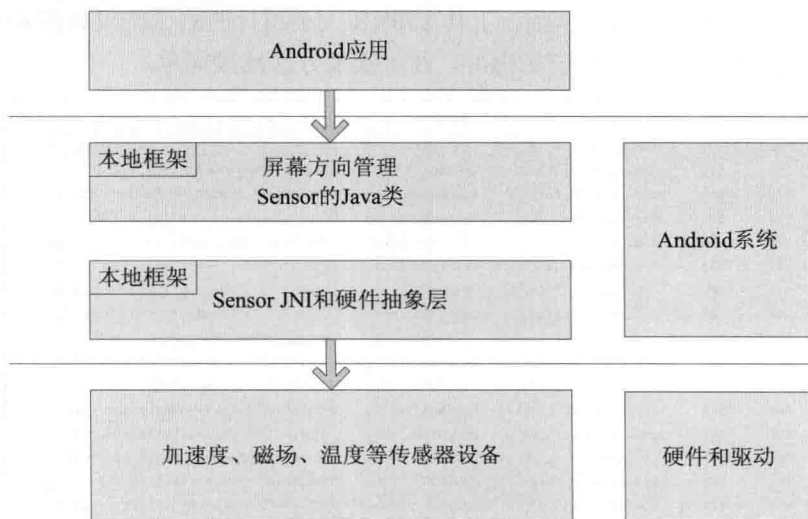


图 7-1 传感器系统的基本层次结构

根据图 7-1 所示的结构，Android 传感器系统从上到下分别是：Java 应用层、Java 框架层（包括各个传感器类）、硬件抽象层、传感器驱动层。各个层的具体说明如下。

(1) 传感器系统的 Java 部分

代码路径是：frameworks/base/include/core/java/android/hardware，对应的实现文件是 Sensor*.java。

(2) 传感器系统的 JNI 部分

代码路径是：frameworks/base/core/jni/android_hardware_SensorManager.cpp，在此部分中提供了对类 android.hardware.Sensor.Manage 的本地支持。

(3) 传感器系统 HAL 层

头文件路径是：hardware/libhardware/include/hardware/sensors.h，传感器系统的硬件抽象层需要开发人员自行编写程序实现。

(4) 驱动层

驱动层的代码路径是：kernel/driver/hwmon/\$(PROJECT)/sensor。

在库 sensor.so 中提供了如下 8 个 API 函数。

- 控制方面：在结构体 sensors_control_device_t 中定义，包括如下函数。
 - int (*open_data_source)(struct sensors_control_device_t *dev)
 - int (*activate)(struct sensors_control_device_t *dev, int handle, int enabled)
 - int (*set_delay)(struct sensors_control_device_t *dev, int32_t ms)
 - int (*wake)(struct sensors_control_device_t *dev)
- 数据方面：在结构体 sensors_data_device_t 中定义，包括如下函数。
 - int (*data_open)(struct sensors_data_device_t *dev, int fd)
 - int (*data_close)(struct sensors_data_device_t *dev)
 - int (*poll)(struct sensors_data_device_t *dev, sensors_data_t* data)
- 模块方面：在结构体 sensors_module_t 中定义，包括如下函数。

int (*get_sensors_list)(struct sensors_module_t* module, struct sensor_t const** list)

在 Java 层 Sensor 的状态是由 SensorService 来负责控制的，其 Java 代码和 JNI 代码分别位于如下文件中。

```
frameworks/base/services/java/com/android/server/SensorService.java
frameworks/base/services/jni/com_android_server_SensorService.cpp
```

SensorManager 负责在 Java 层 Sensor 的数据控制，它的 Java 代码和 JNI 代码分别位于如下文件中。

```
frameworks/base/core/java/android/hardware/SensorManager.java
frameworks/base/core/jni/android_hardware_SensorManager.cpp
```

在 Android 的 Framework 中，通过文件 sensorService.java 和 sensorManager.java 实现与 Sensor 传感器通信。文件 sensorService.java 的通信功能是通过 JNI 调用 sensorService.cpp 中的方法实现的。

文件 sensorManager.java 的具体通信功能是通过 JNI 调用 sensorManager.cpp 中的方法实现的。文件 sensorService.cpp 和 sensorManger.cpp 通过文件 hardware.c 与 sensor.so 通信。其中文件 sensorService.cpp 实现对 sensor 的状态控制，文件 sensorManger.cpp 实现对 sensor 的数据控制。

库 sensor.so 通过 ioctl 控制 sensor driver 的状态，通过打开 sensor driver 对应的设备文件读取 G-sensor 采集的数据。

7.2 使用 SensorSimulator

在进行和传感器相关的开发工作时，使用 SensorSimulator 可以提高开发效率。SensorSimulator 是一个开源免费的传感器小型工具，通过该工具可以在模拟器中调试传感器的应用。搭建 SensorSimulator 开发环境的基本流程如下。

(1) 下载 SensorSimulator，读者可从 <http://code.google.com/p/openintents/wiki/SensorSimulator> 网站找到该工具的下载链接。笔者下载的是 sensorsimulator-1.1.1.zip 版本，如图 7-2 所示。



Filename ▼	Summary + Labels ▼
☆  sensorsimulator-2.0-rc1.zip	SensorSimulator 2.0-rc1
☆  sensorsimulator-1.1.1.zip	SensorSimulator 1.1.0

图 7-2 下载 sensorsimulator-1.1.1.zip

(2) 将下载好的 SensorSimulator 解压到本地根目录，如 C 盘的根目录。

(3) 向模拟器安装 SensorSimulatorSettings-1.1.1.apk。首先在操作系统中选择【开始】|【运行】命令进入“运行”对话框。

(4) 在“运行”对话框中输入 cmd 进入 cmd 命令行，之后通过 cd 命令将当前目录导航到 SensorSimulatorSettings-1.1.1.apk 目录下，然后输入下列命令向模拟器安装该 apk。

```
adb install SensorSimulatorSettings-1.1.1.apk
```

在此需要注意的是，安装 apk 时，一定要保证模拟器正在运行才可以，安装成功后会输出 Success 提示，如图 7-3 所示。

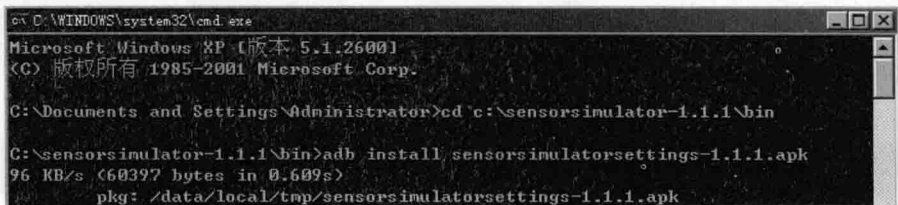


图 7-3 安装 apk

接下来开始配置应用程序，假设我们要在项目 jiaSCH 中使用 SensorSimulator，则配置流程如下。

(1) 在 Eclipse 中打开项目 jiaSCH，然后为该项目添加 JAR 包，使其能够使用 SensorSimulator 工具的种类和方法。添加方法非常简单，在 Eclipse 的 Package Explorer 中找到该项目的文件夹 jiaSCH，然后右击该文件夹并选择 Properties 选项，弹出如图 7-4 所示的 Properties for jiaS 窗口。

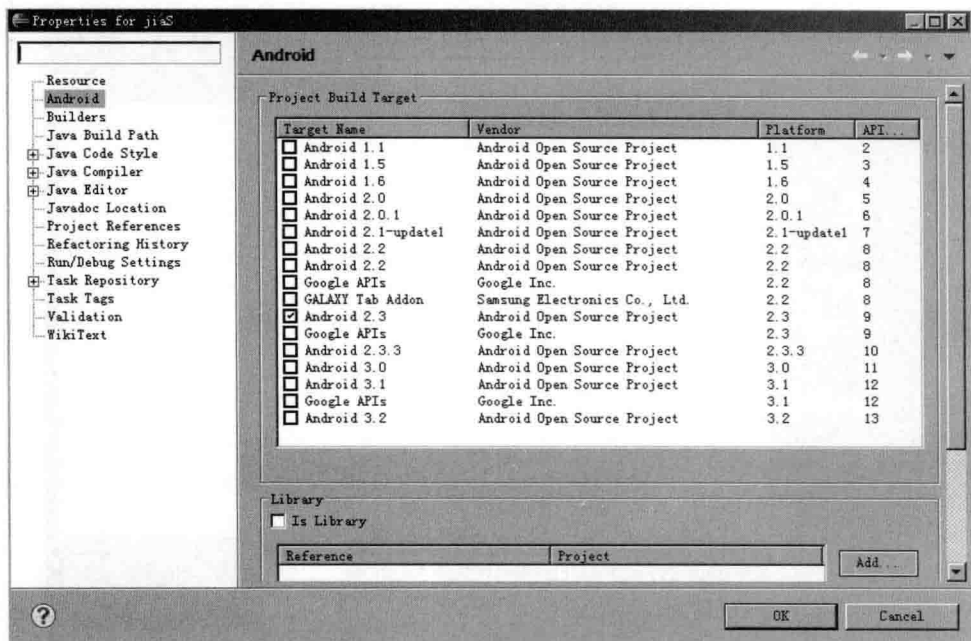


图 7-4 “Properties for jiaSCH” 窗口

(2) 选择左侧的 Java Build Path 选项，然后单击 Libraries 选项卡，如图 7-5 所示。

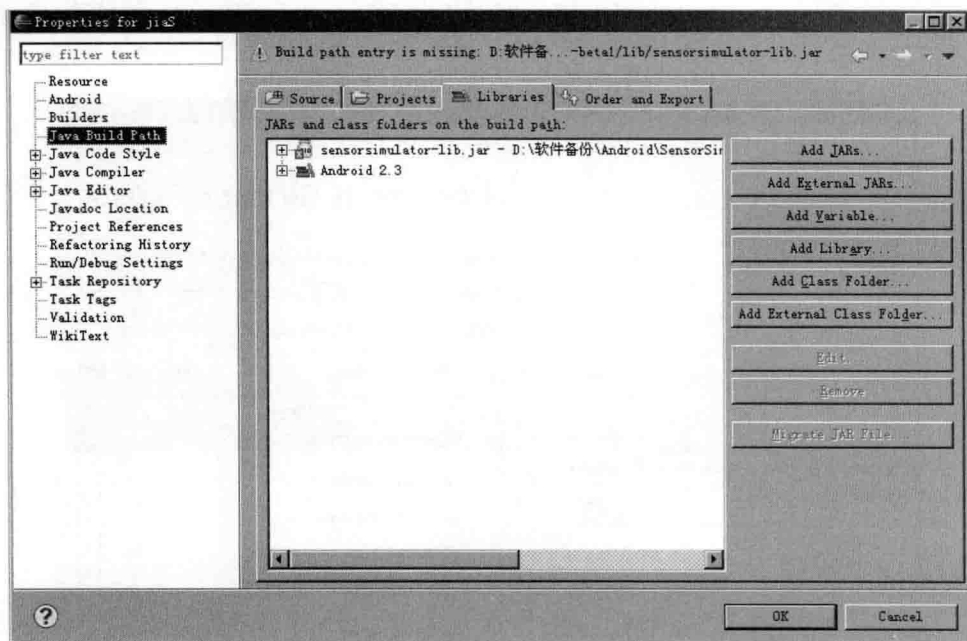


图 7-5 Libraries 选项卡

(3) 单击 Add External JARs 按钮，在弹出的 JAR Selection 对话框中找到 Sensorsimulator 安装目录下的 sensorsimulator-lib-1.1.1.jar，并将其添加到该项目中，如图 7-6 所示。



图 7-6 添加需要的 JAR 包

(4) 开始启动 sensorsimulator.jar, 并对手机模拟器上的 SensorSimulatorSettings 进行必要的配置。首先在 C:\sensorsimulator-1.1.1\bin 目录下找到 sensorsimulator.jar 并启动, 运行后的界面如图 7-7 所示。

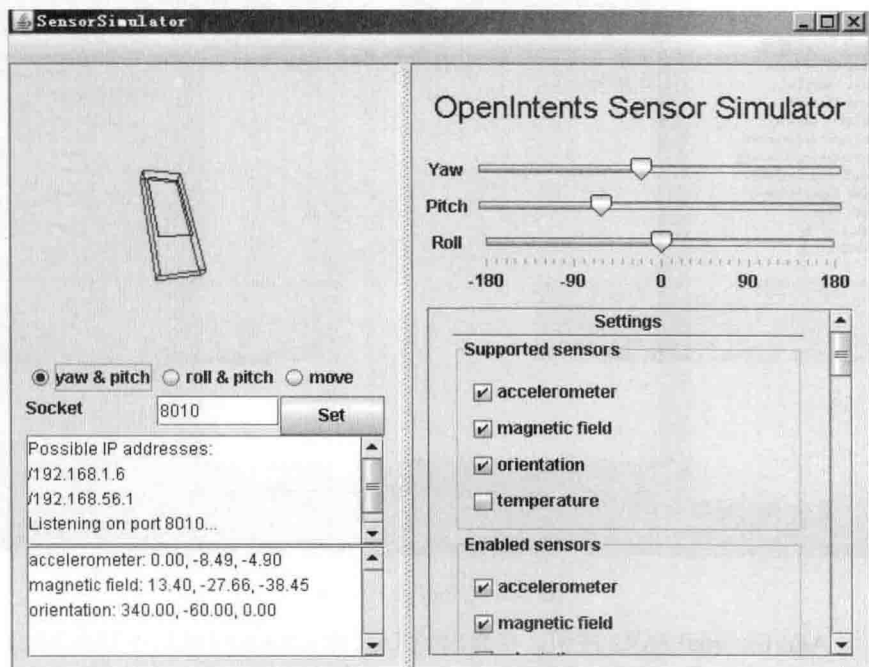


图 7-7 传感器的模拟器

(5) 接下来开始进行手机模拟器和 SensorSimulator 的连接配置工作，运行手机模拟器上安装好的 SensorSimulatorSettings，如图 7-8 所示。

(6) 在图 7-8 中输入 SensorSimulator 启动时显示的 IP 地址和端口号，单击屏幕右上角的 Testing 按钮后会进入测试连接界面，如图 7-9 所示。

(7) 单击屏幕上的 Connect 按钮进入下一界面，如图 7-10 所示。在此界面中可以选择需要监听的传感器，如果能够从传感器中读取到数据，说明 SensorSimulator 与手机模拟器连接成功，可以测试自己开发的应用程序了。

至此，使用 Eclipse 结合 SensorSimulator 配置传感器应用程序的基本流程介绍完毕。

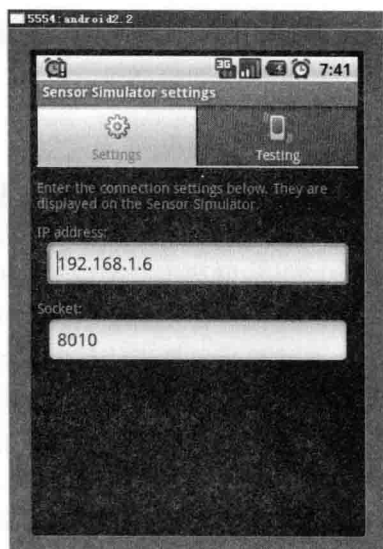


图 7-8 运行手机模拟器上安装好的 SensorSimulatorSettings



图 7-9 测试连接界面



图 7-10 连接界面

7.3 使用传感器

在 Android 系统中支持多种传感器 (Sensor)，传感器系统可以让智能手机的功能更加丰富。Android 系统支持多种传感器，有的传感器已经在 Android 的框架中使用，大多数传感器由应用程序来使用，使用传感器可以开发出包括游戏在内的很多新奇的应用。在 Android 系统中支持的传感器包括加速度传感器 (Accelerometer)、姿态传感器 (Orientation)、磁场传感

器 (Magnetic Field) 和光传感器 (Light) 等。在本节的内容中, 将详细讲解在 Android 系统中使用传感器的基本知识。

7.3.1 光线传感器

光线传感器的好处是可以根据手机所处环境的光线来调节手机屏幕的亮度和键盘灯。比如在光线充足的地方屏幕会很亮, 键盘灯就会关闭。相反, 如果在暗处, 键盘灯就会亮, 屏幕较暗 (与屏幕亮度的设置也有关系), 这样既保护了眼睛又节省了能量。光线传感器在进入睡眠模式时会发出蓝色周期性闪动的光, 非常美观。

光线传感器位于前摄像头旁边的一个小点, 如果在光线充足的情况下 (室外或者灯光充足的室内), 大约在 2~3 秒后键盘灯会自动熄灭, 即使再操作机器键盘灯也不会亮, 除非到了光线比较暗的地方才会自动亮起来。如果在光线充足的情况下用手将光线感应器遮上, 在 2~3 秒后键盘灯会自动亮起来, 在此过程中光线感应器起到了一个节电的功能。

开发光传感器应用时需要监测 `SENSOR_LIGHT`, 代码如下。

```
private SensorListener mySensorListener = new SensorListener() {
    @Override
    public void onAccuracyChanged(int sensor, int accuracy) {}
    //重写 onAccuracyChanged 方法
    @Override
    public void onSensorChanged(int sensor, float[] values) {
        //重写 onSensorChanged 方法
        if(sensor == SensorManager.SENSOR_LIGHT){           //只检查光强度的变化
            myTextView1.setText("光的强度为: "+values[0]);    //将光的强度显示到 TextView
        }
    }
};
@Override
protected void onResume() {                                //重写的 onResume 方法
    mySensorManager.registerListener(                        //注册监听
        mySensorListener,                                   //监听器 SensorListener 对象
        SensorManager.SENSOR_LIGHT,                        //传感器的类型为光的强度
        SensorManager.SENSOR_DELAY_UI                      //频率
    );
    super.onResume();
}
```

在上述代码中, 通过 `if` 语句判断是否为光的强度改变事件。在代码中只对光强度改变事件进行处理, 将得到的光强度显示在屏幕中。光传感器只得到一个数据, 而并不像其他传感器那样得到的是 X、Y、Z 三个方向上的分量。

在注册监听时, 通过传入 `SensorManager.SENSOR_LIGHT` 来通知系统只注册光传感器。

7.3.2 磁场传感器

磁场传感器主要用于感应周围的磁感应强度, 在注册监听器后主要用于捕获如下 3 个参数:

- values[0]
- values[1]
- values[2]

上述 3 个参数分别代表磁感应强度在空间坐标系中 3 个方向轴上的分量。所有数据的单位为 uT，即微特斯拉。

在接下来的实例中，将演示在 Android 中使用磁场传感器的方法。

实 例	功 能	源码路径
实例 7-1	使用磁场传感器	光盘\daima\7\cichangLI

本实例的实现文件是 cichangLI.java，在此文件中定义了监听器类对象和注册监听的方法。主要代码如下。

```
public class cichangLI extends Activity {
    TextView myTextView1;           //x 方向磁场分量
    TextView myTextView2;           //y 方向磁场分量
    TextView myTextView3;           //z 方向磁场分量
    //SensorManager mySensorManager; //引用 SensorManager 对象
    SensorManagerSimulator mySensorManager;
    //声明 SensorManagerSimulator 对象, 调试时用
    @Override
    public void onCreate(Bundle savedInstanceState) { //重写 onCreate 方法
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main); //当前的用户界面
        myTextView1 = (TextView) findViewById(R.id.myTextView1);
        //得到 myTextView1 引用
        myTextView2 = (TextView) findViewById(R.id.myTextView2);
        //得到 myTextView2 引用
        myTextView3 = (TextView) findViewById(R.id.myTextView3);
        //得到 myTextView3 引用
        //调试时用
        mySensorManager = SensorManagerSimulator.getSystemService(this, SENSOR_
SERVICE);
        mySensorManager.connectSimulator(); //连接 Simulator 服务器
    }
    @SuppressWarnings("deprecation")
    private SensorListener mySensorListener = new SensorListener(){
        @Override
        public void onAccuracyChanged(int sensor, int accuracy) {}
        //重写 onAccuracyChanged 方法
        @Override
        public void onSensorChanged(int sensor, float[] values) {
            //重写 onSensorChanged 方法
            if(sensor == SensorManager.SENSOR_MAGNETIC_FIELD){//检查磁场的变化
                myTextView1.setText("x 方向的磁场分量为: "+values[0]);
                //数据显示在 TextView
                myTextView2.setText("y 方向的磁场分量为: "+values[1]);
                //数据显示在 TextView
                myTextView3.setText("z 方向的磁场分量为: "+values[2]);
            }
        }
    }
}
```

```

        //数据显示在 TextView
    }
}

};

@Override
protected void onResume() {
    mySensorManager.registerListener(
        mySensorListener,
        SensorManager.SENSOR_MAGNETIC_FIELD,
        SensorManager.SENSOR_DELAY_UI
    );
    super.onResume();
}

@Override
protected void onPause() {
    //取消注册监听器
    mySensorManager.unregisterListener((SensorEventListener)
mySensorListener);
    super.onPause();
}
}

```

因为本实例比较简单，是根据 SensorSimulator 中附带的开源代码改变的，所以在此不再进行详细介绍，读者只需阅读本书附带光盘中的源码即可。

7.3.3 加速度传感器

传统意义上的加速度传感器是一种能够测量加速力的电子设备。加速力是指当物体在加速过程中作用在物体上的力。加速力既可以是常量，也可以是变量。加速度计有两种，其中一种是角加速度计，是由陀螺仪（角速度传感器）改进而来的，另一种就是线加速度计。

加速度传感器可以帮助机器人了解它现在身处的环境，能够使其分辨出是在爬山，还是在走下坡，是否摔倒等。一个好的程序员能够使用加速度传感器来分辨出上述情形，加速度传感器甚至可以用来分析发动机的振动。

加速度传感器可以测量牵引力产生的加速度。例如，在 IBM Thinkpad 笔记本电脑中就内置了加速度传感器，能够动态地监测出笔记本在使用中的振动。根据这些振动数据，系统会智能地选择关闭硬盘还是让其继续运行，这样可以最大程度地保护电脑。

综上所述，加速度传感器主要应用在手柄振动/摇晃、仪器仪表、汽车制动启动、地震、报警系统、玩具、结构物、环境监视、工程测振、地质勘探、铁路、桥梁、大坝的振动测试与分析，还有鼠标，高层建筑结构动态特性和安全保卫振动侦察上。

在接下来的实例中，将演示在 Android 中使用加速传感器的方法。

实 例	功 能	源码路径
实例 7-2	演示加速传感器的用法	光盘\daima\7\jiaSLI

本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，主要代码如下。

```

<?xml version="1.0" encoding="utf-8"?>      <!-- 声明 xml 的版本及编码格式 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">      <!-- 添加一个垂直的线性布局 -->
    <TextView
        android:id="@+id/title"
        android:gravity="center_horizontal"
        android:textSize="20px"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/title"/>      <!-- 添加一个 TextView 控件 -->
    <TextView
        android:id="@+id/myTextView1"
        android:textSize="18px"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/myTextView1"/> <!-- 添加一个 TextView 控件 -->
    <TextView
        android:id="@+id/myTextView2"
        android:textSize="18px"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/myTextView2"/> <!-- 添加一个 TextView 控件 -->
    <TextView
        android:id="@+id/myTextView3"
        android:textSize="18px"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/myTextView3"/> <!-- 添加一个 TextView 控件 -->
</LinearLayout>

```

(2) 编写主程序文件 jiaSLI.java, 此文件的具体实现流程如下。

- ① 声明 3 个 TextView 的引用, 分别用来显示 3 个方向上的加速度。
- ② 声明对 SensorManager 对象的引用, 此处使用 SensorSimulator 工具来模拟传感器。
- ③ 设置当前的用户界面, 然后得到 XML 文件中配置的各个控件的引用。
- ④ 初始化 SensorManager 对象, 同样因为调试的原因用专用代码替代。
- ⑤ 初始化监听器类, 并重写了该类中的两个方法。
- ⑥ 在 onSensorChanged 方法中只处理加速度的变化, 并将得到的数值显示到 TextView 中。
- ⑦ 重写类 Activity 的 onResume() 方法, 在该方法中为 SensorManager 添加监听, 还需要重写 onPause() 方法, 在方法中取消注册的监听器。

文件 jiaSLI.java 的主要实现代码如下。

```

public class jiaSCH extends Activity {
    TextView myTextView1;          //x 方向加速度
    TextView myTextView2;          //y 方向加速度
    TextView myTextView3;          //z 方向加速度

```

```
//SensorManager mySensorManager;           //SensorManager 对象引用
SensorManagerSimulator mySensorManager;
//声明 SensorManagerSimulator 对象, 调试时用
@Override
public void onCreate(Bundle savedInstanceState) {    //重写 onCreate 方法
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);                //设置当前的用户界面
    myTextView1 = (TextView) findViewById(R.id.myTextView1);
    //得到 myTextView1 引用
    myTextView2 = (TextView) findViewById(R.id.myTextView2);
    //得到 myTextView2 引用
    myTextView3 = (TextView) findViewById(R.id.myTextView3);
    //得到 myTextView3 引用
    //调试时用
    mySensorManager = SensorManagerSimulator.getSystemService(this, SENSOR_
SERVICE);
    mySensorManager.connectSimulator(); //连接 Simulator 服务器
}
private SensorListener mySensorListener = new SensorListener(){
    @Override
    public void onAccuracyChanged(int sensor, int accuracy) {}
    //重写 onAccuracyChanged 方法
    @Override
    public void onSensorChanged(int sensor, float[] values) {
        //重写 onSensorChanged 方法
        if(sensor == SensorManager.SENSOR_ACCELEROMETER){
            //只检查加速度的变化
            myTextView1.setText("x 方向上的加速度为: "+values[0]);
            //将提取的 x 数据显示到 TextView
            myTextView2.setText("y 方向上的加速度为: "+values[1]);
            //将提取的 y 数据显示到 TextView
            myTextView3.setText("z 方向上的加速度为: "+values[2]);
            //将提取的 x 数据显示到 TextView
        }
    }
};
@Override
protected void onResume() {                    //重写的 onResume 方法
    mySensorManager.registerListener(          //注册监听
        mySensorListener,                      //监听 SensorListener 对象
        SensorManager.SENSOR_ACCELEROMETER,    //设置传感器的类型为加速度
        SensorManager.SENSOR_DELAY_UI //用传感器事件传递频度
    );
    super.onResume();
}
@Override
protected void onPause() {                    //重写 onPause 方法
    mySensorManager.unregisterListener(mySensorListener); //取消注册监听器
    super.onPause();
}
}
```


(3) 为了调试本实例代码, 需要为该程序添加网络权限, 因为 SensorSimulator 安装在 Android 模拟器中的客户端需要和桌面端的服务器进行通信。

7.3.4 姿态传感器

姿态传感器与加速度传感器不同, 姿态传感器主要用于感应手机方位的变化, 其中运用了欧拉角的知识。欧拉角的基本思想是将角位移分解为绕 3 个互相垂直轴的 3 个旋转方向组成的序列。其实, 任意 3 个轴和任意顺序都可以, 但最有意义的是使用笛卡儿坐标系并按一定的顺序所组成的旋转序列。

在学习欧拉角知识之前先介绍几种不同概念的坐标系, 以便于读者理解欧拉角知识。

(1) 世界坐标系

世界坐标系是一个特殊的坐标系, 建立了描述其他坐标系所需要的参考框架。能够用世界坐标系描述其他坐标系的位置, 而不能用更大的、外部的坐标系来描述世界坐标系。例如, “向西”、“向东”等词汇就是世界坐标系中的描述词汇。

(2) 物体坐标系

物体坐标系是和特定物体相关联的坐标系, 每个物体都有它们独立的坐标系。当物体移动或改变方向时, 和该物体相关联的坐标系将随之移动或改变方向。例如, “向左”、“向右”等词汇就是物体坐标系中的描述词汇。

(3) 摄像机坐标系

摄像机坐标系是和观察者密切相关的坐标系。在摄像机坐标系中, 摄像机在原点, x 轴向右, z 轴向前 (朝向屏幕内或摄像机方向), y 轴向上 (不是世界的上方而是摄像机本身的上方)。

(4) 惯性坐标系

惯性坐标系是为了简化世界坐标系到物体坐标系的转换而引入的一种新的坐标系。惯性坐标系的原点和物体坐标系的原点重合, 但惯性坐标系的轴平行于世界坐标系的轴。

在欧拉角中, 表示一个物体的方位用 Yaw-Pitch-Roll 约定。在这个系统中, 一个方位被定义为一个 Yaw 角、一个 Pitch 角和一个 Roll 角。欧拉角的基本思想是让物体开始于“标准”方位, 目的是使物体坐标轴和惯性坐标轴对齐。在标准方位上, 让物体作 Yaw、Pitch 和 Roll 旋转, 最后物体到达我们想要描述的方位。

(5) Yaw 轴

Yaw 轴是 3 个方向轴中唯一不变的轴, 其方向总是竖直向上, 和世界坐标系中的 z 轴是等同的, 也就是重力加速度 g 的反方向。

(6) Pitch 轴

Pitch 轴方向依赖于手机沿 Yaw 轴的转动情况, 即当手机沿 Yaw 转过一定的角度后, Pitch 轴也相应围绕 Yaw 轴转动相同的角度。Pitch 轴的位置依赖于手机沿 Yaw 轴转过的角度, 好比 Yaw 轴和 Pitch 轴是两根焊死在一起成 90° 部件。

在接下来的实例中, 将演示在 Android 中使用姿态传感器的方法。

实 例	功 能	源码路径
实例 7-3	演示姿态传感器的用法	光盘\daima\7\zitaiLI

本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，主要代码如下。

```
<TextView
    android:id="@+id/title"
    android:gravity="center_horizontal"
    android:textSize="20px"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/title"/><!-- 添加一个 TextView 控件 -->

<TextView
    android:id="@+id/myTextView1"
    android:textSize="18px"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/myTextView1"/><!-- 添加一个 TextView 控件 -->

<TextView
    android:id="@+id/myTextView2"
    android:textSize="18px"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/myTextView2"/><!-- 添加一个 TextView 控件 -->

<TextView
    android:id="@+id/myTextView3"
    android:textSize="18px"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/myTextView3"/><!-- 添加一个 TextView 控件 -->
```

(2) 编写主程序文件 zitaiLI.java，此文件的具体实现流程如下。

- ① 声明 3 个分别用来显示 Yaw、Pitch、Roll 的 TextView 的引用。
- ② 声明 SensorManager 引用，而因调试原因用 SensorManagerSimulator 代替。
- ③ 重写 Activity 的 onCreate 方法，在方法中设置当前的用户界面，然后得到各个控件的引用，并初始化 SensorManager 或 SensorManagerSimulator。

④ 初始化监听器类的对象，在重写的 onSensorChanged 方法中只对姿态 SENSOR_ORIENTATION 变化进行处理，将 3 个姿态值显示到 TextView 中。

⑤ 重写 Activity 的 onResume 方法，在方法中为 SensorManager 注册监听，此处传入的传感器类型为 SENSOR_ORIENTATION，表示只读取姿态数据。

文件 zitaiCH.java 的主要代码如下。

```
public class zitaiLI extends Activity {
    TextView myTextView1;
    TextView myTextView2;
    TextView myTextView3;
    //声明 SensorManagerSimulator 对象, 调试时用
    SensorManagerSimulator mySensorManager;
    @Override
    public void onCreate(Bundle savedInstanceState) { //重写 onCreate 方法
        super.onCreate(savedInstanceState);
```

```

        setContentView(R.layout.main); //设置用户界面
        myTextView1 = (TextView) findViewById(R.id.myTextView1);
        //myTextView1 的引用
        myTextView2 = (TextView) findViewById(R.id.myTextView2);
        //myTextView2 的引用
        myTextView3 = (TextView) findViewById(R.id.myTextView3);
        //myTextView3 的引用
        //mySensorManager =
        // (SensorManager) getSystemService(SENSOR_SERVICE); //获得 SensorManager
        //调试时用
        mySensorManager = SensorManagerSimulator.getSystemService(this, SENSOR_
        SERVICE);
        mySensorManager.connectSimulator(); //与 Simulator 服务器连接
    }
    private SensorListener mySensorListener = new SensorListener(){
        @Override
        public void onAccuracyChanged(int sensor, int accuracy) {}
        //重写 onAccuracyChanged 方法
        @Override
        public void onSensorChanged(int sensor, float[] values) {
            //重写 onSensorChanged 方法
            if(sensor == SensorManager.SENSOR_ORIENTATION){ //检查姿态变化
                myTextView1.setText("Yaw 为: "+values[0]); //TextView 数据显示
                myTextView2.setText("Pitch 为: "+values[1]); //TextView 数据显示
                myTextView3.setText("Roll 为: "+values[2]); //TextView 数据显示
            }
        }
    };
    @Override
    protected void onResume() { //重写的 onResume 方法
        mySensorManager.registerListener( //注册监听
            mySensorListener, //监听器 SensorListener 对象
            SensorManager.SENSOR_ORIENTATION, //姿态传感器的类型
            SensorManager.SENSOR_DELAY_UI //传感器事件传递频度
        );
        super.onResume();
    }
    @Override
    protected void onPause() { //重写 onPause 方法
        mySensorManager.unregisterListener(mySensorListener); //取消注册监听器
        super.onPause();
    }
}

```

因为此实例比较简单, 是根据 **SensorSimulator** 中附带的开源代码改变的, 所以在此不再进行详细介绍, 读者只需阅读本书附带光盘中的源码即可。

7.3.5 温度传感器

温度传感器是指利用物质的各种物理性质, 随温度变化的规律把温度转换为电量的传感

器。温度传感器是温度测量仪表的核心部分，按测量方式可分为接触式和非接触式两大类，按照传感器材料及电子元件特性分为热电阻和热电偶两类。

温度计通过传导或对流达到热平衡，从而使温度计的示值能直接表示被测对象的温度。在一定的测温范围内，温度计也可测量物体内部的温度分布。但对于运动体、小目标或热容量很小的对象则会产生较大的测量误差，常用的温度计有双金属温度计、玻璃液体温度计、压力式温度计、电阻温度计、热敏电阻和温差电偶等。温度传感器被广泛应用于工业、农业、商业等部门。随着低温技术在国防工程、空间技术、冶金、电子、食品、医药和石油化工等部门的广泛应用和超导技术的研究，测量 120K 以下温度的低温温度计得到了发展，如低温气体温度计、蒸汽压温度计、声学温度计、顺磁盐温度计、量子温度计、低温热电阻和低温温差电偶等。低温温度计要求感温元件体积小、准确度高、复现性和稳定性好。利用多孔高硅氧玻璃渗碳烧结而成的渗碳玻璃热电阻就是低温温度计的一种感温元件，可用于测量 1.6K~300K 范围内的温度。

在接下来的实例中，将演示在 Android 中使用温度传感器的方法。

实 例	功 能	源码路径
实例 7-4	演示温度传感器的用法	光盘\daima\7\wenduLI

本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>      <!-- 声明 xml 的版本及编码格式 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">      <!-- 添加一个垂直的线性布局 -->
    <TextView
        android:id="@+id/title"
        android:gravity="center_horizontal"
        android:textSize="20px"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/title"/>      <!-- 添加一个 TextView 控件 -->
    <TextView
        android:id="@+id/myTextView1"
        android:textSize="18px"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/myTextView1"/> <!-- 添加一个 TextView 控件 -->
</LinearLayout>
```

(2) 编写主程序文件 wenduLI.java，此文件具体实现流程如下。

- ① 声明一个用来显示结果的 TextView 的引用。
 - ② 声明监听器对象，在 onSensorChanged 方法中只对 SENSOR_TEMPERATURE 即温度变化进行检测。
 - ③ 在传感器类型中传入 SENSOR_TEMPERATURE，表示注册的是温度传感器。
- 文件 wenduLI.java 的主要代码如下。


```

public class wenduLI extends Activity {
    TextView myTextView1;                //当前温度
    //SensorManager mySensorManager;    //引用 SensorManager 对象
    SensorManagerSimulator mySensorManager;
    //声明 SensorManagerSimulator 对象, 调试时用
    @Override
    public void onCreate(Bundle savedInstanceState) { //重写 onCreate 方法
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);        //设置当前的用户界面
        myTextView1 = (TextView) findViewById(R.id.myTextView1);
        //得到 myTextView1 的引用
        mySensorManager = SensorManagerSimulator.getSystemService(this,
SENSOR_SERVICE);
        mySensorManager.connectSimulator(); //与 Simulator 连接
    }
    private SensorListener mySensorListener = new SensorListener(){
        @Override
        public void onAccuracyChanged(int sensor, int accuracy) {}
        //重写 onAccuracyChanged 方法
        @Override
        public void onSensorChanged(int sensor, float[] values) {
            //重写 onSensorChanged 方法
            if(sensor == SensorManager.SENSOR_TEMPERATURE){ //只检查温度的变化
                myTextView1.setText("当前的温度为: "+values[0]);
                //在 TextView 显示当前温度
            }
        }
    };
    @Override
    protected void onResume() {           //重写的 onResume 方法
        mySensorManager.registerListener( //注册监听
            mySensorListener,             //监听器 SensorListener 对象
            SensorManager.SENSOR_TEMPERATURE, //传感器的类型为温度
            SensorManager.SENSOR_DELAY_UI //传感器事件传递的频度
        );
        super.onResume();
    }
    @Override
    protected void onPause() {            //重写 onPause 方法
        mySensorManager.unregisterListener(mySensorListener); //取消注册监听器
        super.onPause();
    }
}
.....

```

除了之前介绍过的各种传感器, Android 系统还提供了一些其他的传感器, 如压力传感器 Pressure、距离传感器 Proximity 等, 因本书篇幅有限, 而且这些传感器并不常用, 所以在此不再进行介绍。

人工智能技术

人工智能用英文表示为 Artificial Intelligence, 缩写为 AI, 是研究、开发用于模拟、延伸和扩展人的智能的理论、方法、技术及应用系统的一门新的科学技术。人工智能是计算机科学的一个分支, 它试图了解智能的实质, 并生产出一种新的、能与人类智能相似的方式做出反应的智能机器, 该领域的研究包括机器人、语言识别、图像识别、自然语言处理和专家系统等。本章将详细讲解人工智能技术在 Android 系统中的基本应用知识, 为读者学习本书后面的知识打下基础。

8.1 人工智能基础

本节将首先简要介绍 AI 人工智能的基本知识, 了解 AI 人工智能的基本原理和意义, 为读者学习本章后面的开发部分打好基础。

8.1.1 人工智能概述

“人工智能”一词最初是在 1956 年 Dartmouth 学会上提出的。从此研究者们发展了众多理论和原理, 人工智能这一概念也随之扩展开来。人工智能是一门极富挑战性的科学, 从事这项工作的人必须懂得计算机知识, 心理学和哲学。人工智能包括的内容涉及广泛, 它由不同的领域组成, 如机器学习、计算机视觉等。

人工智能研究的一个主要目标是使机器能够胜任一些通常需要人类智能才能完成的复杂工作。但是在不同的时代、不同的人对这种“复杂工作”的理解是不同的。例如, 繁重的科学和工程计算本来是要人脑来承担的, 现在计算机不但能完成这种计算, 而且能够比人脑做得更快、更准确。正是因为如此, 当代人已不再把这种计算看作“需要人类智能才能完成的复杂任务”, 可见复杂工作的定义是随着时代的发展和技术的进步而变化的, 人工智能这门科学的具体目标也自然随着时代的变化而发展。AI 一方面不断获得新的进展, 一方面又转向更有意义、更加困难的目标。目前能够用来研究人工智能的主要物质手段, 以及能够实现人工智能技术的机器就是计算机。

AI 人工智能的发展历史是和计算机科学技术的发展史联系在一起的。除了计算机科学以外, 人工智能还涉及信息论、控制论、自动化、仿生学、生物学、心理学、数理逻辑、语言学、医学和哲学等多门学科。人工智能学科研究的主要内容包括: 知识表示, 自动推理和搜索方法, 机器学习和知识获取, 知识处理系统, 自然语言理解, 计算机视觉, 智能机器人,

自动程序设计等方面。

8.1.2 两种实现人工智能的方法

人工智能在计算机上有如下两种实现方式。

(1) 采用传统的编程技术,使系统呈现智能的效果,而不考虑所用方法是否与人或动物机体所用的方法相同。这种方法称为工程学方法(Engineering approach),它已在一些领域内作出了成果,如文字识别、电脑下棋等。

(2) 模拟法(Modeling approach),它不仅要看效果,还要求实现方法也和人类或生物机体所用的方法相同或相类似。例如,遗传算法(Generic Algorithm,简称GA)和人工神经网络(Artificial Neural Network,简称ANN)均属后一类型。遗传算法模拟人类或生物的遗传-进化机制,人工神经网络则是模拟人类或动物大脑中神经细胞的活动方式。为了得到相同智能效果,两种方式通常都可使用。

如果采用前一种方法,需要人工详细规定程序逻辑,如果游戏简单,还是方便的。如果游戏复杂,角色数量和活动空间增加,相应的逻辑就会很复杂(按指数式增长),人工编程就非常烦琐,容易出错。而一旦出错,就必须修改原程序,重新编译、调试,最后为用户提供一个新的版本或提供一个新补丁,非常麻烦。当采用后一种方法时,编程者要为每一角色设计一个智能系统(一个模块)来进行控制,这个智能系统(模块)开始什么也不懂,就像初生婴儿那样,但它能够学习,能渐渐地适应环境,应付各种复杂情况。这种系统开始也常犯错误,但它能吸取教训,下一次运行时就可能改正,至少不会永远错下去,不用发布新版本或打补丁。利用这种方法来实现人工智能,要求编程者具有生物学的思考方法,入门难度大一点。一旦入了门,其方法就可得到广泛应用。由于这种方法编程时无须对角色的活动规律做详细规定,应用于复杂问题,通常会比前一种方法更省力。

8.2 图搜索在人工智能中的应用

图论是数学家热衷研究的一个领域,并且已经设计出了无数的算法来搜索或探究一个图的拓扑结构。在游戏设计领域,需要使用图论搜索算法实现人工智能。在本节的内容中,将详细介绍搜索算法是如何实现的,为读者学习后面的知识打下基础。

8.2.1 深度优先搜索(DFS)

深度优先搜索算法(Depth-First-Search)是搜索算法的一种,是指沿着树的深度遍历树的节点,尽可能深地搜索树的分支。当节点v的所有边都已被探寻过,搜索将回溯到发现节点v的那条边的起始节点。这一过程一直进行到已发现从源节点可达的所有节点为止。如果还存在未被发现的节点,则选择其中一个作为源节点并重复以上过程,整个进程反复进行直到所有节点都被访问为止。这种搜索属于盲目搜索。

深度优先搜索是图论中的经典算法,利用深度优先搜索算法可以产生目标图的相应拓扑排序表,利用拓扑排序表可以方便地解决很多相关的图论问题,如最大路径问题等。因发明“深度优先搜索算法”,霍普克洛夫特与陶尔扬共同获得计算机领域的最高奖——图灵奖。图8-1所示为一个对节点进行深度优先搜索的处理顺序。

深度优先算法，是计算机程序的一种编制原理，就是在一个问题出现多种可以实现的方法和技术时，应该优先选择哪个更合适的，这也是一种普遍的逻辑思想，此种思想在运算的过程中，用到计算机程序的一种递归的思想。

1. 性质

依据深度优先搜索可以获得有关图的结构的大量信息。也许深度优先搜索的最基本的特征是它的优先算法图，形成一个由树组成的森林，这是因为深度优先树的结构准确反映了DFS_Visit 中递归调用的结构的缘故，即 $u = \pi[v]$ 当且仅当在搜索 u 的邻接表过程中调用了过程 DFS_Visit(v)。

深度优先搜索的另一重要特性是发现和完成时间具有括号结构，如果把发现顶点 u 用左括号“(u)”表示，完成用右括号“ u)”表示，那么发现与完成的记载在括号被正确套用的前提下就是一个完善的表达式。例如，图 8-2 所示为深度优先搜索的性质。(a) 图是对一个有向图进行深度优先搜索的结果。节点的时间戳与边的类型的表示方式与图 8-2 (b) 相同。(b) 图中的括号表示对应于每个节点的发现时刻和完成时刻组成的区间。列与列之间表示每个矩形跨越相应节点的发现时刻与完成时刻所设定的区间。图中还显示了树枝。如果两个区间有重叠，则必有一个区间嵌套于另一个区间内，且对应于较小区间的节点是对应于较大区间的节点的后裔。(c) 图是对 (a) 图的重新描述，使深度优先树中所有树枝和正向边自上而下，而所有反向边自下而上从后裔指向祖先。下面的定理给出了标记括号结构的另外一种方法。

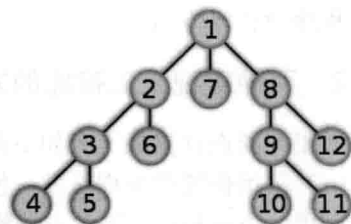


图 8-1 对节点进行深度优先搜索的顺序

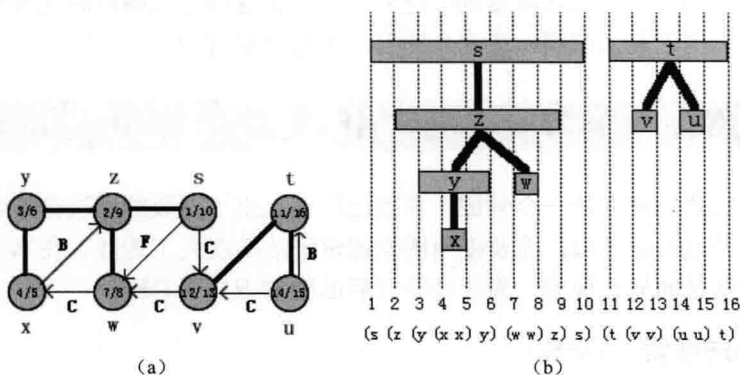


图 8-2 深度优先搜索的性质

2. 算法流程

含有深度界限的深度优先搜索算法的基本流程如下:

- (1) 把起始节点 S 放到未扩展节点 OPEN 表中。如果此节点为一目标节点, 则得到一个解。
- (2) 如果 OPEN 为一空表, 则失败退出。
- (3) 把第一个节点 (节点 n) 从 OPEN 表移到 CLOSED 表。
- (4) 如果节点 n 的深度等于最大深度, 则转向 (2)。
- (5) 扩展节点 n, 产生其全部后裔, 并把它们放入 OPEN 表的前头。如果没有后裔, 则转向 (2)。
- (6) 如果后继节点中有任一个为目标节点, 则求得一个解, 成功退出; 否则, 转向 (2)。

3. 经典问题

深度优先算法最经典的用法是解决迷宫问题和八皇后问题。

1) 迷宫问题

首先, 我们来想象一只老鼠, 在一座不见天日的迷宫内, 老鼠从入口处进去, 要从出口出来。那么老鼠会怎么走? 当然是这样的: 老鼠如果遇到直路, 就一直往前走; 如果遇到分叉路口, 就任意选择其中的一个继续往下走; 如果遇到死胡同, 就退回到最近的一个分叉路口, 选择另一条道路再走下去; 如果遇到了出口, 老鼠的旅途就算结束了。深度优先搜索法的基本原则就是这样: 按照某种条件往前试探搜索, 如果前进中遭到失败 (正如老鼠遇到死胡同), 则退回头另选通路继续搜索, 直到找到条件的目标为止。

① 递归算法

要想实现这一算法, 就要用到编程的一大利器——递归。虽然“递归”是一个很抽象的概念, 但是在日常生活中会经常看到。拿来两面镜子, 把它们面对面, 会看到什么? 会看到镜子中有无数个镜子? A 镜子中有 B 镜子的相, B 镜子中有 A 镜子的相, A 镜子的相就是 A 镜子本身的真实写照, 也就是说 A 镜子的相包括了 A 镜子, 还有 B 镜子在 A 镜子中的相……如果换成计算机语言就是 A 调用 B, 而 B 又调用 A, 这样间接的, A 就调用了 A 本身, 这实现了一个重复的功能。

② 解法

将递归思想运用到上面的迷宫中, 记老鼠现在所在的位置是 (x, y) , 那它现在有前、后、左、右 4 个方向可以走, 分别是 $(x+1, y)$, $(x-1, y)$, $(x, y+1)$, $(x, y-1)$, 其中一个方向是它来时的路, 先不考虑这个方向, 分别尝试其他三个方向, 如果某个方向是路而不是墙, 那么老鼠就向那个方向迈出一步。在新的位置上, 又可以重复前面的步骤。老鼠走到了死胡同又是怎么回事? 就是除了来时的路, 其他 3 个方向都是墙, 这时这条路就走到了尽头, 无法再向深一层发展, 老鼠就应该沿来时的路回去, 尝试另外的方向。

2) 八皇后问题

① 八皇后问题描述

在标准国际象棋的棋盘上 (8×8 格) 准备放置 8 只皇后, 国际象棋中皇后的威力是最大的, 既可以横走竖走, 还可以斜着走, 遇到挡在她前进路线上的敌人就可以将其直接吃掉。要求在棋盘上安放 8 只皇后, 使她们彼此互相都不能吃到对方, 求皇后的放法。

② 解法

这是一个很经典的问题，我们先要明确一下思路，如何运用深度优先搜索法来完成这道实例。先建立一个 $8*8$ 格的棋盘，在棋盘的第一行的任意位置安放一只皇后。紧接着，来放第二行，第二行的安放就要受一些限制，与第一行的皇后在同一竖行或同一对角线的位置上不能安放皇后，接下来是第三行，……或许会遇到这种情况，在摆到某一行时，无论皇后摆放在什么位置，她都会被其他行的皇后吃掉，这说明什么呢？这说明，我们前面的摆放是失败的，也就是说，按照前面的皇后的摆放方法，不可能得到正确的解。那这时怎么办？答案是继续修改！回到上一行，把原先我们摆好的皇后换另外一个位置，接着再回过头摆这一行，如果这样还不行或者上一行的皇后只有一个位置可放，那就回到上一行的上一行，这和老鼠碰了壁就回头是一个道理。就这样不断地尝试、修正，最终会得到正确的结论。

8.2.2 广度优先搜索 (BFS)

广度优先搜索算法 (Breadth-First-Search)，又译作宽度优先搜索，或横向优先搜索，简称 BFS，是一种图形搜索算法。BFS 是从根节点开始，沿着树的宽度遍历树的节点。如果所有节点均被访问，则算法中止。广度优先搜索的实现一般采用 open-closed 表。图 8-3 所示为一个对节点进行广度优先搜索的处理顺序。

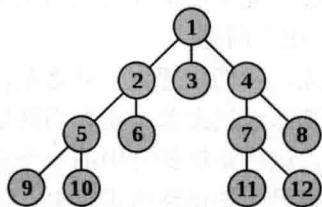


图 8-3 对节点进行广度优先搜索的处理顺序

BFS 是一种盲目搜寻法，目的是系统地展开并检查图中的所有节点，以找寻结果。换句话说，它并不考虑结果的可能位址，彻底地搜索整张图，直到找到结果为止。BFS 并不使用经验法则算法。

从算法的观点来说，所有因为展开节点而得到的子节点都会被加进一个先进先出的队列中。在现实应用中，其邻居节点尚未被检验过的节点会被放置在一个称为 open 的容器中（如队列或链表），而被检验过的节点则被放置在称为 closed 的容器中。

1. 实现流程

- (1) 首先将根节点放入队列中。
- (2) 从队列中取出第一个节点，并检验它是否为目标。
 - ① 如果找到目标，则结束搜寻并回传结果；
 - ② 否则将它所有尚未检验过的直接子节点加入队列中。
- (3) 如果队列为空，表示整张图都检查过了——亦即图中没有欲搜寻的目标。结束搜寻并回传“找不到目标”。
- (4) 重复步骤 (2)。

2. 复杂度

复杂度有两种，分别是空间复杂度和时间复杂度。

1) 空间复杂度

所有节点都必须被储存，因此 BFS 的空间复杂度为 $O(|V| + |E|)$ ，其中 $|V|$ 是节点的数目，

而 $|E|$ 是图中边的数目。还有另外一种说法是称 BFS 的空间复杂度为 $O(BM)$ ，其中 B 是最大分支系数，而 M 是树的最长路径长度。由于对空间的大量需求，因此 BFS 并不适合解非常大的问题。

2) 时间复杂度

在最差情形下，BFS 必须寻找所有到可能节点的所有路径，因此其时间复杂度为 $O(|V| + |E|)$ ，其中 $|V|$ 是节点的数目，而 $|E|$ 是图中边的数目。

3. 完全性

广度优先搜索算法具有完全性的特点，无论图形的种类如何，只要目标存在，则 BFS 一定会找到。但是如果目标不存在，并且图为无限大，则 BFS 将不会结束。

4. 最佳解

如果所有边的长度相等，广度优先搜索算法是最佳解——亦即它找到的第一个解，距离根节点的边数目一定最少；但对一般的图来说，BFS 并不一定回传最佳解。这是因为当图形为加权图（亦即各边长度不同）时，BFS 仍然回传从根节点开始，经过边数目最少的解；而这个解距离根节点的距离不一定最短。这个问题可以考虑使用各边权值的 BFS 改良算法：一致搜寻法（en:uniform-cost search）来解决。然而，若非加权图形，则所有边的长度相等，BFS 就能找到最近的最佳解。

5. 广度优先搜索算法的应用

在现实应用中，广度优先搜索算法可以解决图论中的如下问题。

(1) 寻找图中所有连接元件（Connected Component），一个连接元件是图中的最大相连子图。

(2) 寻找连接元件中的所有节点。

(3) 寻找非加权图中任两点的最短路径。

(4) 测试一图是否为二分图。

(5) (Reverse) Cuthill - McKee 算法。

8.2.3 戴克斯特拉算法（Dijkstra）

戴克斯特拉算法（Dijkstra）是由荷兰计算机科学家艾兹赫尔·戴克斯特拉（Edsger Wybe Dijkstra）发明的。算法解决的是有向图中单个源点到其他顶点的最短路径问题。举例来说，如果图中的顶点表示城市，而边上的权重表示诸城市间开车行经的距离，该算法可以用来找到两个城市之间的最短路径。

在 Dijkstra 算法的输入中包含了一个有权重的有向图 G ，以及 G 中的一个来源顶点 S 。以 V 表示 G 中所有顶点的集合。每一个图中的边，都是两个顶点所形成的有序元素对。 (u, v) 表示从顶点 u 到 v 有路径相连。以 E 表示所有边的集合，而边的权重则由权重函数 $w: E \rightarrow [0, \infty]$ 定义。因此， $w(u, v)$ 就是从顶点 u 到顶点 v 的非负花费值（cost）。边的花费可以想像成两个顶点之间的距离。任两点间路径的花费值，就是该路径上所有边的花费值总和。已知 V 中有顶点 s 及 t ，Dijkstra 算法可以找到 s 到 t 的最低花费路径（例如，最短路径）。这

个算法也可以在一个图中，找到从一个顶点 s 到任何其他顶点的最短路径。

1. 算法描述

Dijkstra 算法是通过为每个顶点 v 保留目前为止所找到的从 s 到 v 的最短路径来工作的。初始时，原点 s 的路径长度值被赋为 0 ($d[s] = 0$)，同时把所有其他顶点的路径长度设为无穷大，即表示我们不知道任何通向这些顶点的路径（对于 V 中所有顶点 v 除 s 外 $d[v] = \infty$ ）。当算法结束时， $d[v]$ 中储存的便是从 s 到 v 的最短路径，或者如果路径不存在则是无穷大。Dijkstra 算法的基础操作是边的拓展：如果存在一条从 u 到 v 的边，那么从 s 到 v 的最短路径可以通过将边 (u, v) 添加到尾部来拓展一条从 s 到 u 的路径。这条路径的长度是 $d[u] + w(u, v)$ 。如果这个值比目前已知的 $d[v]$ 的值要小，则可以用新值来替代当前 $d[v]$ 中的值。拓展边的操作一直执行到所有的 $d[v]$ 都代表从 s 到 v 最短路径的花费。这个算法经过组织因而当 $d[u]$ 达到它最终的值的时候每条边 (u, v) 都只被拓展一次。

算法维护两个顶点集 S 和 Q 。集合 S 保留了我们已知的所有 $d[v]$ 的值已经是最短路径的值顶点，而集合 Q 则保留其他所有顶点。集合 S 初始状态为空，而后每一步都有一个顶点从 Q 移动到 S 。这个被选择的顶点是 Q 中拥有最小的 $d[u]$ 值的顶点。当一个顶点 u 从 Q 中转移到了 S 中，算法对每条外接边 (u, v) 进行拓展。

2. 时间复杂度

可以用符号 O 将该算法的运行时间表示为边数 m 和顶点数 n 的函数。Dijkstra 算法最简单的实现方法是用一个链表或者数组来存储所有顶点的集合 Q ，所以搜索 Q 中最小元素的运算 ($\text{Extract-Min}(Q)$) 只需要线性搜索 Q 中的所有元素。这样，算法的运行时间是 $O(n^2)$ 。

对于边数少于 n^2 的稀疏图来说，可以用邻接表来更有效地实现该算法。同时需要将一个二叉堆或者斐波纳契堆用作优先队列来寻找最小的顶点 (Extract-Min)。当用到二叉堆时，算法所需的时间为 $O((m + n) \log n)$ ，斐波纳契堆能稍微提高一些性能，让算法运行时间达到 $O(m + n \log n)$ 。然而，使用斐波纳契堆进行编程，常常会由于算法常数过大而导致速度没有显著提高。

3. 相关问题及算法

原本 Dijkstra 算法还能够加以修改以扩充其功能。例如，当面对一个问题时，有时我们可能希望取得数学上的次佳解。为了求得这些次佳解，首先先用原本的该算法求出最佳路径；接下来，移除最佳路径中任一段路径，并对剩下来的子集合图再做一次最佳路径计算。对于最佳路径上的每一段路径做一样的操作，就可以得到许多次佳路径解，将这些路径排序后即成为原路径问题的次佳路径解集合。

开放最短路径优先 (OSPF, Open Shortest Path First) 算法是该算法在网络路由中的一个具体实现。与 Dijkstra 算法不同，Bellman-Ford 算法可用于具有负花费边的图，只要图中不存在总花费为负值且从源点 s 可达的环路（如果有这样的环路，则最短路径不存在，因为沿环路循环多次即可无限制地降低总花费）。

与最短路径问题相关最有名的一个问题是旅行商问题 (Traveling salesman problem)，此

类问题要找出恰好通过所有目标点一次且最终回到原点的最短路径。然而该问题为 NP（完全无解的）。也就是说，与最短路径问题不同，旅行商问题不太可能具有多项式时间解法。如果有已知信息可用来估计某一点到目标点的距离，则可改用 A* 搜寻算法，以减小最短路径的搜索范围。

8.2.4 A-Star 算法

A* 算法又称为 A-Star 算法，在游戏中有其典型的用法，是人工智能在游戏中的代表。A* 算法在人工智能中是一种典型的启发式搜索算法，为了讲解 A* 算法，先来了解下面几个概念。

(1) 启发式搜索：启发式搜索就是在状态空间中的搜索对每一个搜索的位置进行评估，得到最好的位置，再从这个位置进行搜索直到目标。这样可以省略大量无谓的搜索路径，提高效率。在启发式搜索中，对位置的估价十分重要。采用不同的估价可以有不同的效果。

(2) 估价函数：从当前节点移动到目标节点的预估费用；这个估计就是启发式的。在寻路问题和迷宫问题中，通常用曼哈顿 (manhattan) 估价函数（下文有介绍）预估费用。

(3) A* 算法与 BFS：可以这样说，BFS 是 A* 算法的一个特例。对于一个 BFS 算法，从当前节点扩展出来的每一个节点（如果没有被访问过）都要放进队列进行进一步扩展。也就是说 BFS 的估计函数 h 永远等于 0，没有任何启发式的信息，可以认为 BFS 是“最差的”A* 算法。

(4) 选取最小估价：读者如果学过数据结构，应该可以知道，对于每次都要选取最小估价的节点，应该用到最小优先级队列（也称最小二叉堆）。在 C++ 的 STL 里有现成的数据结构 `priority_queue`，可以直接使用。当然不要忘了重载自定义节点的比较操作符。

(5) A* 算法的特点：A* 算法在理论上是时间最优的，但是也有缺点——它的空间增长是指数级别的。

(6) IDA* 算法：这种算法称为迭代加深 A* 算法，可以有效地解决 A* 空间增长带来的问题，甚至可以不用优先级队列。

1. 启发式搜索算法基础

在讲解启发式搜索算法之前先简单介绍状态空间搜索。状态空间搜索，就是将问题求解过程表现为从初始状态到目标状态寻找这个路径的过程。通俗来说，就是在解一个问题时，找到一个解题的过程，可从求解的开始到问题的结果。由于求解问题的过程中有很多分支，主要是求解过程中求解条件的不确定性和不完备性造成的，这使得求解的路径很多。这就构成了一个图，通常将这个图称为状态空间。问题的求解实际上就是在这个图中找到一条路径可以从开始到结果。这个寻找的过程就是状态空间搜索。

常用的状态空间搜索有深度优先和广度优先。广度优先是从初始状态一层一层向下找，直到找到目标为止。深度优先是按照一定的顺序查找完一个分支，再查找另一个分支，直到找到目标为止。这两种算法在数据结构书中都有描述，可以参看这些书得到更详细的解释。

前面说的广度和深度优先搜索有一个很大的缺陷就是它们都是在一个给定的状态

空间中穷举。这在状态空间不大的情况下是很合适的算法，可是当状态空间十分大，且不预测的情况下就不可取了。其效率实在太低，甚至不可完成。在这里就要用到启发式搜索。

先来分析估价是如何表示的。启发中的估价是用估价函数表示的，例如：

$$f(n) = g(n) + h(n)$$

其中 $f(n)$ 是节点 n 的估价函数， $g(n)$ 是在状态空间中从初始节点到 n 节点的实际代价， $h(n)$ 是从 n 到目标节点最佳路径的估计代价。在这里主要是 $h(n)$ 体现了搜索的启发信息，因为 $g(n)$ 是已知的。如果说详细点， $g(n)$ 代表了搜索的广度的优先趋势。但是当 $h(n) \gg g(n)$ 时，可以省略 $g(n)$ 而提高效率。

一种具有 $f(n)=g(n)+h(n)$ 策略的启发式算法能成为 A* 算法的充分条件。

(1) 搜索树上存在着从起始点到终了点的最优路径。

(2) 问题域是有限的。

(3) 所有节点的子节点的搜索代价值 > 0 。

(4) $h(n) \leq h^*(n)$ ，其中 $h^*(n)$ 表示实际问题的代价值。

当上述 4 个条件都满足时，一个具有 $f(n)=g(n)+h(n)$ 策略的启发式算法就能成为 A* 算法，并一定能找到最优解。对于一个搜索问题，显然，条件 (1)、(2)、(3) 都是很容易满足的，而条件 (4) 则需要精心设计，因为 $h^*(n)$ 是无法知道的。所以，一个满足条件 (4) 的启发策略 $h(n)$ 就来得难能可贵了。对于图的最优路径搜索和八数码问题，有些相关策略 $h(n)$ 不仅很好理解，而且已经在理论上证明是满足条件 (4) 的，从而为这个算法的推广起到了决定性的作用。不过 $h(n)$ 距离 $h^*(n)$ 的程度不能过大，否则 $h(n)$ 就没有过强的区分能力，算法效率并不会很高。对一个好的 $h(n)$ 的评价是： $h(n)$ 在 $h^*(n)$ 的下界之下，并且尽量接近 $h^*(n)$ 。

当然，估值函数的设计仅是 $f(n)=g(n)+h(n)$ 一种，另外的估值函数“变种”如 $f(n)=w \cdot g(n) + (1-w) \cdot h(n)$ ， $f(n)=g(n)+h(n)+h(n-1)$ 针对不同的具体问题亦会有不同的效果。

2. 初识 A* 算法

启发式搜索其实有很多的算法，比如局部择优搜索法、最好优先搜索法等。当然 A* 算法也是如此。这些算法都使用了启发函数，但在具体的选取最佳搜索节点时的策略不同。例如，局部择优搜索法就是在搜索的过程中选取“最佳节点”后舍弃其他的兄弟节点和父亲节点，并且一直得搜索下去。这种搜索的结果很明显，由于舍弃了其他的节点，可能也把最好的节点都舍弃了，因为求解的最佳节点只是在该阶段的最佳并不一定是全局的最佳。最好优先就比较“聪明”一些，其在搜索时，没有舍弃节点（除非该节点是死节点），在每一步的估价中都把当前的节点和以前的节点的估价值比较得到一个“最佳的节点”。这样可以有效地防止丢失“最佳节点”。那么 A* 算法又是一种什么样的算法呢？其实 A* 算法也是一种最好优先的算法，只不过要加上一些约束条件。由于在对一些问题求解时，我们希望能够求解出状态空间搜索的最短路径，也就是用最快的方法求解问题，这就是 A* 算法的任务。在此先下个定义，如果一个估价函数可以找出最短的路径，就称之为可采纳。A* 算法是一个可采纳的最好优先算法。A* 算法的估价函数可表示为：

$$f'(n) = g'(n) + h'(n)$$

此处的 $f'(n)$ 是估价函数, $g'(n)$ 是起点到终点的最短路径值, $h'(n)$ 是 n 到目标的最短路径的启发值。由于无法预先知道这个 $f'(n)$, 所以用前面的估价函数 $f(n)$ 做近似。 $g(n)$ 代替 $g'(n)$, 但 $g(n) \geq g'(n)$ 才可 (大多数情况下都是满足的, 可以不用考虑), $h(n)$ 代替 $h'(n)$, 但 $h(n) \leq h'(n)$ 才可 (这一点特别重要)。可以证明应用这样的估价函数可以找到最短路径, 也就是可采纳的。应用这种估价函数的最好优先算法就是 A* 算法。

举一个例子, 其实广度优先算法就是 A* 算法的特例。其中 $g(n)$ 是节点所在的层数, $h(n)=0$, 这种 $h(n)$ 肯定小于 $h'(n)$, 所以由前述可知广度优先算法是一种可采纳的算法。实际上正是如此。当然它是一种最差的 A* 算法。

接下来讲解有关 $h(n)$ 启发函数的信息性。 $h(n)$ 的信息性通俗来说其实就是在估计一个节点的值时的约束条件, 如果信息越多或约束条件越多则排除的节点就越多, 估价函数也就越好, 这就是广度优先算法的缺陷, 因为其 $h(n)$ 启发值为 0, 一点启发信息都没有。但在游戏开发中由于实时性的问题, $h(n)$ 的信息越多, 它的计算量就越大, 耗费的时间就越多。就应该适当减小 $h(n)$ 的信息, 即减小约束条件。但算法的准确性就差了, 因为这里就有一个平衡的问题。

3. A* 算法实现框架

在 A* 算法中的几个重要数据解释如下。

- Open Table: 存放所有已探知的但未搜索过点的优先队列。
- Closed Table: 存放搜索过的点的数组, 提取最优路径时有用。
- Start Node: 起始点。
- Target Node: 终止点。
- C Node: 当前点。

提取最优路径的算法并不复杂, 虽然在 close 表中会有许多无效的搜索点, 但是最优路径上各节点的下标一定是按照 close 表中下标的升序排列的。因此, 只要在 close 表中, 就将下标从终止点向起始点移动, 若 $close[i+1]$ 与 $close[i]$ 没有关联, 则剔除 $close[i]$ 。

路径最优问题就是在两个节点之间找一条最短路径。肯定有读者禁不住要问, 这个问题不是已经有 Dijkstra 算法可以解决了吗? 是的, 但是不要忘了 Dijkstra 算法的复杂度是 $O(n^2)$, 一旦节点很多并且需要实时计算, Dijkstra 就无法满足要求了。而用 A* 算法来处理这类有需要实时要求的问题则显得游刃有余。

在路径最优问题中, 用来作为启发函数关键部分的 $h(n)$ 其实很容易选, 那便是当前节点至最终节点的距离, 这个距离既可以是 Hamilton 距离 ($|x1-x2|+|y1-y2|$), 亦可以是 Euclid 距离 (直线距离)。都可以在较快的速度下达到问题的最优解。

4. 深入 A* 算法

A* 算法是最好优先算法的一种, 只是有一些约束条件而已。接下来看一看最好优先算法是如何编写的。在图 8-4 中有如下状态空间。

- 起始位置是 A。
- 目标位置是 P。
- 字母后的数字表示节点的估价值。

在搜索过程中设置两个表，分别是 OPEN 和 CLOSED。OPEN 表保存了所有已生成而未考查的节点，CLOSED 表中记录已访问过的节点。算法中有一步是根据估价函数重排 OPEN 表。这样循环中的每一步只考虑 OPEN 表中状态最好的节点。具体搜索过程如下。

(1) 初始状态：

```
OPEN=[A5]; CLOSED=[];
```

(2) 估算 A5，取得搜有子节点，并放入 OPEN 表中：

```
OPEN=[B4, C4, D6]; CLOSED=[A5]
```

(3) 估算 B4，取得搜有子节点，并放入 OPEN 表中：

```
OPEN=[C4, E5, F5, D6]; CLOSED=[B4, A5]
```

(4) 估算 C4；取得搜有子节点，并放入 OPEN 表中：

```
OPEN=[H3, G4, E5, F5, D6]; CLOSED=[C4, B4, A5]
```

(5) 估算 H3，取得搜有子节点，并放入 OPEN 表中：

```
OPEN=[O2, P3, G4, E5, F5, D6]; CLOSED=[H3, C4, B4, A5]
```

(6) 估算 O2，取得搜有子节点，并放入 OPEN 表中：

```
OPEN=[P3, G4, E5, F5, D6]; CLOSED=[O2, H3, C4, B4, A5]
```

(7) 估算 P3，已得到解。

分析了上述具体过程后，接下来再分析伪程序，此算法的伪程序如下。

```
Best_First_Search()
{
    Open = [起始节点];
    Closed = [];
    while (Open 表非空)
    {
        从 Open 中取得一个节点 X，并从 OPEN 表中删除
        if (X 是目标节点)
        {
            求得路径 PATH;
            返回路径 PATH;
        }
        for (每一个 X 的子节点 Y)
        {
            if (Y 不在 OPEN 表和 CLOSE 表中)
            {
                求 Y 的估价值;
                并将 Y 插入 OPEN 表中;
            }
            //还没有排序
        }
        else if (Y 在 OPEN 表中)
```

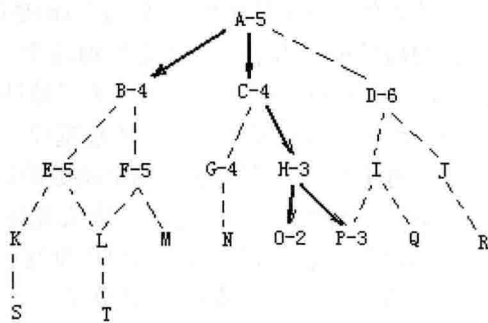


图 8-4 A*算法过程

```

{
    if (Y 的估价值小于 OPEN 表的估价值)
        更新 OPEN 表中的估价值;
    }
    else //Y 在 CLOSE 表中
    {
        if (Y 的估价值小于 CLOSE 表的估价值)
        {
            更新 CLOSE 表中的估价值;
            从 CLOSE 表中移出节点, 并放入 OPEN 表中;
        }
    }
    将 X 节点插入 CLOSE 表中;
    按照估价值将 OPEN 表中的节点排序;
} //end for
} //end while
} //end func

```

具体的 A* 算法程序与上述程序是一样的, 只要注意估价函数中的 $g(n)$ 的 $h(n)$ 约束条件就可以了。

5. 用 A* 算法实现最短路径的搜索

在游戏中, 经常要涉及最短路径的搜索, 现在一个比较好的方法就是用 A* 算法进行设计。A* 算法的核心是估价函数 $f(n)$, 它包括 $g(n)$ 和 $h(n)$ 两部分。 $g(n)$ 是已经走过的代价, $h(n)$ 是 n 到目标的估计代价。在这个例子中 $g(n)$ 表示在状态空间从起始节点到 n 节点的深度, $h(n)$ 表示 n 节点所在地图的位置到目标位置的直线距离。一个是状态空间, 一个是实际的地图。再详细点说, 有一个物体 A, 在地图上的坐标是 (x_a, y_a) , A 所要到达的目标 b 的坐标是 (x_b, y_b) 。则开始搜索时, 设置一个起始节点 1, 生成 8 个子节点 2~9 (因为有 8 个方向)。如图 8-5 所示。

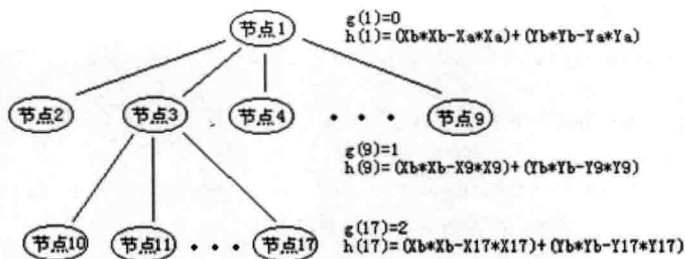


图 8-5 节点图

读者需要仔细看清节点 1、9、17 的 $g(n)$ 和 $h(n)$ 是怎么计算的。接下来开始讲解源程序, 其实这个程序是一个很典型的类似教科书中的程序, 只要看懂了上述伪程序, 这个程序是十分容易理解的。

先看搜索主函数:

```
void AstarPathfinder::FindPath(int sx, int sy, int dx, int dy) {
```

```

NODE *Node, *BestNode;
int TileNumDest;
//得到目标位置, 作判断用
TileNumDest = TileNum(sx, sy);
//生成 Open 和 Closed 表
OPEN = ( NODE* )calloc(1,sizeof( NODE ));
CLOSED=( NODE* )calloc(1,sizeof( NODE ));
//生成起始节点, 并放入 Open 表中
Node=( NODE* )calloc(1,sizeof( NODE ));
Node->g = 0;
//这是计算 h 值
// should really use sqrt().
Node->h = (dx-sx)*(dx-sx) + (dy-sy)*(dy-sy);
//这是计算 f 值, 即估价值
Node->f = Node->g+Node->h;
Node->NodeNum = TileNum(dx, dy);
Node->x = dx; Node->y = dy;
// make Open List point to first node
OPEN->NextNode=Node;
for (;;)
{
    //从 Open 表中取得一个估价值最好的节点
    BestNode=ReturnBestNode();
    //如果该节点是目标节点就退出
    // if we've found the end, break and finish break;
    if (BestNode->NodeNum == TileNumDest)
        //否则生成子节点
        GenerateSuccessors(BestNode,sx,sy);
}
PATH = BestNode;
}

```

再看生成子节点函数:

```

void AstarPathfinder::GenerateSuccessors(NODE *BestNode, int dx, int dy){
    int x, y;
    //依次生成 8 个方向的子节点
    // Upper-Left
    if ( FreeTile(x=BestNode->x-TILESIZE, y=BestNode->y-TILESIZE) )
        GenerateSucc(BestNode,x,y,dx,dy);
    // Upper
    if ( FreeTile(x=BestNode->x, y=BestNode->y-TILESIZE) )
        GenerateSucc(BestNode,x,y,dx,dy);
    // Upper-Right
    if ( FreeTile(x=BestNode->x+TILESIZE, y=BestNode->y-TILESIZE) )
        GenerateSucc(BestNode,x,y,dx,dy);
    // Right
    if ( FreeTile(x=BestNode->x+TILESIZE, y=BestNode->y) )
        GenerateSucc(BestNode,x,y,dx,dy);
    // Lower-Right

```

```

if ( FreeTile(x=BestNode->x+TILESIZE, y=BestNode->y+TILESIZE) )
    GenerateSucc(BestNode,x,y,dx,dy);
// Lower
if ( FreeTile(x=BestNode->x, y=BestNode->y+TILESIZE) )
    GenerateSucc(BestNode,x,y,dx,dy);
// Lower-Left
if ( FreeTile(x=BestNode->x-TILESIZE, y=BestNode->y+TILESIZE) )
    GenerateSucc(BestNode,x,y,dx,dy);
// Left
if ( FreeTile(x=BestNode->x-TILESIZE, y=BestNode->y) )
    GenerateSucc(BestNode,x,y,dx,dy);
}

```

接下来看最重要的 A*算法函数:

```

void AstarPathfinder::GenerateSucc(NODE *BestNode,int x, int y, int dx, int dy)
{
    int g, TileNumS, c = 0;
    NODE *Old, *Successor;
    //计算子节点的 g 值
    // g(Successor)=g(BestNode)+cost of getting from BestNode to Successor
    g = BestNode->g+1;
    // identification purposes
    TileNumS = TileNum(x,y);
    //子节点在 Open 表中吗
    // if equal to NULL then not in OPEN list, else it returns the Node in Old
    if ( (Old=CheckOPEN(TileNumS)) != NULL )
    {
        //若在
        for( c = 0; c < 8; c++)
            // Add Old to the list of BestNode's Children (or Successors).
            if( BestNode->Child[c] == NULL )
                break;
        BestNode->Child[c] = Old;
        //比较 Open 表中的估价值和当前的估价值 (只要比较 g 值即可)
        // if our new g value is < Old's then reset Old's parent to point to BestNode
        if ( g < Old->g )
        {
            //当前的估价值小就更新 Open 表中的估价值
            Old->Parent = BestNode;
            Old->g = g;
            Old->f = g + Old->h;
        }
    }
    else
        //在 Closed 表中吗
        // if equal to NULL then not in OPEN list, else it returns the Node in Old
        if ( (Old=CheckCLOSED(TileNumS)) != NULL )
        {
            //若在

```

```

for( c = 0; c< 8; c++)
// Add Old to the list of BestNode's Children (or Successors).
    if ( BestNode->Child[c] == NULL )
        break;
BestNode->Child[c] = Old;
//比较 Closed 表中的估价值和当前的估价值 (只要比较 g 值即可)
// if our new g value is < Old's then reset Old's parent to point to BestNode
if ( g < Old->g )
{
    //当前的估价值小就更新 Closed 表中的估价值
    Old->Parent = BestNode;
    Old->g = g;
    Old->f = g + Old->h;
    //再依次更新 Old 的所有子节点的估价值
    // Since we changed the g value of Old, we need
    // to propagate this new value downwards, i.e.
    // do a Depth-First traversal of the tree!
    PropagateDown(Old);
}
}
//不在 Open 表中也不在 Closed 表中
else
{
    //生成新的节点
    Successor = ( NODE* )calloc(1,sizeof( NODE ));
    Successor->Parent = BestNode;
    Successor->g = g;
    // should do sqrt(), but since we don't really
    Successor->h = (x-dx)*(x-dx) + (y-dy)*(y-dy);
    // care about the distance but just which branch looks
    Successor->f = g+Successor->h;
    // better this should suffice. Anyayz it's faster.
    Successor->x = x;
    Successor->y = y;
    Successor->NodeNum = TileNumS;
    //再插入 Open 表中, 同时排序
    // Insert Successor on OPEN list wrt f
    Insert(Successor);
    for( c =0; c < 8; c++)
    // Add Old to the list of BestNode's Children (or Successors).
    if ( BestNode->Child[c] == NULL )
        break;
    BestNode->Child[c] = Successor;
}
}

```

上述代码非常简单, 笔者已经在里面加上了详细的注释, 读者应务必结合图 8-5 做到彻底理解。

8.3 实战演练——各种 AI 图搜索算法在 Android 游戏中的用法

经过本章前面内容的学习，读者了解了游戏中人工智能 AI 搜索算法的基本知识。在本节的内容中，将通过一个具体实例的实现过程，演示各种 AI 图搜索算法在 Android 游戏中的具体用法。

实 例	功 能	源码路径
实例 8-1	演示各种 AI 图搜索算法在 Android 游戏中的具体用法	光盘\daima\8\AILI

8.3.1 搭建路径搜索框架

- (1) 将素材图片保存在项目的 `res\drawable-mdpi` 目录下，如图 8-6 所示。
- (2) 实现 UI 界面布局，如图 8-7 所示。

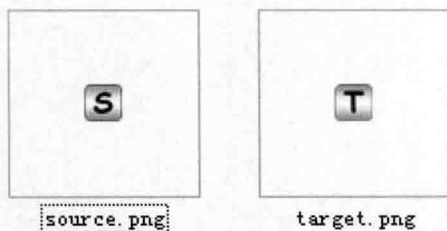


图 8-6 准备素材图片

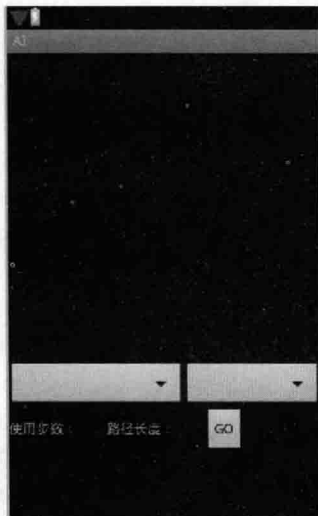


图 8-7 界面布局

布局文件 `main.xml` 的主要代码如下。

```
<LinearLayout
    android:id="@+id/LinearLayout02"
    android:layout_width="wrap_content"
    android:layout_height="320dip">
</LinearLayout>
<LinearLayout
    android:id="@+id/LinearLayout01"
    android:orientation="horizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <Spinner
        android:id="@+id/Spinner01"
        android:layout_width="180dip"
        android:layout_height="wrap_content">
```

```

</Spinner>

<Spinner
    android:id="@+id/Spinner02"
    android:layout_width="140dip"
    android:layout_height="wrap_content">
</Spinner>
</LinearLayout>

<LinearLayout
    android:id="@+id/LinearLayout03"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <TextView
        android:text="@string/sybz"
        android:id="@+id/TextView01"
        android:layout_width="100dip"
        android:layout_height="wrap_content">
    </TextView>

    <TextView
        android:text="@string/ljcd"
        android:id="@+id/TextView02"
        android:layout_width="100dip"
        android:layout_height="wrap_content">
    </TextView>

    <Button
        android:text="@string/GO"
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </Button>

```

(3) 编写文件 Game.java 实现算法类框架，此文件的具体实现流程如下。

- ① 定义绘制类并设置搜索过程。
- ② 定义系统常量来表示不同算法的标志。
- ③ 定义 clearState()实现初始化引用。
- ④ 通过 switch 语句进行相应的算法处理。

文件 Game.java 的主要代码如下。

```

public class Game{
    MySurfaceView mSurfaceView;                //创建绘制类引用
    int[][] map=MapList.map[0];                //需要搜索的地图
    int[] source=MapList.source;                //出发点坐标
    int[] target=MapList.targetA[0];            //目标点 col,row
    int algorithmId=0;                           //算法代号, 0--深度测试

    ArrayList<int[][]> searchProcess=new ArrayList<int[][]>(); //搜索过程

```

```

Stack<int[][]> stack=new Stack<int[][]>(); //深度优先所用栈
LinkedList<int[][]> queue=new LinkedList<int[][]>(); //广度优先所用队列
PriorityQueue<int[][]> astarQueue=new PriorityQueue<int[][]>(100,new
AxingComparator(this)); //A*优先级队列
HashMap<String,int[][]> hm=new HashMap<String,int[][]>(); //结果路径记录
int[][] visited=new int[19][19]; //0 未去过 1 已去过
int[][] length=new int[19][19]; //记录路径长度 for Dijkstra
// 记录到每个点的最短路径 for Dijkstra
HashMap<String,ArrayList<int[][]>> hmPath=new HashMap<String,ArrayList
<int[][]>>();

boolean pathFlag=false; //true 找到路径
int timeSpan=10; //时间间隔
SurfaceHolder holder;
int[][] sequence=
{
    {0,1},{0,-1},
    {-1,0},{1,0},
    {-1,1},{-1,-1},
    {1,-1},{1,1}
};

int tempCount; //记录各搜索方法所用步数
final int DFS_COUNT=1; //深度优先使用步数标志
final int BFS_COUNT=2; //广度优先使用步数标志
final int BFSASTAR_COUNT=3; //广度优先使用步数标志
final int DIJKSTRA_COUNT=4; //Dijkstra 使用步数标志
final int DIJKSTRASTAR_COUNT=5; //DijkstraA*使用步数标志
public Game(MySurfaceView mSurfaceView,SurfaceHolder holder)
{
    this.mSurfaceView=mSurfaceView;
    this.holder=holder;
}
public void clearState() //初始化各引用
{
    searchProcess.clear(); //清空搜索过程列表
    stack.clear(); //清空深度优先所用栈
    queue.clear(); //清空广度优先所用队列
    astarQueue.clear(); //清空 A*优先级队列
    hm.clear(); //清空结果路径记录
    visited=new int[19][19]; //初始化数组
    pathFlag=false; //寻找路径标志位
    hmPath.clear(); //清空 Dijkstra 中记录到每个点的最短路径
    mSurfaceView.paint.setStrokeWidth(0); //初始化画笔
    for(int i=0;i<length.length;i++)
    {
        for(int j=0;j<length[0].length;j++)
        {
            length[i][j]=9999; //设置初始路径的长度（不可能这么大）
        }
    }
}

```

```

    }
}
public void runAlgorithm()
{
    clearState();           //调用初始化方法
    switch(algorithmId)
    {
        case 0:              //深度优先算法
            DFS();
            break;
        case 1:              //广度优先算法
            BFS();
            break;
        case 2:              //广度优先 A*算法
            BFSAStar();
            break;
        case 3:              //Dijkstra 算法
            Dijkstra();
            Log.d("Dijkstra", "algorithmId="+algorithmId);
            break;
        case 4:              //DijkstraA*算法
            DijkstraAStar();
            break;
    }
}
}

```

(4) 编写文件 MapList.java, 此文件的具体实现流程如下。

- ① 定义地图类 MapList, 在此类包含了所有的地图信息。
- ② 通过定义目标点的位置数组可以添加多个目标点。
- ③ 在运行时选择不同的目标点来测试同一种搜索算法在不同情况下的运行效果。

文件 MapList.java 的主要代码如下。

```

public class MapList {
    static int[][][] map=new int[][][]//地图
    {
        {
            {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
            {1,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1},
            {1,0,0,1,0,0,1,1,0,0,0,0,1,1,0,0,0,1},
            {1,0,0,1,0,0,1,1,0,1,1,0,1,1,0,1,1,0,1},
            {1,0,0,1,0,0,1,1,0,1,1,0,1,1,0,1,1,0,1},
            {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
            {1,1,1,1,1,1,1,0,0,1,0,0,0,0,0,1,0,0,1},
            {1,1,1,1,1,1,1,0,1,1,1,0,0,0,1,1,1,0,1},
            {1,1,1,1,1,1,1,0,0,1,0,0,0,0,0,1,0,0,1},
            {1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,1},
            {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
            {0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0},
            {0,0,1,1,1,1,1,0,0,0,0,0,0,1,1,0,0,0},

```

```

        {0,0,1,1,1,1,1,0,0,0,0,0,1,1,1,1,0,0,0},
        {0,0,1,1,1,1,1,0,0,0,0,0,1,1,1,1,0,0,0},
        {0,0,0,1,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
    }
};
static int[] source={2,2};           //出发点坐标
static int[][] targetA=              //目标点坐标
{
    {18,12},{11,9},{12,5},
    {3,18},{15,18}
};
}

```

(5) 编写文件 `MySurfaceView`，在此实现路径搜索功能。此文件的具体实现流程如下。

① 绘制一个大的矩形作为外框，并对地图进行循环检测，为 0 时表示绘制白色矩形，为 1 时表示绘制黑色障碍物。

② 绘制搜索过程，分别绘制深度优先、广度优先、广度优先 A*算法和 Dijkstra 的搜索过程和结果。

③ 在指定位置绘制出发点和目的地。

文件 `MySurfaceView` 的主要实现代码如下。

```

implements SurfaceHolder.Callback {           //实现生命周期回调接口
    MyActivity mActivity;                      //activity 引用
    Paint paint;                              //画笔引用
    Game game=new Game(this, getHolder());    //创建对象
    final float span=15.7f;                   //矩形大小
    final int LJCD_COUNT=6;                   //路径长度
    int LjcdCount;                            //路径长度
    public MySurfaceView(MyActivity mActivity) {
        super(mActivity);
        // TODO Auto-generated constructor stub
        this.mActivity=mActivity;
        this.getHolder().addCallback(this);  //设置生命周期回调接口的实现者
        paint = new Paint();                 //创建画笔
        paint.setAntiAlias(true);            //打开抗锯齿
    }
    public void onDraw(Canvas canvas){
        int map[][]=game.map;                //获取地图
        int row=map.length;                  //地图行数
        int col=map[0].length;               //地图列数
        canvas.drawARGB(255, 128, 128, 128); //设置背景颜色
        int width=(int)span*map.length;      //画布宽度
        int hight=(int)span*map[0].length;    //画布长度
        canvas.setViewport(width, hight);     //设置画布大小
        for(int i=0;i<row;i++)               //绘制地图
        {
            for(int j=0;j<col;j++)
            {

```



```

        if(map[i][j]==1)
        {
            paint.setColor(Color.BLACK);    //设置画笔颜色为黑色
        }
        else if(map[i][j]==0)
        {
            paint.setColor(Color.WHITE);    //设置画笔颜色为白色
        }
    }

    //绘制矩形
    canvas.drawRect(2+j*(span+1),2+i*(span+1),2+j*(span+1)+span,2+i*(span+1)+span,
    paint);
}

//绘制寻找过程
ArrayList<int[][]> searchProcess=game.searchProcess;
for(int k=0;k<searchProcess.size();k++)
{
    int[][] edge=searchProcess.get(k);
    paint.setColor(Color.BLACK);    //设置画笔颜色
    canvas.drawLine
    (
        edge[0][0]*(span+1)+span/2+2, edge[0][1]*(span+1)+span/2+2,
        edge[1][0]*(span+1)+span/2+2, edge[1][1]*(span+1)+span/2+2, paint
    );
}

//绘制结果路径
if(
    mActivity.mySurfaceView.game.algorithmId==0||
    mActivity.mySurfaceView.game.algorithmId==1||
    mActivity.mySurfaceView.game.algorithmId==2
)
{
    //深度优先,广度优先,广度优先A*
    if(game.pathFlag)
    {
        HashMap<String,int[][]> hm=game.hm;
        int[] temp=game.target;
        int count=0;    //路径长度计数器
        while(true)
        {
            int[][] tempA=hm.get(temp[0]+":"+temp[1]); //获取结果路径记录
            paint.setColor(Color.BLACK);    //设置画笔黑色
            paint.setStrokeWidth(3);    //设置画笔宽度
            canvas.drawLine    //绘制线段
            (
                tempA[0][0]*(span+1)+span/2+2,tempA[0][1]*(span+1)+
                span/2+2,
                tempA[1][0]*(span+1)+span/2+2,tempA[1][1]*(span+1)+span/
                2+2,paint
            );
        }
    }
}

```

```

    );

    count++;
    //判断是否到出发点
    if (tempA[1][0]==game.source[0]&&tempA[1][1]==game.source[1])
    {
        break;
    }

    temp=tempA[1];
}
LjcdCount=count; //记录路径长度
mActivity.hd.sendMessage(LJCD_COUNT); //更改路径长度
}
else if(
    mActivity.mySurfaceView.game.algorithmId==3||
    mActivity.mySurfaceView.game.algorithmId==4
)
{
    //Dijkstra 路径绘制
    if (game.pathFlag)
    {
        Log.d(game.pathFlag+"*****", "dijkst");
        HashMap<String, ArrayList<int[] []>> hmPath=game.hmPath;
        ArrayList<int[] []> alPath=hmPath.get(game.target[0]+"-"+game.target[1]);
        for (int[] [] tempA:alPath)
        {
            paint.setColor (Color.BLACK);
            paint.setStrokeWidth (3);
            canvas.drawLine
            (
                tempA[0][0]*(span+1)+span/2+2,tempA[0][1]*(span+1)+span/2+2,
                tempA[1][0]*(span+1)+span/2+2,tempA[1][1]*(span+1)+span/2+2,paint
            );
        }
        LjcdCount=alPath.size(); //记录路径长度
        mActivity.hd.sendMessage(LJCD_COUNT); //更改路径长度
    }
}

//绘制出发点
Bitmap bitmapTmpS=BitmapFactory.decodeResource (mActivity.getResources(),
R.drawable.source);
canvas.drawBitmap (bitmapTmpS, game.source[0]*(span+1),game.source[1]*(span+1), paint);

```

```

        //绘制目标点
        Bitmap bitmapTmpT=BitmapFactory.decodeResource(mActivity.getResources(),
        R.drawable.target);
        canvas.drawBitmap(bitmapTmpT, game.target[0]*(span+1),game.target[1]*
        (span+1), paint);
    }

    public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3) {

    }

    public void surfaceCreated(SurfaceHolder holder) { //创建时被调用
        Canvas canvas = holder.lockCanvas(); //获取画布
        try{
            synchronized(holder){
                onDraw(canvas); //绘制
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }
        finally{
            if(canvas != null){
                holder.unlockCanvasAndPost(canvas);
            }
        }
    }

    public void repaint(SurfaceHolder holder)
    {
        Canvas canvas = holder.lockCanvas(); //获取画布
        try{
            synchronized(holder){
                onDraw(canvas); //绘制
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }
        finally{
            if(canvas != null){
                holder.unlockCanvasAndPost(canvas);
            }
        }
    }

    public void surfaceDestroyed(SurfaceHolder arg0) { //销毁时被调用
    }
}

```

(6) 编写实例文件 AxingComparator.java, 在此文件中定义了一个使用 A*算法的比较器类 AxingComparator, 主要代码如下。

```

public class AxingComparator implements Comparator<int[][]>{
    Game game;
    public AxingComparator(Game game)
    {
        this.game=game;
    }

    public int compare(int[][] o1,int[][] o2)
    {
        int[] t1=o1[1];
        int[] t2=o2[1];

        int[] target=game.target;
        //蒙特卡罗距离
        int a=game.visited[o2[0][1]][o2[0][0]]+Math.abs(t1[0]-target[0])+
            Math.abs(t1[1]-target[1]);
        int b=game.visited[o2[0][1]][o2[0][0]]+Math.abs(t2[0]-target[0])+
            Math.abs(t2[1]-target[1]);
        return a-b;
    }
    public boolean equals(Object obj)
    {
        return false;
    }
}

```

此时执行之后的效果如图 8-8 所示。

8.3.2 实现深度优先算法

在实例文件 Game.java 中定义方法 DFS(), 通过深度优先算法检索路径, 主要代码如下。

```

public void DFS() //深度优先
{
    new Thread()
    {
        public void run()
        {
            boolean flag=true; //线程标志位
            int[][] start= //初始化出发点坐标
            {
                {source[0],source[1]},
                {source[0],source[1]}
            };
            stack.push(start); //入栈
            int count=0; //使用步数计数器
            while(flag)
            {
                int[][] currentEdge=stack.pop(); //从栈中取出边
                int[] tempTarget=currentEdge[1]; //取出此边的目的点
            }
        }
    }
}

```



图 8-8 执行效果


```

//判断目的点是否去过, 若去过, 则直接进入下次循环
if (visited[tempTarget[1]][tempTarget[0]]==1)
{
    continue;
}
count++; //计数器自加
//表示目的点被访问过
visited[tempTarget[1]][tempTarget[0]]=1;

//将临时目标点加入搜索过程中
searchProcess.add(currentEdge);
//记录此临时节点的父节点
hm.put(tempTarget[0]+"."+tempTarget[1], new int[][]{currentEdge[1],
currentEdge[0]});
//重绘画布
mSurfaceView.repaint(holder);
//线程睡眠一定时间
try{Thread.sleep(timeSpan);}catch(Exception e){e.printStackTrace();}
//判断是否到达目的点
if (tempTarget[0]==target[0]&&tempTarget[1]==target[1])
{
    break;
}
//将所有可能的边入栈
int currCol=tempTarget[0]; //取边节点
int currRow=tempTarget[1];
for(int[] rc:sequence) //扫描该点附近所有可能的边
{
    int i=rc[1];
    int j=rc[0];
    if (i==0&&j==0){continue;} //若为 0, 0 结束该次循环
    if (currRow+i>=0&&currRow+i<19&&currCol+j>=0&&currCol+j<19&&
map[currRow+i][currCol+j]!=1) //若在地图内
    {
        int[][] tempEdge=
        {
            {tempTarget[0],tempTarget[1]},
            {currCol+j,currRow+i}
        };
        stack.push(tempEdge); //入栈
    }
}

pathFlag=true; //标志位设为 true
mSurfaceView.repaint(holder); //重绘画布
tempCount=count; //深度优先使用步数
mSurfaceView.mActivity.hd.sendEmptyMessage(DFS_COUNT);
//发送消息更改使用步数数量
mSurfaceView.mActivity.button.setClickable(true);

```



```

        //设置 button 可以点击
    }
    }.start();
}

```

执行后可以在界面中选择“深度优先算法”实现路径选择，如图 8-9 所示为用深度优先算法到目的地 A 的执行效果。

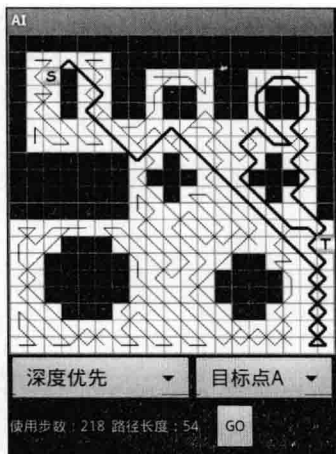


图 8-9 执行效果

8.3.3 实现广度优先算法

在实例文件 Game.java 中定义方法 BFS(), 通过广度优先算法检索路径，主要代码如下。

```

public void BFS()                                //广度优先
{
    new Thread()
    {
        public void run()
        {
            int count=0;                          //计数器
            boolean flag=true;                    //循环标志位
            int[][] start=                        //开始状态
            {
                {source[0],source[1]},
                {source[0],source[1]}
            };
            queue.offer(start);                    //将开始点加入该队列的末尾

            while(flag)
            {
                int[][] currentEdge=queue.poll(); //获取并移除表头
                int[] tempTarget=currentEdge[1];  //取出此边的目的点

                //判断是否去过，若去过则直接进入下次循环
                if(visited[tempTarget[1]][tempTarget[0]]==1)
                {
                    continue;
                }
                count++;                            //计数器自加
                //将去过的点置为 1
                visited[tempTarget[1]][tempTarget[0]]=1;

                //降临时目标点加入搜索过程
                searchProcess.add(currentEdge);
                //记录此临时节点的父节点
                hm.put(tempTarget[0]+"-"+tempTarget[1],new int[][]{currentEdge[1],
                    currentEdge[0]});
            }
        }
    }
}

```

```

//重绘画布
mSurfaceView.repaint(holder);
//线程睡眠一定时间
try{Thread.sleep(timeSpan);}catch(Exception e){e.printStackTrace();}

//判断是否为目的点
if(tempTarget[0]==target[0]&&tempTarget[1]==target[1])
{
    break;
}

//将所有可能的边加入队列
int currCol=tempTarget[0];
int currRow=tempTarget[1];

for(int[] rc:sequence)
{
    int i=rc[1];
    int j=rc[0];

    if(i==0&&j==0){continue;} //若在地图外面,进入下一次循环
    if(currRow+i>=0&&currRow+i<19&&currCol+j>=0&&currCol+j<19&&
    map[currRow+i][currCol+j]!=1) //若为地图内的点
    {
        int[][] tempEdge=
        {
            {tempTarget[0],tempTarget[1]},
            {currCol+j,currRow+i}
        };
        queue.offer(tempEdge); //将该点加入队列末尾
    }
}

pathFlag=true; //标志位设为 true
mSurfaceView.repaint(holder); //重绘画布
tempCount=count; //广度优先使用步数
mSurfaceView.mActivity.hd.sendMessage
(BFS_COUNT);
//发送消息更改使用步数数量
mSurfaceView.mActivity.button.setClickable
(true);
//设置 button 键可以点击
}
}.start();
}

```

执行后可以在界面中选择“广度优先算法”实现路径选择,如图 8-10 所示为用广度优先算法到目的地 A 的执行效果。

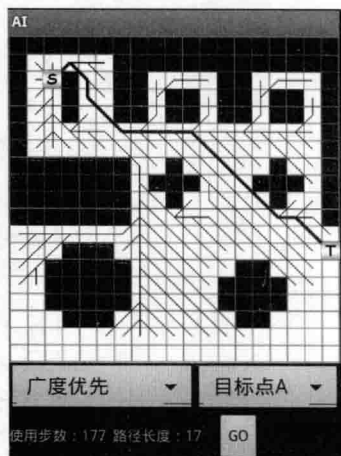


图 8-10 执行效果

8.3.4 实现 Dijkstra 算法

在实例文件 Game.java 中定义方法 Dijkstra(), 通过 Dijkstra 算法检索路径, 主要代码如下。

```
public void Dijkstra()
{
    new Thread()
    {
        public void run()
        {
            int count=0;                //步骤计数器
            boolean flag=true;          //搜索循环设置
            //开始点
            int[] start={source[0],source[1]}; //col,row
            visited[source[1]][source[0]]=1;
            //计算此点所有可以到达点的路径及长度
            for(int[] rowcol:sequence)
            {
                int trow=start[1]+rowcol[1];
                int tcol=start[0]+rowcol[0];
                if(trow<0||trow>18||tcol<0||tcol>18)continue;
                if(map[trow][tcol]!=0)continue;

                //记录路径长度
                length[trow][tcol]=1;
                //计算路径
                String key=tcol+"-"+trow;
                ArrayList<int[][]> al=new ArrayList<int[][]>();
                al.add(new int[][]{{start[0],start[1]},{tcol,trow}});
                hmPath.put(key,al);
                //将去过的点记录
                searchProcess.add(new int[][]{{start[0],start[1]},{tcol,trow}});
                count++;
            }
            mSurfaceView.repaint(holder); //重绘
            outer:while(flag)
            {
                //找到当前扩展点K, 要求扩展点K为从开始点到此点目前路径最短, 且此点未考查过
                int[] k=new int[2];
                int minLen=9999;
                for(int i=0;i<visited.length;i++)
                {
                    for(int j=0;j<visited[0].length;j++)
                    {
                        if(visited[i][j]==0)
                        {
                            if(minLen>length[i][j])
                            {
                                minLen=length[i][j];
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```

        k[0]=j;                //col
        k[1]=i;                //row
    }
}

//设置去过的点
visited[k[1]][k[0]]=1;
//重绘
mSurfaceView.repaint(holder);
//取出开始点到 K 的路径长度
int dk=length[k[1]][k[0]];
//取出开始点到 K 的路径
ArrayList<int[][]> al=hmPath.get(k[0]+":"+k[1]);

//循环计算所有 K 点能直接到的点到开始点的路径长度
for(int[] rowcol:sequence)
{
    //计算出新的要计算的点的坐标
    int trow=k[1]+rowcol[1];
    int tcol=k[0]+rowcol[0];

    //若要计算的点超出地图边界或地图上此位置为障碍物则舍弃考查此点
    if(trow<0||trow>18||tcol<0||tcol>18) continue;
    if(map[trow][tcol]!=0) continue;

    //取出开始点到此点的路径长度
    int dj=length[trow][tcol];
    //计算经 K 点到此点的路径长度
    int dkPluskj=dk+1;

    //若经 K 点到此点的路径长度比原来的小则修改到此点的路径
    if(dj>dkPluskj)
    {
        String key=tcol+":"+trow;
        //克隆开始点到 K 的路径
        ArrayList<int[][]> tempal=(ArrayList<int[][]>)al.clone();
        //在从 K 到此点的路径中加上一步
        tempal.add(new int[][]{{k[0],k[1]},{tcol,trow}});
        //将此路径设置为从开始点到此点的路径
        hmPath.put(key,tempal);
        //修改从开始点到此点的路径长度
        length[trow][tcol]=dkPluskj;
        //如果此点从未计算过路径长度则将此点加入考查过程记录
        if(dj==9999)
        {
            //将去过的点记录
            searchProcess.add(new int[][]{{k[0],k[1]},{tcol,trow}});
            count++;
        }
    }
}

```

```

    }
}

//看是否找到目的点
if (tcol==target[0]&&trow==target[1])
{
    pathFlag=true;
    tempCount=count;           //Dijkstra 使用步数
    mSurfaceView.mActivity.hd.sendMessage(DIJKSTRA_COUNT);
    //发送消息更改使用步数
    mSurfaceView.mActivity.button.setClickable(true);
    mSurfaceView.repaint(holder);
    break outer;
}
}
try{Thread.sleep(timeSpan);}catch
(Exception e){e.printStackTrace();}
}
}.start();
}

```

执行后可以在界面中选择 Dijkstra 实现路径选择, 如图 8-11 所示为用 Dijkstra 算法到目的地 A 的执行效果。

8.3.5 实现广度优先 A*算法

在实例文件 Game.java 中定义方法 BFSAStar(), 通过广度优先 A*算法检索路径, 主要代码如下。

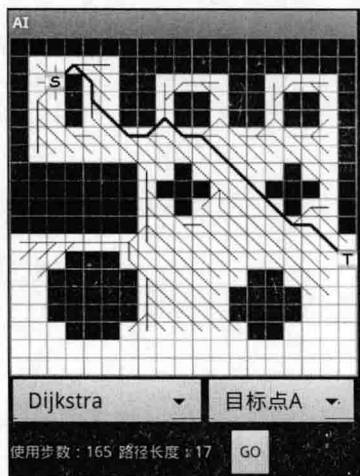


图 8-11 执行效果

```

public void BFSAStar()                                     //广度优先 A*
{
    new Thread()
    {
        public void run()
        {
            boolean flag=true;
            int[][] start=                                  //开始状态
            {
                {source[0],source[1]},
                {source[0],source[1]}
            };
            astarQueue.offer(start);                        //将开始点加入队列末尾
            int count=0;                                    //计数器
            while(flag)
            {
                int[][] currentEdge=astarQueue.poll();     //获取表头, 并将表头移除
                int[] tempTarget=currentEdge[1];           //取此边的目标点
                //判断是否去过, 若去过则直接进入下次循环
                if(visited[tempTarget[1]][tempTarget[0]]!=0)
                {

```



```

        continue;
    }
    count++;
    //表示目标点为访问过
    visited[tempTarget[1]][tempTarget[0]]=visited[currentEdge
    [0][1]][currentEdge[0][0]]+1;
    //将临时目标点加入搜索过程中
    searchProcess.add(currentEdge);
    //记录此临时节点的父节点
    hm.put(tempTarget[0]+":"+tempTarget[1],new int[][]
    {currentEdge[1],currentEdge[0]});
    //重绘画布
    mSurfaceView.repaint(holder);
    try{Thread.sleep(timespan);}catch(Exception e){e.
    printStackTrace();}
    //判断是否为目标点
    if(tempTarget[0]==target[0]&&tempTarget[1]==target[1])
    {
        break;
    }
    //将所有可能的边加入队列
    int currCol=tempTarget[0];
    int currRow=tempTarget[1];
    for(int[] rc:sequence)
    {
        int i=rc[1];
        int j=rc[0];
        if(i==0&&j==0){continue;}
        if(currRow+i>=0&&currRow+i<19&&currCol+j>=0&&currCol+j<19&&
        map[currRow+i][currCol+j]!=1)
        {
            int[][] tempEdge=
            {
                {tempTarget[0],tempTarget[1]},
                {currCol+j,currRow+i}
            };
            astarQueue.offer(tempEdge); //加入队列末尾
        }
    }
}
pathFlag=true;
mSurfaceView.repaint(holder);
tempCount=count; //广度优先 A*使用步数
//发送消息更改使用步数数量
mSurfaceView.mActivity.hd.sendMessage(BFSASTAR_COUNT);
mSurfaceView.mActivity.button.setClickable(true);
//设置 button 的状态为可点击
}
}.start();
}

```

执行后可以在界面中选择“广度优先 A*”实现路径选择,如图 8-12 所示为用广度优先 A*算法到目的地 A 的执行效果。

8.3.6 实现 Dijkstra A*算法

在实例文件 Game.java 中定义方法 DijkstraAStar(),通过 Dijkstra A*算法检索路径,主要代码如下。

```
public void DijkstraAStar() //Dijkstra A*算法
{
    new Thread()
    {
        public void run()
        {
            int count=0;          //步数计数器
            boolean flag=true;    //搜索循环控制
            //开始点
            int[] start={source[0],source[1]};
            visited[source[1]][source[0]]=1;
            //计算此点所有可以到达点的路径
            for(int[] rowcol:sequence)
            {
                int trow=start[1]+rowcol[1];
                int tcol=start[0]+rowcol[0];
                if(trow<0||trow>18||tcol<0||tcol>18)continue;
                if(map[trow][tcol]!=0)continue;

                //记录路径长度
                length[trow][tcol]=1;

                //计算路径
                String key=tcol+"-"+trow;
                ArrayList<int[][]> al=new ArrayList<int[][]>();
                al.add(new int[][]{{start[0],start[1]},{tcol,trow}});
                hmPath.put(key,al);
                //将去过的点记录
                searchProcess.add(new int[][]{{start[0],start[1]},{tcol,trow}});
                count++;
            }
            mSurfaceView.repaint(holder);
            outer:while(flag)
            {
                int[] k=new int[2];
                int minLen=9999;
                boolean iniFlag=true;
                for(int i=0;i<visited.length;i++)
                {
                    for(int j=0;j<visited[0].length;j++)
                    {
```

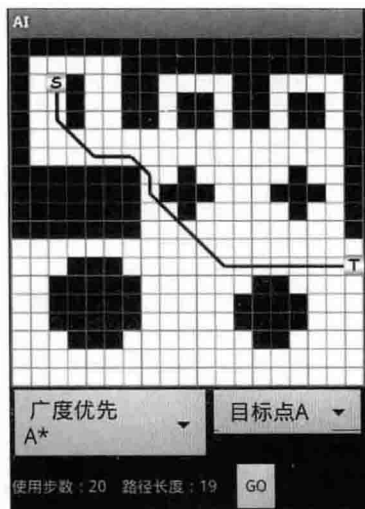


图 8-12 执行效果

```

        if (visited[i][j] == 0)
        {
            //与普通 Dijkstra 算法的区别部分
            if (length[i][j] != 9999)
            {
                if (iniFlag)
                {
                    //第一个找到的可能点
                    minLen = length[i][j] +
                        (int) Math.sqrt((j - target[0]) * (j - target[0]) + (i - target[1]) * (i - target[1]));
                    k[0] = j;
                    k[1] = i;
                    iniFlag = !iniFlag;
                }
                else
                {
                    int tempLen = length[i][j] +
                        (int) Math.sqrt((j - target[0]) * (j - target[0]) + (i - target[1]) * (i - target[1]));
                    if (minLen > tempLen)
                    {
                        minLen = tempLen;
                        k[0] = j;
                        k[1] = i;
                    }
                }
            }
            //与普通 Dijkstra 算法的区别部分
        }
    }

    //设置去过的点
    visited[k[1]][k[0]] = 1;
    //重绘
    mSurfaceView.repaint(holder);
    int dk = length[k[1]][k[0]];
    ArrayList<int[][]> al = hmPath.get(k[0] + ":" + k[1]);
    //循环计算所有 K 点能直接到的点到开始点的路径长度
    for (int[] rowcol : sequence)
    {
        //计算出新的要计算的点的坐标
        int trow = k[1] + rowcol[1];
        int tcol = k[0] + rowcol[0];
        //若要计算的点超出地图边界或地图上此位置为障碍物则舍弃考查此点
        if (trow < 0 || trow > 18 || tcol < 0 || tcol > 18) continue;
        if (map[trow][tcol] != 0) continue;
        //取出开始点到此点的路径长度
        int dj = length[trow][tcol];
        //计算经 K 点到此点的路径长度
        int dkPluskj = dk + 1;
    }
}

```

```

//若经K点到此点的路径长度比原来的小则修改到此点的路径
if(dj>dkPluskj)
{
    String key=tcol+":"+trow;
    ArrayList<int[][]> tempal=(ArrayList<int[][]>)al.clone();
    tempal.add(new int[][]{{k[0],k[1]},{tcol,trow}});
    hmPath.put(key,tempal);
    length[trow][tcol]=dkPluskj;

    if(dj==9999)
    {
        //将去过的点记录
        searchProcess.add(new int[][]{{k[0],k[1]},{tcol,trow}});
        count++;
    }
}
//看是否找到目的点
if(tcol==target[0]&&trow==target[1])
{
    Log.d("target[0]="+target[0], "target[1]="+target[1]);
    pathFlag=true;
    tempCount=count;           //Dijkstra A*使用步数
    mSurfaceView.mActivity.hd.sendEmptyMessage
    (DIJKSTRASTAR_COUNT);      //更改使用步数数量
    mSurfaceView.mActivity.button.setClickable(true);
    mSurfaceView.repaint(holder);
    break outter;
}
}
try{Thread.sleep(timeSpan);}catch(Exception e){e.printStackTrace();}
}
}.start();
}

```

执行后可以在界面中选择“广度优先 A*”实现路径选择，如图 8-13 所示为用 Dijkstra A*算法到目的地 A 的执行效果。

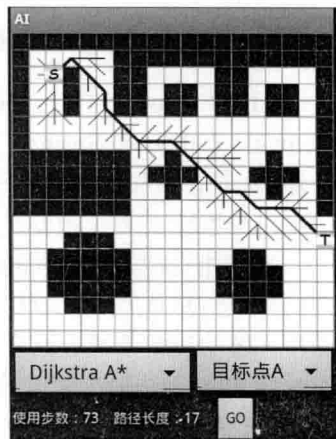


图 8-13 执行效果

语音识别和手势识别

语音识别技术是 Android SDK 中比较重要且比较新颖的一项技术，在 Android 穿戴设备应用中可以通过语音识别技术来控制设备。另外，在 Android 穿戴设备开发过程中，手势识别技术也是最重要的应用之一。在本章的内容中，将详细讲解 Android 语音识别和手势识别技术的基本知识，为读者学习本书后面的知识打下基础。

9.1 语音识别技术

语音识别技术是 Android SDK 中比较重要且比较新颖的一项技术。本节将详细讲解 Android 语音识别技术的基本知识，为读者学习本书后面的知识打下基础。

9.1.1 Text-To-Speech 技术

Text-To-Speech 简称 TTS，是 Android 1.6 版本中比较重要的新功能。将所指定的文本转成不同语言音频输出。它可以方便地嵌入到游戏或者应用程序中，以增强用户体验。在讲解 TTS API 和将这项功能应用到实际项目中的方法之前，先对这套 TTS 引擎有个初步的了解。

1. Text-To-Speech 基础

TTS Engine 依托于当前 AndroidPlatform 所支持的几种主要的语言：English、French、German、Italian 和 Spanish 五大语言。TTS 可以将文本随意地转换成以上任意 5 种语言的语音输出。与此同时，对于个别的语言版本将取决于不同的时区，例如对于 English，在 TTS 中可以分别输出美式和英式两种不同的版本。

既然能支持如此庞大的数据量，TTS 引擎对于资源的优化采取预加载的方法。根据一系列的参数信息从库中提取相应的资源，并加载到当前系统中。尽管当前大部分加载有 Android 操作系统的设备都通过这套引擎来提供 TTS 功能，但由于一些设备的存储空间非常有限，而影响到 TTS 无法最大限度地发挥功能，算是当前的一个技术瓶颈。为此开发小组引入了检测模块，让利用这项技术的应用程序或者游戏针对不同的设备可以有相应的优化调整，从而避免由于此项功能的限制，影响到整个应用程序的使用。比较稳妥的做法是让用户自行选择是

否有足够的空间或者需求来加载此项资源，一个标准的检测方法如下。

```
Intent checkIntent = new Intent();
checkIntent.setAction(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
startActivityForResult(checkIntent, MY_DATA_CHECK_CODE);
```

如果当前系统允许创建一个 `android.speech.tts.TextToSpeech` 的 Object 对象，则说明已经提供 TTS 功能的支持，将检测返回结果中给出 `CHECK_VOICE_DATA_PASS` 的标记。如果系统不支持这项功能，那么用户可以选择是否加载这项功能，从而让设备支持输出多国语言的语音功能 `Multi-lingual Talking`。ACTION_INSTALL_TTS_DATA Intent 将用户引入 Android market 中的 TTS 下载界面。下载完成后将自动完成安装，实现上述过程的完整代码 (androidres.com) 如下。

```
private TextToSpeech mTts;
protected void onActivityResult(
    int requestCode, int resultCode, Intent data) {
    if (requestCode == MY_DATA_CHECK_CODE) {
        if (resultCode == TextToSpeech.Engine.CHECK_VOICE_DATA_PASS) {
            // success, create the TTS instance
            mTts = new TextToSpeech(this, this);
        } else {
            // missing data, install it
            Intent installIntent = new Intent();
            installIntent.setAction(
                TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA);
            startActivity(installIntent);
        }
    }
}
```

TextToSpeech 实体和 OnInitListener 都需要引用当前 Activity 的 Context 作为构造参数。OnInitListener() 的用处是通知系统当前 TTS Engine 已经加载完成，并处于可用状态。

2. Text-To-Speech 的实现流程

(1) 首先检查 TTS 数据是否可用，代码如下。

```
view plaincopy to clipboardprint?
//检查 TTS 数据是否已经安装并且可用
Intent checkIntent = new Intent();
checkIntent.setAction(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
startActivityForResult(checkIntent, REQ_TTS_STATUS_CHECK);
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if(requestCode == REQ_TTS_STATUS_CHECK)
    {
        switch (resultCode) {
            case TextToSpeech.Engine.CHECK_VOICE_DATA_PASS:
                //这个返回结果表明 TTS Engine 可以用
                {
                    mTts = new TextToSpeech(this, this);
```

```

        Log.v(TAG, "TTS Engine is installed!");
    }
    break;
case TextToSpeech.Engine.CHECK_VOICE_DATA_BAD_DATA:
    //需要的语音数据已损坏
case TextToSpeech.Engine.CHECK_VOICE_DATA_MISSING_DATA:
    //缺少需要语言的语音数据
case TextToSpeech.Engine.CHECK_VOICE_DATA_MISSING_VOLUME:
    //缺少需要语言的发音数据
    {
        //这三种情况都表明数据有错,重新下载安装需要的数据
        Log.v(TAG, "Need language stuff:"+resultCode);
        Intent dataIntent = new Intent();
        dataIntent.setAction(TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA);
        startActivity(dataIntent);
    }
    break;
case TextToSpeech.Engine.CHECK_VOICE_DATA_FAIL:
    //检查失败
default:
    Log.v(TAG, "Got a failure. TTS apparently not available");
    break;
}
}
else
{
    //其他 Intent 返回的结果
}
}
}

```

(2) 然后初始化 TTS，代码如下。

```

view plaincopy to clipboardprint?
//实现 TTS 初始化接口
@Override
public void onInit(int status) {
    // TODO Auto-generated method stub
    //TTS Engine 初始化完成
    if(status == TextToSpeech.SUCCESS)
    {
        int result = mTts.setLanguage(Locale.US);
        //设置发音语言
        if(result == TextToSpeech.LANG_MISSING_DATA || result == TextToSpeech.
        LANG_NOT_SUPPORTED)
        //判断语言是否可用
        {
            Log.v(TAG, "Language is not available");
            speakBtn.setEnabled(false);
        }
    }
    else

```

```

        {
            mTts.speak("This is an example of speech synthesis.", TextToSpeech.
                QUEUE_ADD, null);
            speakBtn.setEnabled(true);
        }
    }
}

```

(3) 接下来需要设置发音语言，代码如下。

```

view plaincopy to clipboardprint?
public void onItemSelected(AdapterView<?> parent, View view,
    int position, long id) {
    // TODO Auto-generated method stub
    int pos = langSelect.getSelectedItemPosition();
    int result = -1;
    switch (pos) {
        case 0:
        {
            inputText.setText("I love you");
            result = mTts.setLanguage(Locale.US);
        }
        break;
        case 1:
        {
            inputText.setText("Je t'aime");
            result = mTts.setLanguage(Locale.FRENCH);
        }
        break;
        case 2:
        {
            inputText.setText("Ich liebe dich");
            result = mTts.setLanguage(Locale.GERMAN);
        }
        break;
        case 3:
        {
            inputText.setText("Ti amo");
            result = mTts.setLanguage(Locale.ITALIAN);
        }
        break;
        case 4:
        {
            inputText.setText("Te quiero");
            result = mTts.setLanguage(new Locale("spa", "ESP"));
        }
        break;
        default:
            break;
    }
    //设置发音语言
}

```

```

if(result == TextToSpeech.LANG_MISSING_DATA || result == TextToSpeech.
LANG_NOT_SUPPORTED)
//判断语言是否可用
{
    Log.v(TAG, "Language is not available");
    speakBtn.setEnabled(false);
}
else
{
    speakBtn.setEnabled(true);
}
}

```

(4) 最近设置单击 Button 按钮发出声音，代码如下。

```

view plaincopy to clipboardprint?
public void onClick(View v) {
    //朗读输入框里的内容
    mTts.speak(inputText.getText().toString(), TextToSpeech.QUEUE_ADD, null);
}

```

9.1.2 谷歌的 Voice Recognition 技术

iPhone 的语音识别用的是 Google 的技术，作为 Google 力推的 Android 自然会将其核心技术往 Android 系统里面植入，并结合 Google 的云端技术将其发扬光大。所以 Google Voice Recognition 在 Android 的实现就变得极其轻松，例如，在它自带的 API 例子中，是通过一个 Intent 的 Action 动作来实现语音识别的。主要有以下两种模式。

- ACTION_RECOGNIZE_SPEECH：一般语音识别，在这种模式下可以捕捉到语音的处理后的文字列。
- ACTION_WEB_SEARCH：网络搜索。

下面分析在 Api Demo 源码中提供的语音识别实例，具体实现代码如下。

```

package com.example.android.apis.app;

import com.example.android.apis.R;

import android.app.Activity;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.content.pm.ResolveInfo;
import android.os.Bundle;
import android.speech.RecognizerIntent;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.ListView;

import java.util.ArrayList;
import java.util.List;

```

```
/**
 *用 API 开发的抽象语音识别代码
 */
public class VoiceRecognition extends Activity implements OnClickListener {

    private static final int VOICE_RECOGNITION_REQUEST_CODE = 1234;

    private ListView mList;

    /**
     *呼叫与活动首先被创造
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //从它的 XML 布局描述的 UI
        setContentView(R.layout.voice_recognition);

        //得到最新交互作用的显示项目
        Button speakButton = (Button) findViewById(R.id.btn_speak);
        mList = (ListView) findViewById(R.id.list);

        //检查公认活动是否存在
        PackageManager pm = getPackageManager();
        List<ResolveInfo> activities = pm.queryIntentActivities(
            new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH), 0);
        if (activities.size() != 0) {
            speakButton.setOnClickListener(this);
        } else {
            speakButton.setEnabled(false);
            speakButton.setText("Recognizer not present");
        }
    }

    /**
     *单击“开始识别按钮”后的处理事件
     */
    public void onClick(View v) {
        if (v.getId() == R.id.btn_speak) {
            startVoiceRecognitionActivity();
        }
    }

    /**
     *发送开始语音识别信号
     */
    private void startVoiceRecognitionActivity() {
        Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
            RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
    }
}
```



```

        intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "Speech recognition demo");
        startActivityForResult(intent, VOICE_RECOGNITION_REQUEST_CODE);
    }

    /**
     * 处理识别结果
     */
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == VOICE_RECOGNITION_REQUEST_CODE && resultCode == RESULT_OK) {
            // Fill the list view with the strings the recognizer thought it could
            // have heard
            ArrayList<String> matches = data.getStringArrayListExtra(
                RecognizerIntent.EXTRA_RESULTS);
            mList.setAdapter(new ArrayAdapter<String>(this, android.R.layout.
                simple_list_item_1,
                matches));
        }
        super.onActivityResult(requestCode, resultCode, data);
    }
}

```

上述代码保存在 Google 的 API 开源文件中，原理和实现代码十分简单，感兴趣的读者可以学习一下，上述源码的命令执行后，用户通过单击 **Speak!** 按钮显示界面，如图 9-1 所示；用户说完话后将提交到云端搜索，如图 9-2 所示；在云端搜索完成后将返回打印数据，如图 9-3 所示。



图 9-1 单击 **Speak!** 按钮后



图 9-2 用户说完话后

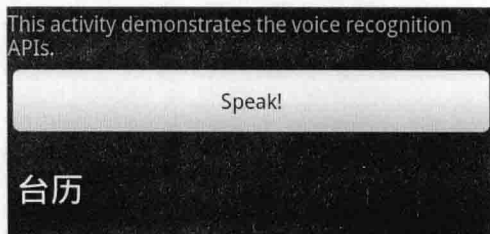


图 9-3 返回识别结果

9.2 手势识别

对于触摸屏设备来说,其消息传递机制包括按下、抬起和移动这几种,用户只需简单地实现重载 `onTouch` 或者设置触摸侦听器 `setOnTouchListener` 即可处理触摸事件。但是有时为了提高应用程序的用户体验,需要识别用户的手势。本节将详细讲解在 Android 设备中实现手势识别的基本知识。

9.2.1 类 `GestureDetector` 基础

在 Android 系统中,专门提供了手势识别类 `GestureDetector`。在 Android 设备中,通过类 `GestureDetector` 可以识别很多手势,通过其 `nTouchEvent(event)` 方法可以完成不同手势的识别。类 `GestureDetector` 对外提供了两个接口: `OnGestureListener` 和 `OnDoubleTapListener`,另外还提供了一个内部类 `SimpleOnGestureListener`。

(1) `GestureDetector.OnDoubleTapListener` 接口:用来通知 `DoubleTap` 事件,类似于鼠标的双击事件。此接口中各个成员的具体说明如下。

- `onDoubleTap(MotionEvent e)`: 在二次双击 `Touch down` 时触发。
- `onDoubleTapEvent(MotionEvent e)`: 通知 `DoubleTap` 手势中的事件,包含 `down`、`up` 和 `move` 事件(这里指的是在双击之间发生的事件,例如,在同一个地方双击会产生 `DoubleTap` 手势,而在 `DoubleTap` 手势里面还会发生 `down` 和 `up` 事件,这两个事件由该函数通知);双击 `Touch down` 和 `up` 都会触发,可用 `e.getAction()` 区分。
- `onSingleTapConfirmed(MotionEvent e)`: 用来判定该次点击是 `SingleTap` 而不是 `DoubleTap`,如果连续点击两次就是 `DoubleTap` 手势,如果只点击一次,系统等待一段时间后没有收到第二次点击则判定该次点击为 `SingleTap` 而不是 `DoubleTap`,然后触发 `SingleTapConfirmed` 事件。这个方法不同于 `onSingleTapUp`,它是在 `GestureDetector` 确信用户在第一次触摸屏幕后,没有紧跟着第二次触摸屏幕,即并非在“双击”的时候触发。

(2) `GestureDetector.OnGestureListener` 接口:用来通知普通的手势事件,该接口有如下 6 个回调函数。

- `onDown(MotionEvent e)`: `down` 事件。
- `onSingleTapUp(MotionEvent e)`: 一次点击 `up` 事件,在 `touch down` 后没有滑动。
- `onLongPress`: 用户长接触摸屏,由多个 `MotionEvent ACTION_DOWN` 触发。
- `onShowPress(MotionEvent e)`: `down` 事件发生而 `move` 或 `up` 还没发生前触发该事件。
- `onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY)`: 滑动手势事件, `Touch` 后滑动一点距离,在 `ACTION_UP` 时才会触发。各个参数的具体说明如下。
 - `e1`: 第 1 个 `ACTION_DOWN MotionEvent` 并且只有一个。
 - `e2`: 最后一个 `ACTION_MOVE MotionEvent`。
 - `velocityX`: X 轴上的移动速度,像素/秒。
 - `velocityY`: Y 轴上的移动速度,像素/秒,触发条件: X 轴的坐标位移大于

FLING_MIN_DISTANCE, 且移动速度大于 FLING_MIN_VELOCITY 像素/秒。

- `onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY)`: 在屏幕上拖动事件。无论用手拖动 view, 还是以抛的动作滚动, 都会多次触发。这个方法在 ACTION_MOVE 动作发生时就会触发。

9.2.2 使用类 GestureDetector

Android 系统的事件处理机制是基于 Listener (监听器) 实现的, 触摸屏相关的事件就是通过 `onTouchListener` 实现的。另外, 在 Android 系统中, 所有 View 的子类都可以通过 `setOnTouchListener()`、`setOnKeyListener()` 等方法来添加对某一类事件的监听器。并且 Listener 一般会以 Interface (接口) 的方式来提供, 其中包含一个或多个 abstract (抽象) 方法, 我们需要实现这些方法来完成 `onTouch()`、`onKey()` 等的操作。这样, 当给某个 view 设置了事件 Listener, 并实现了其中的抽象方法以后, 程序便可以在特定的事件被 dispatch (派送) 到该 view 时, 通过 callback 函数给予适当的响应。

在 Android 开发应用中, 有多种使用类 GestureDetector 的方法。

1. 第一种

(1) 首先通过 GestureDetector 的构造方法将 SimpleOnGestureListener 对象传递进去, 这样 GestureDetector 就能处理不同的手势了。

```
public GestureDetector(Context context, GestureDetector.OnGestureListener listener)
```

(2) 然后在 onTouch 方法中实现 OnTouchListener 监听。

```
private OnTouchListener gestureTouchListener = new OnTouchListener() {
    public boolean onTouch(View v, MotionEvent event) {
        return gDetector.onTouchEvent(event);
    }
};
```

2. 第二种

(1) 首先使用如下方法构建场景。

```
private GestureDetector mGestureDetector;
mGestureListener = new BookOnGestureListener();
```

(2) 然后使用 new 新构造出来的 GestureDetector 对象。

```
mGestureDetector = new GestureDetector(mGestureListener);
class BookOnGestureListener implements OnGestureListener
```

(3) 最后实现事件处理。

```
public boolean onTouchEvent(MotionEvent event) {
    mGestureListener.onTouchEvent(event);
}
```

3. 第三种

(1) 首先在当前类中创建一个 GestureDetector 实例。

```
private GestureDetector mGestureDetector;
```

(2) 然后创建一个 Listener 来实时监听当前面板操作手势。

```
class LearnGestureListener extends GestureDetector.SimpleOnGestureListener
```

(3) 在初始化时, 将 Listener 实例关联当前的 GestureDetector 实例。

```
mGestureDetector = new GestureDetector(this, new LearnGestureListener());
```

(4) 使用方法 onTouchEvent 作为入口检测, 通过传递 MotionEvent 参数来监听操作手势。

```
mGestureDetector.onTouchEvent(event)
```

演示代码如下:

```
private GestureDetector mGestureDetector;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mGestureDetector = new GestureDetector(this, new LearnGestureListener());
}
@Override
public boolean onTouchEvent(MotionEvent event) {
    if (mGestureDetector.onTouchEvent(event))
        return true;
    else
        return false;
}
class LearnGestureListener extends GestureDetector.SimpleOnGestureListener{
    @Override
    public boolean onSingleTapUp(MotionEvent ev) {
        Log.d("onSingleTapUp",ev.toString());
        return true;
    }
    @Override
    public void onShowPress(MotionEvent ev) {
        Log.d("onShowPress",ev.toString());
    }
    @Override
    public void onLongPress(MotionEvent ev) {
        Log.d("onLongPress",ev.toString());
    }
    @Override
    public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX,
        float distanceY) {
        Log.d("onScroll",e1.toString());
        return true;
    }
    @Override
    public boolean onDown(MotionEvent ev) {
        Log.d("onDown",ev.toString());
        return true;
    }
    @Override
    public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
        float velocityY) {
```

```

        Log.d("d",e1.toString());
        Log.d("e2",e2.toString());
        return true;
    }
}

```

4. 第四种

(1) 首先创建一个 `GestureDetector` 的对象，传入 `listener` 对象，在接收到的 `onTouchEvent` 中将 `event` 传给 `GestureDetector` 进行分析，`listener` 会回调给相应的动作。

(2) 通过 `GestureDetector.SimpleOnGestureListener` (Framework 帮我们简化了) 实现了 `OnGestureListener` 和 `OnDoubleTapListener` 两个接口类，只需继承它并重写其中的回调即可。

(3) 设置在第一次单击 `down` 时，给 `Handler` 发送了一个延时的消息，例如，延时 300ms。如果在 300ms 里发生了第二次单击的 `down` 事件，那么就认为是双击事件，并移除之前发送的延时消息。如果 300ms 后仍没有第二次的 `down` 消息，那么就判定为 `SingleTapConfirmed` 事件（当然，此时用户的手指应已完成第一次单击的 `up` 过程）。第三次单击的判定和双击的判定类似，只是多了一次发送延时消息的过程。

演示代码如下：

```

private GestureDetector mGestureDetector;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mGestureDetector = new GestureDetector(this, new MyGestureListener());
}
@Override
public boolean onTouchEvent(MotionEvent event) {
    return mGestureDetector.onTouchEvent(event);
}
class MyGestureListener extends GestureDetector.SimpleOnGestureListener{
    @Override
    public boolean onSingleTapUp(MotionEvent ev) {
        Log.d("onSingleTapUp",ev.toString());
        return true;
    }
    @Override
    public void onShowPress(MotionEvent ev) {
        Log.d("onShowPress",ev.toString());
    }
    @Override
    public void onLongPress(MotionEvent ev) {
        Log.d("onLongPress",ev.toString());
    }
    ...
}

```


9.2.3 通过点击的方式移动图片

实 例	功 能	源码路径
实例 9-1	在屏幕中通过点击的方式移动图片	光盘\daima\9\move

1. 实例说明

在触摸屏手机中，点击移动照片的功能十分常见。在本实例中用 `ImageView` 控件来显示 `Drawale` 中的照片，在程序运行后将照片放在屏幕中央。通过 `onTouchEvent` 来处理点击、拖动、放开等事件来完成拖动图片的功能。并且设置了 `ImageView` 的单击监听事件，让用户在单击图片的同时恢复到图片的初始位置。

2. 具体实现

编写主程序文件，下面开始讲解其具体实现流程。

(1) 通过 `DisplayMetrics` 获取屏幕对象，分别用 `intScreenX` 和 `intScreenY` 取得屏幕解析像素并分别设置图片的宽、高。具体代码如下。

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    /* 取得屏幕对象 */
    DisplayMetrics dm = new DisplayMetrics();
    getWindowManager().getDefaultDisplay().getMetrics(dm);

    /* 取得屏幕解析像素 */
    intScreenX = dm.widthPixels;
    intScreenY = dm.heightPixels;

    /* 设置图片的宽、高 */
    intWidth = 100;
    intHeight = 100;
    .....
}
```

(2) 将图片从 `Drawable` 中赋值给 `ImageView` 控件来呈现在屏幕中，并通过方法 `RestoreButton()` 初始化按钮使其位置居中。具体代码如下。

```
/*通过 findViewById 构造器创建 ImageView 对象*/
mImageView01 =(ImageView) findViewById(R.id.myImageView1);
/*将图片从 Drawable 赋值给 ImageView 来呈现*/
mImageView01.setImageResource(R.drawable.baby);

/* 初始化按钮位置居中 */
RestoreButton();
```

(3) 定义点击监听事件 `setOnClickListener`，当用户点击 `ImageView` 图片时将图片还原到初始位置显示。具体代码如下。

```
/* 当点击 ImageView, 还原初始位置 */
mImageView01.setOnClickListener(new Button.OnClickListener()
```

```

{
    @Override
    public void onClick(View v)
    {
        RestoreButton();
    }
});
}

```

(4) 定义 `onTouchEvent(MotionEvent event)` 覆盖触控事件。首先取得手指触控屏幕的位置，然后实现触控事件的处理，分别实现点击屏幕、移动位置和离开屏幕这三个动作处理。具体代码如下。

```

/*覆盖触控事件*/
public boolean onTouchEvent(MotionEvent event)
{
    /*取得手指触控屏幕的位置*/
    float x = event.getX();
    float y = event.getY();

    try
    {
        /*触控事件的处理*/
        switch (event.getAction())
        {
            /*点击屏幕*/
            case MotionEvent.ACTION_DOWN:
                picMove(x, y);
                break;
            /*移动位置*/
            case MotionEvent.ACTION_MOVE:
                picMove(x, y);
                break;
            /*离开屏幕*/
            case MotionEvent.ACTION_UP:
                picMove(x, y);
                break;
        }
    } catch (Exception e)
    {
        e.printStackTrace();
    }
    return true;
}

```

(5) 定义方法 `picMove(float x, float y)` 来移动屏幕中的图片，具体代码如下。

```

/*移动图片的方法*/
private void picMove(float x, float y)
{
    /*默认微调图片与指针的相对位置*/
    mX=x-(intWidth/2);
}

```

```

mY=y-(intHeight/2);

/*防图片超过屏幕的相关处理*/
/*防止屏幕向右超过屏幕*/
if((mX+intWidth)>intScreenX)
{
    mX = intScreenX-intWidth;
}
/*防止屏幕向左超过屏幕*/
else if(mX<0)
{
    mX = 0;
}
/*防止屏幕向下超过屏幕*/
else if ((mY+intHeight)>intScreenY)
{
    mY=intScreenY-intHeight;
}
/*防止屏幕向上超过屏幕*/
else if (mY<0)
{
    mY = 0;
}
/*通过log 来查看图片位置*/
Log.i("jay", Float.toString(mX)+","+Float.toString(mY));
/* 以 setLayoutParams 方法, 重新安排 Layout 上的位置 */
mImageView01.setLayoutParams
(
    new AbsoluteLayout.LayoutParams
        (intWidth,intHeight,(int) mX,(int)mY)
);
}

```

(6) 定义方法 `RestoreButton()` 来还原 `ImageView` 图片到初始位置, 具体代码如下。

```

/* 还原 ImageView 位置的事件处理 */
public void RestoreButton()
{
    intDefaultX = ((intScreenX-intWidth)/2);
    intDefaultY = ((intScreenY-intHeight)/2);
    /*Toast 还原位置坐标*/
    mMakeTextToast
    (
        "("+
        Integer.toString(intDefaultX)+
        ","+
        Integer.toString(intDefaultY)+")",true
    );

    /* 以 setLayoutParams 方法, 重新安排 Layout 上的位置 */
    mImageView01.setLayoutParams

```

```
(  
    new AbsoluteLayout.LayoutParams  
        (intWidth,intHeight,intDefaultX,intDefaultY)  
    );  
}
```

执行后效果如图 9-4 所示，可以通过鼠标点击的方式移动图片的位置，如图 9-5 所示。



图 9-4 执行效果



图 9-5 移动图片



蓝牙技术基础

BLE (Bluetooth Low Energy, 低功耗蓝牙) 是对传统蓝牙 BR/EDR 技术的补充。尽管 BLE 和传统蓝牙都称为蓝牙标准, 且共享射频, 但是, BLE 是一个完全不一样的技术。BLE 不具备和传统蓝牙 BR/EDR 的兼容性。它是专为小数据率、离散传输的应用而设计的。通信距离上也有所改变, 传统蓝牙的传输距离为几十米到几百米不等, BLE 则规定为 100 米。在本章的内容中, 将详细讲解低功耗蓝牙协议栈的基本知识, 为读者学习本书后面的知识打下基础。

10.1 蓝牙概述



利用“蓝牙”技术, 能够有效地简化移动通信终端设备之间的通信, 也能够成功地简化设备与因特网 Internet 之间的通信, 从而数据传输变得更加迅速、高效, 为无线通信拓宽道路。蓝牙采用分散式网络结构及快跳频和短包技术, 支持点对点及点对多点通信, 工作在全球通用的 2.4GHz ISM (即工业、科学、医学) 频段。其数据速率为 1Mbps。采用时分双工传输方案实现全双工传输。

10.1.1 蓝牙技术的发展历程

蓝牙这一名称来自于第十世纪的一位丹麦国王 Harald Blatand, Blatand 在英文里可以解释为 Bluetooth。因为国王喜欢吃蓝莓, 牙龈每天都是蓝色的所以叫“蓝牙”。蓝牙的创始人是瑞典爱立信公司, 爱立信早在 1994 年就已进行研发。1997 年, 爱立信与其他设备生产商联系, 并激发了其对该项技术的浓厚兴趣。1998 年 2 月, 5 个跨国大公司, 包括爱立信、诺基亚、IBM、东芝及 Intel 组成了一个特殊兴趣小组 (SIG), 他们共同的目标是建立一个全球性的小范围无线通信技术, 即现在的蓝牙。

Bluetooth 无线技术规格供全球的成员公司免费使用。许多行业的制造商都积极地在其产品中实施此技术, 以减少使用零乱的电线, 实现无缝连接、流传输立体声, 传输数据或进行语音通信。Bluetooth 技术在 2.4 GHz 波段运行, 该波段是一种无须申请许可证的工业、科技、医学 (ISM) 无线电波段。正因如此, 使用 Bluetooth 技术不需要支付任何费用。但用户必须向手机提供商注册使用 GSM 或 CDMA, 除了设备费用外, 不需要为使用 Bluetooth 技术再支付任何费用。

Bluetooth 技术得到了空前广泛的应用, 集成该技术的产品从手机、汽车到医疗设备, 使用该技术的用户从消费者、工业市场到企业等, 不一而足。低功耗、小体积及低成本的芯片

解决方案使得 Bluetooth 技术甚至可以应用于极微小的设备中。

10.1.2 蓝牙的特点

Bluetooth 技术是一项即时技术，它不要求固定的基础设施，且易于安装和设置。不需要电缆即可实现连接。新用户使用亦不费力，只需拥有 Bluetooth 品牌产品，检查可用的配置文件，将其连接至使用同一配置文件的另一 Bluetooth 设备即可。后续的 PIN 码流程就如同在 ATM 机器上操作一样简单。外出时，用户可以随身带上其个人局域网（PAN），甚至可以与其他网络连接。

10.2 低功耗蓝牙基础

BLE（Bluetooth Low Energy，低功耗蓝牙）是对传统蓝牙 BR/EDR 技术的补充。本节内容将详细讲解低功耗蓝牙技术的基本知识。

10.2.1 低功耗蓝牙的架构

BLE 协议架构总体上分成 3 层，从下到上分别是：控制器（Controller）、主机（Host）和应用端（Apps）。上述三层既可以在同一芯片类中实现，也可以分别在不同芯片类内实现，控制器（Controller）用于处理射频数据解析、接收和发送，主机（Host）用于控制不同设备之间如何进行数据交换；应用端（Apps）实现具体应用。

（1）控制器 Controller

Controller 实现射频相关的模拟和数字部分，完成最基本的数据发送和接收，Controller 对外接口是天线，对内接口是主机控制器接口 HCI（Host Controller Interface）；控制器包含物理层 PHY（Physical Layer），链路层 LL（Linker Layer），直接测试模式 DTM（Direct Test Mode），以及主机控制器接口 HCI。

● 物理层 PHY

GFSK 信号调制，2 402M~2 480M，40 个 channel，每两个 channel 间隔 2MHz（经典蓝牙协议是 1MHz），数据传输速率是 1Mbps。

● 直接测试模式 DTM

为射频物理层测试接口、射频数据分析之用。

● 链路层 LL

基于物理层 PHY 之上，实现数据通道分发、状态切换、数据包校验、加密等；链路层 LL 分 2 种通道：广播通道（advertising channels）和数据通道（data channels）；广播通道有 3 个，37ch（2 402MHz）、38ch（2 426MHz）和 39ch（2 480MHz），每次广播都会往这 3 个通道同时发送（并不会在这 3 个通道之间跳频），为防止某个通道被其他设备阻塞，以致设备无法配对或广播数据，之所以定 3 个广播通道是一种权衡，少了可能会被阻塞，多了会加大功耗，还有一件有意思的事情是，3 个广播通道刚好避开了 wifi 的 1ch、6ch、11ch，所以在 BLE 广播的时候，不至于被 wifi 影响；当 BLE 匹配之后，链路层 LL 由广播通道切换到数据通道，数据通道有 37 个，数据传输时会在这 37 个通道间切换，切换规则在设备间匹配时约定。

(2) 主机 Host/控制器 Controller 接口 HCI

HCI 作为一种接口,存在于主机 Host 和控制器 Controller 当中,控制器 Host 通过 HCI 发送数据和事件给主机,主机 Host 通过 HCI 发送命令和数据给控制器 Controller。HCI 逻辑上定义一系列的命令、事件;物理上有 UART、SDIO、USB,实际可能包含里面的任意一种或几种。

10.2.2 低功耗蓝牙分类

BLE 通常应用在传感器和智能手机或者平板的通信中。到目前为止,只有很少的智能机和平板支持 BLE,如 iPhone 4S 以后的苹果手机, Motorola Razr 和 the new iPad 及其以后的 iPad。安卓手机也逐渐支持 BLE,安卓的 BLE 标准在 2013 年 7 月 24 日刚发布。智能机和平板会带双模蓝牙的基带和协议栈,协议栈中包括 GATT 及以下的所有部分,但是没有 GATT 之上的具体协议。所以,这些具体的协议需要在应用程序中实现,实现时需要基于各个 GATT API 集。这样有利于在智能机端简单地实现具体协议,也可以在智能机端简单地开发出一套基于 GATT 的私有协议。

在现实应用中,低功耗蓝牙分为单模 (Bluetooth Smart) 和双模 (Bluetooth Smart Ready) 两种设备。BLE 和蓝牙 BR/EDR 有所区分,这样可以用三种方式将蓝牙技术集成到具体设备中。因为不再是所有现有的蓝牙设备可以和另一个蓝牙设备进行互联,所以准确描述产品中蓝牙的版本是非常重要的。

(1) 单模蓝牙

单模蓝牙设备也称为 Bluetooth Smart 设备,并且有专用的 Logo,如图 10-1 所示。

在现实应用中,手表、运动传感器等小型设备通常是基于低功耗单模蓝牙的。为了实现极低的功耗效果,在硬件和软件上都进行了优化,这样的设备只能支持 BLE。单模蓝牙芯片往往是一个带有单模蓝牙协议栈的产品,这个协议栈通常由芯片商免费提供。

(2) 双模蓝牙

双模蓝牙设备也称为 Bluetooth Smart Ready 设备,并且有专用的 Logo,如图 10-2 所示。



图 10-1 Bluetooth Smart 设备



图 10-2 Bluetooth Smart Ready 设备

双模设备支持蓝牙 BR/EDR 和 BLE。在双模设备中, BR/EDR 和 BLE 技术使用同一个射频前端和天线。典型的双模设备有智能手机、平板电脑、PC 和 Gateway。这些设备可以接收到通过 BLE 或者蓝牙 BR/EDR 设备发送过来的数据,这些设备往往都有足够的供电能力。双模设备和 BLE 设备通信的功耗低于双模设备和蓝牙 BR/EDR 设备通信的功耗。在使用双模解决方案时,需要用外部处理器才可以实现蓝牙协议栈。

10.2.3 集成方式

尽管有单模和双模方案的区别，但是在设备中集成蓝牙技术的方式有多种，其中最为常用的方式有模块和芯片。

(1) 模块

在现实应用中，最简单和快速的方式是使用一个嵌入式模块。此类模块包含了天线、嵌入了协议栈并提供多种不同的接口：UART、USB、SPI 和 I²C，可以通过这些接口和用户的处理器连接。模块会提供一种简单的接口来控制蓝牙的功能。很多的模块公司都会提供带 CE、FCC 和 IC 认证的产品。这样的模块可以只是蓝牙 BR/EDR 的、双模式的或者单模式的。

如果是蓝牙 BR/EDR 和双模的方案，还可以采用 HCI 模块。HCI 模块只是不带蓝牙协议栈，其他的和上述的模块一样。所以，这样的模块会更便宜。HCI 模块只是提供了硬件接口，在这样的方案中，蓝牙协议栈需要第三方提供。这样的第三方协议栈需要能在主设备的处理器中运行，如斯图曼提供的 BlueCode+SR。使用 HCI 模块需要将软件移植到最终的硬件中。

从理论上讲，提供单模的 HCI 模块也是可以的。然而，所有的芯片公司都已经将 GATT 集成到他们的芯片中，所以市面上不会有 HCI 单模模块出现（见 5.4 节）。

(2) 芯片

通过芯片来集成 BLE 是从物料角度使成本最低的方式，但是，这需要很多的前期工作和花费大量的时间。虽然在软件上只需将协议栈移植到目标平台之中即可，但是，硬件方面则需要对 RF 的 layout 和天线的设计非常有经验。这些公司提供的 BLE 芯片有：Broadcom、CSR、EM Microelectronic、Nordic 和 TI。

10.2.4 低功耗蓝牙的特点

在实际应用过程中，BLE 的低功耗并不是通过优化空中的无线射频传输来实现的，而是通过改变协议的设计来实现的。为了实现极低的功耗效果，通常 BLE 协议设计为：在不必要射频时，彻底将空中射频关断。

与传统蓝牙 BR/EDR 相比，BLE 通过如下三大特性实现低功耗效果。

- 缩短无线开启时间。
- 快速建立连接。
- 降低收发峰值功耗（具体由芯片决定）。

缩短无线开启时间的第一个技巧是只用 3 个“广告”信道，第二个技巧是通过优化协议栈来降低工作周期。一个在广告的设备可以自动和一个在搜索的设备快速建立连接，所以可以在 3 毫秒内完成连接的建立和数据的传输。

在现实应用中，低功耗设计可能会带来一些“牺牲”，例如，音频数据无法通过 BLE 来进行传输。尽管如此，BLE 仍然是一种非常出色的技术，依然会支持跳频（37 个数据信道），并且采用了一种改进的 GFSK 调制方法来提高链路的稳定性。BLE 也仍是非常安全的技术，因为在芯片级提供了 128 bit AES 加密。

单模设备可以作为 Master 或者 Slave，但是不能同时充当两种角色。这意味着 BLE 只能

建立简单的星状拓扑,不能实现散射网。在 BLE 的无线电规范中,定义了低功耗蓝牙的最高数据率为 305kbps,但是,这只是理论数据。在实际应用中,数据的吞吐量取决于上层协议栈。而 UART 的速度、处理器的能力和主设备都会影响数据吞吐能力。

高的数据吞吐能力的 BLE 只有通过私有方案或者基于 ATT notification 才能实现。事实上,如果是高数据率或高数据量的应用,蓝牙 BR/EDR 通常显得更加省电。

10.2.5 BLE 和传统蓝牙 BR/EDR 技术的对比

BLE 和传统蓝牙 BR/EDR 技术的对比如表 10-1 所示。

表 10-1 BLE 和传统蓝牙 BR/EDR 技术的对比

	Bluetooth BR/EDR	Bluetooth low energy
Frequency	2 400M~2 483.5 MHz	2 400M~2 483.5 MHz
Deep Sleep	~80 μ A	<5 μ A
Idle	~8 mA	~1 mA
Peak Current	10~40 mA	10~30 mA
Range	500m (Class 1) / 50m (Class 2)	100m
Min. Output Power	0 dBm (Class 1) / -6 dBm (Class 2)	-20 dBm
Max. Output Power	+20 dBm (Class 1) / +4 dBm (Class 2)	+10 dBm
Receiver Sensitivity	≥ -70 dBm	≥ -70 dBm
Encryption	64 bit / 128 bit	AES-128 bit
Connection Time	100 ms	3 ms
Frequency Hopping	Yes	Yes
Advertising Channel	32	3
Data Channel	79	37
Voice capable	Yes	No

10.3 蓝牙规范

蓝牙规范即 Bluetooth Profile, Bluetooth SIG 定义了许多 Profile。Profile 的目的是要确保 Bluetooth 设备间的互通性 (interoperability), 但是 Bluetooth 产品无须实现所有的 Bluetooth 规范 Profile。在本节的内容中, 将详细讲解蓝牙规范的基本知识。

10.3.1 Bluetooth 系统中的常用规范

在 Bluetooth 系统中, 定义了如下常用的规范。

(1) 蓝牙立体声音讯传输协议 A2DP

蓝牙立体声音讯传输协议 (Advanced Audio Distribution Profile), 功能是播放立体声。

(2) 基本图像规范

基本图像规范 (Basic Imaging Profile) 的功能是在装置之间传送图像, 可以将其再细分

为如下类别。

- Image Push
- Image Pull
- Advanced Image Printing
- Automatic Archive
- Remote Camera
- Remote Display

(3) 基本打印规范

基本打印规范 (Basic Printing Profile) 可以将文件、电子邮件传至打印机打印, 主要包含如下分类。

- 无线电话规范 (Cordless Telephony Profile), 设置了蓝牙无线电话之间沟通的规范。
- 内通信规范 (Intercom Profile): 是另类的 TCS (Telephone Control protocol Specification) 基底规范, 两个 Bluetooth 通信设备间沟通的规范。
- 拨号网络规范: Baseband、LMP、L2CAP、SDP、RFCOMM 协定所需要的传输需求。
- 传真规范 (Fax Profile): 能传输传真的资料。
- 人机界面规范 (Human Interface Device Profile): 可以支援鼠标和键盘功能。
- 头戴式通话器规范 (Headset Profile): 能够将声音传送到蓝牙耳机设备。
- 序列埠规范 (Serial Port Profile): 用来取代有线的 RS-232 Cable。
- SIM 卡存取规范 (SIM Access Profile): 用于存取手机内的 SIM 卡。
- 同步规范 (Synchronization Profile): 建立在 serial port profile、generic access profile 与 generic access profile 之上。
- 档案传输规范 (File Transfer Profile): Bluetooth 可以利用 OBEX 通信协定来传送档案。
- 泛用存取规范 (Generic Access Profile): 用来建立连线。
- 泛用物件交换规范 (Generic Object Exchange Profile): 使用 OBEX 进行物件交换。
- 物件交换规范 (Object Push Profile): Bluetooth 利用 OBEX 通信协定在两个设备间交换资料。
- 个人局域网络规范 (Personal Area Networking Profile): 可以支持蓝牙网络第三层协定。
- 电话簿存取规范 (Phone Book Access Profile): 可以在装置之间互换电话簿。
- 影像分享规范 (Video Distribution Profile): 可以使用 H.263 编码算法来分享影像信息。

10.3.2 蓝牙协议体系结构

整个蓝牙协议体系结构可分为底层硬件模块、中间协议层和高端应用层三大部分。链路管理层 (LMP)、基带层 (BBP) 和蓝牙无线电信道构成蓝牙的底层模块。BBP 层负责跳频和蓝牙数据及信息帧的传输。LMP 层负责连接的建立和拆除, 以及链路的安全和控制, 它们为上层软件模块提供了不同的访问入口, 但是两个模块接口之间的消息和数据传递必须通过蓝牙主机控制器接口的解释才能进行。也就是说, 中间协议层包括逻辑链路控制与适配协议 (L2CAP)、服务发现协议 (SDP)、串口仿真协议 (RFCOMM) 和电话控制协议规范 (TCS)。L2CAP 完成数据拆装、服务质量控制、协议复用和组提取等功能, 是其他上层协议实现的基

础,因此也是蓝牙协议栈的核心部分。SDP 为上层应用程序提供一种机制来发现网络中可用的服务及其特性。在蓝牙协议栈的最上部是高端应用层,它对应于各种应用模型的剖面,是剖面的一部分,目前定义了 13 种剖面。

(1) 蓝牙低层模块

蓝牙的低层模块是蓝牙技术的核心,是任何蓝牙设备都必须包括的部分。蓝牙工作在 2.4GHz 的 ISM 频段。采用了蓝牙结束的设备将能够提供高达 720kbit/s 的数据交换速率。

蓝牙支持电路交换和分组交换两种技术,分别定义了两种链路类型,即面向连接的同步链路(SCO)和面向无连接的异步链路(ACL)。为了在很低的功率状态下也能使蓝牙设备处于连接状态,蓝牙规定了三种节能状态,即停等(Park)状态、保持(Hold)状态和呼吸(Sniff)状态。这几种工作模式按照节能效率以升序排依次是:Sniff 模式、Hold 模式和 Park 模式。

蓝牙采用 3 种纠错方案,分别是 1/3 前向纠错(FEC)、2/3 前向纠错和自动重发(ARQ)。前向纠错的目的是减少重发的可能性,但同时也增加了额外开销。然而在一个合理的无错误率环境中,多余的投标会减少输出,故分组定义的本身也保持灵活的方式,因此,在软件中可定义是否采用 FEC。一般而言,在信道的噪声干扰比较大时蓝牙系统会使用前向纠错方案,以保证通信质量:对于 SCO 链路,使用 1/3 前向纠错;对于 ACL 链路,使用 2/3 前向纠错。在无编号的自动请求重发方案中,一个时隙传送的数据必须在下一个时隙得到收到的确认。只有数据在收端通过了报头错误检测和循环冗余校验(CRC)后认为无错时,才向发端发回确认消息,否则返回一个错误消息。

蓝牙系统的移动性和开放性使得安全问题变得极其重要。虽然蓝牙系统所采用的调频技术就已经提供了一定的安全保障,但是蓝牙系统仍然需要链路层和应用层的安全管理。在链路层中,蓝牙系统提供了认证、加密和密钥管理等功能。每个用户都有一个人标识码(PIN),它会被译成 128bit 的链路密钥(Link Key)来进行单双向认证。一旦认证完毕,链路就会以不同长度的密码(Encryphon Key)来加密(此密码以 shift 为单位增减,最大的长度为 128bit)。链路层安全机制提供了大量的认证方案和一个灵活的加密方案(即允许协商密码的长度)。当来自不同国家的设备互相通信时,这种机制是极其重要的,因为某些国家会指定最大密码长度。蓝牙系统会选取微微网中各个设备的最小的最大允许密码长度。例如,美国允许 128bit 的密码长度,而西班牙仅允许 48bit,这样当两国的设备互通时,将选择 48bit 来加密。蓝牙系统也支持高层协议栈的不同应用体内的特殊的安全机制。例如,两台计算机在进行商业卡信息交流时,一台计算机就只能访问另一台计算机的该项业务,而无权访问其他业务。蓝牙安全机制依赖 PIN 在设备间建立信任关系,一旦这种关系建立起来了,这些 PIN 就可以存储在设备中以便将来更快捷地连接。

(2) 软件模块

L2CAP 是数据链路层的一部分,位于基带协议之上。L2CAP 向上层提供面向连接的和无连接的数据服务,它的功能包括:协议的复用能力、分组的分割和重新组装(Segmentation And Reassembly),以及提取(Group Abstraction)。L2CAP 允许高层协议和应用发送和接受高达 64K Byte 的数据分组。

SDP 为应用提供了一个发现可用协议和决定这些可用协议的特性的方法。蓝牙环境下的服务发现与传统的网络环境下的服务发现有很大的不同,在蓝牙环境下,移动的 RF 环境变

化很大，因此业务的参数也是不断变换的。SDP 将强调蓝牙环境的独特的特性。蓝牙使用基于客户/服务器机制定义了根据蓝牙服务类型和属性发现服务的方法，还提供了服务浏览的方法。

RFCOMM 是射频通信协议，它可以仿真串行电缆接口协议，符合 ETSI0710 串口仿真协议。通过 RFCOMM，蓝牙可以在无线环境下实现对高层协议，如 PPP、TCP/IP、WAP 等的支持。另外，RFCOMM 可以支持 AT 命令集，从而可以实现移动电话机和传真机及调制解调器之间的无线连接。

蓝牙对语音的支持是它与 WLAN 相区别的一个重要的标志。蓝牙电话控制规范是一个基于 ITU-T 建议 Q.931 的采用面向比特的协议，它定义了用于蓝牙设备间建立语音和呼叫信令数据及用于处理蓝牙 TCS 设备的移动性管理过程。

10.3.3 低功耗（BLE）蓝牙协议

BLE 不再支持传统蓝牙 BR/EDR 的协议，例如，传统蓝牙中的 SPP 协议在 BLE 中就不复存在。在 BLE 应用中，所有的协议或者服务都是基于 GATT（Generic Attribute Profile）的。尽管有些传统蓝牙中的协议，如 HID 被移植到了 BLE 中，但是在 BLE 的应用中必须区分协议和服务。其中服务描述了特点（及它们的 UUID），服务描述自身有什么特点和形式，并且描述清楚如何应用这些特点及需要什么安全机制。而应用协议定义了其使用的服务，说明是传感器端还是接收端，定义 GATT 的角色（Server/Client）和 GAP 的角色（Peripheral/Central）。

和蓝牙 BR/EDR 协议相比，因为所有的功能都是集成在 GATT 终端，这些基于其上的应用协议只是对 GATT 提供的功能的使用，所以基于 GATT 的应用协议非常简单。

10.3.4 现有的基于 GATT 的协议/服务

截至 2013 年 7 月，现有的基于 GATT 的协议/服务如表 10-2 所示。

表 10-2 基于 GATT 的协议/服务

GATT-Based Specifications (Qualifiable)		Adopted Version
ANP	Alert Notification Profile	1.0
ANS	Alert Notification Service	1.0
BAS	Battery Service	1.0
BLP	Blood Pressure Profile	1.0
BLS	Blood Pressure Service	1.0
CPP	Cycling Power Profile	1.0
CPS	Cycling Power Service	1.0
CSCP	Cycling Speed and Cadence Profile	1.0
CSCS	Cycling Speed and Cadence Service	1.0
CTS	Current Time Service	1.0
DIS	Device Information Service	1.1
FMP	Find Me Profile	1.0
GLP	Glucose Profile	1.0
HIDS	HID Service	1.0

续表

GATT-Based Specifications (Qualifiable)		Adopted Version
HOGP	HID over GATT Profile	1.0
HTP	Health Thermometer Profile	1.0
HTS	Health Thermometer Service	1.0
HRP	Heart Rate Profile	1.0
HRS	Heart Rate Service	1.0
IAS	Immediate Alert Service	1.0
LLS	Link Loss Service	1.0
LNP	Location and Navigation Profile	1.0
LNS	Location and Navigation Service	1.0
NDCS	Next DST Change Service	1.0
PASP	Phone Alert Status Profile	1.0
PASS	Phone Alert Status Service	1.0
PXP	Proximity Profile	1.0
RSCP	Running Speed and Cadence Profile	1.0
RSCS	Running Speed and Cadence Service	1.0
RTUS	Reference Time Update Service	1.0
ScPP	Scan Parameters Profile	1.0
ScPS	Scan Parameters Service	1.0
TIP	Time Profile	1.0
TPS	Tx Power Service	1.0

10.3.5 双模协议栈

图 10-3 所示为斯图曼双模协议栈 BlueCode+SR 的具体架构,在此架构图中包含 SPP、HDP 和 GATT 所需要的所有部分。

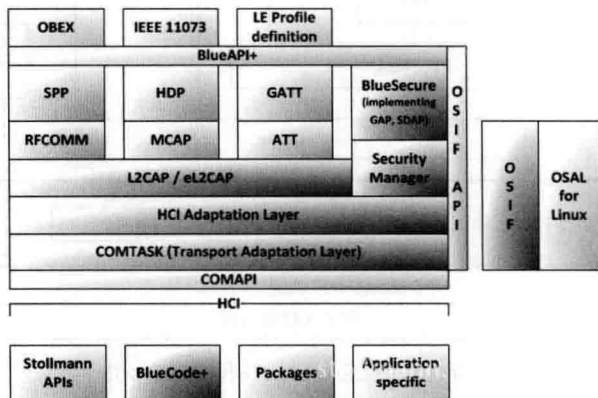


图 10-3 斯图曼双模协议栈 BlueCode+SR 的具体架构

10.3.6 单模协议栈

图 10-4 所示为单模协议栈的一种典型协议栈设计。

在单模协议栈中一般不会包含具体协议，所以需要在具体的应用程序中实现每一个具体应用对应的协议。这和传统蓝牙有非常大的区别，传统蓝牙会在协议栈中实现每个具体应用相关的协议，如 SPP、HDP 等。和双模协议栈相比，BLE 无须一个主处理器来实现它的协议栈，所以极低功耗的集成成为可能。大多数的单模芯片或者模块都自带协议栈。

因为 BLE 单模产品（芯片或者模块）中的协议栈只是实现了 GATT 层，所以通常需要将具体应用对应的协议集成到该单模产品之中。甚至芯片商都开始提供带有具体协议和 sample code 的 SDK。但是，仍然没有真正能拿到手的解决方案。

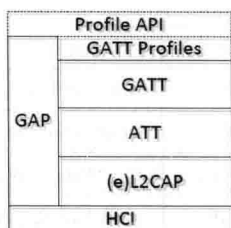


图 10-4 单模协议栈的一种典型协议栈设计

10.4 低功耗蓝牙协议栈详解

提到协议栈时会使人想到开放式系统互联（OSI）协议栈，OSI 协议栈定义了厂商如何才能生产可以与其他厂商的产品一起工作的产品。协议栈是指一组协议的集合，例如把大象装到冰箱里需要 3 步，每步就是一个协议，3 步组成一个协议栈。在本节的内容中，将详细讲解低功耗蓝牙协议栈的基本知识，为读者学习本书后面的知识打下基础。

10.4.1 低功耗蓝牙协议栈基础

蓝牙协议栈就是 SIG（Special Interest Group）定义的一组协议的规范，所有遵循这一规范的蓝牙应用之间可以相互操作。图 10-5 所示为完整蓝牙协议栈和部分 Profile。

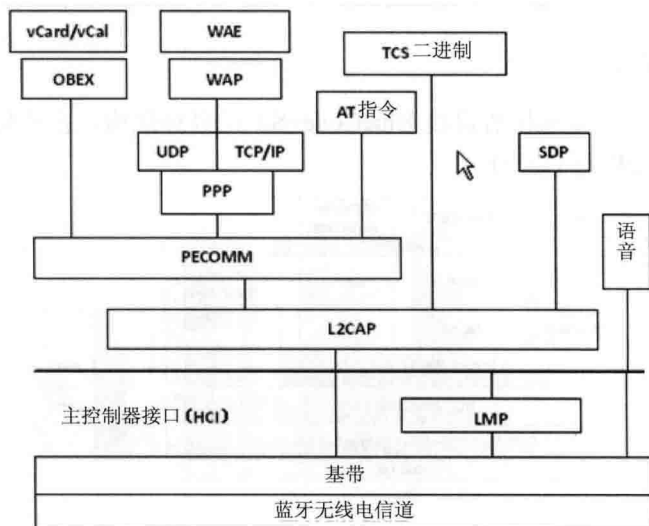


图 10-5 完整蓝牙协议栈和部分 Profile

在蓝牙系统中，Profile 是配置文件，配置文件定义了可能的应用，蓝牙配置文件表达了一般行为，蓝牙设备可以通过这些行为与其他设备进行通信。在蓝牙系统中定义了广泛的配

置文件,描述了许多不同类型的使用案例。按照蓝牙规格中提供的指导,开发商可以创建应用程序以与其他符合蓝牙规格的设备协同工作。到目前为止,在蓝牙系统中一共有 20 多个 Profile,在 www.bluetooth.com 中有各个 Profile 的详细说明文档。在这些众多的协议栈中,r 协议栈的具体构成如下。

- **Widcomm:** 第一个 Windows 中的协议栈,由 Widcomm 公司开发,也就是现在的 Broadcom。
- **Microsoft Windows stack:** Windows XP SP2 中包括这个内建的协议栈,开发者也可以调用其 API 开发第三方软件。
- **Toshiba stack:** 这也是基于 Windows 的,不支持第三方开发,但它把协议栈授权给一些 laptop 商,如 Sony。它支持的 Profile 有 SPP、DUN、FAX、LAP、OPP、FTP、HID、HCRP、PAN、BIP、HSP、HFP、A2DP、AVRCP 和 GAVDP。
- **BlueSoleil:** 著名的 IVT 公司的产品,该产品可以用于桌面和嵌入式,也支持第三方开发,如 DUN、FAX、HFP、HSP、LAP、OBEX、OPP、PAN SPP、AV、BIP、FTP、GAP、HID、SDAP 和 SYNC。
- **Blues:** Linux 官方协议栈,该协议栈的上层用 Socket 封装,便于开发者使用,通过 DBUS 与其他应用程序通信。
- **Affix:** NOKIA 公司的协议栈,在 Symbian 系统中运行。
- **BlueDragon:** 东软公司产品,在 2002 年 6 月就通过了蓝牙的认证,支持的 Profile 有 SDP、Serial-DevB、AVCTP、AVRCP-Controller、AVRCP-Target、Headset-AG、Headset-HS、OPP-Client、OPP-Server、CT-GW、CT-Term、Intercom、FT-Server、FT-Client、GAP、SDAP、Serial-DevA、AVDTP、GAVDP、A2DP-Source 和 A2DP-Sink。
- **BlueMagic:** 美国 Open Interface 公司 for portable embedded divce 的协议栈,iphone(apple),nav-u(sony)等很多电子产品都用该商业的协议栈,BlueMagic 3.0 是第一个通过 bluetooth 协议栈 1.1 认证的协议栈。
- **BCHS-Bluecore Host Software:** 蓝牙芯片 CSR 的协议栈,同时也提供了一些上层应用的 Profile 的库,也是为嵌入式产品提供的服务,支持的 Profile 有 A2DP、AVRCP、PBAP、BIP、BPP、CTP、DUN、FAX、FM API、FTP GAP、GAVDP、GOEP、HCRP、Headset、HF1.5、HID、ICP、JSR82、LAP Message Access Profile、OPP、PAN、SAP、SDAP、SPP、SYNC 和 SYNC ML。
- **Windows CE:** 微软给 Windows CE 开发的协议栈,但是 Windows CE 本身也支持其他的协议栈。
- **BlueLet:** IVT 公司 for embedded product 的轻量级协议栈。

10.4.2 蓝牙协议体系中的协议

在蓝牙协议体系中的协议中,按 SIG 的关注程度分为如下 4 层。

- **核心协议:** BaseBand、LMP、L2CAP 和 SDP。
- **电缆替代协议:** RFCOMM。
- **电话传送控制协议:** TCS-Binary、AT 命令集。
- **选用协议:** PPP、UDP/TCP/IP、OBEX、WAP、vCard、vCal、IrMC 和 WAE。

除上述协议层外,规范还定义了主机控制器接口(HCI),它为基带控制器、连接管理器、

硬件状态和控制寄存器提供命令接口。在图 10-5 中, HCI 位于 L2CAP 的下层, 但 HCI 也可位于 L2CAP 上层。

蓝牙核心协议由 SIG 制定的蓝牙专用协议组成, 绝大部分蓝牙设备都需要核心协议 (加上无线部分), 而其他协议则根据应用的需要而定。总之, 电缆替代协议、电话控制协议和被采用的协议在核心协议基础上构成了面向应用的协议。

在现实应用中, 常用蓝牙核心协议类型如下。

(1) 基带协议

在蓝牙系统中, 基带和链路控制层能够确保微网内各蓝牙设备单元之间是由射频构成的物理连接。蓝牙的射频系统是一个跳频系统, 其任一分组在指定时隙、指定频率上发送。它使用查询和分页进程同步不同设备间的发送频率和时钟, 为基带数据分组提供了两种物理连接方式, 即面向连接 (SCO) 和无连接 (ACL), 而且, 在同一射频上可实现多路数据传送。ACL 适用于数据分组, SCO 适用于语音以及语音与数据的组合, 所有的语音和数据分组都附有不同级别的前向纠错 (FEC) 或循环冗余校验 (CRC), 而且可进行加密。此外, 对于不同数据类型 (包括连接管理信息和控制信息) 都分配一个特殊通道。

可使用各种用户模式在蓝牙设备间传送语音, 面向连接的语音分组只需经过基带传输, 而不到达 L2CAP。语音模式在蓝牙系统内相对简单, 只需开通语音连接就可传送语音。

(2) 连接管理协议 (LMP)

该协议负责各蓝牙设备间连接的建立。它通过连接的发起、交换、核实, 进行身份认证和加密, 通过协商确定基带数据分组大小。它还控制无线设备的电源模式和工作周期, 以及微微网内设备单元的连接状态。

(3) 逻辑链路控制和适配协议 (L2CAP)

该协议是基带的上层协议, 可以认为它与 LMP 并行工作, 它们的区别在于, 当业务数据不经过 LMP 时, L2CAP 为上层提供服务。L2CAP 向上层提供面向连接的和无连接的数据服务, 它采用了多路技术、分割和重组技术、群提取技术。L2CAP 允许高层协议以 64KB 长度收发数据分组。虽然基带协议提供了 SCO 和 ACL 两种连接类型, 但 L2CAP 只支持 ACL。

(4) 服务发现协议 (SDP)

发现服务在蓝牙技术框架中起着至关重要的作用, 它是所有用户模式的基础。使用 SDP 可以查询到设备信息和服务类型, 从而在蓝牙设备间建立相应的连接。

(5) 电缆替代协议 (RFCOMM)

RFCOMM 是基于 ETSI-07.10 规范的串行线仿真协议。它在蓝牙基带协议上仿真 RS-232 控制和数据信号, 为使用串行线传送机制的上层协议 (如 OBEX) 提供服务。

(6) 电话控制协议

- 二元电话控制协议 (TCS-Binary 或 TCSBIN): 面向比特的协议, 它定义了蓝牙设备间建立语音和数据呼叫的控制信令, 定义了处理蓝牙 TCS 设备群的移动管理进程。基于 ITU TQ.931 建议的 TCSBinary 被指定为蓝牙的二元电话控制协议规范。
- AT 命令集电话控制协议: SIG 定义了控制多用户模式下移动电话和调制解调器的 AT 命令集, 该 AT 命令集基于 ITU TV.250 建议和 GSM07.07 规范, 它还可以用于传真业务。

(7) 选用协议

- 点对点协议 (PPP): 在蓝牙技术中, PPP 位于 RFCOMM 上层, 完成点对点的连接。

- **TCP/UDP/IP**：该协议是由互联网工程任务组制定，广泛应用于互联网通信的协议。在蓝牙设备中，使用这些协议是为了与互联网相连接的设备进行通信。
- **对象交换协议（OBEX）**：IrOBEX（简称为 OBEX）是由红外数据协会（IrDA）制定的会话层协议，它采用简单的和自发的方式交换目标。OBEX 是一种类似于 HTTP 的协议，它假设传输层是可靠的，采用客户机/服务器模式，独立于传输机制和传输应用程序接口（API）。

例如，电子名片交换格式（vCard）、电子日历及日程交换格式（vCal）都是开放性规范，它们都没有定义传输机制，而只是定义了数据传输格式。SIG 采用 vCard/vCal 规范，是为了进一步促进个人信息交换。

- **无线应用协议（WAP）**：该协议是由无线应用协议论坛制定的，它融合了各种广域无线网络技术，其目的是将互联网内容和电话传送的业务传送到数字蜂窝电话和其他无线终端上。

10.5 TI 公司的低功耗蓝牙

BLE 低功耗蓝牙协议有很多版本，不同的厂商提供的低功耗蓝牙协议有所区别。在本节的内容中，将详细讲解 TI（德州仪器）公司提供的 BLE 低功耗蓝牙协议的基本知识，为读者学习本书后面的知识打下基础。

10.5.1 获取 TI 公司的低功耗蓝牙协议栈

TI（德州仪器）公司提供了多个版本的 BLE 低功耗蓝牙协议栈，读者可以登录其官方网站来下载，如图 10-6 所示。

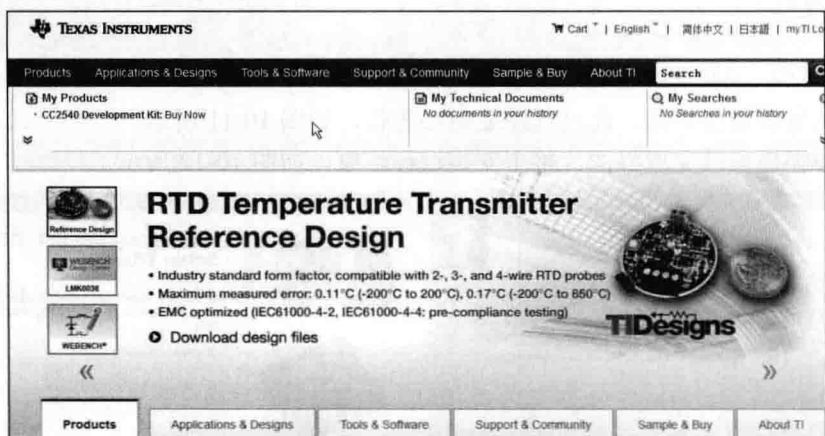


图 10-6 TI 公司的官方网站

笔者下载的版本是 BLE-CC254x-1.3.exe，双击此文件后可以进行安装工作，具体安装过程如下。

- （1）首先弹出“解压缩”界面，在此单击 Next 按钮，如图 10-7 所示。
- （2）弹出同意安装协议界面，在此选择 I accept...选项，单击 Next 按钮，如图 10-8 所示。



图 10-7 “解压缩”界面

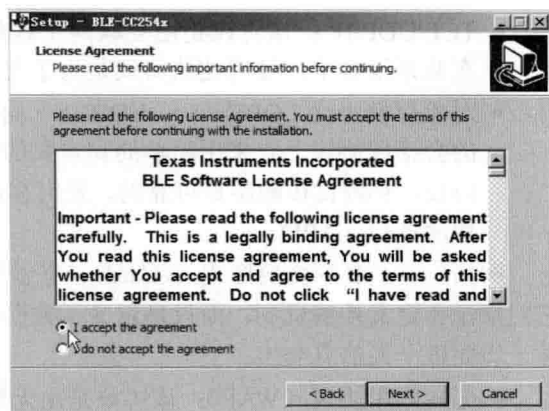


图 10-8 同意安装协议界面

(3) 进入选择安装路径界面，单击 Browse...按钮可以选择安装路径，如图 10-9 所示。

(4) 进入准备安装界面，单击 Install 按钮开始安装，如图 10-10 所示。

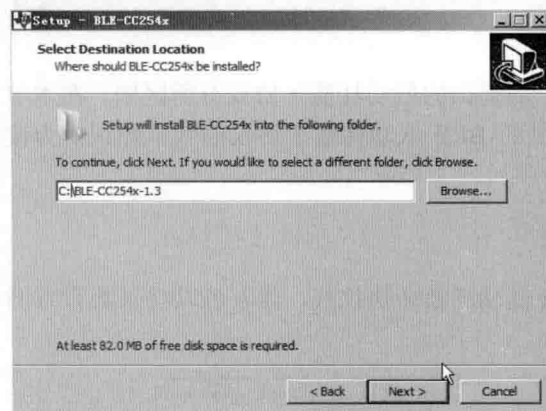


图 10-9 选择安装路径界面

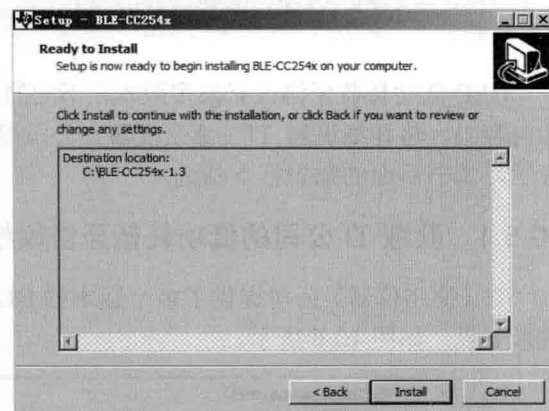


图 10-10 准备安装界面

(5) 弹出安装进度界面，此过程需要耐心等待，如图 10-11 所示。

(6) 最后弹出安装完成界面，整个安装过程结束，如图 10-12 所示。

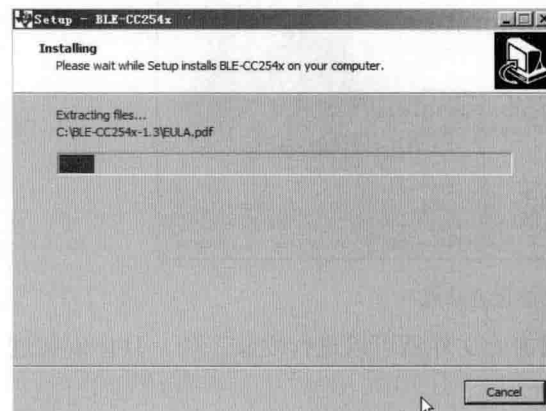


图 10-11 安装进度界面



图 10-12 安装完成界面

安装完成后, 需要使用 IAR 集成开发环境打开工程文件。例如, TI 公司在 Projects\ble\SimpleBLEPeripheral\目录下提供了实例工程, 通过使用 IAR 工具打开.eww 文件的方式可以浏览整个工程。

注意

读者可以自行下载并安装 IAR 集成开发环境。

10.5.2 分析 TI 公司的低功耗蓝牙协议栈

注意

下面的内容参考自 TI 公司的官方资料 *CC2540Bluetooth Low Energy Software Developer's Guide (Rev. B)*, 部分图片也是直接引用自上述参考文档。

1. BLE 蓝牙协议栈结构

BLE 蓝牙协议栈分为两个部分, 分别是控制器和主机。对于 4.0 以前的蓝牙, 这两部分是分开的。所有 profile (姑且称为剧本吧, 用来定义设备或组件的角色) 和应用都建构在 GAP 或 GATT 之上。BLE 蓝牙协议栈的结构如图 10-13 所示。

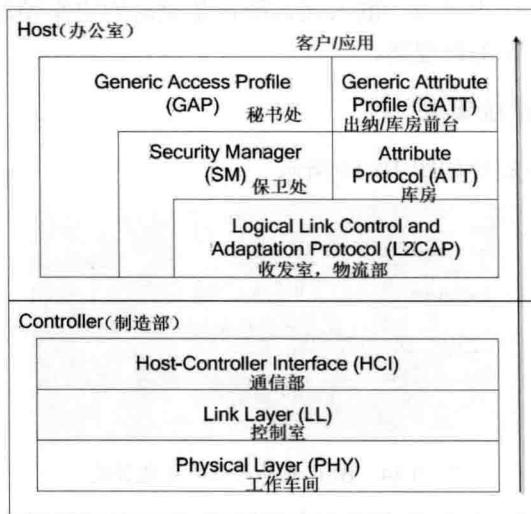


图 10-13 BLE 蓝牙协议栈的结构

在 BLE 蓝牙协议栈的结构中, 从上到下的具体说明如下。

- PHY 层, 工作车间: 1Mbps 自适应跳频 GFSK (高斯频移键控), 运行在免许可证使用的 2.4GHz 频段中。
- LL 层: 为 RF 控制器、控制室、控制设备处于准备 (standby)、广播、监听/扫描 (scan)、初始化、连接, 这 5 种状态中一种。5 种状态切换描述为: 未连接时, 设备广播信息

(向周围邻居讲“我来了”), 另外一个设备一直监听或按需扫描, 两个设备连接初始化(搬几把椅子到院子), 设备连接上了(开聊)。发起聊天的设备为主设备, 接受聊天的设备为从设备, 同一次聊天只能有一个意见领袖, 即主设备和从设备不能切换。

- **HCI 层**: 为接口层、通信部, 向上为主机提供软件应用程序接口 (API), 对外为外部硬件控制接口, 可以通过串口、SPI、USB 来实现设备控制。
- **L2CAP 层**: 物流部, 负责行李打包盒拆封处, 提供数据封装服务。
- **SM 层**: 保卫处, 提供配对和密钥分发, 实现安全连接和数据交换。
- **ATT 层**: 库房, 负责数据检索。
- **GATT 层**: 出纳/库房前台, 出纳负责处理向上与应用打交道, 而库房前台负责向下把检索任务子进程交给 ATT 库房去做, 其关键工作是为检索工作提供合适的 profile 结构, 而 profile 由检索关键词 (characteristics) 组成。
- **GAP 层**: 秘书处, 对上级提供应用程序接口, 对下级管理各级职能部门, 尤其是指示 LL 层控制室 5 种状态切换, 指导保卫处做好机要工作。

蓝牙为了实现同多个设备相连或实现多功能的目标, 也实现了功能扩充, 这就产生了调度问题。因为, 虽然软件和协议栈可扩充, 但终究最底层的执行部门只有一个。为了实现多事件和多任务切换, 需要把事件和任务对应的应用, 以及其相关的提供支撑的“办公室”和“工厂”打包起来, 并命名为 **OSAL 操作系统抽象层**, 类似于集团公司以下的子公司。

如果实现软件和硬件的低耦合, 使软件不经改动或很少改动即可应用在另外的硬件上, 这样就方便硬件改造、升级、迁移后, 软件的移植。**HAL 硬件抽象层**正是用来抽象各种硬件的资源, 告知给软件。其作用类似于嵌入式系统设备驱动的定义硬件资源的“.h”头文件, 其角色类似于现代工厂的设备管理部。

2. BLE 低功耗蓝牙系统架构

BLE 低功耗蓝牙系统架构如图 10-14 所示。

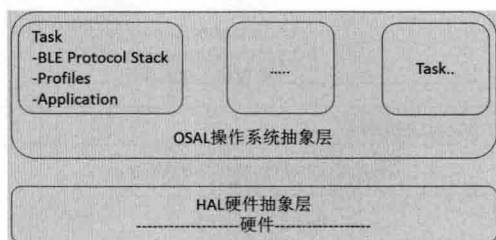


图 10-14 BLE 低功耗蓝牙系统架构

由此而知, BLE 低功耗蓝牙软件有两个主要组成, 分别是 **OSAL 操作系统抽象层**和 **HAL 硬件抽象层**, 多个 Task 任务和事件在 OSAL 管理下工作, 而每个任务和事件又包括 3 个组成部分, 分别是 BLE 协议栈、profiles 和应用程序。

(1) OSAL 操作系统抽象层

OSAL 作为调度核心, BLE 协议栈、profile 定义、所有的应用都围绕它来实现。OSAL 不是传统使用的操作系统, 而是一个允许软件建立和执行事件的循环。软件功能是由任务事件来实现的, 创建的任务事件需要完成如下工作。

- 创建 task identifier 任务 ID。
- 编写任务初始化 (task initialization routine) 进程, 并需要添加到 OSAL 初始化进程中, 这就是说系统启动后不能动态添加功能。
- 编写任务处理程序。
- 提供消息服务。

BLE 协议栈的各层都是以 OSAL 任务方式实现的, 由于 LL 控制室的时间要求最为迫切, 所以其任务优先级最高。为了实现任务管理, OSAL 通过消息处理 (messageprocess)、存储管理、计时器定时等附加服务实现。

(2) 系统启动流程

为了使用 OSAL, 在 main 函数的最后要启动一个名称为 osal_start_system 的进程, 该进程会调用由特定应用决定的启动函数 osalInitTasks (来启动系统)。osalInitTasks 逐个调用 BLE 协议栈各层的启动进程来初始化协议栈。随后, 设置一个任务的 8bit 任务 ID (task ID), 跳入循环等待执行任务, 系统启动完成。

(3) 任务事件与事件处理

• 进程优先级和任务 ID

- 任务优先级决定于任务 ID, 任务 ID 越小, 优先级越高。
- BLE 协议栈各层的任务优先级比应用程序的高。
- 初始化协议栈后, 越早调入的任务, 任务 ID 越高, 优先级越低, 即系统倾向于处理新到的任务。

• 事件变量和旗语

每个事件任务由对应的 16bit 事件变量来标示, 事件状态由旗号 (taskflag) 来标示。如果事件处理程序已经完成, 但其旗号并没有移除, OSAL 会认为事情还没有完成而继续在该程序中不返回。比如, 在 SimpleBLEPeripheral 实例工程中, 当事件 START_DEVICE_EVT 发生, 其处理函数 SimpleBLEPeripheral_ProcessEvent 就运行, 结束后返回 16bit 事件变量, 并清除旗语 SBP_START_DEVICE_EVT。

• 事件处理表单

每当 OSAL 事件检测到了有任务事件, 其相应的处理进程将被添加到由处理进程指针构成的事件处理表单中, 该表单名称为 taskArr (taskarray)。taskArr 中各个事件进程的顺序和 osalInitTasks 初始化函数中任务 ID 的顺序是对应的。

• 事件调度的方法

有两种事件调度的方法, 最简单的方法是使用 osal_set_event 函数 (函数原型在 OSAL.h 文件中), 在这个函数中, 用户可以像定义函数参数一样设置任务 ID 和事件旗语。第二种方法是使用 osal_start_timerEx 函数 (函数原型在 OSAL_Timers.h 文件中), 使用方法同 osal_set_event 函数, 而第三个以毫秒为单位的参数 osal_start_timerEx 则指示该事件处理必须在这个限定时间内, 通过定时器来为事件处理计时。

(4) 存储管理

存储管理类似于 Linux 嵌入式系统内存分配 C 函数 mem_alloc, OSAL 利用 osal_mem_alloc 提供基本的存储管理, 但 osal_mem_alloc 只有一个用于定义 byte 数的参数。对应的内存释放函数为 osal_mem_free。

(5) 进程间通信—通过消息机制实现

不同的子系统通过 OSAL 的消息机制通信。消息即为数据，数据种类和长度都不限定。消息收发过程描述如下。

在接收信息时调用函数 `osal_msg_allocate` 创建消息占用内存空间（已经包含了 `osal_mem_alloc` 函数功能），需要为该函数指定空间大小，该函数返回内存空间地址指针，利用该指针就可把所需数据复制到该空间。

在发送数据时调用函数 `osal_msg_send`，需为该函数指定发送目标任务，OSAL 通过旗语 `SYS_EVENT_MSG` 告知目标任务，目标任务的处理函数调用 `osal_msg_receive` 来接收发来的数据。建议每个 OSAL 任务都有一个消息处理函数，每当任务收到一个消息后，通过消息的种类来确定需要本任务做相应处理。消息接收并处理完成，调用函数 `osal_msg_deallocate` 来释放内存（已经包含了 `osal_mem_free` 函数功能）。

3. 硬件抽象层 HAL

当新的硬件平台做好后，只需修改 HAL，而无须修改 HAL 之上的协议栈的其他组件和应用程序。

4. BLE 低功耗蓝牙协议栈

(1) BLE 库文件

TI 蓝牙协议栈是以单独一个库文件提供的，并没有提供源代码，因此不作深入说明。对于 TI 的 BLE 实例应用来说，这个单独库文件完全够用，因为已经列出了所有的库文件。

(2) GAP 秘书处

● 角色（即服务/功能）

在 TI 实例中，GAP 运行在如下 4 种角色的一种。

- **Broadcaster:** 广播员—“我在，但只可远观，不可连接”。
- **Observer:** 观察员—“看看谁在，但我只远观，不连接”。
- **Peripheral:** 外设（从机）—“我在，谁要我就跟谁走”，协议栈单层连接。
- **Central:** 核心（主机）—“看看谁在，并且愿意跟我走我就带她/他走”，协议栈单层或多层连接，目前最多支持 3 个同时连接。

虽然指标显示 BLE 可以同时扮演多个角色，但是在 TI 提供的 BLE 实例应用中，只演示了 BLE 扮演支持外设角色的情形。每一种角色都由一个剧本（roleprofile）来定义。

● 连接

在主从机连接过程中，一个典型的低功耗蓝牙系统同时包含外设和核心（主机），两者的连接过程是：外设角色向外发送自己的信息（设备地址、名字等），主机收到外设广播信息后，发送扫描请求（scanrequest）给外设，外设响应主机的请求，连接建立完成。

连接参数主要有通信间隙（connectioninterval）、外设鄙视（slavelatency）、最大耐心等待时间（supervisiontimeout）等，具体说明如下。

- **通信间隙**——蓝牙通信是间断的、跳频的，每次连接都可能选择不同的子频带。跳频的好处是避免频道拥塞，间断连接的好处是节省功耗，通信间隙就是指两次连接之间的时间间隔。这个间隔以 1.25ms 为基本单位，最小为 6 单位，最大为 3 200 单

位, 间隙越小通信越及时, 间隙越大功耗越低。

- 外设鄙视——外设与主机建立连接以后, 没事的时候主机总会定期发送问候信息到外设, 外设懒得搭理, 这些主机发送的信息就浮云般飘过。可以忽略的连接事件个数从 0 到 499 个, 最多不超过 32 秒。有效连接间隙= 连接间隙× (1+ 外设鄙视)。
- 最大耐心等待时间——指的是为了创建一个连接, 主机允许的最大等候时间, 在这个时间内, 不停地尝试连接。范围是 10~3 200 个通信间隙基本单位 (1.25ms)。

以上参数大小设置优劣是显而易见的, 连接参数的设置可阅读本小节后面的内容。

假如主机采用从机并不舒坦的参数来请求连接, 有如主从机已经连接了, 但从机有想法了, 要改参数条约。通过“连接参数更新请求 (ConnectionParameter Update Request)”来解决问题, 交由 L2CAP “收发室物流处”处理。

在实现加密处理时可以利用配对实现, 利用密匙来加密授权连接。典型的过程是: 外设向主机请求口令一个 (passkey) 以便进行配对, 待主机发送了正确的口令之后, 连接通信通过主从机互换密码来校验。由于蓝牙通信是间断通信, 如果一个应用需要经常通信, 而每次通信都要重新申请连接, 那将是劳神费力的, 为此 GAP 安全卫士 (SM, security profile) 提供了一种长期签证 (long-termset of keys), 叫作绑定 (bonding), 这样每次建立连接通关流程就简便并快捷。

(3) 出纳 GATT

GATT 负责两个设备间通信的数据交互。共有两种角色: 出纳员 (GATTClient) 和银行 (GATTServer), 银行提供资金, 出纳从银行存、取款。银行可以同时面对多个出纳员。这两种角色和主从机等角色是无关的。

GATT 把工作拆分成几部分来实现: 读关键词 (CharacteristicValue) 和描述符 (CharacteristicDescriptor), 用来去库房查找提取数据, 并写/读关键词和描述符。

GATT 银行 (GATTServer) 的业务部门 (API) 提供两个主要的功能: 一是服务功能, 注册或销毁服务 (serviceattribute), 并作为回调函数 (callbackfunction); 二是管理功能, 添加或删除 GATT 银行业务。

一个角色定义的剧本可以同时定义多个角色, 每个角色的服务、关键词、关键值、描述符 (service, characteristic, characteristic value and descriptors) 都以句柄 (attributes) 形式保存在角色提供的服务上。所有的服务都是一个 gattAttribute_t 类型的 array, 在文件 gatt.h 中定义。

(4) 调用 GAP 和 GATT 的一般过程

调用 GAP 和 GATT 的一般过程如下。

- API 调用。
- 协议栈响应并返回。
- 协议栈发送一个 OSAL 消息 (数据) 去调用相应任务事件。
- 调用任务去接收和处理消息。
- 消息清除。

以设备初始化为 GAP 外设角色来举例说明, 外设角色由其剧本 (GAP peripheral role profile) 来决定, 实例程序在文件 peripheral.c 内。

- 调用 API 函数 GAP_DeviceInit。
- GAP 检查了一下说, 好, 可以初始化, 返回值为 SUCCESS (0x00), 并通知 BLE 工作。

- BLE 协议栈发送 OSAL 消息给外设角色剧本 (peripheral roleprofile), 消息内容包括要干什么 (eventvalue) GAP_MSG_EVENT 和指标是什么 (opcodevalue, 参数)。
- 角色剧本的服务任务就收到了事件请求 SYS_EVENT_MSG, 表示有消息来了。
- 角色剧本接收消息, 并拆看到底是什么事情, 接着把消息数据转换 (cast) 成具体要做的事情, 并完成相应的工作 (这里为 gapDeviceInitDoneEvent_t)。
- 角色剧本清除消息并返回。例如, GATT 客户端设备想从 GATT 服务器端读取数据, 即 GATT 出纳想从 GATT 银行那边取点钱出来。
- 应用程序调用 GATT 子进程 API 函数 GATT_ReadCharValue, 传递的参数为连接句柄、关键词句柄和自身任务的 ID。
- GATT 答应了这个请求, 返回值为 SUCCESS (0x00), 向下告知 BLE 有活干了。
- BLE 协议栈在下次建立蓝牙连接时, 发送取钱的指令给银行, 当银行说: “好, 我们正好有柜员没事”, 于是把钱取出来交给了 BLE。
- BLE 接着就把取到的钱包成消息 (OSAL message), 通过出纳 GATT 返回给了应用程序。消息内包含 GATT_MSG_EVENT 和修改了的 ATT_READ_RSP。
- 应用程序接收到了从 OSAL 来的 SYS_EVENT_MSG 事件, 表示钱可能到了。
- 应用程序接收消息, 拆包检查, 并拿走需要的钱。
- 最后应用程序把包装袋销毁。

(5) GAP 角色剧本 Profiles

在 TI 的 BLE 实例应用中提供了 3 种 GAP 角色剧本, 分别是保卫处角色和几种 GATT 出纳/库管示例程序服务角色。

- GAP 外设剧本。

其 API 函数在 peripheral.h 中定义, 包括如下信息。

- GAPROLE_ADVERT_ENABLED——广播使能。
- GAPROLE_ADVERT_DATA——包含在广播里的信息。
- GAPROLE_SCAN_RSP_DATA——外设用于回复主机扫描请求的信息。
- GAPROLE_ADVERT_OFF_TIME——表示外设关闭广播持续时间, 该值为零表示无限期关闭广播直到下一次广播使能信号到来。
- GAPROLE_PARAM_UPDATE_ENABLE——使能自动更新连接参数, 可以让外设连接失败时自动调整连接参数以便重新连接。
- GAPROLE_MIN_CONN_INTERVAL——设置最小连接间隙, 缺省值为 80 个单位 (每单位 1.25ms)。
- GAPROLE_MIN_CONN_INTERVAL——设置最大连接间隙, 缺省值为 3 200 个单位。
- GAPROLE_SLAVE_LATENCY——外设鄙视参数, 缺省为零。
- GAPROLE_TIMEOUT_MULTIPLIER——最大耐心等待时间, 缺省为 1 000 个单位。

函数 GAPRole_StartDevice 用来初始化 GAP 外设角色, 其唯一的参数是 gapRolesCBs_t, 这个参数是一个包含两个函数指针的结构体, 这两个函数是 pfnStateChange 和 pfnRssiRead, 前者标示状态, 后者标示 RSSI 已经被读走了。

- 多角色同时扮演。

在此以设备同时为外设和广播员两种角色, 方法是去除前文外设的定义剧本 peripheral.c

和 `peripheral.h`，添加新的脚本 `peripheralBroadcaster.c` 和 `peripheralBroadcaster.h`；定义处理器值（`preprocessorvalue`）`PLUS_BROADCASTER`。

- GAP 主机剧本。

与外设剧本相似，主机剧本的 API 函数在 `central.h` 中定义，包括 `GAPCentralRole_GetParameter` 和 `GAPCentralRole_SetParameter` 以及其他。如 `GAPROLE_PARAM_UPDATE_ENABLE` 连接参数自动更新使能的功能，跟外设角色的一样。

`GAPCentralRole_StartDevice` 函数用来初始化 GAP 主机角色，其唯一的参数是 `gapCentralRolesCBs_t`，这个参数是一个包含两个函数指针的结构体，这两个函数是 `eventCB` 和 `rsiCB`，每次 GAP 时间发生，前者都会被调用，后者标示 RSSI 已经被读走。

- GAP 绑定管理器剧本。

GAP 绑定管理器剧本用于保持长期的连接。同时支持外设置置和主机配置。当建立了配对连接后，如果绑定使能，绑定管理器就维护这个连接。主要参数如下：

- `GAPBOND_PAIRING_MODE`
- `GAPBOND_MITM_PROTECTION`
- `GAPBOND_IO_CAPABILITIES`
- `GAPBOND_IO_CAP_DISPLAY_ONLY`
- `GAPBOND_BONDING_ENABLED`

函数 `GAPBondMgr_Register` 用来初始化 GAP 主机角色，其唯一的参数是 `gapBondCBs_t`，这个参数是一个包含两个函数指针的结构体，这两个函数是 `pairStateCB` 和 `passcodeCB`，前者返回状态，后者用于配对时产生 6 为数字口令（`passcode`）。

- 编写一个剧本来创建（定义）新的角色（功能、服务）。

以 `SimpleGATT Profile` 为剧本名称，包含两个文件 `simpleGATTProfile.c` 和 `simpleGATTProfile.h`。包含如下主要 API 函数。

- `SimpleProfile_AddService`——用于初始化的进程，作用是添加服务句柄（`serviceattributes`）到句柄组（`attributetable`）内，寄存器读取和回写。
- `SimpleProfile_SetParameter`——设置剧本（`profile`）关键词（`characteristics`）。
- `SimpleProfile_GetParameter`——获取设置剧本关键词。
- `SimpleProfile_RegisterAppCBs`。
- `simpleProfile_ReadAttrCB`。
- `simpleProfile_WriteAttrCB`
- `simpleProfile_HandleConnStatusCB`。

此实例剧本共有如下 5 个关键词：

- `SIMPLEPROFILE_CHAR1`
- `SIMPLEPROFILE_CHAR2`
- `SIMPLEPROFILE_CHAR3`
- `SIMPLEPROFILE_CHAR4`
- `SIMPLEPROFILE_CHAR5`

TI 公司的低功耗蓝牙协议栈的基本知识介绍完毕。有关此协议栈的具体知识，读者可登录其官方网站查看帮助文档。

Android 蓝牙模块 详解

蓝牙是一种支持设备短距离通信（一般 10m 内）的无线电技术，可以在移动电话、PDA、无线耳机、笔记本电脑、相关外设等众多设备之间进行无线信息交换。在本章的内容中，将首先讲解 Android 系统中蓝牙模块的底层源码和实现原理，为读者学习本书后面的知识打下基础。

11.1 Android 系统中的蓝牙模块

Android 系统包含了对蓝牙网络协议栈的支持，这使得蓝牙设备能够无线连接其他蓝牙设备交换数据。Android 的应用程序框架提供了访问蓝牙功能的 APIs。这些 APIs 让应用程序能够无线连接其他蓝牙设备，实现点对点，或点对多点的无线交互功能。

通过使用蓝牙 APIs，一个 Android 应用程序能够实现如下功能。

- 扫描其他蓝牙设备。
- 查询本地蓝牙适配器（local Bluetooth adapter）用于配对蓝牙设备。
- 建立 RFCOMM 信道（channels）。
- 通过服务发现（service discovery）连接其他设备。
- 数据通信。
- 管理多个连接。

Android 平台的蓝牙系统是基于 BlueZ 实现的，是通过 Linux 中一套完整的蓝牙协议栈开源实现的。当前 BlueZ 被广泛应用于各种 Linux 版本中，并被芯片公司移植到各种芯片平台上使用。在 Linux 2.6 内核中已经包含了完整的 BlueZ 协议栈，在 Android 系统中已经移植并嵌入进了 BlueZ 的用户空间实现，并且随着硬件技术的发展而不断更新。

蓝牙（Bluetooth）技术实际上是一种短距离无线电技术。在 Android 系统的蓝牙模块中，除了使用 Kernel 支持外，还需要用户空间的 BlueZ 的支持。

Android 平台中蓝牙模块的基本层次结构如图 11-1 所示。

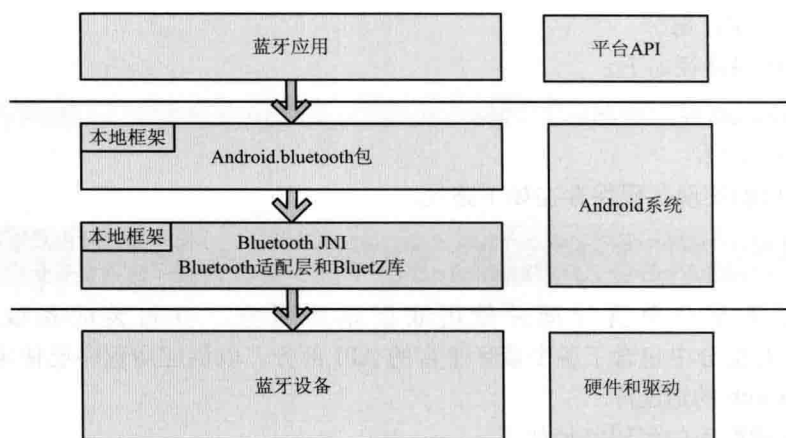


图 11-1 蓝牙模块的基本层次结构

Android 平台中蓝牙系统从上到下主要包括 Java 框架中的 Bluetooth 类、Android 适配库、BlueZ 库、驱动程序和协议，这几部分的系统结构如图 11-2 所示。

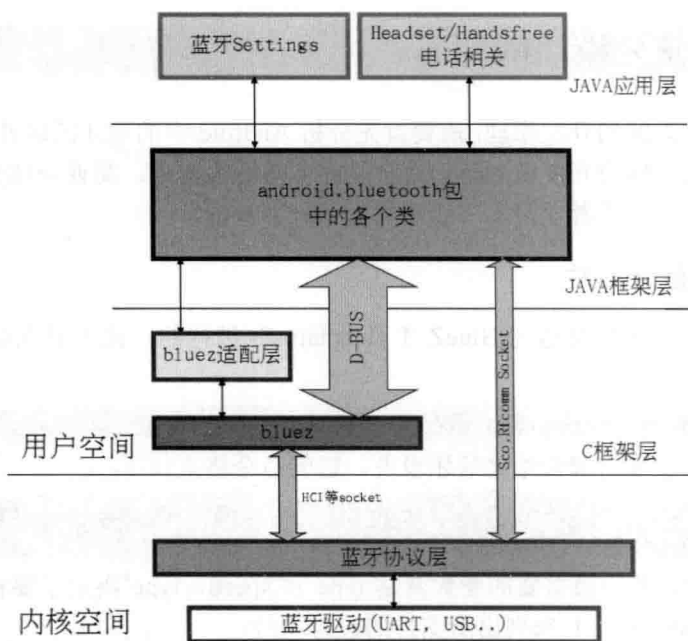


图 11-2 蓝牙系统结构

在图 11-2 中各个层次结构的具体说明如下。

(1) BlueZ 库

Android 蓝牙设备管理的库的路径如下。

```
external/bluez/
```

可以分别生成库 libbluetooth.so、libbluedroid.so 和 hcidump 等众多相关工具和库。BlueZ 库提供了对用户空间蓝牙的支持，在里面包含了主机控制协议 HCI 及其他众多内核实现协议的接口，并且实现了所有蓝牙应用模式 Profile。

(2) 蓝牙的 JNI 部分

此部分的代码路径如下。

```
frameworks/base/core/jni/
```

(3) Java 框架层

Java 框架层的实现代码保存在如下路径。

```
frameworks/base/core/java/android/bluetooth    // 蓝牙部分对应应用程序的 API
frameworks/base/core/java/android/Server       // 蓝牙的服务部分
```

蓝牙的服务部分负责管理并使用底层本地服务，并封装成系统服务。而在 android.bluetooth 部分中包含了各个蓝牙平台的 API 部分，以供应用程序层使用。

(4) Bluetooth 的适配库

Bluetooth 适配库的代码路径如下。

```
system/bluetooth/
```

在此层用于生成库 libbluedroid.so 及相关工具和库，能够实现对蓝牙设备的管理，如蓝牙设备的电源管理。

11.2 分析蓝牙模块的源码

要想掌握蓝牙系统的开发原理，需要首先分析 Android 中的蓝牙源码并了解其核心构造，只有这样才能对蓝牙应用开发做到游刃有余。在本节的内容中，简要介绍开源 Android 中蓝牙模块相关的代码，为读者学习本书后面的知识打下基础。

11.2.1 初始化蓝牙芯片

初始化蓝牙芯片工作是通过 BlueZ 工具 hciattach 进行的，此工具在如下目录的文件中实现。

```
external/bluetooth/tools
```

Hciattach 命令主要用来初始化蓝牙设备，它的命令格式如下。

```
hciattach [-n] [-p] [-b] [-t timeout] [-s initial_speed] <tty> <type | id>
[ speed ] [ flow|noflow ] [ bdaddr ]
```

在上述格式中，其中最重要的参数就是 type 和 speed，type 决定了要初始化的设备的型号，可以使用 hciattach -l 来列出所支持的设备型号。

并不是所有的参数对所有的设备都是适用的，有些设备会忽略一些参数设置，例如，查看 hciattach 的代码就可以看到，多数设备都忽略 bdaddr 参数。Hciattach 命令内部的工作步骤是：首先打开制定的 tty 设备，然后做一些通用的设置，如 flow 等，然后设置波特率为 initial_speed，然后根据 type 调用各自的初始化代码，最后将波特率重新设置为 speed。所以调用 hciattach 时，要根据实际情况，设置好 initial_speed 和 speed。

对于 type BCSP 来说，它的初始化代码只做了一件事，就是完成 BCSP 协议的同步操作，它并不对蓝牙芯片做任何的 pskey 的设置。

11.2.2 蓝牙服务

在蓝牙服务方面一般不要用户自己定义，只需使用初始化脚本文件 `init.rc` 中的默认内容即可。代码如下。

```
service bluetoothd /system/bin/logwrapper /system/bin/bluetoothd -d -n
    socket bluetooth stream 660 bluetooth bluetooth
    socket dbus_bluetooth stream 660 bluetooth bluetooth
    # init.rc does not yet support applying capabilities, so run as root and
    # let bluetoothd drop uid to bluetooth with the right linux capabilities
    group bluetooth net_bt_admin misc
    disabled

# baudrate change 115200 to 1152000 (Bluetooth)
service changebaudrate /system/bin/logwrapper /system/sbin/bccmd_115200 -t
bcsp -d /dev/s3c2410_serial11 psset -r 0x1be 0x126e
    user bluetooth
    group bluetooth net_bt_admin
    disabled
    oneshot

#service hciattach /system/bin/logwrapper /system/bin/hciattach -n -s 1152000
/dev/s3c2410_serial11 bcsp 1152000
service hciattach /system/bin/logwrapper /system/bin/hciattach -n -s 115200
/dev/s3c2410_serial11 bcsp 115200
    user bluetooth
    group bluetooth net_bt_admin misc
    disabled

service hfag /system/bin/sdptool add --channel=10 HFAG
    user bluetooth
    group bluetooth net_bt_admin
    disabled
    oneshot

service hsag /system/bin/sdptool add --channel=11 HSAG
    user bluetooth
    group bluetooth net_bt_admin
    disabled
    oneshot

service opush /system/bin/sdptool add --channel=12 OPUSH
    user bluetooth
    group bluetooth net_bt_admin
    disabled
    oneshot

service pbap /system/bin/sdptool add --channel=19 PBAP
    user bluetooth
    group bluetooth net_bt_admin
    disabled
    oneshot
```

在上述代码中，每一个 `service` 后面都列出了一种 Android 服务。

11.2.3 管理蓝牙电源

在 Android 系统的如下目录中实现了 libbluedroid。

```
system/bluetooth/
```

可以调用 `rftkill` 接口来控制电源管理，如果已经实现了 `rftkill` 接口，则无须再进行配置。如果在文件 `init.rc` 中已经实现了 `hciattach` 服务，则说明在 `libbluedroid` 中已经实现了对其调用以操作蓝牙的初始化。

11.3 和蓝牙相关的类

读者经过本章前面内容的学习，已经了解了 Android 系统中蓝牙的基本知识；根据对上述从底层到应用的学习，了解了蓝牙的工作原理和机制。在本节的内容中，将详细讲解在 Android 系统中和蓝牙相关的类。

11.3.1 BluetoothSocket 类

1. BluetoothSocket 类基础

类 `BluetoothSocket` 的定义格式如下。

```
public static class Gallery.LayoutParams extends ViewGroup.LayoutParams
```

类 `BluetoothSocket` 的定义结构如下。

```
java.lang.Object
android.view.ViewGroup.LayoutParams
android.widget.Gallery.LayoutParams
```

Android 的蓝牙系统和 `Socket` 套接字密切相关，蓝牙端的监听接口和 TCP 的端口类似，都是使用了 `Socket` 和 `ServerSocket` 类。在服务器端，使用 `BluetoothServerSocket` 类来创建一个监听服务端口。当一个连接被 `BluetoothServerSocket` 所接受，它会返回一个新的 `BluetoothSocket` 来管理该连接。在客户端，使用一个单独的 `BluetoothSocket` 类去初始化并管理一个外接连接。

最通常使用的蓝牙端口是 RFCOMM，它被 Android API 支持。RFCOMM 是一个面向连接，通过蓝牙模块进行的数据流传输方式，它也被称为串行端口规范（Serial Port Profile, SPP）。

为了创建一个 `BluetoothSocket` 去连接到一个已知设备，使用方法 `BluetoothDevice.createRfcommSocketToServiceRecord()`。然后调用 `connect()` 方法去尝试一个面向远程设备的连接。这个调用将被阻塞，指导一个连接已经建立或者该链接失效。

为了创建一个 `BluetoothSocket` 作为服务端（或者“主机”），每当该端口连接成功后，无论它初始化为客户端，或者被接受作为服务器端，都通过方法 `getInputStream()` 和 `getOutputStream()` 来打开 IO 流，从而获得各自的 `InputStream` 和 `OutputStream` 对象。

`BluetoothSocket` 类的线程是安全的，因为 `close()` 方法总会马上放弃外界操作并关闭服务器端口。

2. BluetoothSocket 类的公共方法

(1) public void close ()

功能：马上关闭该端口并且释放所有相关的资源。在其他线程的该端口中引起阻塞，从而使系统马上抛出一个 IO 异常。

异常：IOException。

(2) public void connect ()

功能：尝试连接到远程设备。该方法将阻塞，指导一个连接建立或者失效。如果该方法没有返回异常值，则该端口现在已经建立。当设备查找正在进行的时候，创建对远程蓝牙设备的新连接不可被尝试。设备查找在蓝牙适配器上是一个重量级过程，并且肯定会降低一个设备的连接。使用 cancelDiscovery()方法会取消一个外界的查询，因为这个查询并不由活动所管理，而是作为一个系统服务来运行，所以即使它不能直接请求一个查询，应用程序也总会调用 cancelDiscovery()方法。使用方法 close()可以用来放弃从另一线程而来的调用。

异常：IOException，表示一个错误，如连接失败。

(3) public InputStream getInputStream ()

功能：通过连接的端口获得输入数据流。即使该端口未连接，该输入数据流也会返回。不过在该数据流上的操作将抛出异常，直到相关的连接已经建立。

返回值：输入流。

异常：IOException。

(4) public OutputStream getOutputStream ()

功能：通过连接的端口获得输出数据流。即使该端口未连接，该输出数据流也会返回。不过在该数据流上的操作将抛出异常，直到相关的连接已经建立。

返回值：输出流。

异常：IOException。

(5) public BluetoothDevice getRemoteDevice ()

功能：获得该端口正在连接或者已经连接的远程设备。

返回值：远程设备。

11.3.2 BluetoothServerSocket 类

1. BluetoothServerSocket 类基础

类 BluetoothServerSocket 的格式如下。

```
public final class BluetoothServerSocket extends Object implements Closeable
```

类 BluetoothServerSocket 的结构如下。

```
java.lang.Object
android.bluetooth.BluetoothServerSocket
```

2. BluetoothServerSocket 类的公共方法

(1) public BluetoothSocket accept (int timeout)

功能：阻塞直到超时时间内的连接建立。在一个成功建立的连接上返回一个已连接的 BluetoothSocket 类。每当该调用返回时，它可以在此调用去接收以后新来的连接。close() 方法可以用来放弃从另一线程来的调用。

参数 timeout：表示阻塞超时时间。

返回值：已连接的 BluetoothSocket。

异常：IOException，表示出现错误，比如该调用被放弃或超时。

(2) public BluetoothSocket accept ()

功能：阻塞直到一个连接已经建立。在一个成功建立的连接上返回一个已连接的 BluetoothSocket 类。每当该调用返回时，它可以在此调用去接收以后新来的连接。close() 方法可以用来放弃从另一线程来的调用。

返回值：已连接的 BluetoothSocket。

异常：IOException，表示出现错误，比如该调用被放弃或者超时。

(3) public void close ()

功能：马上关闭端口，并释放所有相关的资源。在其他线程的该端口中引起阻塞，从而使系统马上抛出一个 IO 异常。关闭 BluetoothServerSocket 不会关闭接受自 accept() 的任意 BluetoothSocket。

异常：IOException。

11.3.3 BluetoothAdapter 类

1. BluetoothAdapter 类基础

类 BluetoothAdapter 的格式如下。

```
public final class BluetoothAdapter extends Object
```

类 BluetoothAdapter 的结构如下。

```
java.lang.Object  
android.bluetooth.BluetoothAdapter
```

BluetoothAdapter 代表本地的蓝牙适配器设备，通过此类可以让用户能执行基本的蓝牙任务，如初始化设备的搜索，查询可匹配的设备集，使用一个已知的 MAC 地址来初始化一个 BluetoothDevice 类，创建一个 BluetoothServerSocket 类以监听其他设备对本机的连接请求等。

为了得到这个代表本地蓝牙适配器的 BluetoothAdapter 类，需要调用静态方法 getDefaultAdapter()，这是所有蓝牙动作使用的第一步。当拥有本地适配器以后，用户可以获得一系列的 BluetoothDevice 对象，这些对象代表所有拥有 getBondedDevice() 方法的已经匹配的设备；用 startDiscovery() 方法来开始设备的搜寻；或者创建一个 BluetoothServerSocket 类，通过 listenUsingRfcommWithServiceRecord(String, UUID) 方法来监听新来的连接请求。

**注意**

大部分方法需要 BLUETOOTH 权限，一些方法同时需要 BLUETOOTH_ADMIN 权限。

2. BluetoothAdapter 类的常量

(1) String ACTION_DISCOVERY_FINISHED

广播事件：本地蓝牙适配器已经完成设备的搜寻过程。需要 BLUETOOTH 权限接收。

常量值：android.bluetooth.adapter.action.DISCOVERY_FINISHED。

(2) String ACTION_DISCOVERY_STARTED

广播事件：本地蓝牙适配器已经开始对远程设备进行搜寻。它通常涉及一个大概需时 12 秒的查询扫描过程，紧跟着是一个对每个获取到自身蓝牙名称的新设备的页面扫描。用户会发现一个把 ACTION_FOUND 常量通知为远程蓝牙设备的注册。设备查找是一个重量级的过程。当查找正在进行时，用户不能尝试对新的远程蓝牙设备进行连接，同时存在的连接将获得有限的带宽及高等待时间。用户可用 cancelDiscovery() 类来取消正在执行的查找进程。需要 BLUETOOTH 权限接收。

常量值：android.bluetooth.adapter.action.DISCOVERY_STARTED。

(3) String ACTION_LOCAL_NAME_CHANGED

广播活动：本地蓝牙适配器已经更改了它的蓝牙名称。该名称对远程蓝牙设备是可见的，它总是包含了一个带有名称的 EXTRA_LOCAL_NAME 附加域。需要 BLUETOOTH 权限接收。

常量值：android.bluetooth.adapter.action.LOCAL_NAME_CHANGED。

(4) String ACTION_REQUEST_DISCOVERABLE

Activity 活动：显示一个请求被搜寻模式的系统活动。如果蓝牙模块当前未打开，该活动也将请求用户打开蓝牙模块。被搜寻模式和 SCAN_MODE_CONNECTABLE_DISCOVERABLE 等价。当远程设备执行查找进程时，它允许其发现该蓝牙适配器。从隐私安全考虑，Android 不会将被搜寻模式设置为默认状态。该意图的发送者可以选择性地运用 EXTRA_DISCOVERABLE_DURATION 这个附加域去请求发现设备的持续时间。普遍来说，对于每一个请求，默认的持续时间为 120 秒，最大值则可达到 300 秒。

Android 运用 onActivityResult(int, int, Intent) 回收方法来传递该活动结果的通知。被搜寻的时间（以秒为单位）将通过 resultCode 值来显示，如果用户拒绝被搜寻，或者设备产生了错误，则通过 RESULT_CANCELED 值来显示。

每当扫描模式变化时，应用程序可以通过 ACTION_SCAN_MODE_CHANGED 值来监听全局的消息通知。比如，当设备停止被搜寻以后，该消息可以被系统通知给应用程序。需要 BLUETOOTH 权限。

常量值：android.bluetooth.adapter.action.REQUEST_DISCOVERABLE。

(5) String ACTION_REQUEST_ENABLE

Activity 活动：显示一个允许用户打开蓝牙模块的系统活动。当蓝牙模块完成打开工作，或者当用户决定不打开蓝牙模块时，系统活动将返回该值。Android 运用 onActivityResult(int,

int, Intent)回收方法来传递该活动结果的通知。如果蓝牙模块被打开,将通过 resultCode 值 RESULT_OK 来显示;如果用户拒绝该请求,或者设备产生了错误,则通过 RESULT_CANCELED 值来显示。每当蓝牙模块被打开或者关闭,应用程序可以通过 ACTION_STATE_CHANGED 值来监听全局的消息通知。需要 BLUETOOTH 权限。

常量值: android.bluetooth.adapter.action.REQUEST_ENABLE。

(6) String ACTION_SCAN_MODE_CHANGED

广播活动: 指明蓝牙扫描模块或者本地适配器已经发生变化。它总是包含 EXTRA_SCAN_MODE 和 EXTRA_PREVIOUS_SCAN_MODE。这两个附加域各自包含了新的和旧的扫描模式。需要 BLUETOOTH 权限。

常量值: android.bluetooth.adapter.action.SCAN_MODE_CHANGED。

(7) String ACTION_STATE_CHANGED

广播活动: 本来的蓝牙适配器的状态已经改变,例如,蓝牙模块已经被打开或者关闭。它总是包含 EXTRA_STATE 和 EXTRA_PREVIOUS_STATE。这两个附加域各自包含了新的和旧的状态。需要 BLUETOOTH 权限接收。

常量值: android.bluetooth.adapter.action.STATE_CHANGED。

(8) int ERROR

功能: 标记该类的错误值。确保和该类中的任意其他整数常量不相等。它为需要一个标记错误值的函数提供了便利。例如:

```
Intent.getIntExtra(BluetoothAdapter.EXTRA_STATE, BluetoothAdapter.ERROR)
```

常量值: -2147483648 (0x80000000)。

(9) String EXTRA_DISCOVERABLE_DURATION

功能: 试图在 ACTION_REQUEST_DISCOVERABLE 常量中作为一个可选的整型附加域,来为短时间内的设备发现请求一个特定的持续时间。默认值为 120 秒,超过 300 秒的请求将被限制。这些值是可以变化的。

常量值: android.bluetooth.adapter.extra.DISCOVERABLE_DURATION。

(10) String EXTRA_LOCAL_NAME

功能: 试图在 ACTION_LOCAL_NAME_CHANGED 常量中作为一个字符串附加域,来请求本地蓝牙的名称。

常量值: android.bluetooth.adapter.extra.LOCAL_NAME。

(11) String EXTRA_PREVIOUS_SCAN_MODE

功能: 试图在 ACTION_SCAN_MODE_CHANGED 常量中作为一个整型附加域,来请求以前的扫描模式。可以取得如下值。

- SCAN_MODE_NONE
- SCAN_MODE_CONNECTABLE
- SCAN_MODE_CONNECTABLE_DISCOVERABLE

常量值: android.bluetooth.adapter.extra.PREVIOUS_SCAN_MODE。

(12) String EXTRA_PREVIOUS_STATE

功能: 试图在 ACTION_STATE_CHANGED 常量中作为一个整型附加域,来请求以前的

供电状态。可以取得如下值。

- STATE_OFF
- STATE_TURNING_ON
- STATE_ON
- STATE_TURNING_OFF

常量值: `android.bluetooth.adapter.extra.PREVIOUS_STATE`。

(13) String EXTRA_SCAN_MODE

功能: 试图在 ACTION_SCAN_MODE_CHANGED 常量中作为一个整型附加域, 来请求当前的扫描模式, 可以取得如下值。

- SCAN_MODE_NONE
- SCAN_MODE_CONNECTABLE
- SCAN_MODE_CONNECTABLE_DISCOVERABLE

常量值: `android.bluetooth.adapter.extra.SCAN_MODE`。

(14) String EXTRA_STATE

功能: 试图在 ACTION_STATE_CHANGED 常量中作为一个整型附加域, 来请求当前的供电状态。可以取得如下值。

- STATE_OFF
- STATE_TURNING_ON
- STATE_ON
- STATE_TURNING_OFF

常量值: `android.bluetooth.adapter.extra.STATE`。

(15) int SCAN_MODE_CONNECTABLE

功能: 指明在本地蓝牙适配器中, 查询扫描功能失效, 但页面扫描功能有效。因此该设备不能被远程蓝牙设备发现, 但如果以前曾经发现过该设备, 则远程设备可以对其进行连接。

常量值: 21 (0x00000015)。

(16) int SCAN_MODE_CONNECTABLE_DISCOVERABLE

功能: 指明在本地蓝牙适配器中, 查询扫描功能和页面扫描功能都有效。因此该设备既可以被远程蓝牙设备发现, 也可以被其连接。

常量值: 23 (0x00000017)。

(17) int SCAN_MODE_NONE

功能: 指明在本地蓝牙适配器中, 查询扫描功能和页面扫描功能都失效。因此该设备既不可以被远程蓝牙设备发现, 也不可以被其连接。

常量值: 20 (0x00000014)。

(18) int STATE_OFF

功能: 指明本地蓝牙适配器模块已经关闭。

常量值: 10 (0x0000000a)。

(19) int STATE_ON

功能: 指明本地蓝牙适配器模块已经打开, 并且准备被使用。

(20) int STATE_TURNING_OFF

功能：指明本地蓝牙适配器模块正在关闭。本地客户端可以立刻尝试友好地断开任意外部连接。

常量值：13 (0x0000000d)。

(21) int STATE_TURNING_ON

功能：指明本地蓝牙适配器模块正在打开。然而本地客户在尝试使用这个适配器之前需要为 STATE_ON 状态而等待。

常量值：11 (0x0000000b)。

3. BluetoothAdapter 类的公共方法

(1) public boolean cancelDiscovery ()

功能：取消当前的设备发现查找进程，需要 BLUETOOTH_ADMIN 权限。因为对蓝牙适配器而言，查找是一个重量级的过程，因此这个方法必须在尝试连接到远程设备前使用，用 connect() 方法进行调用。发现的过程不会由活动来进行管理，但是它会作为一个系统服务来运行，因此即使它不能直接请求这样的一个查询动作，也必须取消该搜索进程。如果蓝牙状态不是 STATE_ON，这个 API 将返回 false。蓝牙打开后，等待 ACTION_STATE_CHANGED 更新成 STATE_ON。

返回值：成功则返回 true，有错误则返回 false。

(2) public static boolean checkBluetoothAddress (String address)

功能：验证诸如"00:43:A8:23:10:F0"之类的蓝牙地址，字母必须为大写才有效。

参数 address：字符串形式的蓝牙模块地址。

返回值：地址正确则返回 true，否则返回 false。

(3) public boolean disable ()

功能：关闭本地蓝牙适配器——不能在明确关闭蓝牙的用户动作中使用。这个方法友好地停止所有的蓝牙连接，停止蓝牙系统服务，以及对所有基础蓝牙硬件进行断电。没有用户的直接同意，蓝牙永远不能被禁止。这个 disable() 方法只提供了一个应用，该应用包含了一个改变系统设置的用户界面（如“电源控制”应用）。

这是一个异步调用方法：该方法将马上获得返回值，用户要通过监听 ACTION_STATE_CHANGED 值来获取随后的适配器状态改变的通知。如果该调用返回 true 值，则该适配器状态会立刻从 STATE_ON 转向 STATE_TURNING_OFF，稍后则会转为 STATE_OFF 或者 STATE_ON。如果该调用返回 false，那么系统已经有一个保护蓝牙适配器被关闭的问题——比如该适配器已经被关闭。

需要 BLUETOOTH_ADMIN 权限。

返回值：如果蓝牙适配器的停止进程已经开启则返回 true，如果产生错误则返回 false。

(4) public boolean enable ()

功能：打开本地蓝牙适配器——不能在明确打开蓝牙的用户动作中使用。该方法将为基础的蓝牙硬件供电，并且启动所有的蓝牙系统服务。没有用户的直接同意，蓝牙永远不能被禁止。如果用户为了创建无线连接而打开了蓝牙模块，则其需要 ACTION_REQUEST_ENABLE 值，该值将提出一个请求用户允许打开蓝牙模块的会话。这个 enable() 值只提供了一个应用，

该应用包含了一个改变系统设置的用户界面（如“电源控制”应用）。

这是一个异步调用方法：该方法将马上获得返回值，用户要通过监听 `ACTION_STATE_CHANGED` 值来获取随后的适配器状态改变的通知。如果该调用返回 `true` 值，则该适配器状态会立刻从 `STATE_OFF` 转向 `STATE_TURNING_ON`，稍后则会转为 `STATE_OFF` 或者 `STATE_ON`。如果该调用返回 `false`，那么说明系统已经有一个保护蓝牙适配器被打开的问题——比如飞行模式，或者该适配器已经被打开。

需要 `BLUETOOTH_ADMIN` 权限。

返回值：如果蓝牙适配器的打开进程已经开启则返回 `true`，如果产生错误则返回 `false`。

(5) `public String getAddress ()`

功能：返回本地蓝牙适配器的硬件地址，格式如下。

```
00:11:22:AA:BB:CC
```

需要 `BLUETOOTH` 权限。

返回值：字符串形式的蓝牙模块地址。

(6) `public Set<BluetoothDevice> getBondedDevices ()`

功能：返回已经匹配到本地适配器的 `BluetoothDevice` 类的对象集合。如果蓝牙状态不是 `STATE_ON`，这个 API 将返回 `false`。蓝牙打开后，等待 `ACTION_STATE_CHANGED` 更新成 `STATE_ON`。需要 `BLUETOOTH` 权限。

返回值：未被修改的 `BluetoothDevice` 类的对象集合，如果有错误则返回 `null`。

(7) `public static synchronized BluetoothAdapter getDefaultAdapter ()`

功能：获取对默认本地蓝牙适配器的操作权限。目前 `Andoird` 只支持一个蓝牙适配器，但是 API 可以被扩展为支持多个适配器。该方法总是返回默认的适配器。

返回值：返回默认的本地适配器，如果蓝牙适配器在该硬件平台上不能被支持，则返回 `null`。

(8) `public String getName ()`

功能：获取本地蓝牙适配器的蓝牙名称，这个名称对于外界蓝牙设备而言是可见的。需要 `BLUETOOTH` 权限。

返回值：该蓝牙适配器名称，如果有错误则返回 `null`。

(9) `public BluetoothDevice getRemoteDevice (String address)`

功能：为给予的蓝牙硬件地址获取一个 `BluetoothDevice` 对象。合法的蓝牙硬件地址必须为大写，格式类似于“00:11:22:33:AA:BB”。`checkBluetoothAddress(String)`方法可以用来验证蓝牙地址的正确性。`BluetoothDevice` 类对于合法的硬件地址总会产生返回值，即使这个适配器从未见过该设备。

参数：`address` 合法的蓝牙 MAC 地址。

异常：`IllegalArgumentException`，如果地址不合法。

(10) `public int getScanMode ()`

功能：获取本地蓝牙适配器的当前蓝牙扫描模式，蓝牙扫描模式决定本地适配器可连接或可被远程蓝牙设备所连接。需要 `BLUETOOTH` 权限，可能的取值如下。

- `SCAN_MODE_NONE`

- SCAN_MODE_CONNECTABLE
- SCAN_MODE_CONNECTABLE_DISCOVERABLE

如果蓝牙状态不是 STATE_ON，则该 API 将返回 false。蓝牙打开后，等待 ACTION_STATE_CHANGED 更新成 STATE_ON。

返回值：扫描模式。

(11) public int getState ()

功能：获取本地蓝牙适配器的当前状态，需要 BLUETOOTH 类。可能的取值如下。

- STATE_OFF
- STATE_TURNING_ON
- STATE_ON
- STATE_TURNING_OFF

返回值：蓝牙适配器的当前状态。

(12) public boolean isDiscovering ()

功能：如果当前蓝牙适配器正处于设备发现查找进程中，则返回真值。设备查找是一个重量级过程。当查找正在进行时，用户不能尝试对新的远程蓝牙设备进行连接，同时存在的连接将获得有限制的带宽及高等待时间。用户可用 cancelDiscovery()类来取消正在执行的查找进程。

应用程序也可以为 ACTION_DISCOVERY_STARTED 或者 ACTION_DISCOVERY_FINISHED 进行注册，从而当查找开始或者完成时，可以获得通知。

如果蓝牙状态不是 STATE_ON，该 API 将返回 false。蓝牙打开后，等待 ACTION_STATE_CHANGED 更新成 STATE_ON。需要 BLUETOOTH 权限。

返回值：如果正在查找，则返回 true。

(13) public boolean isEnabled ()

功能：如果蓝牙正处于打开状态并可用，则返回真值，与 getBluetoothState()==STATE_ON 等价，需要 BLUETOOTH 权限。

返回值：如果本地适配器已经打开，则返回 true。

(14) public BluetoothServerSocket listenUsingRfcommWithServiceRecord (String name, UUID uuid)

功能：创建一个正在监听的、安全的、带有服务记录的无线射频通信 (RFCOMM) 蓝牙端口。一个对该端口进行连接的远程设备将被认证，对该端口的通信将被加密。使用 accept() 方法可以获取从监听 BluetoothServerSocket 处新来的连接。该系统分配一个未被使用的无线射频通信通道来进行监听。

该系统也将注册一个服务探索协议 (SDP) 记录，该记录带有一个包含了特定的通用唯一识别码 (Universally Unique Identifier, UUID)，服务器名称和自动分配通道的本地 SDP 服务。远程蓝牙设备可以用相同的 UUID 来查询自己的 SDP 服务器，并搜寻连接到了哪个通道上。如果该端口已经关闭，或者如果该应用程序异常退出，则该 SDP 记录会被移除。使用 createRfcommSocketToServiceRecord(UUID)可以从另一使用相同 UUID 的设备来连接到这个端口。需要 BLUETOOTH 权限。

参数如下。

- **name**: SDP 记录下的服务器名。
- **uuid**: SDP 记录下的 UUID。

返回值: 一个正在监听的无线射频通信蓝牙服务端口。

异常: `IOException`, 表示产生错误, 比如蓝牙设备不可用, 或者许可无效, 或者通道被占用。

(15) `public boolean setName (String name)`

功能: 设置蓝牙或者本地蓝牙适配器的昵称, 这个名字对于外界蓝牙设备而言是可见的。合法的蓝牙名称最多拥有 248 位 UTF-8 字符, 但是很多外界设备只能显示前 40 个字符, 有些可能只限制显示前 20 个字符。

如果蓝牙状态不是 `STATE_ON`, 该 API 将返回 `false`。蓝牙打开后, 等待 `ACTION_STATE_CHANGED` 更新成 `STATE_ON`。需要 `BLUETOOTH_ADMIN` 权限。

参数 **name**: 一个合法的蓝牙名称。

返回值: 如果该名称已被设定, 则返回 `true`, 否则返回 `false`。

(16) `public boolean startDiscovery ()`

功能: 开始对远程设备进行查找的进程, 它通常涉及一个大概需时 12 秒的查询扫描过程, 紧接着是一个对每个获取到自身蓝牙名称的新设备的页面扫描。这是一个异步调用方法: 该方法将马上获得返回值, 注册 `ACTION_DISCOVERY_STARTED` and `ACTION_DISCOVERY_FINISHED` 意图准确地确定该探索是处于开始阶段还是完成阶段。注册 `ACTION_FOUND` 以活动远程蓝牙设备已找到的通知。

设备查找是一个重量级过程。当查找正在进行时, 用户不能尝试对新的远程蓝牙设备进行连接, 同时存在的连接将获得有限的带宽及高等待时间。用户可用 `cancelDiscovery()` 类来取消正在执行的查找进程。发现的过程不会由活动来进行管理, 但是它会作为一个系统服务来运行, 因此即使它不能直接请求这样的一个查询动作, 也必须取消该搜索进程。设备搜寻只寻找已经被连接的远程设备。许多蓝牙设备默认不会被搜寻到, 并且需要进入到一个特殊的模式当中。

如果蓝牙状态不是 `STATE_ON`, 该 API 将返回 `false`。蓝牙打开后, 等待 `ACTION_STATE_CHANGED` 更新成 `STATE_ON`。需要 `BLUETOOTH_ADMIN` 权限。

返回值: 成功返回 `true`, 错误返回 `false`。

11.3.4 BluetoothClass.Service 类

类 `BluetoothClass.Service` 的格式如下。

```
public static final class BluetoothClass.Service extends Object
```

类 `BluetoothClass.Service` 的结构如下。

```
java.lang.Object
android.bluetooth.BluetoothClass.Service
```

类 `BluetoothClass.Service` 用于定义所有的服务类常量, 任意 `BluetoothClass` 由 0 或多个服务类编码组成。在类 `BluetoothClass.Service` 中包含如下常量。

- `int AUDIO`

- int CAPTURE
- int INFORMATION
- int LIMITED_DISCOVERABILITY
- int NETWORKING
- int OBJECT_TRANSFER
- int POSITIONING
- int RENDER
- int TELEPHONY

11.3.5 BluetoothClass.Device 类

类 BluetoothClass.Device 的格式如下。

```
public final class BluetoothClass.Device extends Object
```

类 BluetoothClass.Device 的结构如下。

```
java.lang.Object
android.bluetooth.BluetoothClass.Device
```

类 BluetoothClass.Device 用于定义所有的设备类的常量，每个 BluetoothClass 有一个带有主要和较小部分的设备类进行编码。里面的常量代表主要和较小的设备类部分（完整的设备类）的组合。BluetoothClass.Device.Major 的常量只能代表主要设备类，各个常量如下。

BluetoothClass.Device 有一个内部类，此内部类定义了所有的主要设备类常量。内部类的定义格式如下。

```
class BluetoothClass.Device.Major
```



注意

至此，Android 中的蓝牙类介绍完毕。在调用这些类时除了首先确保 API Level 至少为版本 5 以上，并且还需注意添加相应的权限，比如使用通信需要在文件 androidmanifest.xml 中加入 `<uses-permission android:name="android.permission.BLUETOOTH">` 权限，而在开关蓝牙时需要加入 `android.permission.BLUETOOTH_ADMIN` 权限。

11.4 在 Android 平台开发蓝牙应用程序

读者经过前面的学习，了解了在 Android 系统中和蓝牙模块相关的 API 类。其实从查找蓝牙设备到能够相互通信需要经过几个基本步骤，这几个步骤缺一不可。在本节的内容中，将详细讲解在 Android 平台中开发蓝牙应用程序的基本方法。

11.4.1 开发 Android 蓝牙应用程序的基本步骤

在 Android 系统中，开发蓝牙应用程序的基本步骤如下。

(1) 设置权限

在文件 `AndroidManifest.xml` 中声明使用蓝牙的权限，代码如下。

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

(2) 启动蓝牙

首先要查看本机是否支持蓝牙，然后获取 `BluetoothAdapter` 蓝牙适配器对象。代码如下。

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if(mBluetoothAdapter == null){
    //表明此手机不支持蓝牙
    return;
}
if(!mBluetoothAdapter.isEnabled()){ //蓝牙未开启，则开启蓝牙
    Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
}
//.....
public void onActivityResult(int requestCode, int resultCode, Intent data){
    if(requestCode == REQUEST_ENABLE_BT){
        if(resultCode == RESULT_OK){
            //蓝牙已经开启
        }
    }
}
```

(3) 发现蓝牙设备

- 首先使本机蓝牙处于可见状态（即处于易被搜索到的状态），便于其他设备发现本机蓝牙。演示代码如下。

```
//使本机蓝牙在 300 秒内可被搜索
private void ensureDiscoverable() {
    if (mBluetoothAdapter.getScanMode() !=
        BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
        Intent discoverableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
        discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
        startActivity(discoverableIntent);
    }
}
```

- 然后查找已经配对的蓝牙设备，即以前已经配对过的设备。演示代码如下。

```
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
if (pairedDevices.size() > 0) {
    findViewById(R.id.title_paired_devices).setVisibility(View.VISIBLE);
    for (BluetoothDevice device : pairedDevices) {
        //device.getName() +" "+ device.getAddress();
    }
} else {
```

```

        mPairedDevicesArrayAdapter.add("没有找到已匹配的设备");
    }

```

- 最后通过 `mBluetoothAdapter.startDiscovery()` 方法来搜索设备，在此需要注册一个 `BroadcastReceiver` 来获得这个搜索结果。即先注册再获取信息，然后进行处理。演示代码如下。

```

//注册，当一个设备被发现时调用 onReceive
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
this.registerReceiver(mReceiver, filter);
//当搜索结束后调用 onReceive
filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
this.registerReceiver(mReceiver, filter);
//.....
private BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if(BluetoothDevice.ACTION_FOUND.equals(action)){
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.
                EXTRA_DEVICE);
            // 已经配对的则跳过
            if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
                mNewDevicesArrayAdapter.add(device.getName() + "\n" +
                    device.getAddress()); //保存设备地址与名字
            }
        }else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {
            //搜索结束
            if (mNewDevicesArrayAdapter.getCount() == 0) {
                mNewDevicesArrayAdapter.add("没有搜索到设备");
            }
        }
    }
};

```

(4) 建立连接

当查找到蓝牙设备后，接下来需要建立本机与其他设备之间的连接。一般在使用本机搜索其他蓝牙设备时，本机可以作为一个服务端来接收其他设备的连接。启动一个服务器端的线程，死循环等待客户端的连接，这与 `ServerSocket` 极为相似，此线程在准备连接之前启动。演示代码如下。

```

//UUID 可以看作一个端口号
private static final UUID MY_UUID =
    UUID.fromString("fa87c0d0-afac-11de-8a39-0800200c9a66");
//像一个服务器一样时刻监听是否有连接建立
private class AcceptThread extends Thread{
    private BluetoothServerSocket serverSocket;

    public AcceptThread(boolean secure){
        BluetoothServerSocket temp = null;
        try {

```

```

        temp = mBluetoothAdapter.listenUsingRfcommWithServiceRecord(
            NAME_INSECURE, MY_UUID);
    } catch (IOException e) {
        Log.e("app", "listen() failed", e);
    }
    serverSocket = temp;
}

public void run(){
    BluetoothSocket socket=null;
    while(true){
        try {
            socket = serverSocket.accept();
        } catch (IOException e) {
            Log.e("app", "accept() failed", e);
            break;
        }
    }
    if(socket!=null){
        //此时可以新建一个数据交换线程,把此 socket 传进去
    }
}

//取消监听
public void cancel(){
    try {
        serverSocket.close();
    } catch (IOException e) {
        Log.e("app", "Socket Type" + socketType + "close() of server
            failed", e);
    }
}
}

```

(5) 交换数据

当搜索到蓝牙设备后,接下来可以获取设备的地址,通过此地址获取一个 BluetoothDevice 对象,可以将其看作一个客户端,通过对象 device.createRfcommSocketToServiceRecord(MY_UUID)同一个 UUID 可与服务器建立连接获取另一个 socket 对象。因此服务端与客户端各有一个 socket 对象,所以此时它们可以互相交换数据。演示代码如下。

```

//另一个设备去连接本机,相当于客户端
private class ConnectThread extends Thread{
    private BluetoothSocket socket;
    private BluetoothDevice device;
    public ConnectThread(BluetoothDevice device,boolean secure){
        this.device = device;
        BluetoothSocket tmp = null;
        try {

```

```

        tmp = device.createRfcommSocketToServiceRecord(MY_UUID_SECURE);
    } catch (IOException e) {
        Log.e("app", "create() failed", e);
    }
}

public void run(){
    mBluetoothAdapter.cancelDiscovery();    //取消设备查找
    try {
        socket.connect();
    } catch (IOException e) {
        try {
            socket.close();
        } catch (IOException e1) {
            Log.e("app", "unable to close() "+
                " socket during connection failure", e1);
        }
        connetionFailed();    //连接失败
        return;
    }
    //此时可以新建一个数据交换线程，把此 socket 传进去
}

public void cancel() {
    try {
        socket.close();
    } catch (IOException e) {
        Log.e("app", "close() of connect socket failed", e);
    }
}
}

```

(6) 建立数据通信线程

这一阶段的任务是读取通信数据，演示代码如下。

//建立连接后，进行数据通信的线程

```

private class ConnectedThread extends Thread{
    private BluetoothSocket socket;
    private InputStream inStream;
    private OutputStream outputStream;

    public ConnectedThread(BluetoothSocket socket){

        this.socket = socket;
        try {
            //获得输入、输出流
            inStream = socket.getInputStream();
            outputStream = socket.getOutputStream();
        } catch (IOException e) {
            Log.e("app", "temp sockets not created", e);
        }
    }
}

```

```

    }

    public void run() {
        byte[] buff = new byte[1024];
        int len=0;
        //读数据需不断监听，写不需要
        while(true){
            try {
                len = inStream.read(buff);
                //把读取到的数据发送给 UI 进行显示
                Message msg = handler.obtainMessage(BluetoothChat.
                    MESSAGE_READ,
                        len, -1, buff);
                msg.sendToTarget();
            } catch (IOException e) {
                Log.e("app", "disconnected", e);
                connectionLost(); //失去连接
                start();           //重新启动服务器
                break;
            }
        }
    }

    public void write(byte[] buffer) {
        try {
            outputStream.write(buffer);
            handler.obtainMessage(BluetoothChat.MESSAGE_WRITE, -1, -1, buffer)
                .sendToTarget();
        } catch (IOException e) {
            Log.e("app", "Exception during write", e);
        }
    }

    public void cancel() {
        try {
            socket.close();
        } catch (IOException e) {
            Log.e("app", "close() of connect socket failed", e);
        }
    }
}

```

至此为止，一个基本的蓝牙通信的基本操作已经全部完成。读者在开发此类项目时，只需按照上述流程进行即可。

11.4.2 开发一个控制玩具车的蓝牙遥控器

本实例的功能是，开发一个蓝牙遥控器，通过这个遥控器可以控制玩具小车的运动轨迹。下面开始这个实例之旅。

题 目	目 的	源码路径
实例 11-1	通过蓝牙遥控指挥玩具车	光盘\daima\11\lanya

本实例项目的具体实现流程如下。

- (1) 将蓝牙模块放在一辆玩具车上，并为其接通电源。
- (2) 打开 Eclipse 新建一个 Android 工程文件，命名为 lanya。
- (3) 编写布局文件 main.xml，在里面插入 5 个控制按钮，分别实现对玩具车的向前、左转、右转、后退和停止的控制。具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
    android:id="@+id/widget0"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
>
    <Button
        android:id="@+id/btnF"
        android:layout_width="100px"
        android:layout_height="60px"
        android:text="向前"
        android:layout_x="130px"
        android:layout_y="62px"
    >
    </Button>
    <Button
        android:id="@+id/btnL"
        android:layout_width="100px"
        android:layout_height="60px"
        android:text="左转"
        android:layout_x="20px"
        android:layout_y="152px"
    >
    </Button>
    <Button
        android:id="@+id/btnR"
        android:layout_width="100px"
        android:layout_height="60px"
        android:text="右转"
        android:layout_x="240px"
        android:layout_y="152px"
    >
    </Button>
    <Button
        android:id="@+id/btnB"
        android:layout_width="100px"
        android:layout_height="60px"
        android:text="后退"
        android:layout_x="130px"
```

```

android:layout_y="242px"
>
</Button>
<Button
android:id="@+id/btnS"
android:layout_width="100px"
android:layout_height="60px"
android:text="停止"
android:layout_x="130px"
android:layout_y="152px"
>
</Button>
</AbsoluteLayout>

```

(4) 编写蓝牙程序控制文件 `lanya.java`，具体实现流程如下。

① 定义类 `lanya`，然后设置 5 个按钮对象，具体代码如下。

```

public class lanya extends Activity {
    private static final String TAG = "THINBTCLIENT";
    private static final boolean D = true;
    private BluetoothAdapter mBluetoothAdapter = null;
    private BluetoothSocket btSocket = null;

    private OutputStream outputStream = null;
    Button mButtonF;
    Button mButtonB;
    Button mButtonL;
    Button mButtonR;
    Button mButtonS;
    .....
}

```

② 赋值蓝牙设备上的标准串行和要连接的蓝牙设备 MAC 地址，具体代码如下。

```

private static final UUID MY_UUID = UUID.fromString("00011101-0000-1000-8011-00805F9B34FB");
// 蓝牙设备上的标准串行
private static String address = "00:11:03:21:00:42";
// <==要连接的蓝牙设备 MAC 地址

```

③ 编写单击“向前”按钮的处理事件，具体代码如下。

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    // 向前
    mButtonF = (Button) findViewById(R.id.btnF);
    mButtonF.setOnClickListener(new Button.OnClickListener() {

        @Override
        public boolean onTouch(View v, MotionEvent event) {
            // TODO Auto-generated method stub
            String message;
            byte[] msgBuffer;
            int action = event.getAction();

```

```

switch(action)
{
case MotionEvent.ACTION_DOWN:
try {
    outputStream = btSocket.getOutputStream();
} catch (IOException e) {
    Log.e(TAG, "ON RESUME: Output stream creation failed.", e);
}
message = "1";
msgBuffer = message.getBytes();
try {
    outputStream.write(msgBuffer);
} catch (IOException e) {
    Log.e(TAG, "ON RESUME: Exception during write.", e);
}
break;
case MotionEvent.ACTION_UP:
try {
    outputStream = btSocket.getOutputStream();
} catch (IOException e) {
    Log.e(TAG, "ON RESUME: Output stream creation failed.", e);
}
message = "0";
msgBuffer = message.getBytes();
try {
    outputStream.write(msgBuffer);
} catch (IOException e) {
    Log.e(TAG, "ON RESUME: Exception during write.", e);
}
break;
}
return false;
}
});

```

④ 编写单击“后退”按钮的处理事件，具体代码如下。

```

mButtonB=(Button)findViewById(R.id.btnB);
mButtonB.setOnTouchListener(new Button.OnTouchListener(){
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        // TODO Auto-generated method stub
        String message;
        byte[] msgBuffer;
        int action = event.getAction();
        switch(action)
        {
        case MotionEvent.ACTION_DOWN:
        try {
            outputStream = btSocket.getOutputStream();
        } catch (IOException e) {

```

```

        Log.e(TAG, "ON RESUME: Output stream creation failed.", e);
    }
    message = "3";
    msgBuffer = message.getBytes();
    try {
        outputStream.write(msgBuffer);
    } catch (IOException e) {
        Log.e(TAG, "ON RESUME: Exception during write.", e);
    }
    break;

case MotionEvent.ACTION_UP:
    try {
        outputStream = btSocket.getOutputStream();
    } catch (IOException e) {
        Log.e(TAG, "ON RESUME: Output stream creation failed.", e);
    }
    message = "0";
    msgBuffer = message.getBytes();
    try {
        outputStream.write(msgBuffer);
    } catch (IOException e) {
        Log.e(TAG, "ON RESUME: Exception during write.", e);
    }
    break;
}

return false;
}
});

```

⑤ 编写单击“左转”按钮的处理事件，具体代码如下。

```

mButtonL=(Button)findViewById(R.id.btnL);
mButtonL.setOnClickListener(new Button.OnClickListener(){
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        // TODO Auto-generated method stub
        String message;
        byte[] msgBuffer;
        int action = event.getAction();
        switch(action)
        {
            case MotionEvent.ACTION_DOWN:
                try {
                    outputStream = btSocket.getOutputStream();
                } catch (IOException e) {
                    Log.e(TAG, "ON RESUME: Output stream creation failed.", e);
                }
                message = "2";
                msgBuffer = message.getBytes();
                try {

```



```

        outputStream.write(msgBuffer);
    } catch (IOException e) {
        Log.e(TAG, "ON RESUME: Exception during write.", e);
    }
    break;

case MotionEvent.ACTION_UP:
    try {
        outputStream = btSocket.getOutputStream();
    } catch (IOException e) {
        Log.e(TAG, "ON RESUME: Output stream creation failed.", e);
    }
    message = "0";
    msgBuffer = message.getBytes();
    try {
        outputStream.write(msgBuffer);
    } catch (IOException e) {
        Log.e(TAG, "ON RESUME: Exception during write.", e);
    }
    break;
}

return false;

}

});

```

⑥ 编写单击“右转”按钮的处理事件，具体代码如下。

```

mButtonR=(Button)findViewById(R.id.btnR);
mButtonR.setOnTouchListener(new Button.OnTouchListener(){
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        // TODO Auto-generated method stub
        String message;
        byte[] msgBuffer;
        int action = event.getAction();
        switch(action)
        {
            case MotionEvent.ACTION_DOWN:
                try {
                    outputStream = btSocket.getOutputStream();
                } catch (IOException e) {
                    Log.e(TAG, "ON RESUME: Output stream creation failed.", e);
                }
                message = "4";
                msgBuffer = message.getBytes();
                try {
                    outputStream.write(msgBuffer);
                } catch (IOException e) {
                    Log.e(TAG, "ON RESUME: Exception during write.", e);
                }
            }
        }
    }
});

```



```

    }
    break;

    case MotionEvent.ACTION_UP:
        try {
            outputStream = btSocket.getOutputStream();
        } catch (IOException e) {
            Log.e(TAG, "ON RESUME: Output stream creation failed.", e);
        }
        message = "0";
        msgBuffer = message.getBytes();
        try {
            outputStream.write(msgBuffer);
        } catch (IOException e) {
            Log.e(TAG, "ON RESUME: Exception during write.", e);
        }
        break;
    }

    return false;
}

});

```

⑦ 编写单击“停止”按钮的处理事件，具体代码如下。

```

mButtonS=(Button)findViewById(R.id.btnS);
mButtonS.setOnClickListener(new Button.OnClickListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        // TODO Auto-generated method stub
        if(event.getAction()==MotionEvent.ACTION_DOWN)
            try {
                outputStream = btSocket.getOutputStream();
            } catch (IOException e) {
                Log.e(TAG, "ON RESUME: Output stream creation failed.", e);
            }
            String message = "0";
            byte[] msgBuffer = message.getBytes();
            try {
                outputStream.write(msgBuffer);
            } catch (IOException e) {
                Log.e(TAG, "ON RESUME: Exception during write.", e);
            }
            return false;
        }
    });
if (D)
    Log.e(TAG, "+++ ON CREATE +++");
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
    Toast.makeText(this, "蓝牙设备不可用，请打开蓝牙！", Toast.LENGTH_LONG).
        show();
}

```

```

        finish();
        return;
    }
    if (!BluetoothAdapter.isEnabled()) {
        Toast.makeText(this, "请打开蓝牙并重新运行程序!", Toast.LENGTH_LONG).
            show();
        finish();
        return;
    }
    if (D)
        Log.e(TAG, "+++ DONE IN ON CREATE, GOT LOCAL BT ADAPTER +++");
}

```

⑧ 通过套接字建立蓝牙连接，如果连接失败则输出对应的失败提示。主要代码如下。

```

@Override
public void onStart() {
    super.onStart();
    if (D) Log.e(TAG, "++ ON START ++");
}
@Override
public void onResume() {
    super.onResume();
    if (D) {
        Log.e(TAG, "+ ON RESUME +");
        Log.e(TAG, "+ ABOUT TO ATTEMPT CLIENT CONNECT +");
    }
    DisplayToast("正在尝试连接智能小车，请稍后……");
    BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
    try {
        btSocket = device.createRfcommSocketToServiceRecord(MY_UUID);
    } catch (IOException e) {
        DisplayToast("套接字创建失败!");
    }
    DisplayToast("成功连接智能小车！可以开始操控了~~~");
    mBluetoothAdapter.cancelDiscovery();
    try {
        btSocket.connect();
        DisplayToast("连接成功建立，数据连接打开!");
    } catch (IOException e) {
        try {
            btSocket.close();
        } catch (IOException e2) {
            DisplayToast("连接没有建立，无法关闭套接字!");
        }
    }
    if (D)
        Log.e(TAG, "+ ABOUT TO SAY SOMETHING TO SERVER +");
}
@Override

```

```

public void onPause() {
    super.onPause();
    if (D)
        Log.e(TAG, "-- ON PAUSE -");
    if (outStream != null) {
        try {
            outStream.flush();
        } catch (IOException e) {
            Log.e(TAG, "ON PAUSE: Couldn't flush output stream.", e);
        }
    }
    try {
        btSocket.close();
    } catch (IOException e2) {

        DisplayToast("套接字关闭失败!");
    }
}
@Override
public void onStop() {
    super.onStop();
    if (D) Log.e(TAG, "-- ON STOP --");
}
@Override
public void onDestroy() {
    super.onDestroy();
    if (D) Log.e(TAG, "--- ON DESTROY ---");
}
public void DisplayToast(String str)
{
    Toast toast=Toast.makeText(this, str, Toast.LENGTH_LONG);
    toast.setGravity(Gravity.TOP, 0, 220);
    toast.show();
}
}

```

(5) 在文件 `AndroidManifest.xml` 中声明蓝牙权限，对应代码如下。

```

<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".lanya"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

```

至此，蓝牙控制玩具车的实例介绍完毕。本实例的实现比较简单，难度大的是双向控制，即实现每个设备都可以操控另外一个设备的功能，此时就需要有蓝牙功能的电脑或另外一部

Android 手机来完成测试了。在模拟器中因为不具备蓝牙设备，程序执行后会显示“蓝牙设备不可用，请打开蓝牙！”的提示，如图 11-3 所示。

蓝牙设备不可用，请打开蓝牙！

图 11-3 模拟器的运行效果

11.5 在穿戴设备中开发一个蓝牙控制器

本实例的功能是，在 Android 穿戴设备中开发一个蓝牙控制器，通过这个控制器可以实现如下功能。

- 打开蓝牙。
- 关闭蓝牙。
- 允许搜索。
- 开始搜索。
- 客户端。
- 服务器端。
- OBEX 服务器。

题 目	目 的	源码路径
实例 11-2	开发一个 Android 蓝牙控制器	光盘\daima\11\Activity01

11.5.1 界面布局

本实例主界面的布局文件 main.xml，主要实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:padding="10dip">
    <Button android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="打开蓝牙"
        android:onClick="onEnableButtonClicked" />
    <Button android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="关闭蓝牙"
        android:onClick="onDisableButtonClicked" />
    <Button android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="允许搜索"
        android:onClick="onMakeDiscoverableButtonClicked" />
    <Button android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="开始搜索"
        android:onClick="onStartDiscoveryButtonClicked" />
    <Button android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="客户端"
        android:onClick="onOpenClientSocketButtonClicked" />
    <Button android:layout_width="fill_parent"
```

```

        android:layout_height="wrap_content" android:text="服务器端"
        android:onClick="onOpenServerSocketButtonClicked" />
<Button android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="OBEX 服务器"
        android:onClick="onOpenOBEXServerSocketButtonClicked" />
</LinearLayout>

```

此时在执行之后的界面效果如图 11-4 所示。



图 11-4 执行效果

服务器端的界面布局文件是 `server_socket.xml`，主要实现代码如下。

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:padding="10dip">
    <Button android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="Stop server"
        android:onClick="onButtonClicked" />

    <ListView android:id="@+id/android:list" android:layout_width="fill_parent"
        android:layout_height="fill_parent" />
</LinearLayout>

```

其他几个界面的布局文件和上述文件类似，在此不再一一列出。

11.5.2 响应单击按钮

编写主界面的程序文件 `Activity01.java`，功能是根据用户单击屏幕中的按钮来调用对应的处理函数，例如，单击“服务器端”按钮会执行函数 `onOpenServerSocketButtonClicked(View view)`。文件 `Activity01.java` 的主要实现代码如下。

```

public class Activity01 extends Activity
{
    /* 取得默认的蓝牙适配器 */
    private BluetoothAdapter _bluetooth = BluetoothAdapter.
        getDefaultAdapter();
    /* 请求打开蓝牙 */
    private static final int REQUEST_ENABLE = 0x1;

```



```
/* 请求能够被搜索 */
private static final int    REQUEST_DISCOVERABLE    = 0x2;
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
/* 开启蓝牙 */
public void onEnableButtonClicked(View view)
{
    // 用户请求打开蓝牙
    //Intent enabler = new Intent (BluetoothAdapter.ACTION_REQUEST_ENABLE);
    //startActivityForResult(enabler, REQUEST_ENABLE);
    //打开蓝牙
    _bluetooth.enable();
}
/* 关闭蓝牙 */
public void onDisableButtonClicked(View view)
{
    _bluetooth.disable();
}
/* 使设备能够被搜索 */
public void onMakeDiscoverableButtonClicked(View view)
{
    Intent enabler = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
    startActivity(enabler, REQUEST_DISCOVERABLE);
}
/* 开始搜索 */
public void onStartDiscoveryButtonClicked(View view)
{
    Intent enabler = new Intent(this, DiscoveryActivity.class);
    startActivity(enabler);
}
/* 客户端 */
public void onOpenClientSocketButtonClicked(View view)
{
    Intent enabler = new Intent(this, ClientSocketActivity.class);
    startActivity(enabler);
}
/* 服务端 */
public void onOpenServerSocketButtonClicked(View view)
{
    Intent enabler = new Intent(this, ServerSocketActivity.class);
    startActivity(enabler);
}
/* OBEX 服务器 */
public void onOpenOBEXServerSocketButtonClicked(View view)
{
}
```

```

Intent enabler = new Intent(this, OBEXActivity.class);
startActivity(enabler);
}
}

```

11.5.3 和指定的服务器建立连接

编写程序文件 ClientSocketActivity.java, 功能是创建一个 Socket 连接, 和指定的服务器建立连接。文件 ClientSocketActivity.java 的主要实现代码如下。

```

public class ClientSocketActivity extends Activity
{
    private static final String TAG = ClientSocketActivity.class.getSimpleName();
    private static final int REQUEST_DISCOVERY = 0x1;
    private Handler _handler = new Handler();
    private BluetoothAdapter _bluetooth = BluetoothAdapter.getDefaultAdapter();
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_BLUR_BEHIND,
            WindowManager.LayoutParams.FLAG_BLUR_BEHIND);
        setContentView(R.layout.client_socket);
        if (!_bluetooth.isEnabled()) {
            finish();
            return;
        }
        Intent intent = new Intent(this, DiscoveryActivity.class);
        /* 提示选择一个要连接的服务器 */
        Toast.makeText(this, "select device to connect", Toast.LENGTH_SHORT).show();
        /* 跳转到搜索的蓝牙设备列表区, 进行选择 */
        startActivityForResult(intent, REQUEST_DISCOVERY);
    }
    /* 选择了服务器之后进行连接 */
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode != REQUEST_DISCOVERY) {
            return;
        }
        if (resultCode != RESULT_OK) {
            return;
        }
        final BluetoothDevice device = data.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
        new Thread() {
            public void run() {
                /* 连接 */
                connect(device);
            };
        }.start();
    }
    protected void connect(BluetoothDevice device) {
        BluetoothSocket socket = null;
        try {

```

```

// 创建一个 Socket 连接: 只需要服务器在注册时的 UUID 号
// socket = device.createRfcommSocketToServiceRecord(BluetoothProtocols.
OBEX_OBJECT_PUSH_PROTOCOL_UUID);
socket = device.createRfcommSocketToServiceRecord(UUID.fromString(
("a60f35f0-b93a-11de-8a39-08002009c666")));
//连接
socket.connect();
} catch (IOException e) {
    Log.e(TAG, "", e);
} finally {
    if (socket != null) {
        try {
            socket.close();
        } catch (IOException e) {
            Log.e(TAG, "", e);
        }
    }
}
}
}
}

```

11.5.4 搜索附近的蓝牙设备

编写程序文件 `DiscoveryActivity.java`, 功能是搜索设备附近的蓝牙设备, 并在列表中显示搜索到的蓝牙设备。文件 `DiscoveryActivity.java` 的主要实现代码如下。

```

public class DiscoveryActivity extends ListActivity
{
    private Handler _handler = new Handler();
    /* 取得默认的蓝牙适配器 */
    private BluetoothAdapter _bluetooth = BluetoothAdapter.getDefaultAdapter();
    /* 用来存储搜索到的蓝牙设备 */
    private List<BluetoothDevice> _devices = new ArrayList<BluetoothDevice>();
    /* 是否完成搜索 */
    private volatile boolean _discoveryFinished;
    private Runnable _discoveryWorkder = new Runnable() {
        public void run()
        {
            /* 开始搜索 */
            _bluetooth.startDiscovery();
            for (;;)
            {
                if (_discoveryFinished)
                {
                    break;
                }
                try
                {
                    Thread.sleep(100);
                }
                catch (InterruptedException e){}
            }
        }
    };
}

```

```

    }
}

};
/**
 * 接收器
 * 当搜索蓝牙设备完成时调用
 */
private BroadcastReceiver _foundReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        /* 从 intent 中取得搜索结果数据 */
        BluetoothDevice device = intent
            .getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
        /* 将结果添加到列表中 */
        _devices.add(device);
        /* 显示列表 */
        showDevices();
    }
};

private BroadcastReceiver _discoveryReceiver = new BroadcastReceiver() {

    @Override
    public void onReceive(Context context, Intent intent)
    {
        /* 卸载注册的接收器 */
        unregisterReceiver(_foundReceiver);
        unregisterReceiver(this);
        _discoveryFinished = true;
    }
};

protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_BLUR_BEHIND,
        WindowManager.LayoutParams.FLAG_BLUR_BEHIND);
    setContentView(R.layout.discovery);
    /* 如果蓝牙适配器没有打开, 则结果 */
    if (!_bluetooth.isEnabled())
    {
        finish();
        return;
    }
    /* 注册接收器 */
    IntentFilter discoveryFilter = new IntentFilter(BluetoothAdapter.
        ACTION_DISCOVERY_FINISHED);
    registerReceiver(_discoveryReceiver, discoveryFilter);
    IntentFilter foundFilter = new IntentFilter(BluetoothDevice.ACTION_
        FOUND);
    registerReceiver(_foundReceiver, foundFilter);
    /* 显示一个对话框, 正在搜索蓝牙设备 */

```

```

SamplesUtils.indeterminate(DiscoveryActivity.this, _handler, "Scanning...",
_discoveryWorkder, new OnDismissListener() {
    public void onDismiss(DialogInterface dialog)
    {
        for (; _bluetooth.isDiscovering();)
        {
            _bluetooth.cancelDiscovery();
        }
        _discoveryFinished = true;
    }
}, true);
}
/* 显示列表 */
protected void showDevices()
{
    List<String> list = new ArrayList<String>();
    for (int i = 0, size = _devices.size(); i < size; ++i)
    {
        StringBuilder b = new StringBuilder();
        BluetoothDevice d = _devices.get(i);
        b.append(d.getAddress());
        b.append('\n');
        b.append(d.getName());
        String s = b.toString();
        list.add(s);
    }
    final ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, list);
    _handler.post(new Runnable() {
        public void run()
        {
            setListAdapter(adapter);
        }
    });
}
protected void onListItemClick(ListView l, View v, int position, long id)
{
    Intent result = new Intent();
    result.putExtra(BluetoothDevice.EXTRA_DEVICE, _devices.get(position));
    setResult(RESULT_OK, result);
    finish();
}
}

```

11.5.5 建立和 OBEX 服务器的数据传输

编写程序文件 OBEXActivity.java, 功能是建立和 OBEX 服务器的数据传输。OBEX 全称为 Object Exchange, 中文对象交换, 所以也称为对象交换协议。OBEX 协议通过简单地使用 PUT 和 GET 命令实现在不同的设备、不同的平台之间方便、高效地交换信息。支持的设备

广泛，如 PC、PDA、电话、摄像头、自动答录机、计算器、数据采集器和手表等。文件 OBEXActivity.java 的主要实现代码如下。

```
public class OBEXActivity extends Activity
{
    private static final String TAG = "@MainActivity";
    private Handler _handler = new Handler();
    private BluetoothServerSocket _server;
    private BluetoothSocket _socket;
    private static final int OBEX_CONNECT = 0x80;
    private static final int OBEX_DISCONNECT = 0x81;
    private static final int OBEX_PUT = 0x02;
    private static final int OBEX_PUT_END = 0x82;
    private static final int OBEX_RESPONSE_OK = 0xa0;
    private static final int OBEX_RESPONSE_CONTINUE = 0x90;
    private static final int BIT_MASK = 0x000000ff;
    Thread t = new Thread()
    {
        public void run()
        {
            try
            {
                _server = BluetoothAdapter.getDefaultAdapter().
                listenUsingRfcommWithServiceRecord("OBEX", null);
                new Thread()
                {
                    public void run()
                    {
                        Log.d("@Rfcom", "begin close");
                        try
                        {
                            _socket.close();
                        }
                        catch (IOException e)
                        {
                            Log.e(TAG, "", e);
                        }
                        Log.d("@Rfcom", "end close");
                    };
                }.start();
                _socket = _server.accept();
                reader.start();
                Log.d(TAG, "shutdown thread");
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
        };
    };
};
```

```
Thread reader = new Thread()
{
    public void run()
    {
        try
        {
            Log.d(TAG, "getting inputStream");
            InputStream inputStream = _socket.getInputStream();
            OutputStream outputStream = _socket.getOutputStream();
            Log.d(TAG, "got inputStream");
            int read = -1;
            byte[] bytes = new byte[2048];
            ByteArrayOutputStream baos = new ByteArrayOutputStream
                (bytes.length);
            while ((read = inputStream.read(bytes)) != -1)
            {
                baos.write(bytes, 0, read);
                byte[] req = baos.toByteArray();
                int op = req[0] & BIT_MASK;
                Log.d(TAG, "read:" + Arrays.toString(req));
                Log.d(TAG, "op:" + Integer.toHexString(op));
                switch (op)
                {
                    case OBEX_CONNECT:
                        outputStream.write(new byte[] { (byte) OBEX_RESPONSE_
                            OK, 0, 7, 16, 0, 4, 0 });
                        break;
                    case OBEX_DISCONNECT:
                        outputStream.write(new byte[] { (byte) OBEX_RESPONSE_
                            OK, 0, 3, 0 });
                        break;
                    case OBEX_PUT:
                        outputStream.write(new byte[] { (byte) OBEX_RESPONSE_
                            CONTINUE, 0, 3, 0 });
                        break;
                    case OBEX_PUT_END:
                        outputStream.write(new byte[] { (byte) OBEX_RESPONSE_
                            OK, 0, 3, 0 });
                        break;
                    default:
                        outputStream.write(new byte[] { (byte) OBEX_RESPONSE_
                            OK, 0, 3, 0 });
                }
                Log.d(TAG, new String(baos.toByteArray(), "utf-8"));
                baos = new ByteArrayOutputStream(bytes.length);
            }
        }
    }
}
```

```

        Log.d(TAG, new String(baos.toByteArray(), "utf-8"));
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
};

};

private Thread put = new Thread() {
    public void run()
    {
    };
};

public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.obex_server_socket);
    t.start();
}

protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    Log.d(TAG, data.getData().toString());
    switch (requestCode)
    {
        case (1):
            if (resultCode == Activity.RESULT_OK)
            {
                Uri contactData = data.getData();
                @SuppressWarnings("deprecation")
                Cursor c = managedQuery(contactData, null, null, null, null);
                for (; c.moveToNext(); )
                {
                    Log.d(TAG, "c1-----");
                    dump(c);
                    Uri uri = Uri.withAppendedPath(data.getData(),
                        ContactsContract.Contacts.Photo.CONTENT_DIRECTORY);
                    @SuppressWarnings("deprecation")
                    Cursor c2 = managedQuery(uri, null, null, null, null);
                    for (; c2.moveToNext(); )
                    {
                        Log.d(TAG, "c2-----");
                        dump(c2);
                    }
                }
            }
            break;
    }
}
}

```

11.5.6 实现蓝牙服务器端的数据处理

编写文件 `ServerSocketActivity.java`，功能是实现蓝牙服务器端的数据处理，建立服务器端和客户端的连接和监听工作，其中的监听工作和停止服务器工作由独立的函数实现。文件 `ServerSocketActivity.java` 的主要实现代码如下。

```
public class ServerSocketActivity extends ListActivity
{
    /* 一些常量，代表服务器的名称 */
    public static final String PROTOCOL_SCHEME_L2CAP = "btl2cap";
    public static final String PROTOCOL_SCHEME_RFCOMM = "btspp";
    public static final String PROTOCOL_SCHEME_BT_OBEX = "btgoep";
    public static final String PROTOCOL_SCHEME_TCP_OBEX = "tcpobex";
    private static final String TAG = ServerSocketActivity.class.getSimpleName();
    private Handler _handler = new Handler();
    /* 取得默认的蓝牙适配器 */
    private BluetoothAdapter _bluetooth = BluetoothAdapter.getDefaultAdapter();
    /* 蓝牙服务器 */
    private BluetoothServerSocket _serverSocket;
    /* 线程-监听客户端的连接 */
    private Thread _serverWorker = new Thread() {
        public void run() {
            listen();
        };
    };
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_BLUR_BEHIND,
            WindowManager.LayoutParams.FLAG_BLUR_BEHIND);
        setContentView(R.layout.server_socket);
        if (!_bluetooth.isEnabled()) {
            finish();
            return;
        }
        /* 开始监听 */
        _serverWorker.start();
    }
    protected void onDestroy() {
        super.onDestroy();
        shutdownServer();
    }
    protected void finalize() throws Throwable {
        super.finalize();
        shutdownServer();
    }
    /* 停止服务器 */
    private void shutdownServer() {
        new Thread() {
            public void run() {
                _serverWorker.interrupt();
            }
        }.start();
    }
}
```



```

        if (_serverSocket != null) {
            try {
                /* 关闭服务器 */
                _serverSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "", e);
            }
            _serverSocket = null;
        }
    };
    }.start();
}

public void onClicked(View view) {
    shutdownServer();
}

protected void listen() {
    try {
        /* 创建一个蓝牙服务器
        * 参数分别为: 服务器名称、UUID
        */
        _serverSocket = _bluetooth.listenUsingRfcommWithServiceRecord(
            PROTOCOL_SCHEME_RFCOMM,
            UUID.fromString("a60f35f0-b93a-11de-8a39-08002009c666"));
        /* 客户端连线列表 */
        final List<String> lines = new ArrayList<String>();
        _handler.post(new Runnable() {
            public void run() {
                lines.add("Rfcomm server started...");
                ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                    ServerSocketActivity.this,
                    android.R.layout.simple_list_item_1, lines);
                setListAdapter(adapter);
            }
        });
        /* 接受客户端的连接请求 */
        BluetoothSocket socket = _serverSocket.accept();
        /* 处理请求内容 */
        if (socket != null) {
            InputStream inputStream = socket.getInputStream();
            int read = -1;
            final byte[] bytes = new byte[2048];
            for (; (read = inputStream.read(bytes)) > -1;) {
                final int count = read;
                _handler.post(new Runnable() {
                    public void run() {
                        StringBuilder b = new StringBuilder();
                        for (int i = 0; i < count; ++i) {
                            if (i > 0) {
                                b.append(' ');
                            }
                        }
                    }
                });
            }
        }
    }
}

```



```

        String s = Integer.toHexString(bytes[i] & 0xFF);
        if (s.length() < 2) {
            b.append('0');
        }
        b.append(s);
    }
    String s = b.toString();
    lines.add(s);
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(
        ServerSocketActivity.this,
        android.R.layout.simple_list_item_1, lines);
    setListAdapter(adapter);
    }
    });
    }
} catch (IOException e) {
    Log.e(TAG, "", e);
} finally {
}
}
}

```

(7) 在文件 `AndroidManifest.xml` 中声明对蓝牙设备的使用权限，具体代码如下。

```

<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.READ_CONTACTS"/>

```

至此，整个实例全部实现完毕，本实例需要在真实机器上进行测试。



蓝牙 4.0 BLE 详解

蓝牙 4.0 BLE 前身是 NOKIA 开发的 Wibree 技术，主要用于实现移动智能终端与周边配件之间的持续连接，是功耗极低的短距离无线通信技术，并且有效传输距离提升到了 100 米以上，同时只需要一颗纽扣电池就可以工作数年之久，以上诸多技术优势使得 BLE 的发展前景相当可观。在本章的内容中，将首先讲解低功耗蓝牙（Bluetooth Low Energy）技术的基本知识，为读者学习本书后面的知识打下基础。

12.1 短距离无线通信技术概览



在物联网中物与网相连的最后数米，发挥关键作用的是短距离无线传输技术。目前有多种短距离无线传输技术可以应用在物联网中，在我国，除已经得到大规模应用的 RFID 之外，还有 WiFi、ZigBee、蓝牙等比较成熟的技术，以及基于这些技术发展而来的新技术。这些技术各具特点，因对其传输速度、距离、耗电量等方面的要求不同，形成了各自不同的物联网应用场景。在本节的内容中，将简要介绍当今实现短距离无线通信的常用技术。

12.1.1 ZigBee——低功耗、自组网

ZigBee 以其鲜明的技术特点在物联网中受到了高度关注，该技术使用的频段分别为 2.4GHz、868MHz（欧洲）及 915MHz（美国）。其主要的技术特点：一是数据传输速率低，只有 10kbps~250kbps；二是功耗低，低传输速率带来了仅为 1 毫瓦的低发射功率。据估算，ZigBee 设备仅靠两节 5 号电池就可以维持长达 6 个月到两年的使用时间，这是 ZigBee 的一个独特优势；三是成本低，因为 ZigBee 传输速率低、协议简单；四是网络容量大，每个 ZigBee 网络最多可以支持 255 个设备，一个区域内可以同时存在最多 100 个 ZigBee 网络，网络组成灵活。ZigBee 芯片主要企业有德州仪器、飞思卡尔等。市场调研机构 ABI Reserch 的一份数据显示，2005~2012 年，ZigBee 市场的年均复合增长率为 63%。

“ZigBee 是从家庭自动化开始的，在瑞典哥德堡就是从智能电表开始的，然后进一步用到燃气表、水表、热力表等家庭各种计量表。”在 2011 中国无线世界暨物联网大会上 ZigBee 联盟大中华区代表黄家瑞说，“ZigBee 在智能电表里不仅仅是远程抄表工具，它是一个终端，也是一个网关，这些网关结合在一起，整个小区就变成了智能电网小区，智能电表可以搜集家里所有家电的用电信息。”

目前，ZigBee 正在完善其网关标准，2011 年 7 月底发布了第十个标准 ZigBee Gateway

(ZigBee 网关)。ZigBee Gateway 提供了一种简单、高成本效益的互联网连接方式,使服务提供商、企业和个人消费者有机会运行这些设备并将 ZigBee 网络连接至互联网。ZigBee Gateway 是 ZigBee Network Devicesp (ZigBee 网络设备)这一新类别范畴的首个标准,这将使 ZigBee 发展进一步提速。

12.1.2 WiFi——大带宽支持家庭互联

WiFi 是以太网的一种无线扩展技术,如果有多个用户同时通过一个热点接入,带宽将被这些用户共享,WiFi 的速率会降低,处于 2.4GHz 频段的 WiFi 信号受墙壁阻隔的影响较小。WiFi 的传输速率随着技术的演进还在不断提高,我国电信运营商在构建无线城市中采用的 WiFi 技术部分已经升级到 802.11n,最高速率从 802.11g 标准的 11Mbps 提高到 50Mbps 以上。在 WiFi 产业链中,最大的芯片企业是博通。

在笔记本电脑和手机上已经得到广泛应用的 WiFi 正在向消费电子产品渗透,Myron Hattig 说:“除了手机外,已经有 25%的消费类电子设备使用 WiFi,在打印机、洗衣机上都在使用 WiFi,家用电器生产商协会将 WiFi 作为一个更高级别的智能电器沟通技术。WiFi 可以将设备与设备相连,从而使整个家庭的家用电器、电子设备相连。”

最大 WiFi 芯片制造商博通正在推动 WiFi Direct 标准的商用,以支持这种设备到设备的直连。特别是在家庭互联中,相片、视频等大数据量的业务在手机、平板电脑、电视等设备中的直连应用前景广阔。Myron Hattig 说:“直接技术可将平板电脑的内容展示在电视上。”

基于 WiFi 发展起来的 WIGIG 也是未来家庭互联市场有力的竞争技术。该技术可工作在 40GHz~60GHz 的超高频段,其传输速度可以达到 1Gbps 以上,不能穿过墙壁。目前英特尔、高通等芯片企业在支持 WIGIG 发展,目前该技术还在完善中,如需要进一步降低功耗等。

12.1.3 蓝牙——4.0 进入低功耗时代

蓝牙能在包括移动电话、PDA、无线耳机、笔记本电脑、相关外设等众多设备之间进行无线信息交换。蓝牙采用分散式网络结构及快跳频和短包技术,支持点对点及点对多点通信,工作在全球通用的 2.4GHz 频段,其数据速率为 1Mbps。

2010 年 7 月,以低功耗为特点的蓝牙 4.0 标准推出,蓝牙大中华区技术市场经理吕荣良将其看作蓝牙第二波发展高潮的标志,他表示:“蓝牙可以跨领域应用,主要有 4 个生态系统,分别是智能手机与笔记本电脑等终端市场、消费电子市场、汽车前装市场和健身运动器材市场。”

NFC 和 UWB 曾经是十分受关注的短距离无线接入技术,但其发展已经日渐势微。业内专家认为,无线频谱的规划和利用在短距离通信中日益重要。短距离通信技术目前主要采用 2.4GHz 的开放频谱,但随着物联网的发展和大量短距离通信技术的应用,频谱需求会快速增长,视频、图像等大数据量的通信正在寻求更高频段的解决方案。

12.1.4 NFC——必将逐渐远离历史舞台

NFC 是近场通信(Near Field Communication)的缩写,此技术由非接触式射频识别(RFID)演变而来,由飞利浦半导体(现恩智浦半导体)、诺基亚和索尼共同研制开发,其基础是 RFID

及互连技术。NFC 是一种短距高频的无线电技术,在 13.56MHz 频率下运行距离为 20cm。其传输速度有 106 kbps、212 kbps 和 424 kbps 3 种。目前近场通信已经获得 ISO/IEC IS 18092 国际标准、ECMA-340 标准与 ETSI TS 102 190 标准。NFC 采用主动和被动两种读取模式。

NFC 近场通信技术由非接触式射频识别 (RFID) 及互联互通技术整合演变而来,在单一芯片上结合感应式读卡器、感应式卡片和点对点的功能,能在短距离内与兼容设备进行识别和数据交换。工作频率为 13.56MHz。但是使用这种手机支付方案的用户必须更换特制的手机。目前这项技术在日韩被广泛应用。手机用户凭着配置了支付功能的手机就可以行遍全国:他们的手机可以用作机场登机验证、大厦的门禁钥匙、交通一卡通、信用卡、支付卡等。

NFC 和蓝牙 (Bluetooth) 都是短程通信技术,而且都被集成到移动电话。但 NFC 不需要复杂的设置程序。NFC 也可以简化蓝牙连接。NFC 略胜蓝牙的地方在于设置程序较短,但无法达到低功耗蓝牙 (Bluetooth Low Energy) 的速度。在两台 NFC 设备相互连接的设备识别过程中,使用 NFC 来替代人工设置会使创建连接的速度大大加快,所用时间会少于 1/10 秒。

12.2 蓝牙 4.0 BLE 基础

蓝牙 4.0 也称为 Bluetooth Smart,而 BLE 是 Bluetooth Low Energy 的缩写,属于蓝牙低功耗协议,Android 4.3 以上版本及苹果手机等都支持蓝牙 BLE,主要面向传感器应用市场提供短时间、小数据传输,如健康领域的手机监测血压和体育领域的手机计步器等。在本节的内容中,将详细讲解蓝牙 4.0 BLE 的基础知识。

12.2.1 蓝牙 4.0 的最杰出表现是低功耗

蓝牙 4.0 是 2012 年最新蓝牙版本,是 3.0 的升级版本;较 3.0 版本更省电、成本低、3 毫秒低延迟、超长有效连接距离、AES-128 加密等,通常被用在蓝牙耳机、蓝牙音箱等设备上。

蓝牙 4.0 最重要的特性是省电,极低的运行和待机功耗可以使一粒纽扣电池连续工作数年之久。此外,低成本和跨厂商互操作性,3 毫秒低延迟、AES-128 加密等诸多特色,可以用于计步器、心律监视器、智能仪表、传感器物联网等众多领域,大大扩展蓝牙技术的应用范围。

蓝牙 4.0 已经走向了商用,在最新款的 Xperia Z、Galaxy S3、S4、Note2、Note3、SurfaceRT、iPhone 5S、iPhone 5、iPhone 4S、魅族 MX3、Moto Droid Razr、HTC One X、小米手机 2、The New iPad、iPad 4、MacBook Air、Macbook Pro,以及台商 ACER AS3951 系列/Getway NV57 系列,ASUS UX21/31 三星 NOTE 系列上都已应用了蓝牙 4.0 技术。很多品牌已推出蓝牙 4.0 版本周边设备,同时支持蓝牙 4.0 和 NFC 的 WOOWI HERO、jabra MOTION、WOOWI 泡我等,支持蓝牙 4.0 的音箱有 Big jambox、Braven 等产品。

蓝牙技术联盟 (Bluetooth SIG) 2010 年 7 月 7 日宣布,正式采纳蓝牙 4.0 核心规范 (Bluetooth Core Specification Version 4.0),并启动对应的认证计划。会员厂商可以提交其产品进行测试,通过后将获得蓝牙 4.0 标准认证。该技术拥有极低的运行和待机功耗,使用一粒纽扣电池甚至可连续工作数年之久。

蓝牙 4.0 的主要特性如下。

- 超低的峰值、平均和待机模式功耗。
- 使用标准纽扣电池可运行一年乃至数年。
- 低成本。
- 不同厂商设备交互性。
- 无线覆盖范围增强。
- 完全向下兼容。
- 低延迟 (APT-X)。

12.2.2 蓝牙 4.0 的优势

蓝牙 4.0 将传统蓝牙技术、高速技术和低功耗技术这三种规格集一体，与 3.0 版本相比最大的不同就是低功耗。4.0 版本的功耗较老版本降低了 90%，这样更省电。蓝牙技术联盟大中华区技术市场经理吕荣良表示：“随着蓝牙技术由手机、游戏、耳机、便携电脑和汽车等传统应用领域向物联网、医疗等新领域的扩展，对低功耗的要求会越来越高。4.0 版本强化了蓝牙在数据传输上的低功耗性能。”

低功耗版本使蓝牙技术得以延伸到采用纽扣电池供电的一些新兴市场。蓝牙低功耗技术是基于蓝牙低功耗无线技术核心规格的升级版，为开拓钟表、远程控制、医疗保健及运动感应器等广大新兴市场的应用奠定基础。

这项技术将应用于每年出售的数亿台蓝牙耳机、个人电脑及掌上电脑。以最低耗能提供持久的无线连接，有效扩大相关应用产品的覆盖距离，开辟全新的网络服务。低功耗无线技术的特点在于超低的峰值、平均值及待机耗能；使装置配件和人机介面装置 (HIDs) 具备超低成本和轻巧的特性；更能使手机及个人电脑相关配件的成本降至最低、体积更小；全球适用之外，更加直观，且能确保多种设备连接的互操作性。

蓝牙 4.0 对个人健身和健康市场的影响很大。无论在跑步机上，或是在办公室的小工具上，Fitbit 无线师，耐克公司的新 Fuelband，摩托罗拉 MOTACTV 和时尚的基带都是很好的例子。而且健身手表也承诺使用蓝牙跟踪体力活动和心率。

另外蓝牙 4.0 依旧向下兼容，包含经典蓝牙技术规范 and 最高速度 24Mbps 的蓝牙高速技术规范。三种技术规范可单独使用，也可同时运行。

12.2.3 Bluetooth 4.0 BLE 推动了可穿戴设备的兴起

到目前为止，谈到可穿戴设备时都要提到一个问题：支持蓝牙还是用无线网络与智能手机相连。这是衡量可穿戴设备是否能与智能手机上的软件顺利“对话”的主要依据。其实在过去的一段时间内，人们已经习惯了“Bluetooth X.0 版”的说法，从 Bluetooth 4.0 开始，这项技术被 Bluetooth SIG (Special Interest Group，负责推动蓝牙技术标准的开发和将其授权给制造商的非营利组织) 改名为 Bluetooth Smart 或 Bluetooth Smart Ready。Bluetooth SIG 首席营销官 Suke Jawanda 对 PingWest 说：“未来 Bluetooth SIG 也将继续淡化 X.0 的概念，将更加强调 Bluetooth Smart，原因是 X.0 是说给极客听的，而 Bluetooth SIG 希望普通消费者也能听懂。”

可穿戴设备与智能手机之间的数据传输方式对蓝牙技术的要求也与以往不同。Suke

Jawanda 用自己手腕上的 Fitbit Flex 举例：“过去当我们谈到蓝牙技术和数据传输，主要考虑的是类似 Spotify 这种在一个较长的时间段里输送数据的需求，现在像 Fitbit Flex 是先收集数据，再断续地在某些‘时刻’里将数据传送到用户的手机上，两种数据传输方式不同。Bluetooth Smart 的低耗能技术就可以满足这一需求了。”

Suke Jawanda 向 PingWest 解释说：“现在不少设备制造商都在强调自己支持蓝牙低功耗技术（Bluetooth Low Energy），其实它只是 Bluetooth Smart 其中的一个功能。支持 Bluetooth Smart 的设备都支持蓝牙低功耗技术。”

虽然特意强调低功耗技术是给消费者造成一种“省电、省流量”的印象，但是换个角度来看，每个产品说明自己支持 Bluetooth Smart 的背后就是可穿戴设备为什么在此时流行的重要原因之一——Bluetooth Smart 对操作系统和硬件设备的支持情况，决定了可穿戴设备能否以较低的成本与软件进行数据传输，接下来才是解决软件获得数据之后怎么处理的问题。

根据 Suke Jawanda 的介绍，Bluetooth Smart 对可穿戴设备的支持分成硬件和软件。以 Fitbit 为例，如果 Fitbit 开发一款新的产品，支持 Bluetooth Smart，同时要求软件，也就是从 iOS 或者 Android——操作系统层面要支持 Bluetooth Smart，苹果是从 iOS 5（iPhone 4S 及以上版本的手机）开始支持 Bluetooth Smart，而 Google 直到 Android 4.3 才开始支持。

当然 Bluetooth Smart 并不是唯一推动穿戴设备发展的原因，有些可穿戴设备可以用其他方式传输数据，如无线网络，但是人们不可能一直在无线网络环境下生活。

除了可穿戴设备，汽车呢？除了用蓝牙接打电话，还能做些什么？Suke Jawanda 说：“现在我们知道汽车能做到的是通过蓝牙进行语音操作、接打电话，未来我们想象是用蓝牙技术可以不再用钥匙，你的手机就可以作为车钥匙；另一个是利用更多的传感器收集数据，让车与车之间‘对话’，例如你的车可以知道前后三辆车的时速，当他们减速时你的车能提醒你前方的车在减速可能是遇到什么情况等。但这里最大的问题是汽车行业技术滞后，比如你現在看到的一个汽车领域的新技术，真正应用到生产、被推广恐怕是 2~3 年后的事情，而且人们买一辆车的期待是要用 10~15 年的，也就是你买了一辆车之后 10 年内可能都体验不到汽车领域的新技术，这个问题现在还没有很好的解决方案。”



注意

12.2.3 节的内容引用自“ZOL 网的科技频道：<http://news.zol.com.cn/article/179109.html>”。

12.2.4 BLE 推动了 Android 可穿戴设备的发展

2013 年 9 月，Texas Instruments（德州仪器）引入了一款基于 Android 4.3 的 app Bluetooth Smart SensorTag，通过该 app 每个人都将有能力开发可连接 BLE（Bluetooth low energy，低功耗蓝牙）传感器的应用。软硬件结合的 Bluetooth 电子瓷片 Tile 通过 Selfstarter 筹得 260 万美元，该数字是其原始目标的 130 倍，而 Tile 正是使用了 BLE 和相应 app 来为人们跟踪钱包、自行车、手提箱等贵重东西，防止丢失。诸如 Tile、Davis、StickNFind 等如雨后春笋般冒出的新鲜事物，不免让人们好奇：BLE 是否正引发智能标签的革命浪潮？

2013 年 5 月，当 Google 宣布在 Android 4.3 系统 Jelly Bean 中支持 BLE 或许是促成此势的原因之一。而随着市场上 BLE 的插件越来越多，Google 发现重新设计 Bluetooth 整套软件组件

已经不可拖延。当 BLE 几乎成为必不可少的物联网协议时，这次 Google 采取的升级行动瞬间给开发者们打开了一扇新的大门，从此后可以运用 BLE 为相互连接的设备开发 app。

这对 Android 设备的使用者来说无疑是件好事，他们有机会尝试到最新鲜的 BLE app 和服务。不过，市面上要有较多支持 Jelly Bean 4.3 的设备，恐怕还要等到 2013 年秋季。目前只有 Nexus 7 平板及一些 Nexus 硬件设备支持 Jelly Bean 4.3，而平板电脑绝非用来管理门锁、钥匙、钱包的最佳设备。

而在未来，当 BLE 把每个人的 Android 设备变为一个传感器 tag 时，它所能做的就不仅仅是通过 Tile 和 StickNFind 来找些东西而已。BLE 拥有巨大的可拓展性，好比 Tile app 和传感器可以用来构建一个 P2P 的网络——模拟 GPS 的功能。同时，Mari Silbey 还在一篇关于 StickNFind 的文章中写道：“当 Bluetooth 传感器无处不在时，会有更广泛的应用可能出现。”

这背后有着巨大的商机，无论是现有的跟踪 app，或者是即将来临的 task-launcher 功能。这个 launcher 可以在设备接近某些 Bluetooth 标签时，自动执行某些任务。比如基于地理位置的自动化可以让家居、汽车、生产线都变得更加智能。

曾经有一个开发者建议，可以开发一个 app，当主人回到家中时该 app 可以自动发送进门确认的 email。而另一位开发者则想避免司机在车上用手机打字的情况，因此他想开发一个锁键盘的功能——当车主携带智能手机进入车中时，汽车中自带的相应 Bluetooth 标签，即会激活锁键盘的功能。

上述交互场景的实现，将把我们带入一个可以远程控制设备，并且可以按照一定设置随时被触发的世界。最重要的是，Bluetooth 与 RFID（射频识别技术）、NFC 都不同，Bluetooth 已经普遍存在了，它更为广泛的普及以及更深一层的传感支持，只需再等待一些时日，但 NFC 却至今都还没有被足够多的硬件设备所支持。



注意

12.2.4 的内容引用自：

<http://gigaom.com/2013/07/29/bluetooth-le-android-and-why-the-smart-tag-revolutions-is-up-on-us/>

12.3 低功耗蓝牙协议栈详解

从 Android 4.2 版本开始，Google 便更换了 Android 的蓝牙协议栈，从 Bluez 换成 BlueDroid。从 Android 4.3 版本开始，提供了对蓝牙 4.0 BLE 的支持。在本节的内容中，将详细讲解 Android 系统中的蓝牙 4.0 BLE 的基本知识，为读者学习本书后面的知识打下基础。

12.3.1 低功耗蓝牙协议栈基础

为了确保 Android 系统可以更好地支持蓝牙 4.0 BLE，Broadcom 公司特意推出了适应于 Android 平台的开源低功耗蓝牙协议栈 BlueDroid，其开发文档和 API 是开源代码，保存在如

下地址中。

<https://github.com/briandbl/framework>

在上述开源代码中，低功耗蓝牙 API 支持 Android 平台上的低功耗蓝牙通信功能。通过使用 BlueDroid 协议栈，Android 应用程序可以枚举、发现并访问低功耗蓝牙的外部设备，并且实现了低功耗蓝牙规范。

从 Android 4.2 版本开始，低功耗蓝牙模块的整体结构如图 12-1 所示。

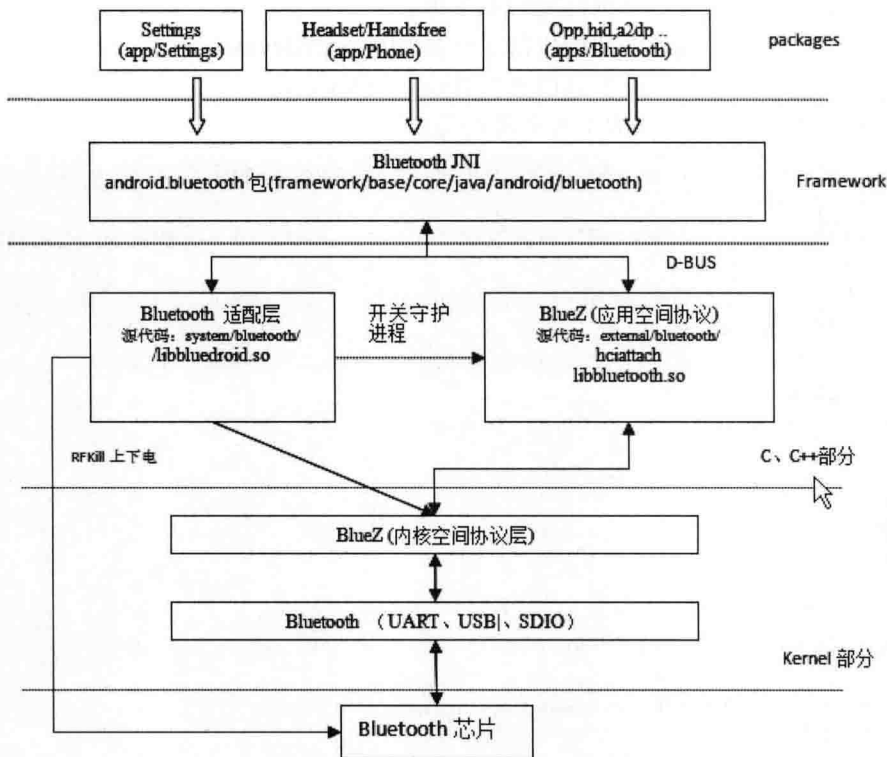


图 12-1 低功耗蓝牙模块的整体结构

注意

虽然从 Android 4.2 版本开始，JNI 部分的代码在 packages 层中实现。但是为了便于读者从视觉上更加容易接受，所以将 JNI 部分绘制在了 Framework 层中。

12.3.2 低功耗蓝牙 API 详解

Broadcom 公司推出的低功耗蓝牙协议栈 BlueDroid 的开发文档和 API 是开源代码，保存在如下地址中。

<https://github.com/briandbl/framework>

在接下来的内容中，将详细讲解主要 API 的基本功能和具体原理。

(1) 本地蓝牙适配器设备

本地蓝牙适配器设备功能不是由 Broadcom 公司提供的, 而是由 Android SDK 提供的, 源码位于如下目录中。

framework/base/core/java/android.bluetooth/BluetoothAdapter.java

文件 BluetoothAdapter.java 实现了所有蓝牙交互的入口。通过使用类 BluetoothAdapter 可以实现如下功能。

- 发现其他的蓝牙设备, 查询匹配的设备集。
- 使用一个已知蓝牙地址来初始化蓝牙设备 BluetoothDevice。
- 创建一个能够监听其他设备通信的类 BluetoothSocket。

文件 BluetoothAdapter.java 的主要实现代码如下。

```
public static synchronized BluetoothAdapter getDefaultAdapter() {
    if (sAdapter == null) {
        IBinder b = ServiceManager.getService(BLUETOOTH_MANAGER_SERVICE);
        if (b != null) {
            IBluetoothManager managerService = IBluetoothManager.Stub.
                asInterface(b);
            sAdapter = new BluetoothAdapter(managerService);
        } else {
            Log.e(TAG, "Bluetooth binder is null");
        }
    }
    return sAdapter;
}

/**
 * Use {@link #getDefaultAdapter} to get the BluetoothAdapter instance.
 */
BluetoothAdapter(IBluetoothManager managerService) {

    if (managerService == null) {
        throw new IllegalArgumentException("bluetooth manager service is null");
    }
    try {
        mService = managerService.registerAdapter(mManagerCallback);
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    mManagerService = managerService;
    mLeScanClients = new HashMap<LeScanCallback, GattCallbackWrapper>();
}

public BluetoothDevice getRemoteDevice(byte[] address) {
    if (address == null || address.length != 6) {
        throw new IllegalArgumentException("Bluetooth address must have 6 bytes");
    }
    return new BluetoothDevice(String.format("%02X:%02X:%02X:%02X:%02X:%02X",
        address[0], address[1], address[2], address[3], address[4], address[5]));
}

public int getState() {
    try {
```

```

        synchronized(mManagerCallback) {
            if (mService != null)
            {
                int state= mService.getState();
                if (VDBG) Log.d(TAG, "" + hashCode() + ": getState(). Returning " + state);
                return state;
            }
            // TODO(BT) there might be a small gap during STATE_TURNING_ON that
            //          mService is null, handle that case
        }
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    if (DBG) Log.d(TAG, "" + hashCode() + ": getState() : mService = null.
    Returning STATE_OFF");
    return STATE_OFF;
}

public String getAddress() {
    try {
        return mManagerService.getAddress();
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    return null;
}

public String getName() {
    try {
        return mManagerService.getName();
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    return null;
}

public int getScanMode() {
    if (getState() != STATE_ON) return SCAN_MODE_NONE;
    try {
        synchronized(mManagerCallback) {
            if (mService != null) return mService.getScanMode();
        }
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    return SCAN_MODE_NONE;
}

*/

public boolean setScanMode(int mode, int duration) {
    if (getState() != STATE_ON) return false;
    try {
        synchronized(mManagerCallback) {
            if (mService != null) return mService.setScanMode(mode, duration);
        }
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    return false;
}

/** @hide */
public boolean setScanMode(int mode) {

```



```
if (getState() != STATE_ON) return false;
/* getDiscoverableTimeout() to use the latest from NV than use 0 */
return setScanMode(mode, getDiscoverableTimeout());
}

/** @hide */
public int getDiscoverableTimeout() {
    if (getState() != STATE_ON) return -1;
    try {
        synchronized(mManagerCallback) {
            if (mService != null) return mService.getDiscoverableTimeout();
        }
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    return -1;
}

/** @hide */
public void setDiscoverableTimeout(int timeout) {
    if (getState() != STATE_ON) return;
    try {
        synchronized(mManagerCallback) {
            if (mService != null) mService.setDiscoverableTimeout(timeout);
        }
    } catch (RemoteException e) {Log.e(TAG, "", e);}
}

public boolean startDiscovery() {
    if (getState() != STATE_ON) return false;
    try {
        synchronized(mManagerCallback) {
            if (mService != null) return mService.startDiscovery();
        }
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    return false;
}

public boolean cancelDiscovery() {
    if (getState() != STATE_ON) return false;
    try {
        synchronized(mManagerCallback) {
            if (mService != null) return mService.cancelDiscovery();
        }
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    return false;
}

public boolean isDiscovering() {
    if (getState() != STATE_ON) return false;
    try {
        synchronized(mManagerCallback) {
            if (mService != null) return mService.isDiscovering();
        }
    } catch (RemoteException e) {Log.e(TAG, "", e);}
}
```

```

        return false;
    }
    public Set<BluetoothDevice> getBondedDevices() {
        if (getState() != STATE_ON) {
            return toDeviceSet(new BluetoothDevice[0]);
        }
        try {
            synchronized(mManagerCallback) {
                if (mService != null) return toDeviceSet(mService.getBondedDevices());
            }
            return toDeviceSet(new BluetoothDevice[0]);
        } catch (RemoteException e) {Log.e(TAG, "", e);}
        return null;
    }
    public int getConnectionState() {
        if (getState() != STATE_ON) return BluetoothAdapter.STATE_DISCONNECTED;
        try {
            synchronized(mManagerCallback) {
                if (mService != null) return mService.getAdapterConnectionState();
            }
        } catch (RemoteException e) {Log.e(TAG, "getConnectionState:", e);}
        return BluetoothAdapter.STATE_DISCONNECTED;
    }
    public int getProfileConnectionState(int profile) {
        if (getState() != STATE_ON) return BluetoothProfile.STATE_DISCONNECTED;
        try {
            synchronized(mManagerCallback) {
                if (mService != null) return mService.getProfileConnectionState(profile);
            }
        } catch (RemoteException e) {
            Log.e(TAG, "getProfileConnectionState:", e);
        }
        return BluetoothProfile.STATE_DISCONNECTED;
    }
    public BluetoothServerSocket listenUsingRfcommOn(int channel) throws IOException {
        BluetoothServerSocket socket = new BluetoothServerSocket(
            BluetoothSocket.TYPE_RFCOMM, true, true, channel);
        int errno = socket.mSocket.bindListen();
        if (errno != 0) {
            //TODO(BT): Throw the same exception error code
            // that the previous code was using.
            //socket.mSocket.throwErrnoNative(errno);
            throw new IOException("Error: " + errno);
        }
        return socket;
    }
    public BluetoothServerSocket listenUsingRfcommWithServiceRecord(String name,
        UUID uuid)
        throws IOException {
        return createNewRfcommSocketAndRecord(name, uuid, true, true);
    }

```

```

}
private BluetoothServerSocket createNewRfcommSocketAndRecord(String name,
    UUID uuid,
        boolean auth, boolean encrypt) throws IOException {
    BluetoothServerSocket socket;
    socket = new BluetoothServerSocket(BluetoothSocket.TYPE_RFCOMM, auth,
        encrypt, new ParcelUuid(uuid));
    socket.setServiceName(name);
    int errno = socket.mSocket.bindListen();
    if (errno != 0) {
        //TODO(BT): Throw the same exception error code
        // that the previous code was using.
        //socket.mSocket.throwErrnoNative(errno);
        throw new IOException("Error: " + errno);
    }
    return socket;
}

```

(2) 请求远程蓝牙设备

请求远程蓝牙设备功能也不是由 Broadcom 公司提供的，而是由 Android SDK 提供的，源码位于如下目录中。

framework/base/core/java/android.bluetooth/BluetoothDevice.java

文件 BluetoothDevice.java 代表一个远程蓝牙设备，可以支持 BLE 低功耗设备、BR/EDR 设备或 Dual-mode 类型的设备。通过使用类 BluetoothDevice 可以实现如下功能。

- 请求获取远程蓝牙设备的连接。
- 查询获取远程蓝牙设备的名称、地址、类和链接状态。

文件 BluetoothDevice.java 的主要实现代码如下。

```

static IBluetooth getService() {
    synchronized (BluetoothDevice.class) {
        if (sService == null) {
            BluetoothAdapter adapter = BluetoothAdapter.getDefaultAdapter();
            sService = adapter.getBluetoothService(mStateChangeCallback);
        }
    }
    return sService;
}

static IBluetoothManagerCallback mStateChangeCallback = new
IBluetoothManagerCallback.Stub() {

    public void onBluetoothServiceUp(IBluetooth bluetoothService)
        throws RemoteException {
        synchronized (BluetoothDevice.class) {
            sService = bluetoothService;
        }
    }

    public void onBluetoothServiceDown()

```

```

        throws RemoteException {
            synchronized (BluetoothDevice.class) {
                sService = null;
            }
        }
    };

    /*package*/ BluetoothDevice(String address) {
        getService(); // ensures sService is initialized
        if (!BluetoothAdapter.checkBluetoothAddress(address)) {
            throw new IllegalArgumentException(address + " is not a valid
            Bluetooth address");
        }

        mAddress = address;
    }

    @Override
    public boolean equals(Object o) {
        if (o instanceof BluetoothDevice) {
            return mAddress.equals(((BluetoothDevice)o).getAddress());
        }
        return false;
    }

    @Override
    public int hashCode() {
        return mAddress.hashCode();
    }

    public static final Parcelable.Creator<BluetoothDevice> CREATOR =
        new Parcelable.Creator<BluetoothDevice>() {
            public BluetoothDevice createFromParcel(Parcel in) {
                return new BluetoothDevice(in.readString());
            }
            public BluetoothDevice[] newArray(int size) {
                return new BluetoothDevice[size];
            }
        };

    public void writeToParcel(Parcel out, int flags) {
        out.writeString(mAddress);
    }

    public String getAddress() {
        if (DBG) Log.d(TAG, "mAddress: " + mAddress);
        return mAddress;
    }

    public String getName() {
        if (sService == null) {
            Log.e(TAG, "BT not enabled. Cannot get Remote Device name");
            return null;
        }
    }

```



```
try {
    return sService.getRemoteName(this);
} catch (RemoteException e) {Log.e(TAG, "", e);}
return null;
}

public int getType() {
    if (sService == null) {
        Log.e(TAG, "BT not enabled. Cannot get Remote Device type");
        return DEVICE_TYPE_UNKNOWN;
    }
    try {
        return sService.getRemoteType(this);
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    return DEVICE_TYPE_UNKNOWN;
}

public String getAlias() {
    if (sService == null) {
        Log.e(TAG, "BT not enabled. Cannot get Remote Device Alias");
        return null;
    }
    try {
        return sService.getRemoteAlias(this);
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    return null;
}

public boolean setAlias(String alias) {
    if (sService == null) {
        Log.e(TAG, "BT not enabled. Cannot set Remote Device name");
        return false;
    }
    try {
        return sService.setRemoteAlias(this, alias);
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    return false;
}

public String getAliasName() {
    String name = getAlias();
    if (name == null) {
        name = getName();
    }
    return name;
}

public boolean createBond() {
    if (sService == null) {
        Log.e(TAG, "BT not enabled. Cannot create bond to Remote Device");
        return false;
    }
    try {
        return sService.createBond(this);
    } catch (RemoteException e) {Log.e(TAG, "", e);}
```



```

        return false;
    }
    public boolean createBondOutOfBand(byte[] hash, byte[] randomizer) {
        //TODO(BT)
        /*
        try {
            return sService.createBondOutOfBand(this, hash, randomizer);
        } catch (RemoteException e) {Log.e(TAG, "", e);}*/
        return false;
    }
    public boolean cancelBondProcess() {
        if (sService == null) {
            Log.e(TAG, "BT not enabled. Cannot cancel Remote Device bond");
            return false;
        }
        try {
            return sService.cancelBondProcess(this);
        } catch (RemoteException e) {Log.e(TAG, "", e);}
        return false;
    }
    public boolean removeBond() {
        if (sService == null) {
            Log.e(TAG, "BT not enabled. Cannot remove Remote Device bond");
            return false;
        }
        try {
            return sService.removeBond(this);
        } catch (RemoteException e) {Log.e(TAG, "", e);}
        return false;
    }
    public int getBondState() {
        if (sService == null) {
            Log.e(TAG, "BT not enabled. Cannot get bond state");
            return BOND_NONE;
        }
        try {
            return sService.getBondState(this);
        } catch (RemoteException e) {Log.e(TAG, "", e);}
        catch (NullPointerException npe) {
            // Handle case where bluetooth service proxy
            // is already null.
            Log.e(TAG, "NullPointerException for getBondState() of device (" +
                getAddress() + ")", npe);
        }
        return BOND_NONE;
    }
    public BluetoothClass getBluetoothClass() {
        if (sService == null) {
            Log.e(TAG, "BT not enabled. Cannot get Bluetooth Class");
            return null;
        }
    }

```

```

    }
    try {
        int classInt = sService.getRemoteClass(this);
        if (classInt == BluetoothClass.ERROR) return null;
        return new BluetoothClass(classInt);
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    return null;
}

public boolean fetchUuidsWithSdp() {
    try {
        return sService.fetchRemoteUuids(this);
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    return false;
}

public boolean setPin(byte[] pin) {
    if (sService == null) {
        Log.e(TAG, "BT not enabled. Cannot set Remote Device pin");
        return false;
    }
    try {
        return sService.setPin(this, true, pin.length, pin);
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    return false;
}

public boolean setPasskey(int passkey) {
    //TODO(BT)
    /*
    try {
        return sService.setPasskey(this, true, 4, passkey);
    } catch (RemoteException e) {Log.e(TAG, "", e);}*/
    return false;
}

public boolean setPairingConfirmation(boolean confirm) {
    if (sService == null) {
        Log.e(TAG, "BT not enabled. Cannot set pairing confirmation");
        return false;
    }
    try {
        return sService.setPairingConfirmation(this, confirm);
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    return false;
}
}

```

(3) 实现客户端的低功耗蓝牙规范

在 Broadcom 公司提供的源码中，文件 `BleClientProfile.java` 的功能是实现客户端的低功耗蓝牙规范。在应用中要想访问远程设备中的低功耗蓝牙规范，就必须继承于类 `BleClientProfile`，并且需要提供要访问规范的必需参数和服务标识。通过 `BleClientProfile` 的派生类可以发起一个远程设备的连接，并且一个 `BleClientProfile` 类可能会包含多个 `BleClientService` 对象的实例。文件 `BleClientProfile.java` 的具体实现代码如下。

```
//下面是构造方法，功能是给当前规范的 UUID 和客户端应用上下文创建一个 BleClientProfile
public BleClientProfile(Context context, BleGattID profileUuid)
{
    Log.d(TAG, "new profile" + profileUuid.toString());

    this.mContext = context;
    this.mAppUuid = profileUuid;

    this.mConnectedDevices = new ArrayList<BluetoothDevice>();
    this.mConnectingDevices = new ArrayList<BluetoothDevice>();
    this.mDisconnectingDevices = new ArrayList<BluetoothDevice>();

    this.mClientIDToDeviceMap = new HashMap<Integer, BluetoothDevice>();
    this.mDeviceToClientIDMap = new HashMap<BluetoothDevice, Integer>();

    this.mCallback = new BleClientCallback();
    this.mSvcConn = new GattServiceConnection(context);
}

/**
 * 初始化 BleClientProfile 对象
 */
public void init(ArrayList<BleClientService> requiredServices,
                ArrayList<BleClientService> optionalServices)
{
    Log.d(TAG, "init (" + this.mAppUuid + ")");

    this.mRequiredServices = requiredServices;
    this.mOptionalServices = optionalServices;

    IBinder b = ServiceManager.getService(BleConstants.BLUETOOTH_LE_
    SERVICE);
    if (b == null) {
        throw new RuntimeException("Bluetooth Low Energy service not available");
    }
    this.mSvcConn.onServiceConnected(null, b);
}

/**
 * 清除和此规范有关的资源
 */
public synchronized void finish()
{
    if (this.mSvcConn != null) {
        this.mContext.unbindService(this.mSvcConn);
        this.mSvcConn = null;
    }
}

@Override
```

```
/**
 * 返回此规范是否已经成功注册到蓝牙协议栈中
 * @see {@link #registerProfile()}
 */
public boolean isProfileRegistered()
{
    Log.d(TAG, "isProfileRegistered (" + this.mAppUuid + ")");
    return this.mClientIf != BleConstants.GATT_SERVICE_PRIMARY;
}

/**
 * 注册规范到蓝牙协议栈
 */
public int registerProfile()
{
    int ret = BleConstants.GATT_SUCCESS;
    Log.d(TAG, "registerProfile (" + this.mAppUuid + ")");

    if (this.mClientIf == BleConstants.GATT_SERVICE_PRIMARY)
    {
        try
        {
            this.mService.registerApp(this.mAppUuid, this.mCallback);
        } catch (RemoteException e) {
            Log.e(TAG, e.toString());
            ret = BleConstants.SERVICE_UNAVAILABLE;
        }
    }

    return ret;
}

/**
 * 注销蓝牙协议栈中的规范
 */
public void deregisterProfile()
{
    Log.d(TAG, "deregisterProfile (" + this.mAppUuid + ")");

    if (this.mClientIf != BleConstants.GATT_SERVICE_PRIMARY)
        try {
            this.mService.unregisterApp(this.mClientIf);
        } catch (RemoteException e) {
            Log.e(TAG, "deregisterProfile() - " + e.toString());
        }
    }

}

/**
 * 设置一个活跃连接设备的加密等级
```

```
*/
public void setEncryption(BluetoothDevice device, byte action)
{
    try
    {
        this.mService.setEncryption(device.getAddress(), action);
    } catch (RemoteException e) {
        Log.e(TAG, e.toString());
    }
}

/**
 * 当请求后台连接时，定义本地设备扫描远程低功耗设备的强度
 */
public void setScanParameters(int scanInterval, int scanWindow)
{
    try
    {
        this.mService.setScanParameters(scanInterval, scanWindow);
    } catch (RemoteException e) {
        Log.e(TAG, e.toString());
    }
}

/**
 * 建立一个到远程设备的 GATT 连接
 */
public int connect(BluetoothDevice device)
{
    Log.d(TAG, "connect (" + this.mAppUuid + ") " + device.getAddress());

    int ret = BleConstants.GATT_SUCCESS;

    synchronized (this.mConnectingDevices) {
        this.mConnectingDevices.add(device);
    }

    synchronized (this.mDisconnectingDevices) {
        this.mDisconnectingDevices.remove(device);
    }

    try
    {
        {
            this.mService.open(this.mClientIf, device.getAddress(), true);
        } catch (RemoteException e) {
            Log.e(TAG, e.toString());
            ret = BleConstants.GATT_ERROR;
        }
    }

    return ret;
}
```



```
/**
 * 准备一个到远程蓝牙设备的后台连接
 */
public int connectBackground(BluetoothDevice device)
{
    Log.d(TAG,
        "connectBackground (" + this.mAppUuid + ") " + device.getAddress());

    int ret = BleConstants.GATT_SUCCESS;

    synchronized (this.mConnectingDevices) {
        this.mConnectingDevices.add(device);
    }

    synchronized (this.mDisconnectingDevices) {
        this.mDisconnectingDevices.remove(device);
    }

    try
    {
        this.mService.open(this.mClientIf, device.getAddress(), false);
    } catch (RemoteException e) {
        Log.e(TAG, e.toString());
        ret = BleConstants.GATT_ERROR;
    }

    return ret;
}

/**
 * 停止监听远程蓝牙设备试图发起的链接
 */
public int cancelBackgroundConnection(BluetoothDevice device)
{
    Log.d(TAG, "cancelBackgroundConnection (" + this.mAppUuid
        + ") - device " + device.getAddress());

    int ret = BleConstants.GATT_SUCCESS;
    try
    {
        this.mService.close(this.mClientIf, device.getAddress(), 0, false);
    } catch (RemoteException e) {
        Log.e(TAG, e.toString());
        ret = BleConstants.GATT_ERROR;
    }

    return ret;
}

/**
```

```

*断开一个到远程设备的 GATT 连接
*/
public int disconnect(BluetoothDevice device)
{
    Log.d(TAG,
        "disconnect (" + this.mAppUuid + ") - device " + device.
            getAddress());

    synchronized (this.mDisconnectingDevices) {
        this.mDisconnectingDevices.add(device);
    }

    int ret = BleConstants.GATT_SUCCESS;
    try
    {
        this.mService.close(this.mClientIf,
            device.getAddress(),
            ((Integer) this.mDeviceToClientIDMap.get(device)).intValue(),
            true);
    } catch (RemoteException e) {
        Log.e(TAG, e.toString());
        ret = BleConstants.GATT_ERROR;
    }
    return ret;
}

/**
 * 刷新当前客户端的规范
 */
public int refresh(BluetoothDevice device)
{
    Log.d(TAG,
        "refresh (" + this.mAppUuid + ") - address = " + device.
            getAddress());

    if (isDeviceDisconnecting(device)) {
        Log.d(TAG, "refresh (" + this.mAppUuid
            + ") - Device unavailable!");
        return BleConstants.GATT_ERROR;
    }

    this.mRequiredServices.get(BleConstants.GATT_SERVICE_PRIMARY).refresh
        (device);

    return BleConstants.GATT_SUCCESS;
}

/**
 * 刷新当前规范包含的特定服务
 */

```

```
public int refreshService(BluetoothDevice device, BleClientService service)
{
    Log.d(TAG, "refreshService (" + this.mAppUuid + ") address = s "
        + device.getAddress() + "service = " + service.getServiceId());

    return 0;
}

/**
 * 在已经连接的设备列表中查找指定蓝牙设备的地址
 */
public BluetoothDevice findConnectedDevice(String address)
{
    BluetoothDevice ret = null;
    synchronized (this.mConnectedDevices) {
        for (int i = 0; i != this.mConnectedDevices.size(); i++) {
            BluetoothDevice d = (BluetoothDevice) this.mConnectedDevices.
                get(i);
            if (address.equalsIgnoreCase(d.getAddress())) {
                ret = d;
                break;
            }
        }
    }
    return ret;
}

/**
 * 返回当前连接和等待连接中的所有远程设备集合
 */
public BluetoothDevice[] getPendingConnections()
{
    return (BluetoothDevice[]) this.mConnectingDevices.toArray(new
        BluetoothDevice[0]);
}

/**
 * 设置一个蓝牙设备地址，在等待连接设备列表中查找一个远程设备
 */
public BluetoothDevice findDeviceWaitingForConnection(String address)
{
    BluetoothDevice ret = null;
    synchronized (this.mConnectingDevices) {
        for (int i = 0; i < this.mConnectingDevices.size(); i++) {
            BluetoothDevice d = (BluetoothDevice) this.mConnectingDevices.
                get(i);
            if (address.equalsIgnoreCase(d.getAddress())) {
                ret = d;
                break;
            }
        }
    }
}
```

```

        }
    }
    return ret;
}

/**
 * @hide
 * @return
 */
IBluetoothGatt getGattService()
{
    return this.mService;
}

/**
 * @hide
 * @param d
 * @return
 */
int getConnIdForDevice(BluetoothDevice d)
{
    if (!this.mDeviceToClientIDMap.containsKey(d)) {
        return 65535;
    }

    return ((Integer) this.mDeviceToClientIDMap.get(d)).intValue();
}

void onServiceRefreshed(BleClientService s, BluetoothDevice device)
{
    int i = this.mRequiredServices.indexOf(s);
    if (i + 1 < this.mRequiredServices.size()) {
        Log.d(TAG, "Refreshing next service");
        ((BleClientService) this.mRequiredServices.get(i + 1)).refresh(
            device);
    } else {
        onRefreshed(device);
    }
}

public void onInitialized(boolean success)
{
    Log.d(TAG, "onInitialized");
    if (success)
        registerProfile();
}

public void onServiceConnected(ComponentName name, IBinder service)
{
    Log.d(TAG, "Connected to GattService!");

    if (service != null)

```

```
try {
    BleClientProfile.this.mService = IBluetoothGatt.Stub.
        asInterface(service);

    for (int i = 0; i < BleClientProfile.this.mRequiredServices.
        size(); i++) {
        BleClientProfile.this.mRequiredServices.get(i)
            .setProfile(BleClientProfile.this);
    }

    if (BleClientProfile.this.mOptionalServices != null) {
        for (int i = 0; i < BleClientProfile.this.mOptionalServices.
            size(); i++) {
            BleClientProfile.this.mOptionalServices
                .get(i).setProfile(BleClientProfile.this);
        }
    }

    BleClientProfile.this.onInitialized(true);
} catch (Throwable t) {
    Log.e(TAG, "Unable to get Binder to GattService", t);
    BleClientProfile.this.onInitialized(false);
}

}

public void onServiceDisconnected(ComponentName name)
{
    Log.d(TAG, "Disconnected from GattService!");
}

}

class BleClientCallback extends IBleClientCallback.Stub
{
    BleClientCallback()
    {
    }

    public void onAppRegistered(byte status, byte client_if)
    {
        Log.d(TAG, "BleClientCallback::onAppRegistered ("
            + BleClientProfile.this.mAppUuid + ") status = " + status +
            " client_if = "
            + client_if);

        BleClientProfile.this.mClientIf = client_if;
        BleClientProfile.this.onProfileRegistered();
    }

    public void onAppDeregistered(byte client_if) {
```



```

Log.d(TAG, "BleClientCallback::onAppDeregistered ("
    + BleClientProfile.this.mAppUuid + ") client_if = " + client_if);

BleClientProfile.this.mClientIf = BleConstants.GATT_SERVICE_PRIMARY;
BleClientProfile.this.onProfileDeregistered();
}

public void onConnected(String deviceAddress, int connID) {
    Log.d(TAG, "BleClientCallback::OnConnected ("
        + BleClientProfile.this.mAppUuid + ") " + deviceAddress +
        "connID = " + connID);

    BluetoothDevice d = BleClientProfile.this
        .findDeviceWaitingForConnection(deviceAddress);

    if (null == d)
    {
        d = BluetoothAdapter.getDefaultAdapter().getRemoteDevice
            (deviceAddress);
        synchronized (BleClientProfile.this.mConnectedDevices) {
            BleClientProfile.this.mConnectedDevices.add(d);
        }

        synchronized (BleClientProfile.this.mConnectingDevices) {
            BleClientProfile.this.mConnectingDevices.remove(d);
        }

        synchronized (BleClientProfile.this.mDisconnectingDevices) {
            BleClientProfile.this.mDisconnectingDevices.remove(d);
        }
    }

    if (d.getBondState() == BluetoothDevice.BOND_BONDED) {
        Log.d(TAG,
            "onConnected device is bonded start encrypt the link");
        BleClientProfile.this.setEncryption(d, (byte) 1);
    }
    BleClientProfile.this.mClientIDToDeviceMap.put(new Integer
        (connID), d);
    BleClientProfile.this.mDeviceToClientIDMap.put(d, new Integer
        (connID));

    BleClientProfile.this.mPeerServices.clear();
    try
    {
        BleClientProfile.this.mService.searchService(connID, null);
    } catch (RemoteException e) {
        Log.d(TAG, "Error calling searchService " + e.toString());
    }
}

```

```
public void onDisconnected(int connID, String deviceAddress)
{
    Log.d(TAG, "BleClientCallback::onDisconnected ("
        + BleClientProfile.this.mAppUuid + ") connID = " + connID);

    BleClientProfile.this
        .onDeviceDisconnected((BluetoothDevice) BleClientProfile.this.
            mClientIDToDeviceMap
                .get(new Integer(connID)));

    BluetoothDevice d = (BluetoothDevice) BleClientProfile.this.
        mClientIDToDeviceMap
            .get(new Integer(connID));

    BleClientProfile.this.mDeviceToClientIDMap.remove(d);
    BleClientProfile.this.mClientIDToDeviceMap.remove(new Integer
        (connID));
    BleClientProfile.this.mConnectedDevices.remove(d);
    BleClientProfile.this.mDisconnectingDevices.remove(d);
}

public void onSearchResult(int connID, BluetoothGattID svcId) {
    Log.d(TAG, "BleClientCallback::onSearchResult ("
        + BleClientProfile.this.mAppUuid + ") connID = " + connID +
        " svcId: id = "
        + svcId.toString() + " inst id = " + svcId.getInstanceID());

    BleClientProfile.this.mPeerServices.add(BleApiHelper.gatt2BleID
        (svcId));
}

public void onSearchCompleted(int connID, int status) {
    Log.d(TAG, "BleClientCallback::onSearchCompleted ("
        + BleClientProfile.this.mAppUuid + ") connID = " + connID +
        "status = "
        + status);

    int nServicesFound = 0;

    if (BleClientProfile.this.mRequiredServices == null) {
        Log.d(TAG, "mRequiredServices is null");
        return;
    }

    if (BleClientProfile.this.mPeerServices == null) {
        Log.d(TAG, "mPeerServices is null");
        return;
    }
}
```

```

for (int i = 0; i < BleClientProfile.this.mRequiredServices.size();
i++) {
    for (int j = 0; j < BleClientProfile.this.mPeerServices.size();
j++) {
        if (((BleGattID) BleClientProfile.this.mPeerServices.get(j))
            .toString().equalsIgnoreCase(
                BleClientProfile.this.mRequiredServices
                    .get(i).getServiceId().toString())) {
            BleClientProfile.this.mRequiredServices.get(i)
                .setInstanceID(
                    BleClientProfile.this.mClientIDToDeviceMap.
                        get(new Integer(
                            connID)),
                    BleClientProfile.this.mPeerServices.get
                        (j).getInstanceID());
            nServicesFound++;
            break;
        }
    }
}

Log.d(TAG, "BleClientCallback::onSearchResult - found "
    + nServicesFound + " out of " + BleClientProfile.this.
        mRequiredServices.size()
    + " services needed for this profile");
BluetoothDevice device = (BluetoothDevice) BleClientProfile.this.
    mClientIDToDeviceMap
        .get(new Integer(connID));

if (device == null) {
    Log.d(TAG,
        "No bluetooth device in the device map for connid = " + connID);
}
else if (BleClientProfile.this.isDeviceDisconnecting(device)) {
    Log.d(TAG, "Device disconnecting...");
}
else if (nServicesFound == BleClientProfile.this.mRequiredServices.
    size()) {
    Log.d(TAG,
        "the num of Srvs found match the required srv size ");
    BleClientProfile.this.onDeviceConnected(device);
} else {
    Log.d(TAG,
        "the num of Srvs found DOES NOT match the required srv size ");
}
}
}
}
}

```

(4) 创建一个代表客户端角色设备上的低功耗蓝牙服务派生类

在 Broadcom 公司提供的源码中, 文件 `BleClientService.java` 的功能是定义了一个派生类, 此派生类代表了客户端角色设备上的低功耗蓝牙服务。通过这个派生类可以允许应用程序读写低功耗蓝牙服务的特征, 并在特征改变时注册通知。文件 `BleClientService.java` 的主要实现代码如下。

```
//定义代表客户端的低功耗服务
public abstract class BleClientService
{
    private static String TAG = "BleClientService";

    private BleClientProfile mProfile = null;
    private BleGattID mServiceId = null;
    private HashMap<BluetoothDevice, ArrayList<ServiceData>> mdeviceToDataMap =
        new HashMap<BluetoothDevice, ArrayList<ServiceData>>();
    private BleCharacteristicDataCallback mCallback =
        new BleCharacteristicDataCallback();
    private boolean mReadDescriptors = true;

    /**
     * 创建一个新的低功耗蓝牙服务的 UUID
     *
     * @param serviceId
     */
    public BleClientService(BleGattID serviceId)
    {
        mServiceId = serviceId;
        if (mServiceId.getServiceType() == BleConstants.GATT_UNDEFINED)
            mServiceId.setServiceType(BleConstants.GATT_SERVICE_PRIMARY);
    }

    /**
     * 返回服务的 UUID
     */
    public BleGattID getServiceId()
    {
        return mServiceId;
    }

    /**
     * 写操作远程设备上的一个特性
     */
    public int writeCharacteristic(BluetoothDevice remoteDevice, int instanceId,
        BleCharacteristic characteristic)
    {
        Log.d(TAG, "writeCharacteristic");

        int ret = BleConstants.GATT_SUCCESS;
        int connID = BleConstants.GATT_INVALID_CONN_ID;
```

```

if ((connID = mProfile.getConnIdForDevice(remoteDevice)) == BleConstants.
GATT_INVALID_CONN_ID) {
    return BleConstants.GATT_INVALID_CONN_ID;
}
ServiceData s = getServiceData(remoteDevice, instanceId);

if (s == null) {
    return ret;
}
s.writeIndex = s.characteristics.indexOf(characteristic);

if ((s.characteristics != null) && (s.writeIndex >= BleConstants.
GATT_SERVICE_PRIMARY)) {
    Log.d(TAG, "writeCharacteristic found characteristic in array:");
    Log.d(TAG,
        "Service = [instanceID = " + instanceId + " svcid = "
            + mServiceId.toString() + " serviceType = "
            + mServiceId.getServiceType());
    Log.d(TAG, "CharID = [instanceID = " + characteristic.getInstanceID()
        + " svcid = " + characteristic.getID().toString());
    BleGattID svcId = new BleGattID(instanceId, mServiceId.getUuid(),
        mServiceId.getServiceType());
    BleGattID cID = characteristic.getID();
    BluetoothGattCharID charID = new BluetoothGattCharID(svcId, cID);
    try
    {
        if (characteristic.isDirty()) {
            if (characteristic.getWriteType() == BleConstants.GATT_
                SUCCESS)
                characteristic.setWriteType(2);
            characteristic.setDirty(false);
            mProfile.getGattService().writeCharValue(connID, charID,
                characteristic.getWriteType(), characteristic.
                getAuthReq(),
                characteristic.getValue());
        }
        else if (!characteristic.getDirtyDescQueue().isEmpty()) {
            ArrayList<BleDescriptor> descList =
                characteristic.getDirtyDescQueue();
            BleDescriptor descObj = descList.get(0);

            Log.d(TAG, "writeCharacteristic - descriptor = "
                + descObj.getID().toString());
            if (descObj.isDirty()) {
                BluetoothGattCharDescrID descID = new BluetoothGattCharDescrID(
                    svcId, cID, descObj.getID());
                descObj.setDirty(false);
                mProfile.getGattService().writeCharDescrValue(connID,
                    descID, descObj.getWriteType(), descObj.getAuthReq(),
                    descObj.getValue());
            }
        }
    }
}

```



```

    }

    }
    else
    {
        onWriteCharacteristicComplete(0, remoteDevice, characteristic);
    }
} catch (RemoteException e) {
    ret = BleConstants.GATT_ERROR;
}
} else {
    onWriteCharacteristicComplete(0, remoteDevice, characteristic);
}
return ret;
}

/**
 *检索服务包含的所有特征
 */
public ArrayList<BleCharacteristic> getAllCharacteristics(BluetoothDevice
remoteDevice)
{
    Log.d(TAG, "getAllCharacteristics");

    ServiceData s = getServiceData(remoteDevice, mServiceId.getInstanceID());
    if (null != s) {
        return s.characteristics;
    }
    return null;
}

/**
 *返回一个基于它的 ID 的服务特性
 */
public BleCharacteristic getCharacteristic(BluetoothDevice remoteDevice,
BleGattID characteristicID)
{
    Log.d(TAG, "getCharacteristic charID = [" + characteristicID.toString()
        + "] instance ID = [" + characteristicID.getInstanceID() + "]");
    ServiceData s = getServiceData(remoteDevice, mServiceId.getInstanceID());
    if (s == null) {
        Log.d(TAG, "getCharacteristic - Service data not found");
        return null;
    }
    for (int i = 0; i < s.characteristics.size(); i++) {
        BleCharacteristic c = s.characteristics.get(i);
        if (c != null) {
            if (c.getID() != null) {
                if ((c.getID().toString().equals(characteristicID.toString()))
                    && (c.getInstanceID() == characteristicID.

```

```

        getInstanceID()))
    {
        return c;
    }
}
else
    Log.d(TAG, "Error: Characteristic ID is null");
}
else {
    Log.d(TAG, "Error: Cannot retrieve characteristic");
}
}
return null;
}

/**
 * 返回所有服务实例的 IDs 列表
 */
public int[] getAllServiceInstanceIds(BluetoothDevice remoteDevice)
{
    Log.d(TAG, "getAllServiceInstanceIds");
    ArrayList<ServiceData> s = mdeviceToDataMap.get(remoteDevice);
    if (s != null) {
        int[] instanceIds = new int[s.size()];

        for (int i = 0; i < s.size(); i++) {
            instanceIds[i] = s.get(i).instanceID;
        }

        return instanceIds;
    }

    return null;
}

/**
 * 刷新远程服务的所有势力的所有特性
 */
public void refresh(BluetoothDevice remoteDevice)
{
    Log.d(TAG, "Refresh (" + mServiceId.toString() + ")");

    ArrayList<ServiceData> s = mdeviceToDataMap.get(remoteDevice);
    if (s != null) {
        ServiceData sd = s.get(0);
        Log.e(TAG,
            "refresh() - Service data found, reading first characteristic...
            (serviceType = "
                + sd.serviceType + ")");
        readFirstCharacteristic(remoteDevice, new BleGattID(sd.instanceID,

```

```

        getServiceId().getUuid(), sd.serviceType));
    } else {
        Log.e(TAG, "refresh() - Service data not found");
    }
}

/**
 * 从远程设备上读取指定的特性
 */
 * @see {@link #onReadCharacteristicComplete(BluetoothDevice,
 *      BleCharacteristic)}
 */
public int readCharacteristic(BluetoothDevice remoteDevice,
    BleCharacteristic characteristic)
{
    int ret = BleConstants.GATT_SUCCESS;
    int connID = BleConstants.GATT_INVALID_CONN_ID;
    Log.d(TAG,
        "readCharacteristic - svc UUID = " + getServiceId().getUuid().
        toString()
        + ", characteristic = " + characteristic.getID());

    BluetoothGattCharID charID = new BluetoothGattCharID(new BleGattID(
        characteristic.getInstanceID(), getServiceId().getUuid(),
        getServiceId().
            .getServiceType()), characteristic.getID());

    if ((connID = mProfile.getConnIdForDevice(remoteDevice)) != BleConstants.
        GATT_INVALID_CONN_ID)
        readCharacteristicValue(connID, charID, characteristic.getAuthReq());
    else {
        ret = BleConstants.GATT_INVALID_CONN_ID;
    }
    return ret;
}

/**
 * 设置一个远程设备特性的写操作已经完成的函数
 */
public void onWriteCharacteristicComplete(int status, BluetoothDevice
remoteDevice,
    BleCharacteristic characteristic)
{
    Log.d(TAG, "onWriteCharacteristicComplete 1 status=" + status);
    if (status == BleConstants.GATT_INSUF_AUTHENTICATION) {
        Log.d(TAG,
            "onWriteCharacteristicComplete rcv GATT_INSUF_AUTHENTICATION
            issue createBond");
        if (remoteDevice.createBond())
            Log.d(TAG, "onWriteCharacteristicComplete createBond request

```

```

        Accepted");
    else {
        Log.e(TAG, "onWriteCharacteristicComplete createBond request
        FAILED");
    }
}
else if (status == BleConstants.GATT_INSUF_ENCRYPTION) {
    Log.d(TAG,
        "onWriteCharacteristicComplete rcv GATT_INSUF_ENCRYPTION
        check link can be encrypt or not");
    if (remoteDevice.getBondState() == BluetoothDevice.BOND_BONDED) {
        Log.d(TAG,
            "device bonded start to encrypt the link. !!!! This case
            should not happen !!!!");
    } else {
        Log.d(TAG, "device is Not bonded start to pair");
        remoteDevice.createBond();
    }
}
}

/**
 * 设置一个远程特性已经改变的回调函数
 */
public void onCharacteristicChanged(BluetoothDevice remoteDevice,
    BleCharacteristic characteristic)
{
    Log.d(TAG, "onCharacteristicChanged");
}

/**
 * 设置服务已经完成一次刷新的回调函数
 */
public void onRefreshComplete(BluetoothDevice remoteDevice)
{
    Log.d(TAG, "onRefreshComplete");
}

/**
 * 当服务需要给一个个顶的特性设置授权请求时触发的回调函数
 */
public void onSetCharacteristicAuthRequirement(BluetoothDevice remoteDevice,
    BleCharacteristic characteristic, int instanceID)
{
    Log.d(TAG, "onSetCharacteristicAuthRequirement");
}

/**
 * 当一个设置的特性被更新，并且读取其值和描述符时会被调用
 */

```

```

public void onReadCharacteristicComplete(BluetoothDevice remoteDevice,
    BleCharacteristic characteristic)
{
    Log.d(TAG, "onReadCharacteristicComplete");
}

public void onReadCharacteristicComplete(int status, BluetoothDevice
remoteDevice,
    BleCharacteristic characteristic)
{
    Log.d(TAG, "onReadCharacteristicComplete status=" + status);
    if (status == BleConstants.GATT_INSUF_AUTHENTICATION) {
        Log.d(TAG,
            "onReadCharacteristicComplete rcv GATT_INSUF_AUTHENTICATION
            issue createBond");
        remoteDevice.createBond();
        return;
    }
    if (status != BleConstants.GATT_INSUF_ENCRYPTION) {
        return;
    }

    Log.d(TAG,
        "onReadCharacteristicComplete rcv GATT_INSUF_ENCRYPTION check
        link can be encrypt or not");
    if (remoteDevice.getBondState() == BluetoothDevice.BOND_BONDED) {
        Log.d(TAG,
            "device bonded start to encrypt the link. !!!! This case
            should not happen !!!!");
    } else {
        Log.d(TAG, "device is Not bonded start to pair");
        remoteDevice.createBond();
    }
}

/**
 * 注册此服务在服务器上的通知
 */
public int registerForNotification(BluetoothDevice remoteDevice, int
instanceID,
    BleGattID characteristicID)
{
    int ret = BleConstants.GATT_SUCCESS;
    Log.d(TAG, "registerForNotification address: " + remoteDevice.
    getAddress());
    try {
        BleGattID svcId = new BleGattID(instanceID, getServiceId().
        getUuid(),
            getServiceId().getServiceType());
    }
}

```



```

BluetoothGattCharID charId = new BluetoothGattCharID(svcId,
characteristicID);

    mProfile.getGattService().registerForNotifications(
        mProfile.getClientIf(), remoteDevice.getAddress(), charId);
} catch (RemoteException e) {
    ret = BleConstants.GATT_ERROR;
}
return ret;
}

/**
 *取消从服务器的通知服务
 */
public int unregisterNotification(BluetoothDevice remoteDevice, int instanceID,
    BleGattID characteristicID)
{
    int ret = BleConstants.GATT_SUCCESS;
    Log.d(TAG, "unregisterNotification address: " + remoteDevice.
        getAddress());
    try {
        BleGattID svcId = new BleGattID(instanceID, getServiceId().
            getUuid(),
            getServiceId().getServiceType());
        BluetoothGattCharID charId = new BluetoothGattCharID(svcId,
            characteristicID);

        mProfile.getGattService().deregisterForNotifications(
            mProfile.getClientIf(), remoteDevice.getAddress(), charId);
    } catch (RemoteException e) {
        ret = BleConstants.GATT_ERROR;
    }

    return ret;
}

void setInstanceId(BluetoothDevice remoteDevice, int instanceId)
{
    Log.d(TAG, "setInstanceId address = " + remoteDevice.getAddress());

    ServiceData sd = getServiceData(remoteDevice, instanceId);
    mServiceId.setInstanceId(instanceId);
    if (null == sd) {
        Log.d(TAG, "setInstanceId setting instance id (" + instanceId + ")");

        sd = new ServiceData();
        sd.instanceID = mServiceId.getInstanceID();
        sd.serviceType = mServiceId.getServiceType();
        ArrayList<ServiceData> s = mdeviceToDataMap.get(remoteDevice);
        if (null == s) {

```

```

        s = new ArrayList<ServiceData>();
    }
    s.add(sd);
    mdeviceToDataMap.put(remoteDevice, s);
}

sd.instanceID = mServiceId.getInstanceID();
sd.serviceType = mServiceId.getServiceType();
try
{
    int connID = BleConstants.GATT_INVALID_CONN_ID;

    if ((connID = mProfile.getConnIdForDevice(remoteDevice)) != BleConstants.
        GATT_INVALID_CONN_ID)
        mProfile.getGattService().registerServiceDataCallback(connID,
            mServiceId, remoteDevice.getAddress(), mCallback);
} catch (RemoteException e)
{
    Log.d(TAG, e.toString());
}
}

void removeInstanceID(BluetoothDevice remoteDevice, int instanceID)
{
}

void setProfile(BleClientProfile profile)
{
    mProfile = profile;
}

protected ServiceData getServiceData(BluetoothDevice remoteDevice, int
instanceID)
{
    Log.d(TAG, "getServiceData address = " + remoteDevice.getAddress()
        + " instanceID = " + instanceID);

    ServiceData sData = null;
    ArrayList<ServiceData> s = mdeviceToDataMap.get(remoteDevice);
    if (s != null) {
        for (int i = 0; i < s.size(); i++) {
            if (s.get(i).instanceID == instanceID) {
                sData = s.get(i);
                break;
            }
        }
    }
    return sData;
}

protected ServiceData getNextServiceData(BluetoothDevice remoteDevice,

```

```

        int currentInstanceID)
    {
        Log.d(TAG, "getServiceData address = " + remoteDevice.getAddress()
            + " currentInstanceID = " + currentInstanceID);

        ServiceData sData = null;
        ArrayList<ServiceData> s = mdeviceToDataMap.get(remoteDevice);
        if (s != null) {
            for (int i = 0; i < s.size(); i++) {
                if (s.get(i).instanceID != currentInstanceID)
                    continue;
                if (s.size() > i + 1) {
                    sData = s.get(i + 1);
                    break;
                }
            }
        }

        return sData;
    }

protected int getFirstIncludedService(BluetoothDevice remoteDevice, int
instanceID)
{
    int ret = BleConstants.GATT_SUCCESS;
    int connID = BleConstants.GATT_INVALID_CONN_ID;

    Log.d(TAG, "getFirstIncludedService address = " + remoteDevice.
        getAddress());
    ServiceData sd = getServiceData(remoteDevice, instanceID);
    try {
        if ((connID = mProfile.getConnIdForDevice(remoteDevice)) !=
            BleConstants.GATT_INVALID_CONN_ID) {
            BleGattID svcId = new BleGattID(sd.instanceID, mServiceId.
                getUuid(),
                mServiceId.getServiceType());
            mProfile.getGattService().getFirstIncludedService(connID, svcId,
                null);
        }
        else {
            ret = BleConstants.GATT_INVALID_CONN_ID;
        }
    } catch (RemoteException e) {
        Log.d(TAG, "getFirstIncludedService " + e.toString());
        ret = BleConstants.GATT_ERROR;
    }

    return ret;
}

```

```

protected int getNextIncludedService(BluetoothDevice remoteDevice,
    BluetoothGattInclSrvclD inclsvclD)
{
    int ret = BleConstants.GATT_SUCCESS;
    int connID = BleConstants.GATT_INVALID_CONN_ID;

    Log.d(TAG, "getNextIncludedService address = " + remoteDevice.
        getAddress());
    try
    {
        if ((connID = mProfile.getConnIdForDevice(remoteDevice)) != BleConstants.
            GATT_INVALID_CONN_ID)
            mProfile.getGattService().getNextIncludedService(connID, inclsvclD,
                null);
        else
            ret = BleConstants.GATT_INVALID_CONN_ID;
    } catch (RemoteException e) {
        Log.d(TAG, "getNextIncludedService " + e.toString());
        ret = BleConstants.GATT_ERROR;
    }

    return ret;
}

protected int readFirstCharacteristic(BluetoothDevice remoteDevice,
    BleGattID svcId)
{
    int ret = BleConstants.GATT_SUCCESS;
    int connID = BleConstants.GATT_INVALID_CONN_ID;

    Log.d(TAG, "readFirstCharacteristic address = " + remoteDevice.
        getAddress());
    try
    {
        if ((connID = mProfile.getConnIdForDevice(remoteDevice)) !=
            BleConstants.GATT_INVALID_CONN_ID)
            mProfile.getGattService().getFirstChar(connID, svcId, null);
        else
            ret = BleConstants.GATT_INVALID_CONN_ID;
    } catch (RemoteException e) {
        Log.d(TAG, "getFirstCharacteristic " + e.toString());
        ret = BleConstants.GATT_ERROR;
    }

    return ret;
}

protected int readNextCharacteristic(BluetoothDevice remoteDevice,
    BleGattID svcId,

```

```

        BleGattID characteristicID)
    {
        Log.d(TAG,
            "readNextCharacteristic characteristicID = " + characteristicID.
            toString()
            + " char inst id = " + characteristicID.getInstanceID());

        int ret = BleConstants.GATT_SUCCESS;
        int connID = BleConstants.GATT_INVALID_CONN_ID;
        ServiceData sd = getServiceData(remoteDevice, svcId.getInstanceID());
        BluetoothGattCharID charId = new BluetoothGattCharID(svcId, characteristicID);
        try
        {
            if ((connID = mProfile.getConnIdForDevice(remoteDevice)) !=
                BleConstants.GATT_INVALID_CONN_ID)
                mProfile.getGattService().getNextChar(connID, charId, null);
            else
                ret = BleConstants.GATT_INVALID_CONN_ID;
        } catch (RemoteException e) {
            Log.d(TAG, "getFirstCharacteristic " + e.toString());
            ret = BleConstants.GATT_ERROR;
        }

        return ret;
    }

    /** @hide */
    protected int readFirstCharDescriptor(BluetoothDevice remoteDevice,
        BleGattID svcId,
        BleGattID characteristicId)
    {
        int ret = BleConstants.GATT_SUCCESS;
        int connID = BleConstants.GATT_INVALID_CONN_ID;

        Log.d(TAG, "readFirstCharDescriptor address = " + remoteDevice.
            getAddress());
        BluetoothGattCharID charId = new BluetoothGattCharID(svcId, characteristicId);
        try
        {
            if ((connID = mProfile.getConnIdForDevice(remoteDevice)) != BleConstants.
                GATT_INVALID_CONN_ID)
                mProfile.getGattService().getFirstCharDescr(connID, charId, null);
            else
                ret = BleConstants.GATT_INVALID_CONN_ID;
        } catch (RemoteException e) {
            Log.d(TAG, "getFirstCharDescriptor " + e.toString());
            ret = BleConstants.GATT_ERROR;
        }

        return ret;
    }

```



```

    }

    protected int readNextCharDescriptor(BluetoothDevice remoteDevice,
        BleGattID svcId,
        BleGattID charId, BleGattID descriptorId)
    {
        int ret = BleConstants.GATT_SUCCESS;
        int connID = BleConstants.GATT_INVALID_CONN_ID;
        Log.d(TAG, "readNextCharDescriptor address = " + remoteDevice.
            getAddress());
        Log.d(TAG, "svcId " + svcId);
        Log.d(TAG, "charId " + charId);
        Log.d(TAG, "descId " + descriptorId);
        if (descriptorId == null)
            descriptorId = charId;

        BluetoothGattCharDescrID descId = new BluetoothGattCharDescrID(svcId,
            charId,
            descriptorId);
        try
        {
            if ((connID = mProfile.getConnIdForDevice(remoteDevice)) !=
                BleConstants.GATT_INVALID_CONN_ID) {
                try {
                    mProfile.getGattService().getNextCharDescr(connID, descId,
                        descriptorId);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
            else
                ret = BleConstants.GATT_INVALID_CONN_ID;
        } catch (Exception e) {
            Log.d(TAG, "getNextCharDescriptor " + e.toString());
            ret = BleConstants.GATT_ERROR;
        }

        return ret;
    }

    protected int readCharacteristicValue(int connID, BluetoothGattCharID
        charID, byte authReq)
    {
        Log.d(TAG, "readCharacteristicValue ");
        int ret = BleConstants.GATT_SUCCESS;
        try {
            mProfile.getGattService().readChar(connID, charID, authReq);
        } catch (RemoteException e) {
            Log.d(TAG, "readCharacteristicValue" + e.toString());
            ret = BleConstants.GATT_ERROR;
        }
    }

```

```

    }
    return ret;
}

/** @hide */
protected int readCharDescriptorValue(int connID, BluetoothGattCharDescrID
charDescrID,
    byte authReq)
{
    Log.d(TAG, "readCharDescriptor");
    int ret = BleConstants.GATT_SUCCESS;
    try
    {
        mProfile.getGattService().readCharDescr(connID, charDescrID, authReq);
    } catch (RemoteException e) {
        Log.d(TAG, "readCharacteristicExtProp" + e.toString());
        ret = BleConstants.GATT_ERROR;
    }

    return ret;
}

protected int sendIndicationConfirmation(int connId, BluetoothGattCharID
charId)
{
    int ret = BleConstants.GATT_SUCCESS;
    Log.d(TAG, "sendIndicationConfirmation");
    try
    {
        mProfile.getGattService().sendIndConfirm(connId, charId);
    } catch (RemoteException e) {
        Log.d(TAG, "sendIndicationConfirmation" + e.toString());
        ret = BleConstants.GATT_ERROR;
    }
    return ret;
}

protected void onServiceRefreshed(int connID)
{
    Log.d(TAG, "onServiceRefreshed");
    onRefreshComplete(mProfile.getDeviceForConnId(connID));
    mProfile.onServiceRefreshed(this, mProfile.getDeviceForConnId(connID));
}

class BleCharacteristicDataCallback extends IBleCharacteristicDataCallback.Stub
{
    BleCharacteristicDataCallback()
    {
    }
}

```

```
@Override
public void onGetFirstCharacteristic(int connID, int status,
BluetoothGattID svcId,
    BluetoothGattID characteristicID)
{
    Log.d(BleClientService.TAG,
        "onGetFirstCharacteristic " + characteristicID.toString() +
        " status = "
        + status);

    BluetoothDevice device = BleClientService.this.mProfile
        .getDeviceforConnId(connID);
    if ((device == null)
        || (BleClientService.this.mProfile.isDeviceDisconnecting(
            device))) {
        Log.e(BleClientService.TAG,
            "onGetFirstCharacteristic() - Device is disconnecting...");
        return;
    }

    if (status == 0) {
        ServiceData s = BleClientService.this.getServiceData(
            BleClientService.this.mProfile.getDeviceforConnId(
                connID),
            svcId.getInstanceID());
        s.characteristics.clear();

        Log.d(BleClientService.TAG,
            "characteristic ID = " + characteristicID.toString() +
            "instance ID = "
            + characteristicID.getInstanceID());

        BleCharacteristic characteristic = null;
        if (characteristicID.getUuidType() == 16)
            characteristic = new BleCharacteristic(new BleGattID(
                characteristicID.getUuid()));
        else
            characteristic = new BleCharacteristic(new BleGattID(
                characteristicID.getUuid16()));
        characteristic.setInstanceID(characteristicID.getInstanceID());

        BleClientService.this.onSetCharacteristicAuthRequirement(
            BleClientService.this.mProfile.getDeviceforConnId(
                connID), characteristic,
            svcId.getInstanceID());

        s.characteristics.add(characteristic);
        BleClientService.this.readFirstCharDescriptor(
            BleClientService.this.mProfile.getDeviceforConnId(
                connID),
```

```

        BleApiHelper.gatt2BleID(svcId),
        BleApiHelper.gatt2BleID(characteristicID));
    }
    else
    {
        ServiceData s = BleClientService.this.getNextServiceData(
            BleClientService.this.mProfile.getDeviceforConnId(
                connID),
            svcId.getInstanceID());
        if (null != s)
        {
            BleClientService.this.readFirstCharacteristic(
                BleClientService.this.mProfile.getDeviceforConnId(
                    connID),
                new BleGattID(s.instanceID,
                    svcId.getUuid(),
                    svcId.getServiceType()));
        }
        else
        {
            BleClientService.this.onServiceRefreshed(connID);
        }
    }
}

@Override
public void onGetFirstCharacteristicDescriptor(int connID, int status,
    BluetoothGattID svcId, BluetoothGattID characteristicID,
    BluetoothGattID descriptorID)
{
    Log.d(BleClientService.TAG, "onGetFirstCharacteristicDescriptor "
        + characteristicID.toString() + " status = " + status);

    BluetoothDevice device = BleClientService.this.mProfile
        .getDeviceforConnId(connID);
    if ((device == null)
        || (BleClientService.this.mProfile.isDeviceDisconnecting(
            device))) {
        Log.e(BleClientService.TAG,
            "onGetFirstCharacteristicDescriptor() - Device is
            disconnecting...");
        return;
    }

    if (status != BleConstants.GATT_SUCCESS) {
        BleClientService.this.readNextCharacteristic(
            BleClientService.this.mProfile.getDeviceforConnId(
                connID),
            BleApiHelper.gatt2BleID(svcId),
            BleApiHelper.gatt2BleID(characteristicID));
    }
}

```

```

        return;
    }

    Log.d(BleClientService.TAG,
        "characteristic ID = " + characteristicID.toString() +
        "instance ID = "
            + characteristicID.getInstanceID());

    BleCharacteristic characteristic = findCharacteristic(connID,
        BleApiHelper.gatt2BleID(svcId), BleApiHelper.gatt2BleID(
            characteristicID));

    if (descriptorID.getUuidType() == BleConstants.GATT_UUID_TYPE_128) {
        String uuid128 = descriptorID.getUuid().toString();
        if (uuid128.equals("00002900-0000-1000-8000-00805f9b34fb"))
            characteristic.addDescriptor(new BleExtProperty());
        else if (uuid128.equals("00002902-0000-1000-8000-00805f9b34fb"))
            characteristic.addDescriptor(new BleClientConfig());
        else if (uuid128.equals("00002903-0000-1000-8000-00805f9b34fb"))
            characteristic.addDescriptor(new BleServerConfig());
        else if (uuid128.equals("00002904-0000-1000-8000-00805f9b34fb"))
            characteristic.addDescriptor(new BlePresentationFormat());
        else if (uuid128.equals("00002901-0000-1000-8000-00805f9b34fb"))
            characteristic.addDescriptor(new BleUserDescription());
        else
            characteristic.addDescriptor(new BleDescriptor(new BleGattID(
                descriptorID.getUuid())));
    }
    else {
        switch (descriptorID.getUuid16()) {
            case BleConstants.GATT_UUID_CHAR_EXT_PROP16:
                characteristic.addDescriptor(new BleExtProperty());
                break;
            case BleConstants.GATT_UUID_CHAR_CLIENT_CONFIG16:
                characteristic.addDescriptor(new BleClientConfig());
                break;
            case BleConstants.GATT_UUID_CHAR_SRVR_CONFIG16:
                characteristic.addDescriptor(new BleServerConfig());
                break;
            case BleConstants.GATT_UUID_CHAR_PRESENT_FORMAT16:
                characteristic.addDescriptor(new BlePresentationFormat());
                break;
            case BleConstants.GATT_UUID_CHAR_DESCRIPTION16:
                characteristic.addDescriptor(new BleUserDescription());
                break;
            default:
                characteristic.addDescriptor(new BleDescriptor(new
                    BleGattID(
                        descriptorID.getUuid16())));
        }
    }
}

```



```

    }

    BleClientService.this.readNextCharDescriptor(
        BleClientService.this.mProfile.getDeviceforConnId(connID),
        BleApiHelper.gatt2BleID(svcId), BleApiHelper.gatt2BleID(
            characteristicID),
        BleApiHelper.gatt2BleID(descriptorID));
    }

    @Override
    public void onGetNextCharacteristicDescriptor(int connID, int status,
        BluetoothGattID svcId, BluetoothGattID characteristicID,
        BluetoothGattID descriptorID)
    {
        Log.d(BleClientService.TAG, "onGetNextCharacteristicDescriptor");
        Log.d(BleClientService.TAG, "onGetNextCharacteristicDescriptor "
            + characteristicID.toString() + " status = " + status);

        BluetoothDevice device = BleClientService.this.mProfile
            .getDeviceforConnId(connID);
        if ((device == null)
            || (BleClientService.this.mProfile.isDeviceDisconnecting(
                device))) {
            Log.e(BleClientService.TAG,
                "onGetNextCharacteristicDescriptor() - Device is
                disconnecting...");
            return;
        }

        if (status != BleConstants.GATT_SUCCESS) {
            BleClientService.this.readNextCharacteristic(
                BleClientService.this.mProfile.getDeviceforConnId(connID),
                BleApiHelper.gatt2BleID(svcId), BleApiHelper.gatt2BleID(
                    characteristicID));
            return;
        }

        Log.d(BleClientService.TAG,
            "characteristic ID = " + characteristicID.toString() +
            "instance ID = "
            + characteristicID.getInstanceID());

        BleCharacteristic characteristic = findCharacteristic(connID,
            BleApiHelper.gatt2BleID(svcId), BleApiHelper.gatt2BleID(
                characteristicID));

        if (descriptorID.getUuidType() == BleConstants.GATT_UUID_TYPE_128) {
            characteristic.addDescriptor(new BleDescriptor(new BleGattID(

```

```

        descriptorID.getUuid())));
    }
    else {
        characteristic.addDescriptor(new BleDescriptor(new BleGattID(
            descriptorID.getUuid16())));
    }

    BleClientService.this.readNextCharDescriptor(
        BleClientService.this.mProfile.getDeviceforConnId(connID),
        BleApiHelper.gatt2BleID(svcId), BleApiHelper.gatt2BleID
            (characteristicID),
        BleApiHelper.gatt2BleID(descriptorID));
}

@Override
public void onGetNextCharacteristic(int connID, int status,
    BluetoothGattID svcId,
    BluetoothGattID characteristicID)
{
    Log.d(BleClientService.TAG,
        "onGetNextCharacteristic " + characteristicID.toString() +
        "instance id = "
            + characteristicID.getInstanceID() + " status = " +
            status);

    BluetoothDevice device = BleClientService.this.mProfile
        .getDeviceforConnId(connID);
    if ((device == null)
        || (BleClientService.this.mProfile.isDeviceDisconnecting
            (device))) {
        Log.e(BleClientService.TAG,
            "onGetNextCharacteristic() - Device is disconnecting...");
        return;
    }

    if (status == BleConstants.GATT_SUCCESS) {
        ServiceData s = BleClientService.this.getServiceData(
            BleClientService.this.mProfile.getDeviceforConnId
                (connID),
            svcId.getInstanceID());

        Log.d(BleClientService.TAG,
            "characteristic ID = " + characteristicID.toString() +
            "instance ID = "
                + characteristicID.getInstanceID());

        BleCharacteristic characteristic = null;
        if (characteristicID.getUuidType() == BleConstants.GATT_UUID_
            TYPE_128)

```

```

        characteristic = new BleCharacteristic(new BleGattID(
            characteristicID.getUuid()));
    else
        characteristic = new BleCharacteristic(new BleGattID(
            characteristicID.getUuid16()));
    characteristic.setInstanceId(characteristicID.getInstanceID());

    BleClientService.this.onSetCharacteristicAuthRequirement(
        BleClientService.this.mProfile.getDeviceforConnId(connID),
        characteristic,
        svcId.getInstanceID());

    s.characteristics.add(characteristic);

    BleClientService.this.readNextCharDescriptor(
        BleClientService.this.mProfile.getDeviceforConnId(
            connID),
        BleApiHelper.gatt2BleID(svcId),
        BleApiHelper.gatt2BleID(characteristicID),
        null);
    }
    else
    {
        ServiceData s = BleClientService.this.getNextServiceData(
            BleClientService.this.mProfile.getDeviceforConnId(
                connID),
            svcId.getInstanceID());

        if (null != s)
        {
            BleClientService.this.readFirstCharacteristic(
                BleClientService.this.mProfile.getDeviceforConnId(
                    connID),
                new BleGattID(s.instanceID, BleClientService.this.
                    getServiceId()
                        .getUuid(), BleClientService.this.getServiceId()
                        .getServiceType()));
        }
        else
        {
            BleClientService.this.onServiceRefreshed(connID);
        }
    }
}

@Override
public void onReadCharacteristicValue(int connID, int status,
    BluetoothGattID svcId,
    BluetoothGattID characteristicID, byte[] data)
{

```

```

        Log.d(BleClientService.TAG, "onReadCharacteristicValue charID = "
            + characteristicID.toString() + " status = " + status);

        BleCharacteristic c = findCharacteristic(connID,
            BleApiHelper.gatt2BleID(svcId), BleApiHelper.gatt2BleID
            (characteristicID));

        if (c == null) {
            Log.e(BleClientService.TAG,
                "onReadCharacteristicValue() - Characteristic not found");
        }

        if (status != BleConstants.GATT_SUCCESS) {
            BleClientService.this.onReadCharacteristicComplete(status,
                BleClientService.this.mProfile.getDeviceforConnId
                (connID), c);
            return;
        }

        c.setValue(data);
        if (BleClientService.this.mReadDescriptors)
        {
            ArrayList<BleDescriptor> descList = c.getDirtyDescQueue();
            if (!descList.isEmpty()) {
                BleDescriptor descObj = descList.get(0);
                BleClientService.this.readCharDescriptorValue(
                    connID,
                    new BluetoothGattCharDescrID(svcId, characteristicID,
                        descObj
                            .getID()), descObj.getAuthReq());
            }
            else {
                BleClientService.this.onReadCharacteristicComplete(
                    BleClientService.this.mProfile.getDeviceforConnId
                    (connID), c);
            }
        }
        else {
            BleClientService.this.onReadCharacteristicComplete(
                BleClientService.this.mProfile.getDeviceforConnId
                (connID), c);
        }
    }

    @Override
    public void onReadCharDescriptorValue(int connID, int status,
        BluetoothGattID svcId,
        BluetoothGattID characteristicID, BluetoothGattID descr, byte[]
        data)
    {
        Log.d(BleClientService.TAG, "onReadCharacteristicExtProp charID = "

```

```

        + characteristicID.toString() + " status = " + status);

BleDescriptor d = findDescriptor(connID, BleApiHelper.gatt2BleID
(svcId),
    BleApiHelper.gatt2BleID(characteristicID), BleApiHelper.
gatt2BleID(descr));

BleCharacteristic c = findCharacteristic(connID,
    BleApiHelper.gatt2BleID(svcId), BleApiHelper.gatt2BleID
(characteristicID));

if (d != null)
    if (status == 0) {
        d.setValue(data);

        if (BleClientService.this.mReadDescriptors)
        {
            if (c != null)
            {
                c.updateDirtyDescQueue();
                ArrayList<BleDescriptor> descList = c.getDirtyDescQueue();
                if (!descList.isEmpty()) {
                    BleDescriptor descObj = descList.get(0);
                    BleClientService.this.readCharDescriptorValue
                        (connID,
                            new BluetoothGattCharDescrID(svcId,
                                characteristicID,
                                    descObj.getID(), descObj.
                                        getAuthReq());
                        )
                }
            }
            else
            {
                BleClientService.this.mReadDescriptors = false;
                BleClientService.this.onReadCharacteristicComplete(
                    BleClientService.this.mProfile.
                        getDeviceforConnId(connID),
                        c);
            }
        }
    }
    else if (c != null) {
        BleClientService.this.onReadCharacteristicComplete(status,
            BleClientService.this.mProfile.getDeviceforConnId
                (connID), c);
    }
}

@Override
public void onWriteCharValue(int connID, int status, BluetoothGattID

```



```

        svcId,
        BluetoothGattID characteristicID)
    {
        Log.d(BleClientService.TAG, "onWriteCharValue connID " + connID +
            " status "
            + status);
        BleCharacteristic c = findCharacteristic(connID,
            BleApiHelper.gatt2BleID(svcId), BleApiHelper.gatt2BleID
            (characteristicID));

        if (status == 0) {
            BleClientService.this.writeCharacteristic(
                BleClientService.this.mProfile.getDeviceforConnId
                (connID),
                svcId.getInstanceID(), c);
        }
        else
            BleClientService.this.onWriteCharacteristicComplete(status,
                BleClientService.this.mProfile.getDeviceforConnId
                (connID), c);
    }

    @Override
    public void onWriteCharDescrValue(int connID, int status, BluetoothGattID
        svcId,
        BluetoothGattID characteristicID, BluetoothGattID descr)
    {
        Log.d(BleClientService.TAG, "onWriteCharDescrValue status=" + status);

        BleCharacteristic c = findCharacteristic(connID,
            BleApiHelper.gatt2BleID(svcId), BleApiHelper.gatt2BleID
            (characteristicID));

        if (status == 0) {
            BleClientService.this.writeCharacteristic(
                BleClientService.this.mProfile.getDeviceforConnId(connID),
                svcId.getInstanceID(), c);
        }
        else
            BleClientService.this.onWriteCharacteristicComplete(status,
                BleClientService.this.mProfile.getDeviceforConnId
                (connID), c);
    }

    BleCharacteristic findNextCharacteristic(int connID, BleCharacteristic c,
        int instanceID)
    {
        Log.d(BleClientService.TAG,
            "findNextCharacteristic " + connID + " current characteristic : "
            + c.getID().toString() + " instance id = " +

```

```

        c.getInstanceID());

    ServiceData s = BleClientService.this.getServiceData(
        BleClientService.this.mProfile.getDeviceforConnId(connID),
        instanceID);

    int i;
    for (i = 0; i < s.characteristics.size(); i++) {
        BleCharacteristic cTemp = s.characteristics.get(i);
        if (c.getID().equals(cTemp.getID()))
        {
            break;
        }
    }
    if (i + 1 < s.characteristics.size()) {
        Log.d(BleClientService.TAG, "findNextCharacteristic position = " + i
            + " connID = " + connID + " next characteristic : "
            + s.characteristics.get(i + 1).getID().toString());

        return s.characteristics.get(i + 1);
    }
    return null;
}

BleCharacteristic findCharacteristic(int connID, BleGattID svcId,
    BleGattID characteristicID)
{
    Log.d(BleClientService.TAG,
        "findCharacteristic charID = [" + characteristicID.
        toString()
        + "] instance ID = [" + characteristicID.getInstanceID()
        + "]");

    ServiceData s = BleClientService.this.getServiceData(
        BleClientService.this.mProfile.getDeviceforConnId(connID),
        svcId.getInstanceID());

    for (int i = 0; i < s.characteristics.size(); i++) {
        BleCharacteristic c = s.characteristics.get(i);
        if ((!c.getID().toString().equals(characteristicID.toString()))
            || (c.getInstanceID() != characteristicID.
                getInstanceID()))
            continue;
        Log.d(BleClientService.TAG, "findCharacteristic - found");
        return c;
    }

    return null;
}

```

```

BleDescriptor findDescriptor(int connID, BleGattID svcId, BleGattID
characteristicID,
    BleGattID descriptorID)
{
    Log.d(BleClientService.TAG,
        "findCharacteristic charID = [" + characteristicID.
        toString()
            + "] instance ID = [" + characteristicID.getInstanceID()
            + "]");

    ServiceData s = BleClientService.this.getServiceData(
        BleClientService.this.mProfile.getDeviceforConnId(connID),
        svcId.getInstanceID());

    BleCharacteristic charObj = null;
    for (int i = 0; i < s.characteristics.size(); i++) {
        BleCharacteristic c = s.characteristics.get(i);

        if (c.getID().equals(characteristicID.getUuid())) {
            Log.d(BleClientService.TAG, "findCharacteristic - found");
            charObj = c;
        }
    }
    if (charObj != null) {
        for (int i = 0; i < charObj.getAllDescriptors().size(); i++) {
            BleDescriptor d = charObj.getAllDescriptors().get(i);
            if (d.getID().equals(descriptorID)) {
                Log.d(BleClientService.TAG, "findDescriptor - found");
                return d;
            }
        }
    }

    return null;
}

@Override
public void onRegForNotifications(int connId, int status, BluetoothGattID
svcId,
    BluetoothGattID charId)
{
    Log.d(BleClientService.TAG, "onRegForNotifications " + status);
}

@Override
public void onUnregisterNotifications(int connId, int status,
BluetoothGattID svcId,
    BluetoothGattID charId)
{
    Log.d(BleClientService.TAG, "onUnregisterNotifications" + status);
}

```

```

    }

    @Override
    public void onNotify(int connId, String address, BluetoothGattID svcId,
        BluetoothGattID characteristicID, boolean isNotify, byte[] data)
    {
        Log.d(BleClientService.TAG, "onNotify " + connId + " " + address);

        BleCharacteristic c = findCharacteristic(connId,
            BleApiHelper.gatt2BleID(svcId), BleApiHelper.gatt2BleID
                (characteristicID));

        if (c != null)
        {
            c.setValue(data);
            BleClientService.this.onCharacteristicChanged(
                BleClientService.this.mProfile.getDeviceforConnId
                    (connId), c);
        } else {
            Log.d(BleClientService.TAG, "onNotify Characteristic not found"
                + connId
                + " " + address);
        }

        if (!isNotify)
        {
            BluetoothGattCharID charId = new BluetoothGattCharID(svcId,
                characteristicID);

            BleClientService.this.sendIndicationConfirmation(connId, charId);
        }
    }

    @Override
    public void onGetFirstIncludedService(int connID, int status,
        BluetoothGattID svcId,
        BluetoothGattID inclsvcId)
    {
        Log.d(BleClientService.TAG, "onGetFirstIncludedService");

        if (status == 0) {
            BluetoothGattInclSrvID cursvcId = new BluetoothGattInclSrvID
                (svcId,
                inclsvcId);

            BleClientService.this.getNextIncludedService(
                BleClientService.this.mProfile.getDeviceforConnId
                    (connID), cursvcId);
        } else {
            BleClientService.this.onServiceRefreshed(connID);
        }
    }

```



```

    }
}

@Override
public void onGetNextIncludedService(int connID, int status,
    BluetoothGattID svcId,
    BluetoothGattID inclsvcId)
{
    Log.d(BleClientService.TAG, "onGetNextIncludedService");
    if (status == 0) {
        BluetoothGattInclSrvCID cursvcId = new BluetoothGattInclSrvCID
            (svcId,
            inclsvcId);

        BleClientService.this.getNextIncludedService(
            BleClientService.this.mProfile.getDeviceforConnId
            (connID), cursvcId);
    }
    else {
        BleClientService.this.onServiceRefreshed(connID);
    }
}

}

class ServiceData
{
    public int instanceID = -1;
    public int writeIndex = -1;
    public int serviceType = -1;

    public ArrayList<BleCharacteristic> characteristics = new ArrayList
    <BleCharacteristic>();

    ServiceData()
    {
    }
}
}
}

```

(5) 定义服务器端的角色低功耗规范

在 Broadcom 公司提供的源码中，文件 `BleServerProfile.java` 的功能是定义了服务器端的角色低功耗规范，在创建一个新的低功耗规范之前，需要先继承于这个类，并提供标识要访问规范所必需的参数和服务。通常来说，一个 `BleServerProfile` 派生的类包含一个或多个 `BleServerService` 对象。在 `BleServerProfile` 派生的类中，包含低功耗规范中定义服务的 `BleServerService` 对象的集合。文件 `BleServerProfile.java` 的主要实现代码如下。


```

public abstract class BleServerProfile
{
    private static final boolean D = true;
    private static final String TAG = "BleServerProfile";
    private Context mCtxt = null;
    private BleGattID mAppid;
    ArrayList<BleServerService> mServiceArr = null;
    private HashMap<String, Integer> mConnMap = null;
    private HashMap<Integer, Integer> mMtuMap = null;
    private IBluetoothGatt mService;
    private int mSvcCreated = 0;
    private int mSvcStarted = 0;
    private byte mAppHandle = -1;
    private int mProfileStatus = 2;
    private GattServiceConnection mSvcConn;

    public BleServerProfile(Context ctxt, BleGattID appId,
        ArrayList<BleServerService> serviceArr)
    {
        mAppid = appId;
        mCtxt = ctxt;
        mServiceArr = serviceArr;
        mConnMap = new HashMap<String, Integer>();
        mMtuMap = new HashMap<Integer, Integer>();
        mSvcConn = new GattServiceConnection(null);
        Intent i = new Intent();
        i.setClassName("com.broadcom.bt.app.system",
            "com.broadcom.bt.app.system.GattService");
        mCtxt.bindService(i, mSvcConn, 1);

        throw new RuntimeException("Not implemented");
    }
    /*取消和此规范相关的资源*/
    public synchronized void finish()
    {
        if (mSvcConn != null) {
            mCtxt.unbindService(mSvcConn);
            mSvcConn = null;
        }
    }

    public void finalize()
    {
        finish();
    }

    byte getAppHandle()
    {
        return mAppHandle;
    }
}

```

```
HashMap<String, Integer> getConnMap() {
    return mConnMap;
}
/*初始化相关的服务*/
void initProfile()
{
    Log.i("BleServerProfile", "initProfile()");
    try {
        mService.registerServerProfileCallback(mAppid,
            new BleServerProfileCallback(this));
    } catch (Throwable t) {
        Log.e("BleServerProfile", "Unable to start profile", t);
    }
}

void notifyAction(int event)
{
    if ((event == 0) && (++mSvcCreated == mServiceArr.size()))
    {
        Log.i("BleServerProfile",
            "All services created successfully. Calling onInitialized");
        onInitialized(true);
    } else if ((event == 4) && (--mSvcCreated == 0))
    {
        Log.i("BleServerProfile",
            "All services stopped successfully. Calling onStoped");
        onStoped();
    } else if ((event == 2) && (++mSvcStarted == mServiceArr.size()))
    {
        Log.i("BleServerProfile",
            "All services started successfully. Calling onStart");
        onStart(true);
    } else if (event == 1) {
        Log.i("BleServerProfile",
            "One of the services creation failed. Calling onInitialized");
        mProfileStatus = 2;
        onInitialized(false);
    } else if (event == 3) {
        Log.i("BleServerProfile",
            "One of the services start failed. Calling onStart");
        mProfileStatus = 2;
        onStart(false);
    } else {
        Log.e("BleServerProfile", "Unknown action from a service");
    }
}
/*启用和此规范有关的所有服务*/
public void startProfile()
{

```

```

    Log.i("BleServerProfile", "startProfile()");
    if (mService == null) {
        Log.i("BleServerProfile", "Remote service object is null.. Returning..");
        return;
    }

    for (int i = 0; i < mServiceArr.size(); i++) {
        if (!((BleServerService) mServiceArr.get(i)).isRegistered()) {
            Log.i("BleServerProfile",
                "One of the services is not registered. Stopping all the
                services");
            stopProfile();
            return;
        }

        ((BleServerService) mServiceArr.get(i)).startService();
    }
}

/*停止和此规范有关的所有服务*/
public void stopProfile()
{
    Log.i("BleServerProfile", "stopProfile()");
    for (int i = 0; i < mServiceArr.size(); i++)
        ((BleServerService) mServiceArr.get(i)).stopService();
}

/*注销所有相关的服务*/
public void finishProfile()
{
    Log.i("BleServerProfile", "finishProfile()");
    for (int i = 0; i < mServiceArr.size(); i++) {
        ((BleServerService) mServiceArr.get(i)).deleteService();
    }
    try
    {
        mService.unregisterServerProfileCallback(mAppHandle);
    } catch (Throwable t) {
        Log.e("BleServerProfile", "Unable to stop profile", t);
        return;
    }
}

/*为连接设置最大传输单元*/
public void setMtuSize(int connId, int mtuSize)
{
    Log.i("BleServerProfile", "setMtuSize");
    mMtuMap.put(Integer.valueOf(connId), Integer.valueOf(mtuSize));
}

/*为一个活跃的连接设置需要的加密等级*/
public void setEncryption(String bdaddr, byte action)
{
    try

```

```

    {
        mService.setEncryption(bdaddr, action);
    } catch (Throwable t) {
        Log.e("BleServerProfile", "Unable to set encryption for connection", t);
    }
}

/*当已经请求一个后台链接时, 定义本地设备扫描远程低功耗设备的强度*/
public void setScanParameters(int scanInterval, int scanWindow)
{
    try
    {
        mService.setScanParameters(scanInterval, scanWindow);
    } catch (Throwable t) {
        Log.e("BleServerProfile", "Unable to set scan parameters", t);
    }
}

/*打开一个外设 GAP 客户端的连接*/
public void open(String bdaddr, boolean isDirect)
{
    try
    {
        mService.GATTServer_Open(mAppHandle, bdaddr, isDirect);
    } catch (Throwable t) {
        Log.e("BleServerProfile", "Unable to open Gatt connection", t);
    }
}

/*取消一个正在进行中对外设 GATT 客户端的打开操作*/
public void cancelOpen(String bdaddr, boolean isDirect)
{
    try
    {
        mService.GATTServer_CancelOpen(mAppHandle, bdaddr, isDirect);
    } catch (Throwable t) {
        Log.e("BleServerProfile", "Unable to open Gatt connection", t);
        return;
    }
}

/*关闭一个到远程低功耗规范客户端的连接*/
public void close(String bdaddr)
{
    try
    {
        mService.GATTServer_Close(((Integer) mConnMap.get(bdaddr))
            .intValue());
    } catch (Throwable t) {
        Log.e("BleServerProfile", "Unable to open Gatt connection", t);
        return;
    }
}
}

```

```

protected void onAppRegisterCompleted(int status, int serIf)
{
    if (status == 0) {
        mAppHandle = (byte) serIf;
        for (int i = 0; i < mServiceArr.size(); i++)
            ((BleServerService) mServiceArr.get(i)).onServiceAvailable(this,
                mService, mAppid);
    } else {
        Log.e("BleServerProfile", "Application registration failed.. Aborting");
    }
}

public abstract void onInitialized(boolean paramBoolean);

public abstract void onStarted(boolean paramBoolean);

public abstract void onStopped();

public abstract void onClientConnected(String paramString, boolean
paramBoolean);

public abstract void onOpenCompleted(int paramInt);

public abstract void onOpenCancelCompleted(int paramInt);

public abstract void onCloseCompleted(int paramInt);

private class BleServerProfileCallback extends IBleProfileEventCallback.Stub
{
    private BleServerProfile mProfile;

    public BleServerProfileCallback(BleServerProfile profile)
    {
        mProfile = profile;
    }
}

/*当一个客户端建立或断开连接时触发的回调函数*/
public void onClientConnected(int connId, String bdaddr, boolean
isConnected)
{
    Log.i("BleServerProfile", "onClientConncted addr is " + bdaddr + "
connId is "
        + connId);

    mProfile.onClientConnected(bdaddr, isConnected);
    if (isConnected)
        mProfile.mConnMap.put(bdaddr, Integer.valueOf(connId));
    else
        mProfile.mConnMap.remove(bdaddr);
}

```



```

    public void onAttributeMtuExchange(String address, int connId, int
    transId, int mtuSize)
    {
        Log.i("BleServerProfile", "onAttributeMtuExchange");
    }

    public void onAppRegisterCompleted(int status, int serIf) {
        Log.i("BleServerProfile", "onAppRegisterCompleted");
        mProfile.onAppRegisterCompleted(status, serIf);
    }
}

private class GattServiceConnection
    implements ServiceConnection
{
    private Context context;

    private GattServiceConnection(Context c) {
        context = c;
    }

    public void onServiceConnected(ComponentName name, IBinder service)
    {
        Log.d("BleServerProfile", "Connected to GattService!");

        if (service != null)
            try {
                BleServerProfile.this.mService =
                    IBluetoothGatt.Stub.asInterface(service);
                BleServerProfile.this.initProfile();
            } catch (Throwable t) {
                Log.e("BleServerProfile", "Unable to get Binder to
                GattService", t);
            }
    }

    public void onServiceDisconnected(ComponentName name)
    {
        Log.d("BleServerProfile", "Disconnected from GattService!");
    }
}
}

```

(6) 创建低功耗服务

在 Broadcom 公司提供的源码中，文件 `BleServerService.java` 的功能是创建一个低功耗服务，这是服务器端角色上的低功耗规范的一部分。在 `BleServerService` 的派生类包含了一个或多个 `BleCharacteristic` 对象。在应用程序中，需要重写类 `BleServerService` 来实现一个服务。文件 `BleServerService.java` 的主要实现代码如下。

```

public abstract class BleServerService
{
    private final String TAG = "BleServerService";

    private HashMap<Integer, BleCharacteristic> mCharHdlMap = null;
    private HashMap<Integer, BleServerService> mServiceHdlMap = null;
    private HashMap<Integer, AttributeRequestInfo> mAttrReqMap = null;

    private ArrayList<BleCharacteristic> mCharQueue = null;
    private ArrayList<BleDescriptor> mDirtyDescQueue = null;
    private BleGattID mServiceId;
    private BleGattID mAppUuid;
    private BleServerProfile mProfileHandle;
    private IBluetoothGatt mService;
    private int mSvcHandle = -1;
    private byte mSupTransport;
    private BleServiceCallback mGattServiceCallback;
    private boolean isServiceAvailable = false;
    private int mSvcInstance = 0;

    private boolean isPrimary = false;
    private int mNumHandles;
    private final int CHAR_ADDED = 0;
    private final int CHAR_DESC_ADDED = 1;
    private final int ATTRIBUTE_WRITE = 2;
    private final int ATTRIBUTE_READ = 3;
    private final int HDL_VAL_INDICATION = 4;
    private final int HDL_VAL_NOTIFICATION = 5;
    private final int MTU_EXCHANGE = 6;
    private final int EXECUTE_WRITE = 7;

    private Handler mHandler = new Handler()
    {
        public void handleMessage(Message msg)
        {
        }
    };

    int getConnId(String address)
    {
        if (this.mProfileHandle == null)
            return -1;
        HashMap connMap = this.mProfileHandle.getConnMap();
        return ((Integer) connMap.get(address)).intValue();
    }

    /**构造函数, 使用给定的 ID 构造一个低功耗服务*/
    public BleServerService(BleGattID serviceId, int numHandles)
    {
        /**

```

```

        * TODO: implement
        */
        this.mServiceId = serviceId;
        this.mNumHandles = numHandles;
        this.mSupTransport = 2;
        this.mGattServiceCallback = new BleServiceCallback(this);
        this.mCharHdlMap = new HashMap();
        this.mServiceHdlMap = new HashMap();
        this.mCharQueue = new ArrayList();
        this.mAttrReqMap = new HashMap();

        if (this.mServiceId.getServiceType() == -1)
            this.mServiceId.setServiceType(0);
        throw new RuntimeException("not implemented");
    }

    /*构造函数，使用给定的 ID 构造一个新的低功耗服务*/
    public BleServerService(BleGattID serviceId, byte supTransport, int numHandles)
    {
        this.mServiceId = serviceId;
        this.mNumHandles = numHandles;
        this.mSupTransport = supTransport;
        this.mGattServiceCallback = new BleServiceCallback(this);

        this.mCharHdlMap = new HashMap();
        this.mCharQueue = new ArrayList();
        this.mServiceHdlMap = new HashMap();
        this.mAttrReqMap = new HashMap();

        if (this.mServiceId.getServiceType() == -1)
            this.mServiceId.setServiceType(0);
        throw new RuntimeException("not implemented");
    }

    /*初始化服务*/
    protected void initService()
    {
        if (this.mService != null)
            try {
                this.mService.registerServerServiceCallback(this.mServiceId,
                    this.mAppUuid, this.mGattServiceCallback);
            } catch (Throwable t) {
                Log.e("BleServerService", "initService", t);
            }
    }

    /*注册服务到蓝牙协议栈*/
    public void createService()
    {
        if (this.mService != null)
            try {

```

```

        this.mService.GATTServer_CreateService(this.mProfileHandle.
            getAppHandle(),
            this.mServiceId, this.mNumHandles);
    } catch (Throwable t)
    {
        Log.e("BleServerService", "createService", t);
    }
}

/*从蓝牙协议栈注销服务*/
public void deleteService()
{
    if (this.mService != null)
        try {
            this.mService.GATTServer_DeleteService(this.mSvcHandle);
        } catch (Throwable t) {
            Log.e("BleServerService", "deleteService ", t);
        }
}

/*启用服务*/
public void startService()
{
    if (this.mService != null)
        try {
            this.mService.GATTServer_StartService(this.mSvcHandle, this.
                mSupTransport);
        } catch (Throwable t) {
            Log.e("BleServerService", "startService ", t);
        }
}

/*停止服务*/
public void stopService()
{
    if (this.mService != null) {
        this.mProfileHandle.notifyAction(4);
        try {
            this.mService.unregisterServerServiceCallback(this.mSvcHandle);
            this.mService.GATTServer_StopService(this.mSvcHandle);
        } catch (Throwable t) {
            Log.e("BleServerService", "stopService ", t);
        }
    }
}

/*为此服务添加一个包含的服务*/
public void addIncludedService(BleServerService service)
{
    if (this.mService != null)
        try {
            if (service.isRegistered()) {
                this.mServiceHdlMap.put(Integer.valueOf(service.
                    getServiceHandle()),

```



```

        service);
        this.mService.GATTServer_AddIncludedService(this.mSvcHandle,
            service.getServiceHandle());
    }
    else {
        Log.i("BleServerService",
            "addIncludedService: Service to be included is not
            registered.");
    }
} catch (Throwable t) {
    Log.e("BleServerService", "addIncludedService", t);
}
}

/*更新一个特性或描述符*/
public void updateCharacteristic(BleCharacteristic charObj)
{
    addCharacteristic(charObj);
}

/*当客户端已经请求读或写一个本地特性属性后发送一个响应*/
public void sendResponse(String address, int transId, byte[] data, int
    statusCode)
{
    Log.d("BleServerService", "sendResponse() address = " + address + ",
        transId = "
        + transId + ",statusCode = " + statusCode);

    if (this.mService == null) {
        Log.e("BleServerService", "sendResponse(): error. GattService not
            available");
        return;
    }

    AttributeRequestInfo attrInfo = (AttributeRequestInfo) this.mAttrReqMap
        .remove(Integer.valueOf(transId));
    if (attrInfo == null)
    {
        Log.e("BleServerService",
            "sendResponse() error. attrInfo not found with transId " +
            transId);
        return;
    }

    byte[] dataToSend = null;
    if (attrInfo.mOffset == 0) {
        dataToSend = data;
    } else {
        dataToSend = new byte[data.length - attrInfo.mOffset];
        System.arraycopy(data, attrInfo.mOffset, dataToSend, 0, dataToSend.
            length);
    }
}

```



```

try
{
    this.mService
        .GATTServer_SendRsp(
            attrInfo.mConnId,
            attrInfo.mTransId,
            (byte) statusCode,
            attrInfo.mAttrHandle,
            attrInfo.mOffset,
            dataToSend,
            (byte) 0,
            false);
} catch (Throwable t)
{
    Log.e("BleServerService", "sendResponse(): error", t);
}
}

/*当客户端已经请求读或写一个本地特性属性后发送一个响应*/
public void sendResponse(String address, int transId, int handle, int offset,
    byte[] data,
    int statusCode, boolean isWrite)
{
    int connId = getConnId(address);
    if ((connId != -1) && (this.mService != null))
        try {
            this.mService.GATTServer_SendRsp(connId, transId, (byte) statusCode,
                handle, offset, data, (byte) 0, isWrite);
        } catch (Throwable t)
        {
            Log.e("BleServerService", "sendResponse", t);
        }
}

/*当本地属性改变时发送一个通知给客户端*/
public void sendNotification(String address, int attrHandle, byte[] value)
{
    int connId = getConnId(address);
    if ((connId != -1) && (this.mService != null))
        try {
            this.mService
                .GATTServer_HandleValueNotification(connId, attrHandle,
                    value);
        } catch (Throwable t) {
            Log.e("BleServerService", "sendNotification", t);
        }
}

/*发送一个指示到客户端*/
public void sendIndication(String address, int attrHandle, byte[] value)
{
    int connId = getConnId(address);

```

```

        if ((connId != -1) && (this.mService != null))
            try {
                this.mService.GATTServer_HandleValueIndication(connId, attrHandle,
                    value);
            } catch (Throwable t) {
                Log.e("BleServerService", "sendIndication", t);
            }
    }

    /*添加新特性到此服务*/
    private void addCharacteristic(BleCharacteristic charObj, boolean addtoQueue) {
        Log.i("BleServerService", "GATTServer_AddCharacteristic");
        ArrayList dirtyDescQueue = charObj.getDirtyDescQueue();

        if ((this.mService == null) || (charObj == null) || (charObj.getID()
            == null)) {
            Log
                .i("BleServerService",
                    "GattService/Characteristic object passed in is null..
                    Cannot add the chanaracteristic...");

            return;
        }
        try
        {
            if (charObj.isRegistered()) {
                Log.d("BleServerService", "Starting to add descriptors,
                    dirtyDesc size ="
                        + dirtyDescQueue.size());
                boolean dirtyMask = charObj.isDirty();
                if (!dirtyDescQueue.isEmpty())
                {
                    BleDescriptor descObj = (BleDescriptor) dirtyDescQueue.
                        get(0);
                    Log.i("BleServerService", "GATTServer_AddCharDescriptor");
                    this.mService.GATTServer_AddCharDescriptor(this.mSvcHandle,
                        descObj.getPermission(), descObj.mID);
                }
                else if (dirtyMask) {
                    Log.i("BleServerService", "GATTServer_AddCharValue");
                    HashMap<String, Integer> connMap = this.mProfileHandle.
                        getConnMap();

                    int clientCfg;
                    for (Map.Entry<String, Integer> entry : connMap.entrySet()) {
                        String address = (String) entry.getKey();
                        int connId = ((Integer) entry.getValue()).intValue();
                        clientCfg = 0;
                    }

                    return;
                }
            }
        }
    }

```

```

    }
    } else {
        if (addtoQueue) {
            synchronized (this) {
                this.mCharQueue.add(charObj);
                Log.e("BleServerService",
                    "Adding a new characteristic... SIZE IS " +
                    this.mCharQueue.size()
                    + "Uuid=" + charObj.getID());

                if (this.mCharQueue.size() > 1)
                    return;
            }
        }
        BleGattID uuid = charObj.getID();
        this.mService.GATTServer_AddCharacteristic(this.mSvcHandle, uuid,
            charObj.getPermission(), charObj.getProperty(), charObj.
            isDirty(),
            dirtyDescQueue.size());
    }
} catch (Throwable t)
{
    Log.e("BleServerService", "addCharacteristic", t);
}
}

/*这是一个回调函数，当添加一个包含的服务时触发*/
public void onIncludedServiceAdded(byte status, BleServerService
includedService)
{
    Log.d("BleServerService", "OnIncludedServiceAdded : status=" + status
        + "Included service" + includedService.getUuid());
}

/*当添加一个特性时调用*/
public void onCharacteristicAdded(byte status, BleCharacteristic charObj)
{
    Log.d("BleServerService", "OnCharacteristicAdded : Characteristic uuid = "
        + charObj.getID() + "status=" + status);
}

/*当一个读或写操作的响应已经发送时调用*/
public void onResponseSendCompleted(byte status, BleCharacteristic charObj)
{
    Log.d("BleServerService", "onResponseSendCompleted : status=" + status);
}

/*这是一个回调函数，当添加一个特性时调用*/
public void onCharacteristicRead(String address, int transId, int attrHandle,
BleCharacteristic charObj)
{
    AttributeRequestInfo attrInfo = (AttributeRequestInfo) this.mAttrReqMap
        .remove(Integer.valueOf(transId));

```

```
Log.d("BleServerService", "Inside onCharacteristicRead()");
if (attrInfo == null)
{
    Log.e("BleServerService",
        "onCharacteristicRead() error. attrInfo not found with
        transId " + transId);
    return;
}
if (charObj == null)
{
    Log.e("BleServerService", "onCharacteristicRead() error. charObj
    is null");
    return;
}

byte[] data = charObj.getValueByHandle(attrHandle);
if (data == null) {
    Log.d("BleServerService", "Attribute not found with handle " +
    attrHandle);
    try
    {
        this.mService.GATTServer_SendRsp(attrInfo.mConnId,
            attrInfo.mTransId,
            (byte) 10,
            attrInfo.mAttrHandle,
            attrInfo.mOffset, null,
            (byte) 0,
            false);
    } catch (Throwable t)
    {
        Log.e("BleServerService", "onCharacteristicRead(): error", t);
    }
    return;
}

int dataLength = data == null ? 0 : data.length;

if (attrInfo.mOffset >= dataLength) {
    Log.e("BleServerService",
        "onCharacteristicRead() error. dataLength < attrInfo.mOffset");
    try
    {
        this.mService.GATTServer_SendRsp(
            attrInfo.mConnId,
            attrInfo.mTransId,
            (byte) 7,
            attrInfo.mAttrHandle,
            attrInfo.mOffset,
            null,
            (byte) 0,
```

```

        false);
    } catch (Throwable t)
    {
        Log.e("BleServerService", "onCharacteristicRead(): error", t);
    }
    return;
}

byte[] dataToSend = null;
if (attrInfo.mOffset == 0) {
    dataToSend = data;
} else {
    dataToSend = new byte[data.length - attrInfo.mOffset];
    System.arraycopy(data, attrInfo.mOffset, dataToSend, 0, dataToSend.
        length);
}

try
{
    this.mService.GATTServer_SendRsp(attrInfo.mConnId, attrInfo.
        mTransId, (byte) 0,
        attrInfo.mAttrHandle, attrInfo.mOffset, dataToSend, (byte)
        0, false);
} catch (Throwable t)
{
    Log.e("BleServerService", "sendResponse(): error", t);
}
}

/*这是一个回调函数，当特性的写操作完成时被触发*/
public void onCharacteristicWrite(String address, BleCharacteristic charObj)
{
    Log.d("BleServerService", "onCharacteristicWrite : modified
        characteristic="
        + charObj.getID());
}

private class BleServiceCallback extends IBleServiceCallback.Stub {
    private BleServerService mGattService;

    public BleServiceCallback(BleServerService service) {
        this.mGattService = service;
    }
}

/*注册服务*/
public void onServiceRegistered(byte status, BluetoothGattID svcId) {
    Log.i("BleServerService", "onServiceRegistered");
    if (status == 0)
    {
        this.mGattService.setServiceInstance(svcId.getInstanceID());
        this.mGattService.createService();
    } else {

```



```

        Log.e("BleServerService", "#####Service registration failed...");
        BleServerService.this.mProfileHandle.notifyAction(1);
    }
}
/*创建服务*/
public void onServiceCreated(byte status, int svcHandle) {
    Log.i("BleServerService", "onServiceCreated");
    if (status == 0) {
        this.mGattService.setServiceHandle(svcHandle);
        BleServerService.this.mProfileHandle.notifyAction(0);
    } else {
        BleServerService.this.mProfileHandle.notifyAction(1);
    }
}
}

```

(7) 描述低功耗蓝牙服务的特性

在 Broadcom 公司提供的源码中，文件 `BleCharacteristic.java` 的功能是描述低功耗蓝牙服务的特性。在特性中包含了描述符、实际值和元数据，提供了表现格式或便于阅读值的描述。文件 `BleCharacteristic.java` 的主要实现代码如下。

```

public class BleCharacteristic extends BleAttribute
    implements Parcelable
{
    private static final String TAG = "BleCharacteristic";
    private HashMap<BleGattID, BleDescriptor> mDescriptorMap = new HashMap<BleGattID, BleDescriptor>();
    private ArrayList<BleDescriptor> mDirtyDescQueue = new ArrayList<BleDescriptor>();
    private int mProp;
    private int mWriteType;
    private byte mAuthReq;
    private int mPermission = 0;

    /** @hide */
    @SuppressWarnings({
        "rawtypes", "unchecked"
    })
    public static final Parcelable.Creator<BleCharacteristic> CREATOR = new
        Parcelable.Creator()
    {
        @Override
        public BleCharacteristic createFromParcel(Parcel source) {
            return new BleCharacteristic(source);
        }
    };

    /*获取 GATT 的 ID 值*/
    private BleGattID getBleGattId(int handle)
    {
        for (Map.Entry<BleGattID, Integer> entry : mHandleMap.entrySet()) {

```

```

        if (handle == entry.getValue().intValue()) {
            return entry.getKey();
        }
    }
    return null;
}

/**
 * 返回该特征的实例的 ID
 * 实例 ID 的 BLE 配置文件和服务用于标识属于一个给定的实例的服务或轮廓的特征
 */
public int getInstanceID()
{
    return mID.getInstanceID();
}

/**
 * 指定一个实例 ID 的这一特性
 */
* @see {@link #getInstanceID()}
*/
public void setInstanceID(int instanceID)
{
    mID.setInstanceId(instanceID);
}

/**
 * 根据特性向一个给定的偏移量设置原始值的字节
 */
public byte setValue(byte[] value, int offset, int len, int handle, int
totalsize,
    String address)
{
    int uuid = -1;
    int uuidType = -1;
    Log.e("BleCharacteristic", "#### handle is " + handle + " total size is "
        + totalsize);

    BleGattID gattUuid = getBleGattId(handle);
    if (gattUuid == null) {
        Log.e("BleCharacteristic", "setValue: Invalid handle");
        return BleConstants.GATT_INVALID_HANDLE;
    }

    if (gattUuid.equals(mID)) {
        Log.i("BleCharacteristic", "##Writing a characteristic value..");
        Log.i("BleCharacteristic", "##offset=" + offset + " mMaxLength="
            + mMaxLength + " totalsize=" + totalsize);
        return setValue(value, offset, len, gattUuid, totalsize, address);
    }
}

```

```

BleDescriptor descObj = mDescriptorMap.get(gattUuid);
if (descObj != null) {
    Log.i("BleCharacteristic", "##Writing descriptor value..");
    Log.i("BleCharacteristic",
        "##offset=" + offset + " mMaxSize=" + descObj.getMaxLength()
        + " totalsize="
        + totalsize + "desc uuid =" + descObj.getID());
    if (offset > descObj.getMaxLength())
        return BleConstants.GATT_INVALID_OFFSET;
    if (offset + totalsize > descObj.getMaxLength())
        return BleConstants.GATT_INVALID_ATTR_LEN;
    Log.i("BleCharacteristic", "find the user defined descriptor ");
    return descObj.setValue(value, offset, len, gattUuid, totalsize,
        address);
}
Log.e("BleCharacteristic", "Failed to write the value correctly!!!");
return -127;
}

/**
 * 一个给定的偏移量设置原始值的字节*/
@Override
public byte setValue(byte[] value, int offset, int len, BleGattID gattUuid,
    int totalsize,
    String address)
{
    return super.setValue(value, offset, len, gattUuid, totalsize, address);
}

/**
 * Gets the characteristic properties value (bit field).
 */
public int getProperty()
{
    return mProp;
}

/**
 * 设置特性属性集
 */
public void setProperty(int Prop)
{
    mProp = Prop;
}

/**
 * 获取一个基于 UUID 的描述符
 */
public BleDescriptor getDescriptor(BleGattID descriptor)
{

```

```

        BleDescriptor descObj = mDescriptorMap.get(descriptor);
        if (descObj != null) {
            return descObj;
        }

        return null;
    }

    /**
     * 添加一个描述符对象
     */
    public void addDescriptor(BleGattID descId, BleDescriptor descriptor)
    {
        Log.d("BleCharacteristic", "Inside add descriptor");
        mDescriptorMap.put(descId, descriptor);
        mDirtyDescQueue.add(descriptor);
        descriptor.setCharRef(this);
    }

    /**
     * 添加一个描述符对象
     */
    public void addDescriptor(BleDescriptor descriptor)
    {
        mDescriptorMap.put(descriptor.mID, descriptor);
        mDirtyDescQueue.add(descriptor);
        descriptor.setCharRef(this);
    }

    /**
     * 返回所有用户定义的、都包含在这一特性数组描述符
     */
    public ArrayList<BleDescriptor> getAllDescriptors()
    {
        ArrayList<BleDescriptor> descList = new ArrayList<BleDescriptor>();
        for (Map.Entry<BleGattID, BleDescriptor> entrySet : mDescriptorMap.
            entrySet()) {
            descList.add(entrySet.getValue());
        }
        return descList;
    }

    /**
     * 映射这个属性句柄值属性
     */
    public void addHandle(BleGattID uuid, int handle)
    {
        mHandleMap.put(uuid, Integer.valueOf(handle));
    }

```



```
/**
 * 返回一个对给定属性 ID 的处理结果
 */
public int getHandle(BleGattID uuid)
{
    Integer tmp;
    if ((tmp = mHandleMap.get(uuid)) != null) {
        return tmp.intValue();
    }
    return -1;
}

/**
 * 设置所需的读/写这个属性的验证水平
 */
@Override
public void setAuthReq(byte AuthReq)
{
    mAuthReq = AuthReq;
}

/**
 * 返回是否允许签写这个特性
 */
public boolean isAuthenticated()
{
    return (mProp & BleConstants.GATT_CHAR_PROP_BIT_AUTH) == BleConstants.
        GATT_CHAR_PROP_BIT_AUTH;
}

/**
 * 返回一个基于先前分配的句柄值这一特性属性
 */
@Override
public byte[] getValueByHandle(int handle)
{
    BleGattID gattUuid = getBleGattId(handle);
    if (gattUuid == null) {
        Log.w("BleCharacteristic", "Attribute UUID not found with handle "
            + handle);
        return null;
    }
    int uuidType = gattUuid.getUuidType();
    if (uuidType == BleConstants.GATT_UUID_TYPE_16)
        return getValueByUUID16(gattUuid);
    if (uuidType == BleConstants.GATT_UUID_TYPE_128) {
        return getValueByUUID128(gattUuid);
    }
    Log.w("BleCharacteristic", "Invalid UUID type.");
    return null;
}
```



```

/**
 *检索一个基于 16 位 UUID 这个特征的属性
 */
public byte[] getValueByUUID16(BleGattID uuid)
{
    int uuid16 = uuid.getUuid16();
    if (uuid16 == -1) {
        Log.w("BleCharacteristic", "Invalid UUID16.");
        return null;
    }

    int thisAttrUuid16 = mID == null ? -1 : mID.getUuid16();
    if (uuid16 == thisAttrUuid16)
    {
        return getValue();
    }

    BleDescriptor descObj = mDescriptorMap.get(uuid);

    if (descObj != null) {
        Log.d("BleCharacteristic", "Descriptor UUID = " + descObj.getID().
            getUuid16());
        return descObj.getValue();
    }

    Log.w("BleCharacteristic", "Attribute query not supported for uuid16
    value "
        + uuid16);
    return null;
}

/**
 *检索一个基于 128 位 UUID 这个特征的属性*/
public byte[] getValueByUUID128(BleGattID uuid)
{
    UUID uuid128 = uuid.getUuid();
    if (uuid128 == null) {
        return null;
    }

    if ((mID != null) && (uuid128.equals(mID.getUuid()))) {
        return getValue();
    }

    BleDescriptor descObj = mDescriptorMap.get(uuid);
    if (descObj != null) {
        return descObj.getValue();
    }

    Log.w("BleCharacteristic", "Attribute query not supported for uuid128

```

```

        value "
            + uuid128);
        return null;
    }
}

```

(8) 低功耗描述符

在 Broadcom 公司提供的源码中，文件 `BleDescriptor.java` 是 `BleCharacteristic` 的一部分，功能是定义了一个低功耗描述符。文件 `BleDescriptor.java` 的主要实现代码如下。

```

public class BleDescriptor extends BleAttribute
    implements Parcelable
{
    private static final String TAG = "BleDescriptor";
    private BleCharacteristic mCharObj;
    protected HashMap<String, Integer> mClientcfgMap = new HashMap();

    /** @hide */
    @SuppressWarnings({
        "unchecked", "rawtypes"
    })
    public static final Parcelable.Creator<BleDescriptor> CREATOR = new
        Parcelable.Creator()
    {
        public BleDescriptor createFromParcel(Parcel source) {
            return new BleDescriptor(source);
        }

        public BleDescriptor[] newArray(int size)
        {
            return new BleDescriptor[size];
        }
    };

    /**
     * 从一个给定的偏移设置原始值的字节的描述符
     *
     * @return {@link BleConstants#GATT_SUCCESS} if successful
     */
    @Override
    public byte setValue(byte[] value, int offset, int length, BleGattID
        gattUuid,
        int totalSize, String address)
    {
        int uuidType = gattUuid.getUuidType();
        int uuid = -1;
        Log.e("BleDescriptor", "#### UUID type=" + gattUuid.getUuidType());

        if (uuidType == 2) {
            uuid = gattUuid.getUuid16();

```

```

        if (uuid == -1) {
            Log.e("BleDescriptor", "setValue: Invalid handle (UUID16 not
            found)");
            return 1;
        }
    }
    if (uuid == 10500) {
        Log.i("BleDescriptor", "##Writing a Presentation format..");
    } else if (uuid == 10498) {
        Log.i("BleDescriptor", "##Writing a characteristic client config");
        if (totalSize > this.mMaxLength)
            return 13;
        int valueInt = 0;
        for (int i = 0; i < length; i++) {
            int shift = (length - 1 - i) * 8;
            valueInt += ((value[i] & 0xFF) << shift);
        }
        this.mClientcfgMap.put(address, Integer.valueOf(valueInt));
    } else if (gattUuid.equals(this.mID)) {
        Log.i("BleDescriptor", "##Writing a descriptor value..");
        Log.i("BleDescriptor", "##offset=" + offset + " mMaxLength=" +
        this.mMaxLength
            + " length=" + length);
        super.setValue(value, offset, length, gattUuid, totalSize, address);
    }
    this.mDirty = true;
    return 0;
}
}
}

```

(9) 标识低功耗蓝牙规范、服务和特性

在 Broadcom 公司提供的源码中, 文件 `BleGattID.java` 的功能是定义了一个标识低功耗蓝牙规范、服务和特性的类, 此类使用 16 位或 128 位的 UUIDs 来标识一个给定的低功耗蓝牙实体, 这个实体包含规范、服务和特性。文件 `BleGattID.java` 的主要实现代码如下。

```

/**
 *标识一个蓝牙 GATT 特性或属性
 */
public final class BleGattID extends BluetoothGattID
    implements Parcelable
{
    private static final String BASE_UUID_TPL = "%08x-0000-1000-8000-00805f9b34fb";
    @SuppressWarnings({
        "rawtypes", "unchecked"
    })
    public static final Parcelable.Creator<BleGattID> CREATOR = new Parcelable.
    Creator() {
        public BleGattID createFromParcel(Parcel source) {
            int instId = source.readInt();
            int type = source.readInt();
            int serviceType = source.readInt();

```

```
        if (type == 16) {
            String sUuid = source.readString();
            return new BleGattID(instId, sUuid, serviceType);
        }
        int uuid = source.readInt();
        return new BleGattID(instId, uuid, serviceType);
    }

    public BleGattID[] newArray(int size)
    {
        return new BleGattID[size];
    }
};

public BleGattID(int instId, UUID uuid)
{
    super(instId, uuid);
}

public BleGattID(int instId, UUID uuid, int serviceType) {
    super(instId, uuid, serviceType);
}

public BleGattID(int instId, long uuidLsb, long uuidMsb) {
    super(instId, uuidLsb, uuidMsb);
}

public BleGattID(long uuidLsb, long uuidMsb, int uuidType) {
    super(uuidLsb, uuidMsb, uuidType);
}

public BleGattID(int instId, int uuidType, long uuidLsb, long uuidMsb) {
    super(instId, uuidType, uuidLsb, uuidMsb);
}

public BleGattID(int instId, long uuidLsb, long uuidMsb, int serviceType) {
    super(instId, uuidLsb, uuidMsb, serviceType);
}

public BleGattID(int instId, int uuidType, long uuidLsb, long uuidMsb, int serviceType) {
    super(instId, uuidType, uuidLsb, uuidMsb, serviceType);
}

public BleGattID(int instId, String sUUID) {
    super(instId, sUUID);
}
```



```

public BleGattID(int instId, String sUUID, int serviceType) {
    super(instId, sUUID, serviceType);
}

public BleGattID(int instId, int uuid) {
    super(instId, uuid);
}

public BleGattID(int instId, int iUUID, int serviceType) {
    super(instId, iUUID, serviceType);
}

public BleGattID(UUID uuid) {
    super(uuid);
}

public BleGattID(UUID uuid, int serviceType) {
    super(uuid, serviceType);
}

public BleGattID(String sUUID) {
    super(sUUID);
}

public BleGattID(String sUUID, int serviceType) {
    super(sUUID, serviceType);
}

public BleGattID(int uuid) {
    super(uuid);
}

/**
 *得到的 UUID 型的 BleGattid 标识 (16 或 128 位)
 */
public UUID getUuid() {
    if (getUuidType() == BleConstants.GATT_UUID_TYPE_128)
        return super.getUuid();
    return UUID.fromString(String.format(BASE_UUID_TPL,
        new Object[] {
            Integer.valueOf(getUuid16())
        }));
}

public int getUuid16() {
    return super.getUuid16();
}

public int getUuidType()
{

```



```

        return super.getUuidType();
    }

    public void setInstanceId(int instanceId) {
        super.setInstId(instanceId);
    }

    public int getInstanceID() {
        return super.getInstanceID();
    }

    public int getServiceType() {
        return super.getServiceType();
    }

    public void setServiceType(int serviceType) {
        super.setServiceType(serviceType);
    }

    public int hashCode() {
        return getUuid().hashCode();
    }

    public boolean equals(Object target) {
        if (target == null) {
            return false;
        }

        if (this == target) {
            return true;
        }

        if (!(target instanceof BleGattID)) {
            return false;
        }

        BleGattID lhs = (BleGattID) target;
        return getUuid().equals(lhs.getUuid());
    }
}

```

(10) 为远程蓝牙设备提供额外信息

在 Broadcom 公司提供的源码中，文件 `BleAdapter.java` 的功能是为远程蓝牙设备提供了额外的信息，能够判断远程设备是否是低功耗设备、BR/EDR 传统蓝牙设备或双模设备（同时支持低功耗和传统设备）。文件 `BleAdapter.java` 的主要实现代码如下。

```

/**
 *提供帮助的功能和相关的常数扩展蓝牙功能的低能耗信息
 */
public class BleAdapter
{
    private static final String TAG = "BleAdapter";
}

```

```

private static final boolean D = true;

private static final int API_LEVEL = 5;
private static IBluetoothGatt mService;
private GattServiceConnection mSvcConn;
private Context mContext;

/**
 * 设置远程 ACTION_FOUND 设备的额外信息
 *
 * @see {@link #DEVICE_TYPE_BREDR}, {@link #DEVICE_TYPE_BLE},
 *      {@link #DEVICE_TYPE_DUMO}
 */
public static final String EXTRA_DEVICE_TYPE = "android.bluetooth.device.
extra.DEVICE_TYPE";

/**
 * Identifies a remote Bluetooth device as type BR/EDR, not capable of
 * accepting Bluetooth Low Energy connections.
 */
public static final byte DEVICE_TYPE_BREDR = 1;

/**
 * Designates a remote device as a Bluetooth Low Energy (only) device.
 */
public static final byte DEVICE_TYPE_BLE = 2;
public static final byte DEVICE_TYPE_DUMO = 3;
public static final String ACTION_UUID = "android.bluetooth.le.device.
action.UUID";

public static final String EXTRA_UUID = "android.bluetooth.le.device.
extra.UUID";
public static final String EXTRA_DEVICE = "android.bluetooth.le.device.
extra.DEVICE";

private static boolean startService() {
    if (mService != null)
        return true;
    IBinder service = ServiceManager.getService(BleConstants.BLUETOOTH_LE_
SERVICE);
    if (service != null)
        mService = IBluetoothGatt.Stub.asInterface(service);
    return mService != null;
}

/**
 * 构建一种新的 BleAdapter 对象
 */
public BleAdapter(Context ctx) {
    this.mContext = ctx;
}

```

```

        if (startService()==false)
            throw new RuntimeException("failed connecting to service");
        this.init();
    }

    public static boolean checkAPIAvailability() {
        return startService();
    }

    public static int getApiLevel()
    {
        return API_LEVEL;
    }

    /**
     * 返回蓝牙设备的类型 (LE, BR/EDR or dual-mode).
     *
     * @param device - The remote device who's type is to be determined
     * @return The type of the remote device
     * @see {@link #DEVICE_TYPE_BREDR}, {@link #DEVICE_TYPE_BLE},
     *      {@link #DEVICE_TYPE_DUMO}
     */
    public static byte getDeviceType(BluetoothDevice device)
    {
        if (!startService())
            throw new RuntimeException("service not available");
        if (device != null) {
            try {
                return mService.getDeviceType(device.getAddress());
            } catch (RemoteException e) {
                Log.e(TAG, "error", e);
            }
        }

        return 0;
    }

    /**
     * 启动远程设备中的蓝牙服务，发现使用{@link #ACTION_UUID}的意图
     *
     * @param deviceAddress - Bluetooth address of the remote device in
     *      00:11:22:33:44:55 format
     * @return true if the device discovery was started successfully
     */
    public static boolean getRemoteServices(String deviceAddress)
    {
        if (!startService())
            throw new RuntimeException("service not available");
        BluetoothAdapter adapter = BluetoothAdapter.getDefaultAdapter();
        if (adapter == null)
            return false;
    }

```

```

    try {
        mService.getUUIDs(deviceAddress);
        return true;
    } catch (RemoteException e) {
        e.printStackTrace();
        if (D)
            Log.e(TAG, "error", e);
    }
    return false;
}

/**
 * 初始化 BleAdapter 对象, 连接蓝牙 GATT 服务失效回调
 * service is made and the onInitialized(boolean) callback is invoked.
 */
public void init()
{
    /**
     * TODO: implement
     */
    Log.d("BleAdapter", "init");
    Intent i = new Intent();
    i.setClassName("com.broadcom.bt.app.system",
        "com.broadcom.bt.app.system.GattService");
    mContext.bindService(i, mSvcConn, 1);
}

public synchronized void finish()
{
    if (mSvcConn != null) {
        mContext.unbindService(mSvcConn);
        mSvcConn = null;
    }
}

public void setScanParameters(int scanInterval, int scanWindow)
{
    try
    {
        mService.setScanParameters(scanInterval, scanWindow);
    } catch (RemoteException e) {
        Log.e("BleAdapter", e.toString());
    }
}

void observeStart(int duration)
{
    try
    {
        if (mService != null)
            mService.observe(true, duration);
    }
}

```



```

        } catch (RemoteException e) {
            Log.d("BleAdapter", "Error calling observe: " + e.toString());
        }
    }

    void observeStop()
    {
        try
        {
            if (mService != null)
                mService.observe(false, 0);
        } catch (RemoteException e) {
            Log.d("BleAdapter", "Error calling observe: " + e.toString());
        }
    }

    void filterEnable(boolean enable)
    {
        try
        {
            if (mService != null)
                mService.filterEnable(enable);
        } catch (RemoteException e) {
            Log.d("BleAdapter", "Error calling filterEnable: " + e.toString());
        }
    }

    void filterEnableBDA(boolean enable, int addr_type, String address)
    {
        try
        {
            if (mService != null)
                mService.filterEnableBDA(enable, addr_type, address);
        } catch (RemoteException e) {
            Log.d("BleAdapter", "Error calling filterEnableBDA: " + e.toString());
        }
    }

    void filterManufacturerData(int company, byte[] data1, byte[] data2, byte[]
    data3,
        byte[] data4)
    {
        try
        {
            if (mService != null)
                mService.filterManufacturerData(company, data1, data2, data3,
                data4);
        } catch (RemoteException e) {
            Log.d("BleAdapter", "Error calling filterManufacturerData: " +
            e.toString());
        }
    }

```



```

    }
}

void filterManufacturerDataBDA(int company, byte[] data1, byte[] data2,
byte[] data3,
    byte[] data4, boolean has_bda, int addr_type, String address)
{
    try
    {
        if (mService != null)
            mService.filterManufacturerDataBDA(company, data1, data2,
                data3, data4,
                    has_bda, addr_type, address);
    } catch (RemoteException e) {
        Log.d("BleAdapter", "Error calling filterManufacturerDataBDA: " +
            e.toString());
    }
}

void clearManufacturerData()
{
    try
    {
        if (mService != null)
            mService.clearManufacturerData();
    } catch (RemoteException e) {
        Log.d("BleAdapter", "Error calling observe: " + e.toString());
    }
}

private class GattServiceConnection implements ServiceConnection
{
    private Context context;

    public GattServiceConnection(Context ctx) {
        context = ctx;
    }

    public void onServiceConnected(ComponentName name, IBinder service) {
        if (service != null)
            try {
                BleAdapter.this.mService =
                    IBluetoothGatt.Stub.asInterface(service);
                BleAdapter.this.onInitialized(true);
            } catch (Throwable t) {
                Log.e("BleAdapter", "Unable to get Binder to GattService", t);
                BleAdapter.this.onInitialized(false);
            }
    }

    public void onServiceDisconnected(ComponentName name) {

```

```

        Log.d("BleAdapter", "Disconnected from GattService!");
    }
}

```

(11) 保存和 GATT 相关的常量

在 Broadcom 公司提供的源码中，文件 `BleConstants.java` 的功能是定义保存各种和 GATT 相关的常量，这些常量用于表示各种实现低功耗功能函数的属性和返回值。文件 `BleConstants.java` 的主要实现代码如下。

```

public abstract class BleConstants
{
    public static final int GATT_UNDEFINED = -1;
    public static final int GATT_SERVICE_CREATION_SUCCESS = 0;
    public static final int GATT_SERVICE_CREATION_FAILED = 1;
    public static final int GATT_SERVICE_START_SUCCESS = 2;
    public static final int GATT_SERVICE_START_FAILED = 3;
    public static final int GATT_SERVICE_STOPPED = 4;
    public static final int SERVICE_UNAVAILABLE = 1;
    public static final int GATT_SERVICE_PRIMARY = 0;
    public static final int GATT_SERVICE_SECONDARY = 1;
    public static final int GATT_SERVER_PROFILE_INITIALIZED = 0;
    public static final int GATT_SERVER_PROFILE_UP = 1;
    public static final int GATT_SERVER_PROFILE_DOWN = 2;
    public static final int GATT_SUCCESS = 0;
    public static final int GATT_INVALID_HANDLE = 1;
    public static final int GATT_READ_NOT_PERMIT = 2;
    public static final int GATT_WRITE_NOT_PERMIT = 3;
    public static final int GATT_INVALID_PDU = 4;
    public static final int GATT_INSUF_AUTHENTICATION = 5;
    public static final int GATT_REQ_NOT_SUPPORTED = 6;
    public static final int GATT_INVALID_OFFSET = 7;
    public static final int GATT_INSUF_AUTHORIZATION = 8;
    public static final int GATT_PREPARE_Q_FULL = 9;
    public static final int GATT_NOT_FOUND = 10;
    public static final int GATT_NOT_LONG = 11;
    public static final int GATT_INSUF_KEY_SIZE = 12;
    public static final int GATT_INVALID_ATTR_LEN = 13;
    public static final int GATT_ERR_UNLIKELY = 14;
    public static final int GATT_INSUF_ENCRYPTION = 15;
    public static final int GATT_UNSUPPORT_GRP_TYPE = 16;
    public static final int GATT_INSUF_RESOURCE = 17;
    public static final int GATT_ILLEGAL_PARAMETER = 135;
    public static final int GATT_NO_RESOURCES = 128;
    public static final int GATT_INTERNAL_ERROR = 129;
    public static final int GATT_WRONG_STATE = 130;
    public static final int GATT_DB_FULL = 131;
    public static final int GATT_BUSY = 132;
    public static final int GATT_ERROR = 133;
    public static final int GATT_CMD_STARTED = 134;
    public static final int GATT_PENDING = 136;
}

```

```
public static final int GATT_AUTH_FAIL = 137;
public static final int GATT_MORE = 138;
public static final int GATT_INVALID_CFG = 139;
public static final byte GATT_AUTH_REQ_NONE = 0;
public static final byte GATT_AUTH_REQ_NO_MITM = 1;
public static final byte GATT_AUTH_REQ_MITM = 2;
public static final byte GATT_AUTH_REQ_SIGNED_NO_MITM = 3;
public static final byte GATT_AUTH_REQ_SIGNED_MITM = 4;
public static final int GATT_PERM_READ = 1;
public static final int GATT_PERM_READ_ENCRYPTED = 2;
public static final int GATT_PERM_READ_ENC_MITM = 4;
public static final int GATT_PERM_WRITE = 16;
public static final int GATT_PERM_WRITE_ENCRYPTED = 32;
public static final int GATT_PERM_WRITE_ENC_MITM = 64;
public static final int GATT_PERM_WRITE_SIGNED = 128;
public static final int GATT_PERM_WRITE_SIGNED_MITM = 256;
public static final byte GATT_CHAR_PROP_BIT_BROADCAST = 1;
public static final byte GATT_CHAR_PROP_BIT_READ = 2;
public static final byte GATT_CHAR_PROP_BIT_WRITE_NR = 4;
public static final byte GATT_CHAR_PROP_BIT_WRITE = 8;
public static final byte GATT_CHAR_PROP_BIT_NOTIFY = 16;
public static final byte GATT_CHAR_PROP_BIT_INDICATE = 32;
public static final byte GATT_CHAR_PROP_BIT_AUTH = 64;
public static final byte GATT_CHAR_PROP_BIT_EXT_PROP = -128;
public static final byte SVC_INF_INVALID = -1;
public static final int GATTC_TYPE_WRITE_NO_RSP = 1;
public static final int GATTC_TYPE_WRITE = 2;
public static final int GATT_FORMAT_RES = 0;
public static final int GATT_FORMAT_BOOL = 1;
public static final int GATT_FORMAT_2BITS = 2;
public static final int GATT_FORMAT_NIBBLE = 3;
public static final int GATT_FORMAT_UINT8 = 4;
public static final int GATT_FORMAT_UINT12 = 5;
public static final int GATT_FORMAT_UINT16 = 6;
public static final int GATT_FORMAT_UINT24 = 7;
public static final int GATT_FORMAT_UINT32 = 8;
public static final int GATT_FORMAT_UINT48 = 9;
public static final int GATT_FORMAT_UINT64 = 10;
public static final int GATT_FORMAT_UINT128 = 11;
public static final int GATT_FORMAT_SINT8 = 12;
public static final int GATT_FORMAT_SINT12 = 13;
public static final int GATT_FORMAT_SINT16 = 14;
public static final int GATT_FORMAT_SINT24 = 15;
public static final int GATT_FORMAT_SINT32 = 16;
public static final int GATT_FORMAT_SINT48 = 17;
public static final int GATT_FORMAT_SINT64 = 18;
public static final int GATT_FORMAT_SINT128 = 19;
public static final int GATT_FORMAT_FLOAT32 = 20;
public static final int GATT_FORMAT_FLOAT64 = 21;
public static final int GATT_FORMAT_SFLOAT = 22;
```

```
public static final int GATT_FORMAT_FLOAT = 23;
public static final int GATT_FORMAT_DUINT16 = 24;
public static final int GATT_FORMAT_UTF8S = 25;
public static final int GATT_FORMAT_UTF16S = 26;
public static final int GATT_FORMAT_STRUCT = 27;
public static final int GATT_FORMAT_MAX = 28;
public static final String GATT_UUID_CHAR_EXT_PROP = "00002900-0000-1000-8000-00805f9b34fb";
public static final String GATT_UUID_CHAR_DESCRIPTION = "00002901-0000-1000-8000-00805f9b34fb";
public static final String GATT_UUID_CHAR_CLIENT_CONFIG = "00002902-0000-1000-8000-00805f9b34fb";
public static final String GATT_UUID_CHAR_SRVR_CONFIG = "00002903-0000-1000-8000-00805f9b34fb";
public static final String GATT_UUID_CHAR_PRESENT_FORMAT = "00002904-0000-1000-8000-00805f9b34fb";
public static final String GATT_UUID_CHAR_AGG_FORMAT = "00002905-0000-1000-8000-00805f9b34fb";
public static final int GATT_UUID_CHAR_EXT_PROP16 = 10496;
public static final int GATT_UUID_CHAR_DESCRIPTION16 = 10497;
public static final int GATT_UUID_CHAR_CLIENT_CONFIG16 = 10498;
public static final int GATT_UUID_CHAR_SRVR_CONFIG16 = 10499;
public static final int GATT_UUID_CHAR_PRESENT_FORMAT16 = 10500;
public static final int GATT_UUID_CHAR_AGG_FORMAT16 = 10501;
public static final int GATT_TRANSPORT_BREDR_LE = 2;
public static final int GATT_TRANSPORT_BREDR = 1;
public static final int GATT_TRANSPORT_LE = 0;
public static final int GATT_UUID_TYPE_128 = 16;
public static final int GATT_UUID_TYPE_32 = 4;
public static final int GATT_UUID_TYPE_16 = 2;
public static final int PREPARE_QUEUE_SIZE = 200;
public static final int GATT_MAX_CHAR_VALUE_LENGTH = 100;
public static final int GATT_CLIENT_CONFIG_NOTIFICATION_BIT = 1;
public static final int GATT_CLIENT_CONFIG_INDICATION_BIT = 2;
public static final int GATT_INVALID_CONN_ID = 65535;
public static final int VALUE_DIRTY = 1;
public static final int USER_DESCRIPTION_DIRTY = 2;
public static final int EXT_PROP_DIRTY = 4;
public static final int PRESENTATION_FORMAT_DIRTY = 8;
public static final int CLIENT_CONFIG_DIRTY = 16;
public static final int SERVER_CONFIG_DIRTY = 32;
public static final int AGGREGATED_FORMAT_DIRTY = 64;
public static final int USER_DESCRIPTOR_DIRTY = 128;
public static final int ALL_DIRTY = 127;
public static final byte GATT_ENCRYPT_NONE = 0;
public static final byte GATT_ENCRYPT = 1;
public static final byte GATT_ENCRYPT_NO_MITM = 2;
public static final byte GATT_ENCRYPT_MITM = 3;
static final String ACTION_OBSERVE_RESULT = "com.broadcom.bt.app.gatt.OBSERVE_RESULT";
```

```
static final String ACTION_OBSERVE_COMPLETED = "com.broadcom.bt.app.gatt.OBSERVE_COMPLETED";
static final String EXTRA_ADDRESS = "ADDRESS";
static final String EXTRA_ADDR_TYPE = "ADDR_TYPE";
static final String EXTRA_RSSI = "RSSI";
static final String EXTRA_ADV_DATA = "ADV_DATA";
static final String EXTRA_NUM_RESULTS = "NUM_RESULTS";
static final String GATT_SVC_PKG_NAME = "com.broadcom.bt.app.system";
static final String GATT_SVC_NAME = "com.broadcom.bt.app.system.GattService";

/**
 * @hide
 */
public static final String BLUETOOTH_LE_SERVICE = "com.manuelnaranjo.btle";
}
```

至此为止，Broadcom 公司推出的低功耗蓝牙协议栈 BlueDroid 的开发文档和 API 源码分析完毕。因为本书篇幅的限制，只分析了主要的模块类，对于其他类的实现代码的功能和原理，读者可参阅其源码中的注释说明。

项目实战—— 开发智能心率计

近几年以来，各大科技巨头都纷纷推出了智能可穿戴设备。随着人们对新技术接受度的提高，可穿戴设备必将成为未来人们生活的必需品之一。开发可穿戴设备需要掌握相关的硬件和软件技术。在本章的内容中，将详细讲解开发 Android 智能心率计的基本知识，为读者轻松步入现实工作打下基础。

13.1 什么是心率

心率 (Heart Rate): 用来描述心动周期的专业术语，是指心脏每分钟跳动的次数，以第一声音为准。现代汉语将心率解释为“心脏跳动的频率”。频率就是在单位时间内，某件事情发生的次数。两种解释合起来就是，心脏在一定时间内跳动的次数，也就是在一定时间内，心脏跳动快慢的意思。

据科学研究发现，正常成年人安静时的心率有显著的个体差异，平均在 75 次/分左右(60~100 次/分之间)。心率可因年龄、性别及其他生理情况的不同而不同。初生儿的心率很快，可达 130 次/分以上。在成年人中，女性的心率一般比男性稍快。同一个人，在安静或睡眠时心率减慢，运动时或情绪激动时心率加快，在某些药物或神经体液因素的影响下，会使心率发生加快或减慢。经常进行体力劳动和体育锻炼的人，平时心率较慢。近年，国内大样本健康人群调查发现：国人男性静息心率的正常范围为 50~95 次/分，女性为 55~95 次/分。所以，心率随年龄、性别和健康状况变化而变化。

健康成人的心率为 60~100 次/分，大多数为 60~80 次/分，女性稍快；3 岁以下的小儿常在 100 次/分以上；老年人偏慢。成人每分钟心率超过 100 次（一般不超过 160 次/分）或婴幼儿超过 150 次/分者，称为窦性心动过速。常见于正常人运动、兴奋、激动、吸烟、饮酒和喝浓茶后。也可见于发热、休克、贫血、甲亢、心力衰竭及应用阿托品、肾上腺素、麻黄素等。如果心率为 160~220 次/分，常称为阵发性心动过速。心率低于 60 次/分者（一般在 40 次/分以上），称为窦性心动过缓。可见于长期从事重体力劳动和运动员；病理性的见于甲状腺机能低下、颅内压增高、阻塞性黄疸、以及洋地黄、奎尼丁或心得安类药物过量或中毒。如心率低于 40 次/分，应考虑有房室传导阻滞。心率过快超过 160 次/分，或低于 40 次/分，大多见于心脏病病人，病人常有心悸、胸闷、心前区不适，应及早进行详细检查，以便

针对病因进行治疗。心脏每次收缩时由心室向动脉输出的血量叫作每搏输出量，心脏每分钟输出的血量叫作每分输出量，正常人在安静状态下每搏输出量为 70 毫升，如果心率按每分钟 75 次计算，每分输出量约为 5 250 毫升。心输出量的多少，是衡量心脏工作能力的一项指标。

13.2 什么是心率表

所谓心率表，就是在运动过程中能够实时准确记录我们运动心率的手表，心率表在有目的性锻炼中的作用非常明显。我们都知道运动量越大心跳越快这个简单的道理，运动时，通过监测心率观察便可带来事半功倍的效果。心率表分为两大类，有胸带心率表和无胸带心率表。有胸带的心率表有 SUUNTO、POLAR 和 FITBOX 等，这些都是通过佩戴在胸口的胸带上的传感器检测心跳并无线传递给手表并显示的。由于其直接放置在胸口上，所以其检测的准确性比较高也可以实时测量心率，是被广泛运用的一种心率表。无胸带的心率表的形式有 MIO，则是通过表背上或者表面的传感器来检测脉搏的。其特点是比较方便，但是准确性比较差。

心率表的测量原理常见的有两种，一种是心动电流测量法，还有一种是光电透射测量法。

(1) 心动电流测量法

人体每次心跳都会产生心动电流，无线心率胸带就是这样一种可以感应心动电流的仪器。感应器的极片位于胸带前方两侧，使用者带上胸带后，胸带内的极片采集锻炼者的心动电流波动幅度，再通过无线传输技术发送给心率表转化为便于观察的心跳 BPM 数值，目前这个是主流也是比较准确的运动心率测量方法。其原理和心电图测量原理一致。这种测量心率方法的另外一个优点是可以在运动中持续测量心率。

(2) 光电透射测量法

光电测量方法是利用血液中血红蛋白的吸光度的变化来测量脉搏的。手表装有红外发射光束回路和接收反射回路。这种方法测量心率的优点是非常简便且无须胸带，但是由于信号极为微弱且非常容易受到外界干扰而造成测量数据不准确，一般需要安静的状态下测量，不适合运动中持续测量心率。

13.3 开发一个 Android 版测试心率系统

实 例	功 能	源码路径
实例 13-1	心率测试系统	光盘:\daima\13\xinlv

本实例的功能是，在 Android 系统中开发一个心率测试应用程序。在使用时，只要将食指放在摄像头 10 秒钟后就可以测试出心率。具体原理是：每次心跳都会使血液里的氧含量增加，当身体消耗掉之后血液的氧含量又会降低，所以血液的颜色由于氧含量的变化会发生周期性的改变。由此可见，本应用程序是通过记录手指透过的光的颜色的改变来记录心率数据的。在使用本应用程序时，一定要有充足的光线，并且在将手指放在摄像头之后使之

朝向阳光或者灯光。如果有闪光灯的手持设备，还可以打开闪光灯进行测量。当获得具体结果时，使用绘图机制在设备屏幕中绘制一个心率表。

在本节的内容中，将详细讲解本实例的具体实现流程。

13.3.1 系统主界面

本系统的主界面 Activity 是 MainActivity，界面布局文件是 activity_main.xml，具体实现代码如下。

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    tools:context=".MainActivity" >

    <LinearLayout
        android:id="@+id/root"
        android:layout_width="fill_parent"
        android:layout_height="150dip"
        android:layout_alignParentRight="true"
        android:background="#000000"
        android:orientation="vertical" />

    <Button
        android:id="@+id/close"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/root"
        android:layout_marginRight="32dp"
        android:layout_marginTop="39dp"
        android:text="end" />

    <Button
        android:id="@+id/open"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/close"
        android:layout_alignBottom="@+id/close"
        android:layout_marginRight="29dp"
        android:layout_toLeftOf="@+id/close"
        android:text="start" />

    <TextView
        android:id="@+id/heart_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/open"
        android:layout_below="@+id/root"
        android:layout_marginLeft="32dp"
```

```

/>
</RelativeLayout>

```

通过上述代码，在屏幕中分别通过 Button 控件显示了 open 按钮和 close 按钮，分别用于开启心率测试功能和关闭心率测试功能。通过 TextView 控件显示测试的心率值。并且还通过文件 main.xml 构建了一个列表区域，在区域中用于显示心率表的值。文件 main.xml 的具体实现代码如下。

```

<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:title="@string/action_settings"/>
</menu>

```

主界面 Activity 的实现文件是 MainActivity.java，具体实现流程如下。

- (1) 定义定时分析摄像头数据的定时器函数 Timer()，以固定周期绘制心率曲线。
 - (2) 启动摄像头和闪光灯捕获心率数据。
 - (3) 通过 setOnClickListener 监听关闭摄像头事件，在获取并设置摄像头参数后关闭闪光灯。
 - (4) 编写函数 onPause()，用于当单击 close 按钮时停止测试工作。
 - (5) 编写函数 startTimer()，用于当单击 open 按钮时启动测试工作。
 - (6) 编写函数 onPreviewFrame 实现摄像头数据回调接口，实现获取摄像头数据尺寸的功能。如果获取不到摄像头数据尺寸，则抛出对应的异常。
 - (7) 编写函数 init()来初始化曲线图。
- 文件 MainActivity.java 的具体实现代码如下。

```

package com.xinlv;

import java.util.Timer;
import java.util.TimerTask;

import com.xinlv.R;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.hardware.Camera;
import android.hardware.Camera.Parameters;
import android.hardware.Camera.PreviewCallback;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.Menu;
import android.view.SurfaceHolder;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

```

```

import android.widget.LinearLayout;
import android.widget.TextView;

public class MainActivity extends Activity implements SurfaceHolder.Callback,
PreviewCallback{

    private DrawChart view;
    /**
     * 定时分析摄像头数据的定时器
     */
    private Timer timer = new Timer();
    private TimerTask task;
    public static TextView textView;
    @SuppressWarnings("HandlerLeak")
    private Handler handler=new Handler(){
        public void handleMessage(Message msg) {
            //调用相机回调接口由于 MainActivity 已经实现了回调接口，所以 MainActivity.
            this 即可
            camera.setOneShotPreviewCallback(MainActivity.this);
            view.invalidate();
        };
    };

    private static Camera camera = null;
    private Button openCamer,closeCamer;
    private static int heartbeat=0;
    public static void setHeartbeat(int heart){
        heartbeat=heart;
    }
    public static int getHeartbeat(){
        return heartbeat;
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //通过 findViewById 控件
        openCamer=(Button) findViewById(R.id.open);
        closeCamer=(Button) findViewById(R.id.close);
        textView=(TextView) findViewById(R.id.heart_text);
        openCamer.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                //摄像头开始捕获预览帧数据
                camera.startPreview();
                //获取相机当前设置参数
                Camera.Parameters parameter = camera.getParameters();
                //启动闪光灯
                parameter.setFlashMode(Parameters.FLASH_MODE_TORCH);
                camera.setParameters(parameter);
            }
        });
    }
}

```



```

        //开启定时器
        startTimer();
    }
});
closeCamer.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        //获取摄像头参数
        Camera.Parameters parameter = camera.getParameters();
        //设置摄像头参数 关闭闪光灯
        parameter.setFlashMode(Parameters.FLASH_MODE_OFF);

        camera.setParameters(parameter);
        pauseTimer();
    }
});
init();
}

public void onPause() {
    super.onPause();
    //暂停定时器
    pauseTimer();
    //将相机的回调接口设为 NULL 即不再接受回调数据
    camera.setPreviewCallback(null);
    camera.stopPreview();
    //断开相机并释放对象
    camera.release();
    camera = null;
}
/**
 * 开始测试方法
 */
public void startTimer(){
    if(timer==null){
        timer=new Timer();
    }
    if(task==null){
        task = new TimerTask() {
            @Override
            public void run() {
                // TODO Auto-generated method stub
                Message message = new Message();
                message.what = 1;
                handler.sendMessage(message);
            }
        };
    }
    if(timer!=null&&task!=null){
        timer.schedule(task, 500, 500);
    }
}

```

```

    }
    /**
     * 结束测试方法
     */
    public void pauseTimer(){
        if(timer!=null){
            timer.cancel();
            timer=null;
        }
        if(task!=null){
            task.cancel();
            task=null;
        }
    }
    @Override
    protected void onResume() {
        // TODO Auto-generated method stub
        super.onResume();
        //打开摄像头
        camera = Camera.open();
    }
    @Override
    public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3) {
        // TODO Auto-generated method stub
    }
    @Override
    public void surfaceCreated(SurfaceHolder arg0) {
        // TODO Auto-generated method stub
    }
    @Override
    public void surfaceDestroyed(SurfaceHolder arg0) {
        // TODO Auto-generated method stub
    }
    /**
     * 摄像头数据回调接口
     */
    @Override
    public void onPreviewFrame(byte[] data, Camera camera) {
        Log.i("onPreviewFrame", "onPreviewFrame 数据接口回调");
        if (data == null) throw new NullPointerException();
        //获取摄像头数据的尺寸
        Camera.Size size = camera.getParameters().getPreviewSize();
        //如果获取不到摄像头数据尺寸抛出异常
        if (size == null) throw new NullPointerException();
        //获取摄像头数据长度与高度
        int width = size.width;
        int height = size.height;
        //接口回调并获取到相机数据后调用 YUV 转换 RGB 方法将数据进行转换
    }

```

```

int imgAvg = ImageProcessing.decodeYUV420SPtoRedAvg(data.clone(),
height, width);
Log.i("imgAvg", "imgAvg===== "+imgAvg);
//限定心率值范围, 只有值在合理的范围内也就是在 40 到 150 之间才会显示出来
if (imgAvg > 40 & imgAvg < 151) {
    //将 Imagvg 写入图像方法中做出不规则波线
    setHeraTbeat(imgAvg);
    textView.setText(""+imgAvg+"次/m");
}else{
    setHeraTbeat(0);
    textView.setText("请将手指覆盖摄像头");
}
}

private void init() {
    //初始化曲线图
    LinearLayout layout=(LinearLayout) findViewById(R.id.root);
    view = new DrawChart(this);
    view.invalidate();
    layout.addView(view);
}
}

```

13.3.2 绘制心率表

编写文件 DrawChart.java, 功能是绘制心率曲线形成一个心率表。此文件的核心功能是通过函数 drawTable()实现的, 此函数可以绘制出波浪线的表格及虚线, 并且在曲线表中绘制具体的数值。文件 DrawChart.java 的具体实现代码如下。

```

package com.xinlv;

import java.util.ArrayList;
import java.util.List;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.DashPathEffect;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.PathEffect;
import android.graphics.Point;
import android.graphics.Rect;
import android.util.Log;
import android.view.View;

public class DrawChart extends View {
    private int CHARTH = 200;
    private int CHARTW = 460;
    private int OFFSET_LEFT = 10;
    private int OFFSET_TOP = 20;
    private int X_INTERVAL = 20;

```

```

private List<Point> plist;
MainActivity activity;
public DrawChart(Context context) {
    super(context);
    plist = new ArrayList<Point>();
    activity=(MainActivity) context;
    Log.i("DrawChart", "DrawChart1");
    //initPlist();
}

@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    Log.i("DrawChart", "onDraw2");
    drawTable(canvas);
    prepareLine();
    drawCurve(canvas);
}
/**
 * 画出波浪线的表格及虚线
 * @param canvas
 */
private void drawTable(Canvas canvas){
    Log.i("DrawChart", "drawTable3");
    Paint paint = new Paint();
    paint.setColor(Color.WHITE);

    //画外框
    paint.setStyle(Paint.Style.STROKE);
    Rect chartRec = new Rect(OFFSET_LEFT, OFFSET_TOP, CHARTW+OFFSET_LEFT,
    CHARTH+OFFSET_TOP);
    canvas.drawRect(chartRec, paint);

    //画左边的文字
    Path textPath = new Path();
    paint.setStyle(Paint.Style.FILL);
    textPath.moveTo(30, 420);
    textPath.lineTo(30, 300);
    paint.setTextSize(15);
    paint.setAntiAlias(true);
    canvas.drawTextOnPath("信号强度 [dBm]", textPath, 0, 0, paint);
    //画表格中的虚线
    Path path = new Path();
    PathEffect effects = new DashPathEffect(new float[]{2,2,2,2},1);
    paint.setStyle(Paint.Style.STROKE);
    paint.setAntiAlias(false);
    paint.setPathEffect(effects);
    for(int i = 1 ; i<10 ; i++){
        path.moveTo(OFFSET_LEFT, OFFSET_TOP + CHARTH/10*i);
    }
}

```

```

        path.lineTo(OFFSET_LEFT+CHARTW,OFFSET_TOP + CHARTH/10*i);
        canvas.drawPath(path, paint);
    }
}

private void drawCurve(Canvas canvas){
    Log.i("DrawChart", "drawCurve4");
    Paint paint = new Paint();
    paint.setColor(Color.WHITE);
    paint.setStrokeWidth(3);
    paint.setAntiAlias(true);
    if(plist.size() >= 2){
        for(int i = 0; i<plist.size()-1; i++){
            canvas.drawLine(plist.get(i).x, plist.get(i).y, plist.get(i+1).
                x, plist.get(i+1).y, paint);
        }
    }
}

private void prepareLine(){
    Log.i("DrawChart", "prepareLine5");
    int py=120;
    if(activity.getHeartbeat()!=0&&activity.getHeartbeat()>30&&activity.
        getHeartbeat()<150){
        py = OFFSET_TOP + (int) (Math.random()*(activity.getHeartbeat() -
            OFFSET_TOP));
    }
    Point p = new Point(OFFSET_LEFT + CHARTW,py);

    if(plist.size() > 24){
        Log.i("prepareLine", "plist==" +plist.get(0).x);
        plist.remove(0);
        for(int i = 0; i<23; i++){
            if(i == 0) plist.get(i).x -= (X_INTERVAL - 2);
            else plist.get(i).x -= X_INTERVAL;
        }
        plist.add(p);
    }
    else{
        for(int i = 0; i<plist.size()-1; i++){
            plist.get(i).x -= X_INTERVAL;
        }

        plist.add(p);
    }
}
}
}

```

然后编写文件 ImageProcessing.java 来优化绘制心率曲线的过程,此文件是通过图像线程的调度实现的。文件 ImageProcessing.java 的具体实现代码如下。


```

package com.xinlv;

import android.util.Log;

public abstract class ImageProcessing {
    private static int decodeYUV420SPtoRedSum(byte[] yuv420sp, int width, int
height) {
        if (yuv420sp==null) return 0;

        final int frameSize = width * height;

        int sum = 0;
        for (int j = 0, yp = 0; j < height; j++) {
            int uvp = frameSize + (j >> 1) * width, u = 0, v = 0;
            for (int i = 0; i < width; i++, yp++) {
                int y = (0xff & ((int) yuv420sp[yp])) - 16;

                if (y < 0) y = 0;

                if ((i & 1) == 0) {
                    v = (0xff & yuv420sp[uvp++]) - 128;
                    u = (0xff & yuv420sp[uvp++]) - 128;
                }
                int r=(int) (y+1.14*v);
                int g=(int) (y - 0.394*u - 0.581*v);
                int b=(int) (u+2.203*v);
                if(r>170){
                    sum+=(int) ((r * 0.3)+(g * 0.59) + (b * 0.11));
                }
            }
        }
        return sum;
    }

    public static int decodeYUV420SPtoRedAvg(byte[] yuv420sp, int width,
int height) {
        if (yuv420sp==null) return 0;
        final int frameSize = width * height;
        int sum = decodeYUV420SPtoRedSum(yuv420sp,width,height);
        Log.i("decode", "decode==" +sum);
        Log.i("decode", "frameSize==" +frameSize);
        sum=sum/frameSize;
        return sum;
    }

    //这个公式用来算亮度
    public static double getRGB(BufferedImage image){
        int x=image.getWidth();
        int y=image.getHeight();

```

```

        long sum=0;
        for(int i=1;i<x;i=i+2){
            for(int j=1;j<y;j=j+2){
                //循环获取每个像素的 RGB 值
                int rgb=image.getRGB(i, j);
                //单独获取 R,G,B
                int r =(rgb & 0xff0000 ) >> 16 ;
                int g= (rgb & 0xff00 ) >> 8 ;
                int b= (rgb & 0xff );
                sum+=((r*0.3)+(g*0.59)+(b*0.11));
            }
        }
        return sum;
    }
}

//这个算峰值数, 15 秒里探测的峰值数 count 乘以 4, 就是一分钟的心率
public int getCount(){
    int count=0;
    for (int i=0;i<imageRGB.size();i++) {
        if(i>2&&i<imageRGB.size()-2){
            if(imageRGB.get(i)>imageRGB.get(i+1)&&imageRGB.get(i)>
                imageRGB.get(i-1)){
                count++;
            }
        }
    }
    return count;
}*/
}
}

```

最后, 需要在文件 `AndroidManifest.xml` 中开启 `CAMERA` 权限, 具体实现代码如下。

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.xinlv"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
    <uses-permission android:name="android.permission.FLASHLIGHT" />
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-feature android:name="android.hardware.camera" />
    <uses-feature android:name="android.hardware.autofocus"/>
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.xinlv.MainActivity"
            android:label="@string/app_name" >

```

```

<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
</application>
</manifest>

```

至此，整个实例介绍完毕，执行后的效果如图 13-1 所示。

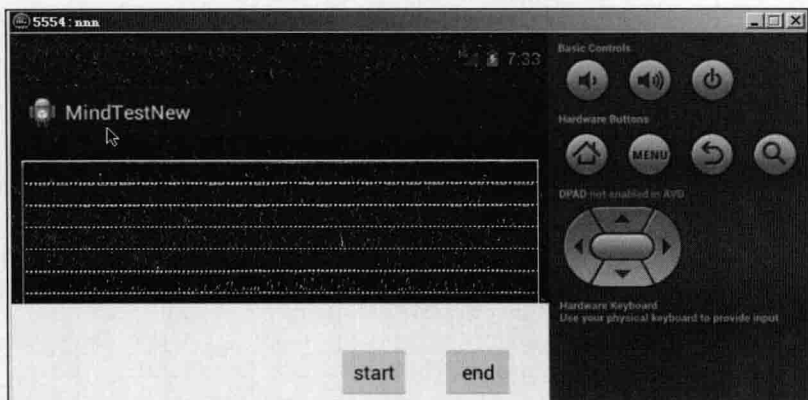


图 13-1 执行效果

注意

本实例只有在真机中测试运行，才能得到心率表效果。



项目实战—— 开发计步器

随着人们生活水平的日益提高，高血压、高血脂和高血糖患者越来越多。2008 奥运会在北京的成功举办，使全民健身潮深入民心。如果拥有一款科学的穿戴设备——计步器，不仅是一件新潮的事情，还可用来科学健身。在本章的内容中，将详细讲解开发一个 Android 计步器的基本知识，为读者轻松步入现实工作岗位打下基础。

14.1 系统功能模块介绍



本章计步器系统的功能是，统计穿戴者徒步行走的步数和距离，计算行走速度和消耗的热量。并且还具有启动和关闭功能，还可以对整个系统进行灵活设置。本章计步器系统的构成模块结构如图 14-1 所示。

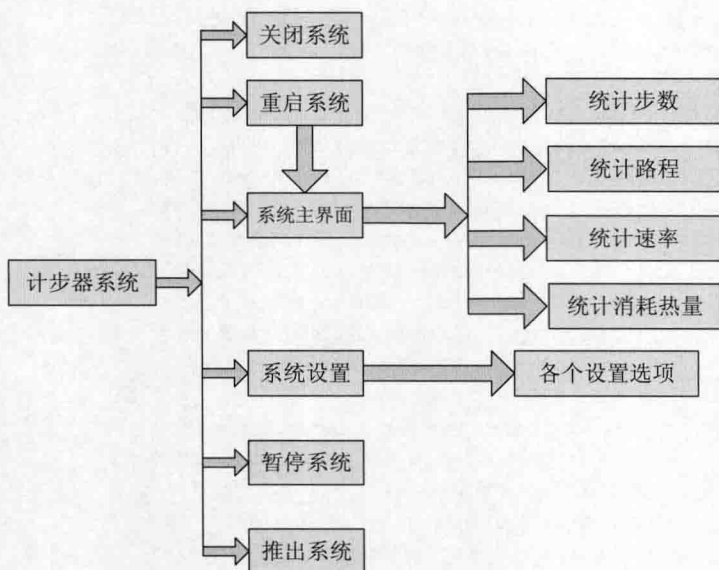


图 14-1 计步器系统的构成模块结构

14.2 系统主界面

系统主界面是运行程序后首先呈现在用户面前的界面。在本节的内容中，将详细讲解系统主界面的具体实现流程。

14.2.1 布局文件

本系统主界面的布局文件是 `main.xml`，具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="@dimen/margin"
    android:background="@color/screen_background">

    <LinearLayout android:id="@+id/row_1"
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="@dimen/row_spacing">

        <LinearLayout android:id="@+id/box_steps"
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:gravity="center_horizontal"
            android:paddingRight="@dimen/margin"
            android:layout_weight="1">

            <TextView android:id="@+id/step_value"
                android:textSize="@dimen/value"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:gravity="center_horizontal"
                android:background="@color/display_background"
                android:paddingLeft="@dimen/padding"
                android:paddingRight="@dimen/padding"
                android:paddingTop="@dimen/padding"
                android:text=""/>

            <TextView android:id="@+id/step_units"
                android:gravity="center_horizontal"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:textSize="@dimen/units"
                android:text="@string/steps"
                android:background="@color/display_background"
                android:paddingBottom="@dimen/padding"/>
        </LinearLayout>
    </LinearLayout>
</LinearLayout>
```



```

</LinearLayout>

<LinearLayout android:id="@+id/box_distance"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal"
    android:layout_weight="1">

    <TextView android:id="@+id/distance_value"
        android:textSize="@dimen/value"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:background="@color/display_background"
        android:paddingTop="@dimen/padding"
        android:paddingRight="@dimen/padding"
        android:paddingLeft="@dimen/padding"
        android:text=""/>

    <TextView android:id="@+id/distance_units"
        android:gravity="center_horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="@dimen/units"
        android:text="@string/kilometers"
        android:background="@color/display_background"
        android:paddingBottom="@dimen/padding"/>

</LinearLayout>
</LinearLayout>

<LinearLayout android:id="@+id/row_2"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:paddingBottom="@dimen/row_spacing">

    <LinearLayout android:id="@+id/box_pace"
        android:orientation="vertical"
        android:layout_height="wrap_content"
        android:paddingRight="@dimen/margin"
        android:layout_width="fill_parent"
        android:layout_weight="1">

        <TextView android:id="@+id/pace_value"
            android:gravity="center_horizontal"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:background="@color/display_background"

```

```

        android:textSize="@dimen/small_value"
        android:paddingLeft="@dimen/padding"
        android:paddingRight="@dimen/padding"
        android:paddingTop="@dimen/padding"
        android:text=""/>
<TextView android:id="@+id/pace_units"
    android:gravity="center_horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="@dimen/units"
    android:text="@string/steps_per_minute"
    android:paddingBottom="@dimen/padding"
    android:background="@color/display_background"/>

```

```
</LinearLayout>
```

```

<LinearLayout android:id="@+id/box_speed"
    android:orientation="vertical"
    android:paddingRight="@dimen/margin"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:layout_weight="1">

```

```

    <TextView android:id="@+id/speed_value"
        android:gravity="center_horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@color/display_background"
        android:textSize="@dimen/small_value"
        android:paddingLeft="@dimen/padding"
        android:paddingRight="@dimen/padding"
        android:paddingTop="@dimen/padding"
        android:text=""/>

```

```

    <TextView android:id="@+id/speed_units"
        android:gravity="center_horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="@dimen/units"
        android:text="@string/kilometers_per_hour"
        android:paddingBottom="@dimen/padding"
        android:background="@color/display_background"/>

```

```
</LinearLayout>
```

```

<LinearLayout android:id="@+id/box_calories"
    android:orientation="vertical"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:layout_weight="1">

```

```

    <TextView android:id="@+id/calories_value"

```

```

        android:gravity="center_horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@color/display_background"
        android:textSize="@dimen/small_value"
        android:paddingLeft="@dimen/padding"
        android:paddingRight="@dimen/padding"
        android:paddingTop="@dimen/padding"
        android:text=""/>
<TextView android:id="@+id/calories_units"
    android:gravity="center_horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="@dimen/units"
    android:text="@string/calories_burned"
    android:paddingBottom="@dimen/padding"
    android:background="@color/display_background"/>

</LinearLayout>

</LinearLayout>

<!-- Desired pace/speed row -->
<LinearLayout
    android:id="@+id/desired_pace_control"
    android:paddingTop="@dimen/row_spacing"
    android:gravity="center_horizontal"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1">

    <!-- Button "-", for decrementing desired pace/speed -->
    <Button android:id="@+id/button_desired_pace_lower"
        android:text="-"
        android:textSize="@dimen/button_sign"
        android:layout_width="@dimen/button"
        android:layout_height="@dimen/button"/>

    <!-- Container for desired pace/speed -->
    <LinearLayout
        android:gravity="center_horizontal"
        android:orientation="vertical"
        android:layout_width="@dimen/desired_pace_width"
        android:layout_height="wrap_content">

        <TextView android:id="@+id/desired_pace_label"
            android:gravity="center_horizontal"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"

```

```

        android:text="@string/desired_pace"/>

        <!-- Current desired pace/speed -->
        <TextView android:id="@+id/desired_pace_value"
            android:gravity="center_horizontal"
            android:textSize="@dimen/desired_pace"
            android:layout_width="@dimen/desired_pace_width"
            android:layout_height="wrap_content"/>

    </LinearLayout>

    <!-- Button "+", for incrementing desired pace/speed -->
    <Button android:id="@+id/button_desired_pace_raise"
        android:text="+"
        android:textSize="@dimen/button_sign"
        android:layout_width="@dimen/button"
        android:layout_height="@dimen/button"/>

</LinearLayout>

</LinearLayout>

```

通过上述代码，在屏幕中分别显示了穿戴者徒步行走的步数、行走距离、行走速度和消耗的热量等信息。

14.2.2 系统主 Activity

本系统的主 Activity 是 Pedometer，实现文件是 Pedometer.java，具体实现流程如下。

(1) 定义类 Pedometer，功能是定义系统中需要的变量和初始值，具体实现代码如下。

```

public class Pedometer extends Activity {
    private static final String TAG = "Pedometer";
    private SharedPreferences mSettings;
    private PedometerSettings mPedometerSettings;
    private Utils mUtils;

    private TextView mStepValueView;
    private TextView mPaceValueView;
    private TextView mDistanceValueView;
    private TextView mSpeedValueView;
    private TextView mCaloriesValueView;
    TextView mDesiredPaceView;
    private int mStepValue;           //mStepValueView 的值
    private int mPaceValue;           //mPaceValueView 的值
    private float mDistanceValue;     //mDistanceValueView 的值
    private float mSpeedValue;        //mSpeedValueView 的值
    private int mCaloriesValue;       //mCaloriesValueView 的值
    private float mDesiredPaceOrSpeed; //
    private int mMaintain;             //是否是爬山
    private boolean mIsMetric;         //公制和米制切换标志
}

```

```
private float mMaintainInc;           //
private boolean mQuitting = false;    //
private boolean mIsRunning;           //程序是否运行的标志位
```

(2) 编写函数 `onStart()` 用于启动计步器，具体代码如下。

```
//开始函数，重写该函数，加入日志
@Override
protected void onStart() {
    Log.i(TAG, "[ACTIVITY] onStart");
    super.onStart();
}
```

(3) 编写函数 `onResume()` 用于恢复计步器系统，具体代码如下。

```
//重写回复函数
@Override
protected void onResume() {
    Log.i(TAG, "[ACTIVITY] onResume");
    super.onResume();

    mSettings = PreferenceManager.getDefaultSharedPreferences(this);
    mPedometerSettings = new PedometerSettings(mSettings);

    mUtils.setSpeak(mSettings.getBoolean("speak", false));

    // Read from preferences if the service was running on the last onPause
    mIsRunning = mPedometerSettings.isServiceRunning();

    // Start the service if this is considered to be an application start (last
    onPause was long ago)
    if (!mIsRunning && mPedometerSettings.isNewStart()) {
        startStepService();
        bindStepService();
    }
    else if (mIsRunning) {
        bindStepService();
    }

    mPedometerSettings.clearServiceRunning();

    mStepValueView = (TextView) findViewById(R.id.step_value);
    mPaceValueView = (TextView) findViewById(R.id.pace_value);
    mDistanceValueView = (TextView) findViewById(R.id.distance_value);
    mSpeedValueView = (TextView) findViewById(R.id.speed_value);
    mCaloriesValueView = (TextView) findViewById(R.id.calories_value);
    mDesiredPaceView = (TextView) findViewById(R.id.desired_pace_value);

    mIsMetric = mPedometerSettings.isMetric();
    ((TextView) findViewById(R.id.distance_units)).setText(getString(
        mIsMetric
        ? R.string.kilometers
        : R.string.miles
```



```

));
((TextView) findViewById(R.id.speed_units)).setText(getString(
    mIsMetric
    ? R.string.kilometers_per_hour
    : R.string.miles_per_hour
));

mMaintain = mPedometerSettings.getMaintainOption();
((LinearLayout) this.findViewById(R.id.desired_pace_control)).setVisibility(
    mMaintain != PedometerSettings.M_NONE
    ? View.VISIBLE
    : View.GONE
);
if (mMaintain == PedometerSettings.M_PACE) {
    mMaintainInc = 5f;
    mDesiredPaceOrSpeed = (float)mPedometerSettings.getDesiredPace();
}
else
if (mMaintain == PedometerSettings.M_SPEED) {
    mDesiredPaceOrSpeed = mPedometerSettings.getDesiredSpeed();
    mMaintainInc = 0.1f;
}
Button button1 = (Button) findViewById(R.id.button_desired_pace_lower);
button1.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        mDesiredPaceOrSpeed -= mMaintainInc;
        mDesiredPaceOrSpeed = Math.round(mDesiredPaceOrSpeed * 10) / 10f;
        displayDesiredPaceOrSpeed();
        setDesiredPaceOrSpeed(mDesiredPaceOrSpeed);
    }
});
Button button2 = (Button) findViewById(R.id.button_desired_pace_raise);
button2.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        mDesiredPaceOrSpeed += mMaintainInc;
        mDesiredPaceOrSpeed = Math.round(mDesiredPaceOrSpeed * 10) / 10f;
        displayDesiredPaceOrSpeed();
        setDesiredPaceOrSpeed(mDesiredPaceOrSpeed);
    }
});
if (mMaintain != PedometerSettings.M_NONE) {
    ((TextView) findViewById(R.id.desired_pace_label)).setText(
        mMaintain == PedometerSettings.M_PACE
        ? R.string.desired_pace
        : R.string.desired_speed
    );
}

displayDesiredPaceOrSpeed();
}

```

(4) 编写函数 `displayDesiredPaceOrSpeed()`，用于显示想要的步伐节奏或速率，具体实现代码如下。

```
private void displayDesiredPaceOrSpeed() {
    if (mMaintain == PedometerSettings.M_PACE) {
        mDesiredPaceView.setText("" + (int)mDesiredPaceOrSpeed);
    }
    else {
        mDesiredPaceView.setText("" + mDesiredPaceOrSpeed);
    }
}
```

(5) 定义函数 `onPause()` 用于暂停计步器，具体实现代码如下。

```
@Override
protected void onPause() {
    Log.i(TAG, "[ACTIVITY] onPause");
    if (mIsRunning) {
        unbindStepService();
    }
    if (mQuitting) {
        mPedometerSettings.saveServiceRunningWithNullTimestamp(mIsRunning);
    }
    else {
        mPedometerSettings.saveServiceRunningWithTimestamp(mIsRunning);
    }

    super.onPause();
    savePaceSetting();
}
```

(6) 定义函数 `onStop()` 用于停止计步器系统，具体实现代码如下。

```
@Override
protected void onStop() {
    Log.i(TAG, "[ACTIVITY] onStop");
    super.onStop();
}
```

(7) 定义函数 `onDestroy()` 用于退出整个计步器系统，具体实现代码如下。

```
protected void onDestroy() {
    Log.i(TAG, "[ACTIVITY] onDestroy");
    super.onDestroy();
}
```

(8) 编写函数 `onRestart()` 用于重启计步器，具体代码如下。

```
protected void onRestart() {
    Log.i(TAG, "[ACTIVITY] onRestart");
    super.onDestroy();
}
```

(9) 编写函数 `setDesiredPaceOrSpeed`，用于设置想要的步伐节奏或速率，具体实现代码如下。

```
private void setDesiredPaceOrSpeed(float desiredPaceOrSpeed) {
    if (mService != null) {
        if (mMaintain == PedometerSettings.M_PACE) {
            mService.setDesiredPace((int)desiredPaceOrSpeed);
        }
        else
            if (mMaintain == PedometerSettings.M_SPEED) {
                mService.setDesiredSpeed(desiredPaceOrSpeed);
            }
    }
}

private void savePaceSetting() {
    mPedometerSettings.savePaceOrSpeedSetting(mMaintain, mDesiredPaceOrSpeed);
}
```

(10) 编写函数 `startStepService()`，用于启动计步服务，具体实现代码如下。

```
private void startStepService() {
    if (! mIsRunning) {
        Log.i(TAG, "[SERVICE] Start");
        mIsRunning = true;
        startService(new Intent(Pedometer.this,
                                StepService.class));
    }
}
```

(11) 编写函数 `bindStepService()`，用于绑定计步服务，具体实现代码如下。

```
private void bindStepService() {
    Log.i(TAG, "[SERVICE] Bind");
    bindService(new Intent(Pedometer.this,
                            StepService.class), mConnection, Context.BIND_AUTO_CREATE +
                Context.BIND_DEBUG_UNBIND);
}
```

(12) 编写函数 `unbindStepService()`，用于解除对当前计步服务的绑定，具体实现代码如下。

```
private void unbindStepService() {
    Log.i(TAG, "[SERVICE] Unbind");
    unbindService(mConnection);
}
```

(13) 编写函数 `stopStepService()`，用于停止当前的计步服务，具体实现代码如下。

```
private void stopStepService() {
    Log.i(TAG, "[SERVICE] Stop");
    if (mService != null) {
        Log.i(TAG, "[SERVICE] stopService");
        stopService(new Intent(Pedometer.this,
                                StepService.class));
    }
    mIsRunning = false;
}
```

(14) 编写函数 `stopStepService()`，用于重置当前计步器中的各个数值，具体实现代码如下。

```
private void resetValues(boolean updateDisplay) {
    if (mService != null && mIsRunning) {
        mService.resetValues();
    }
    else {
        mStepValueView.setText("0");
        mPaceValueView.setText("0");
        mDistanceValueView.setText("0");
        mSpeedValueView.setText("0");
        mCaloriesValueView.setText("0");
        SharedPreferences state = getSharedPreferences("state", 0);
        SharedPreferences.Editor stateEditor = state.edit();
        if (updateDisplay) {
            stateEditor.putInt("steps", 0);
            stateEditor.putInt("pace", 0);
            stateEditor.putFloat("distance", 0);
            stateEditor.putFloat("speed", 0);
            stateEditor.putFloat("calories", 0);
            stateEditor.commit();
        }
    }
}
```

(15) 编写函数 `onPrepareOptionsMenu(Menu menu)`，功能是创建系统菜单中的各个选项，具体实现代码如下。

```
public boolean onPrepareOptionsMenu(Menu menu) {
    menu.clear();
    if (mIsRunning) {
        menu.add(0, MENU_PAUSE, 0, R.string.pause)
            .setIcon(android.R.drawable.ic_media_pause)
            .setShortcut('1', 'p');
    }
    else {
        menu.add(0, MENU_RESUME, 0, R.string.resume)
            .setIcon(android.R.drawable.ic_media_play)
            .setShortcut('1', 'p');
    }
    menu.add(0, MENU_RESET, 0, R.string.reset)
        .setIcon(android.R.drawable.ic_menu_close_clear_cancel)
        .setShortcut('2', 'r');
    menu.add(0, MENU_SETTINGS, 0, R.string.settings)
        .setIcon(android.R.drawable.ic_menu_preferences)
        .setShortcut('8', 's')
        .setIntent(new Intent(this, Settings.class));
    menu.add(0, MENU_QUIT, 0, R.string.quit)
        .setIcon(android.R.drawable.ic_lock_power_off)
        .setShortcut('9', 'q');
    return true;
}
```

(16) 编写函数 `onOptionsItemSelected(MenuItem item)`，用于根据用户选择的选项进行处理，具体实现代码如下。

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case MENU_PAUSE:
            unbindStepService();
            stopStepService();
            return true;
        case MENU_RESUME:
            startStepService();
            bindStepService();
            return true;
        case MENU_RESET:
            resetValues(true);
            return true;
        case MENU_QUIT:
            resetValues(false);
            unbindStepService();
            stopStepService();
            mQuitting = true;
            finish();
            return true;
    }
    return false;
}
```

(17) 通过如下代码在屏幕文本框中显示统计数据的变化状况，具体实现代码如下。

```
// TODO: unite all into 1 type of message
private StepService.ICallback mCallback = new StepService.ICallback() {
    public void stepsChanged(int value) {
        mHandler.sendMessage(mHandler.obtainMessage(STEPS_MSG, value, 0));
    }
    public void paceChanged(int value) {
        mHandler.sendMessage(mHandler.obtainMessage(PACE_MSG, value, 0));
    }
    public void distanceChanged(float value) {
        mHandler.sendMessage(mHandler.obtainMessage(DISTANCE_MSG,
(int) (value*1000), 0));
    }
    public void speedChanged(float value) {
        mHandler.sendMessage(mHandler.obtainMessage(SPEED_MSG,
(int) (value*1000), 0));
    }
    public void caloriesChanged(float value) {
        mHandler.sendMessage(mHandler.obtainMessage(CALORIES_MSG,
(int) (value), 0));
    }
};

private static final int STEPS_MSG = 1;
```



```
private static final int PACE_MSG = 2;
private static final int DISTANCE_MSG = 3;
private static final int SPEED_MSG = 4;
private static final int CALORIES_MSG = 5;
```

(18) 编写函数 `handleMessage(Message msg)`，在文本框中显示具体统计的数值，具体实现代码如下。

```
private Handler mHandler = new Handler() {
    @Override public void handleMessage(Message msg) {
        switch (msg.what) {
            case STEPS_MSG:
                mStepValue = (int)msg.arg1;
                mStepValueView.setText("" + mStepValue);
                break;
            case PACE_MSG:
                mPaceValue = msg.arg1;
                if (mPaceValue <= 0) {
                    mPaceValueView.setText("0");
                }
                else {
                    mPaceValueView.setText("" + (int)mPaceValue);
                }
                break;
            case DISTANCE_MSG:
                mDistanceValue = ((int)msg.arg1)/1000f;
                if (mDistanceValue <= 0) {
                    mDistanceValueView.setText("0");
                }
                else {
                    mDistanceValueView.setText(
                        ("" + (mDistanceValue + 0.000001f)).substring(0, 5)
                    );
                }
                break;
            case SPEED_MSG:
                mSpeedValue = ((int)msg.arg1)/1000f;
                if (mSpeedValue <= 0) {
                    mSpeedValueView.setText("0");
                }
                else {
                    mSpeedValueView.setText(
                        ("" + (mSpeedValue + 0.000001f)).substring(0, 4)
                    );
                }
                break;
            case CALORIES_MSG:
                mCaloriesValue = msg.arg1;
                if (mCaloriesValue <= 0) {
                    mCaloriesValueView.setText("0");
                }
                else {
                    mCaloriesValueView.setText("" + (int)mCaloriesValue);
                }
            }
        }
    }
}
```

```

        }
        break;
    default:
        super.handleMessage(msg);
    }
};
}

```

至此，整个系统主界面的开发介绍完毕，执行后的效果如图 14-2 所示。



图 14-2 系统主界面的执行效果

14.3 系统设置模块


在系统主界面中，当按下  按钮后会在屏幕下方弹出如图 14-3 所示的界面。



图 14-3 弹出的新界面

单击图 14-3 中的  选项后会进入系统设置界面，如图 14-4 所示。

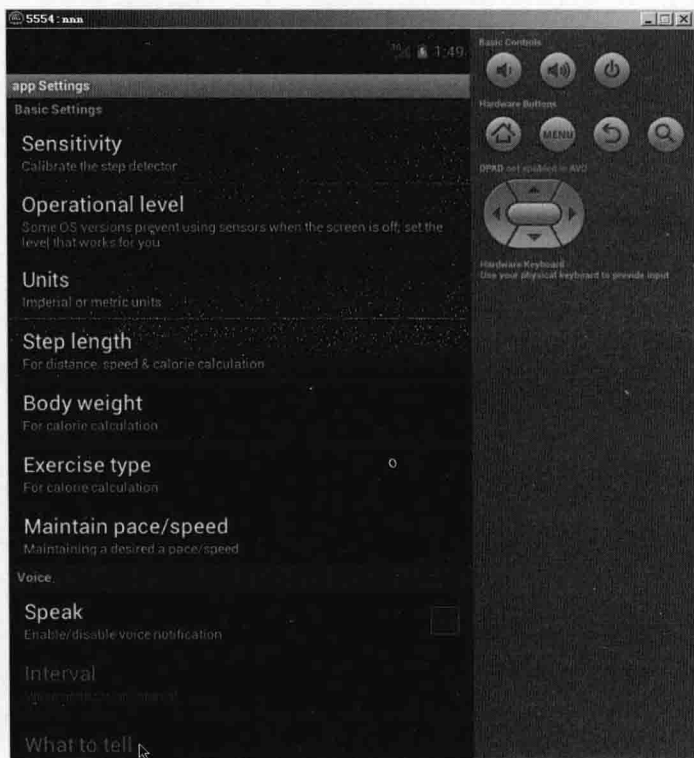


图 14-4 系统设置界面

在图 14-4 所示的界面中，可以对整个系统的参数进行设置。在本节的内容中，将详细讲解系统设置模块的具体实现过程。

14.3.1 系统设置 Activity

本系统设置 Activity 的实现文件是 Settings.java，功能是载入系统设置界面的布局文件 preferences.xml，具体实现代码如下。

```
public class Settings extends PreferenceActivity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        addPreferencesFromResource(R.xml.preferences);
    }
}
```

系统设置界面的布局文件是 preferences.xml，具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
```

```
<PreferenceCategory
    android:title="@string/steps_settings_title">
    <ListPreference
        android:key="sensitivity"
        android:title="@string/sensitivity_setting"
        android:summary="@string/sensitivity_setting_details"
        android:entries="@array/sensitivity_preference"
        android:entryValues="@array/sensitivity_preference_values"
        android:dialogTitle="@string/sensitivity_setting_title"
        android:defaultValue="30" />
    <ListPreference
        android:key="operation_level"
        android:title="@string/operation_level_setting"
        android:summary="@string/operation_level_setting_details"
        android:entries="@array/operation_level_preference"
        android:entryValues="@array/operation_level_preference_values"
        android:dialogTitle="@string/operation_level_setting_title"
        android:defaultValue="30" />
    <ListPreference
        android:key="units"
        android:title="@string/units_setting"
        android:summary="@string/units_setting_details"
        android:entries="@array/units_preference"
        android:entryValues="@array/units_preference_values"
        android:dialogTitle="@string/units_setting_title"
        android:defaultValue="imperial" />
    <name.step.preferences.StepLengthPreference
        android:key="step_length"
        android:title="@string/step_length_setting"
        android:summary="@string/step_length_setting_details"
        android:dialogTitle="@string/step_length_setting_title"
        android:defaultValue="20" />
    <name.step.preferences.BodyWeightPreference
        android:key="body_weight"
        android:title="@string/body_weight_setting"
        android:summary="@string/body_weight_setting_details"
        android:dialogTitle="@string/body_weight_setting_title"
        android:defaultValue="50" />
    <ListPreference
        android:key="exercise_type"
        android:title="@string/exercise_type_setting"
        android:summary="@string/exercise_type_setting_details"
        android:entries="@array/exercise_type_preference"
        android:entryValues="@array/exercise_type_preference_values"
        android:dialogTitle="@string/exercise_type_setting_title"
        android:defaultValue="running" />
    <ListPreference
        android:key="maintain"
        android:title="@string/maintain_setting"
        android:summary="@string/maintain_setting_details"
```

```

        android:entries="@array/maintain_preference"
        android:entryValues="@array/maintain_preference_values"
        android:dialogTitle="@string/maintain_setting_title"
        android:defaultValue="none" />
</PreferenceCategory>
<PreferenceCategory
    android:title="@string/voice_settings_title">
    <CheckBoxPreference
        android:key="speak"
        android:title="@string/voice_setting"
        android:summary="@string/voice_setting_details"
        android:defaultValue="false" />

    <ListPreference
        android:key="speaking_interval"
        android:title="@string/speaking_interval_setting"
        android:summary="@string/speaking_interval_setting_details"
        android:entries="@array/speaking_interval_preference"
        android:entryValues="@array/speaking_interval_preference_values"
        android:dependency="speak"
        android:defaultValue="1" />

    <PreferenceScreen
        android:key="tell_what"
        android:title="@string/tell_what"
        android:dependency="speak">
        <PreferenceCategory
            android:title="@string/tell_what">
            <CheckBoxPreference
                android:key="tell_steps"
                android:title="@string/tell_steps_setting"
                android:summary="@string/tell_steps_setting_details"
                android:defaultValue="false" />

            <CheckBoxPreference
                android:key="tell_pace"
                android:title="@string/tell_pace_setting"
                android:summary="@string/tell_pace_setting_details"
                android:defaultValue="false" />

            <CheckBoxPreference
                android:key="tell_distance"
                android:title="@string/tell_distance_setting"
                android:summary="@string/tell_distance_setting_details"
                android:defaultValue="false" />

            <CheckBoxPreference
                android:key="tell_speed"
                android:title="@string/tell_speed_setting"
                android:summary="@string/tell_speed_setting_details"
                android:defaultValue="false" />

            <CheckBoxPreference
                android:key="tell_calories"
                android:title="@string/tell_calories_setting"
                android:summary="@string/tell_calories_setting_details"

```



```

        android:defaultValue="false" />
        <CheckBoxPreference
            android:key="tell_fasterslower"
            android:title="@string/tell_fasterslower_setting"
            android:summary="@string/tell_fasterslower_setting_details"
            android:defaultValue="false" />
    </PreferenceCategory>
</PreferenceScreen>
</PreferenceCategory>
</PreferenceScreen>

```

14.3.2 获取各个设置值

编写文件 `PedometerSettings.java`，功能是定义各种系统参数的设置函数，获取图 14-4 所示的各个设置值。文件 `PedometerSettings.java` 的具体实现代码如下。

```

public class PedometerSettings {
    SharedPreferences mSettings;

    public static int M_NONE = 1;
    public static int M_PACE = 2;
    public static int M_SPEED = 3;

    public PedometerSettings(SharedPreferences settings) {
        mSettings = settings;
    }
    //公制和米制切换标志 imperial 公制 metric 米制
    public boolean isMetric() {
        return mSettings.getString("units", "imperial").equals("metric");
    }

    //取步长
    public float getStepLength() {
        try {
            return Float.valueOf(mSettings.getString("step_length", "20").trim());
        }
        catch (NumberFormatException e) {
            // TODO: reset value, & notify user somehow
            return 0f;
        }
    }

    //取体重
    public float getBodyWeight() {
        try {
            return Float.valueOf(mSettings.getString("body_weight", "50").trim());
        }
        catch (NumberFormatException e) {
            // TODO: reset value, & notify user somehow

```

```

        return 0f;
    }
}
//判断是走路还是跑步
public boolean isRunning() {
    return mSettings.getString("exercise_type", "running").equals(
        "running");
}
//登山还是平时走路
public int getMaintainOption() {
    String p = mSettings.getString("maintain", "none");
    return
        p.equals("none") ? M_NONE : (
            p.equals("pace") ? M_PACE : (
                p.equals("speed") ? M_SPEED : (0)
            )
        );
}
//-----
// Desired pace & speed:
// these can not be set in the preference activity, only on the main
// screen if "maintain" is set to "pace" or "speed"

public int getDesiredPace() {
    return mSettings.getInt("desired_pace", 180); // steps/minute
}

public float getDesiredSpeed() {
    return mSettings.getFloat("desired_speed", 4f); // km/h or mph
}

public void savePaceOrSpeedSetting(int maintain, float desiredPaceOrSpeed) {
    SharedPreferences.Editor editor = mSettings.edit();
    if (maintain == M_PACE) {
        editor.putInt("desired_pace", (int)desiredPaceOrSpeed);
    }
    else
    if (maintain == M_SPEED) {
        editor.putFloat("desired_speed", desiredPaceOrSpeed);
    }
    editor.commit();
}
//-----
// Speaking:

public boolean shouldSpeak() {
    return mSettings.getBoolean("speak", false);
}
}

```

```
public float getSpeakingInterval() {
    try {
        return Float.valueOf(mSettings.getString("speaking_interval", "1"));
    }
    catch (NumberFormatException e) {
        // This could not happen as the value is selected from a list.
        return 1;
    }
}

public boolean shouldTellSteps() {
    return mSettings.getBoolean("speak", false)
        && mSettings.getBoolean("tell_steps", false);
}

public boolean shouldTellPace() {
    return mSettings.getBoolean("speak", false)
        && mSettings.getBoolean("tell_pace", false);
}

public boolean shouldTellDistance() {
    return mSettings.getBoolean("speak", false)
        && mSettings.getBoolean("tell_distance", false);
}

public boolean shouldTellSpeed() {
    return mSettings.getBoolean("speak", false)
        && mSettings.getBoolean("tell_speed", false);
}

public boolean shouldTellCalories() {
    return mSettings.getBoolean("speak", false)
        && mSettings.getBoolean("tell_calories", false);
}

public boolean shouldTellFasterslower() {
    return mSettings.getBoolean("speak", false)
        && mSettings.getBoolean("tell_fasterslower", false);
}

public boolean wakeAggressively() {
    return mSettings.getString("operation_level", "run_in_background").
        equals("wake_up");
}

public boolean keepScreenOn() {
    return mSettings.getString("operation_level", "run_in_background").
        equals("keep_screen_on");
}
```

```
//
// Internal

public void saveServiceRunningWithTimestamp(boolean running) {
    SharedPreferences.Editor editor = mSettings.edit();
    editor.putBoolean("service_running", running);
    editor.putLong("last_seen", Utils.currentTimeMillis());
    editor.commit();
}

public void saveServiceRunningWithNullTimestamp(boolean running) {
    SharedPreferences.Editor editor = mSettings.edit();
    editor.putBoolean("service_running", running);
    editor.putLong("last_seen", 0);
    editor.commit();
}

public void clearServiceRunning() {
    SharedPreferences.Editor editor = mSettings.edit();
    editor.putBoolean("service_running", false);
    editor.putLong("last_seen", 0);
    editor.commit();
}

public boolean isServiceRunning() {
    return mSettings.getBoolean("service_running", false);
}

public boolean isNewStart() {
    // activity last paused more than 10 minutes ago
    return mSettings.getLong("last_seen", 0) < Utils.currentTimeMillis()
        - 1000*60*10;
}
}
```

14.3.3 系统服务设置

编写文件 StepService.java，功能是设置计步器系统的服务功能。这里的服务是指与用户进行交互的后台服务。文件 StepService.java 的具体实现流程如下。

(1) 定义服务类 StepService，定义系统需要的变量并设置初始值，具体实现代码如下。

```
public class StepService extends Service {
    private static final String TAG = "name.bagi.levente.pedometer.StepService";
    private SharedPreferences mSettings;
    private PedometerSettings mPedometerSettings;
    private SharedPreferences mState;
    private SharedPreferences.Editor mStateEditor;
    private Utils mUtils;
    private SensorManager mSensorManager;
    private Sensor mSensor;
    private StepDetector mStepDetector;
    // private StepBuzzer mStepBuzzer; // used for debugging
}
```

```

private StepDisplayer mStepDisplayer;
private PaceNotifier mPaceNotifier;
private DistanceNotifier mDistanceNotifier;
private SpeedNotifier mSpeedNotifier;
private CaloriesNotifier mCaloriesNotifier;
private SpeakingTimer mSpeakingTimer;

private PowerManager.WakeLock wakeLock;
private NotificationManager mNM;

private int mSteps;
private int mPace;
private float mDistance;
private float mSpeed;
private float mCalories;

public class StepBinder extends Binder {
    StepService getService() {
        return StepService.this;
    }
}

```

(2) 编写函数 `onCreate()`，载入系统初始设置的系统参数值，并且开启检测功能。具体实现代码如下。

```

public void onCreate() {
    Log.i(TAG, "[SERVICE] onCreate");
    super.onCreate();

    mNM = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
    showNotification();

    // Load settings
    mSettings = PreferenceManager.getDefaultSharedPreferences(this);
    mPedometerSettings = new PedometerSettings(mSettings);
    mState = getSharedPreferences("state", 0);

    mUtils = Utils.getInstance();
    mUtils.setService(this);
    mUtils.initTTS();

    acquireWakeLock();

    // 开启检测功能
    mStepDetector = new StepDetector();
    mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    registerDetector();

    // 注册 ACTION_SCREEN_OFF，将电话进入待机模式
    IntentFilter filter = new IntentFilter(Intent.ACTION_SCREEN_OFF);
    registerReceiver(mReceiver, filter);
}

```



```

mStepDisplayer = new StepDisplayer(mPedometerSettings, mUtils);
mStepDisplayer.setSteps(mSteps = mState.getInt("steps", 0));
mStepDisplayer.addListener(mStepListener);
mStepDetector.addStepListener(mStepDisplayer);

mPaceNotifier = new PaceNotifier(mPedometerSettings, mUtils);
mPaceNotifier.setPace(mPace = mState.getInt("pace", 0));
mPaceNotifier.addListener(mPaceListener);
mStepDetector.addStepListener(mPaceNotifier);

mDistanceNotifier = new DistanceNotifier(mDistanceListener,
mPedometerSettings, mUtils);
mDistanceNotifier.setDistance(mDistance = mState.getFloat("distance", 0));
mStepDetector.addStepListener(mDistanceNotifier);

mSpeedNotifier = new SpeedNotifier(mSpeedListener, mPedometerSettings, mUtils);
mSpeedNotifier.setSpeed(mSpeed = mState.getFloat("speed", 0));
mPaceNotifier.addListener(mSpeedNotifier);

mCaloriesNotifier = new CaloriesNotifier(mCaloriesListener,
mPedometerSettings, mUtils);
mCaloriesNotifier.setCalories(mCalories = mState.getFloat("calories", 0));
mStepDetector.addStepListener(mCaloriesNotifier);

mSpeakingTimer = new SpeakingTimer(mPedometerSettings, mUtils);
mSpeakingTimer.addListener(mStepDisplayer);
mSpeakingTimer.addListener(mPaceNotifier);
mSpeakingTimer.addListener(mDistanceNotifier);
mSpeakingTimer.addListener(mSpeedNotifier);
mSpeakingTimer.addListener(mCaloriesNotifier);
mStepDetector.addStepListener(mSpeakingTimer);

// 启动声音
reloadSettings();

// Tell the user we started.
Toast.makeText(this, getText(R.string.started), Toast.LENGTH_SHORT).show();
}

```

(3) 编写函数 `onStart(Intent intent, int startId)` 以启动系统服务，具体实现代码如下。

```

public void onStart(Intent intent, int startId) {
    Log.i(TAG, "[SERVICE] onStart");
    super.onStart(intent, startId);
}

```

(4) 编写函数 `onDestroy()` 以销毁系统服务，具体实现代码如下。

```

public void onDestroy() {
    Log.i(TAG, "[SERVICE] onDestroy");
    mUtils.shutdownTTS();
}

```

```

// 取消接收机制
unregisterReceiver(mReceiver);
unregisterDetector();

mStateEditor = mState.edit();
mStateEditor.putInt("steps", mSteps);
mStateEditor.putInt("pace", mPace);
mStateEditor.putFloat("distance", mDistance);
mStateEditor.putFloat("speed", mSpeed);
mStateEditor.putFloat("calories", mCalories);
mStateEditor.commit();

mNM.cancel(R.string.app_name);

wakeLock.release();

super.onDestroy();

// 停止检测
mSensorManager.unregisterListener(mStepDetector);

// 通知用户停止
Toast.makeText(this, getText(R.string.stopped), Toast.LENGTH_SHORT).show();
}

```

(5) 编写函数 `registerDetector()` 以注册探测服务，具体实现代码如下。

```

private void registerDetector() {
    mSensor = mSensorManager.getDefaultSensor(
        Sensor.TYPE_ACCELEROMETER /*|
        Sensor.TYPE_MAGNETIC_FIELD |
        Sensor.TYPE_ORIENTATION*/);
    mSensorManager.registerListener(mStepDetector,
        mSensor,
        SensorManager.SENSOR_DELAY_FASTEST);
}

```

(6) 编写函数 `unregisterDetector()` 以注销探测服务，具体实现代码如下。

```

private void unregisterDetector() {
    mSensorManager.unregisterListener(mStepDetector);
}

```

(7) 编写函数 `onBind(Intent intent)` 以绑定系统服务，具体实现代码如下。

```

public IBinder onBind(Intent intent) {
    Log.i(TAG, "[SERVICE] onBind");
    return mBinder;
}

private final IBinder mBinder = new StepBinder();

public interface ICallback {
    public void stepsChanged(int value);
    public void paceChanged(int value);
}

```

```

    public void distanceChanged(float value);
    public void speedChanged(float value);
    public void caloriesChanged(float value);
}

private ICallback mCallback;

public void registerCallback(ICallback cb) {
    mCallback = cb;
    //mStepDisplayer.passValue();
    //mPaceListener.passValue();
}

private int mDesiredPace;
private float mDesiredSpeed;

```

(8) 编写函数 `setDesiredPace(int desiredPace)`，功能是重设用户修改后的每步距离值，具体实现代码如下。

```

public void setDesiredPace(int desiredPace) {
    mDesiredPace = desiredPace;
    if (mPaceNotifier != null) {
        mPaceNotifier.setDesiredPace(mDesiredPace);
    }
}

```

(9) 编写函数 `setDesiredSpeed(float desiredSpeed)`，功能是重设用户修改后的速率值，具体实现代码如下。

```

public void setDesiredSpeed(float desiredSpeed) {
    mDesiredSpeed = desiredSpeed;
    if (mSpeedNotifier != null) {
        mSpeedNotifier.setDesiredSpeed(mDesiredSpeed);
    }
}

```

(10) 编写函数 `reloadSettings()`，功能是重新载入系统设置值，具体实现代码如下。

```

public void reloadSettings() {
    mSettings = PreferenceManager.getDefaultSharedPreferences(this);

    if (mStepDetector != null) {
        mStepDetector.setSensitivity(
            Float.valueOf(mSettings.getString("sensitivity", "10"))
        );
    }

    if (mStepDisplayer != null) mStepDisplayer.reloadSettings();
    if (mPaceNotifier != null) mPaceNotifier.reloadSettings();
    if (mDistanceNotifier != null) mDistanceNotifier.reloadSettings();
    if (mSpeedNotifier != null) mSpeedNotifier.reloadSettings();
    if (mCaloriesNotifier != null) mCaloriesNotifier.reloadSettings();
    if (mSpeakingTimer != null) mSpeakingTimer.reloadSettings();
}

```

(11) 编写函数 `resetValues()`，功能是重新设置值，具体实现代码如下。

```
public void resetValues() {
    mStepDisplayer.setSteps(0);
    mPaceNotifier.setPace(0);
    mDistanceNotifier.setDistance(0);
    mSpeedNotifier.setSpeed(0);
    mCaloriesNotifier.setCalories(0);
}
```

(12) 通过如下代码从 `PaceNotifie` 获取前进速率值。

```
private StepDisplayer.Listener mStepListener = new StepDisplayer.Listener() {
    public void stepsChanged(int value) {
        mSteps = value;
        passValue();
    }
    public void passValue() {
        if (mCallback != null) {
            mCallback.stepsChanged(mSteps);
        }
    }
};
```

(13) 通过如下代码从 `PaceNotifier` 获取步长值。

```
private PaceNotifier.Listener mPaceListener = new PaceNotifier.Listener() {
    public void paceChanged(int value) {
        mPace = value;
        passValue();
    }
    public void passValue() {
        if (mCallback != null) {
            mCallback.paceChanged(mPace);
        }
    }
};
```

(14) 通过如下代码从 `DistanceNotifier` 获取距离值。

```
private DistanceNotifier.Listener mDistanceListener = new DistanceNotifier.
Listener() {
    public void valueChanged(float value) {
        mDistance = value;
        passValue();
    }
    public void passValue() {
        if (mCallback != null) {
            mCallback.distanceChanged(mDistance);
        }
    }
};
```

(15) 通过如下代码从 **SpeedNotifier** 获取速率值。

```
private SpeedNotifier.Listener mSpeedListener = new SpeedNotifier.Listener() {
    public void valueChanged(float value) {
        mSpeed = value;
        passValue();
    }
    public void passValue() {
        if (mCallback != null) {
            mCallback.speedChanged(mSpeed);
        }
    }
};
```

(16) 通过如下代码从 **CaloriesNotifier** 获取热量值。

```
private CaloriesNotifier.Listener mCaloriesListener = new CaloriesNotifier.
Listener() {
    public void valueChanged(float value) {
        mCalories = value;
        passValue();
    }
    public void passValue() {
        if (mCallback != null) {
            mCallback.caloriesChanged(mCalories);
        }
    }
};
```

(17) 编写函数 **showNotification()**，功能是显示一个“该服务正在运行”的通知，具体实现代码如下。

```
private void showNotification() {
    CharSequence text = getText(R.string.app_name);
    Notification notification = new Notification(R.drawable.ic_notification, null,
        System.currentTimeMillis());
    notification.flags = Notification.FLAG_NO_CLEAR | Notification.FLAG_
        ONGOING_EVENT;
    Intent pedometerIntent = new Intent();
    pedometerIntent.setComponent(new ComponentName(this, Pedometer.class));
    pedometerIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
        pedometerIntent, 0);
    notification.setLatestEventInfo(this, text,
        getText(R.string.notification_subtitle), contentIntent);

    mNM.notify(R.string.app_name, notification);
}
```

(18) 通过如下代码处理广播接收机制 **ACTION_SCREEN_OFF**。

```
private BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
```



```
// Check action just to be on the safe side.
if (intent.getAction().equals(Intent.ACTION_SCREEN_OFF)) {
    // Unregisters the listener and registers it again.
    StepService.this.unregisterDetector();
    StepService.this.registerDetector();
    if (mPedometerSettings.wakeAggressively()) {
        wakeLock.release();
        acquireWakeLock();
    }
}
};
```

(19) 编写函数 `acquireWakeLock()` 获取唤醒锁，具体实现代码如下。

```
private void acquireWakeLock() {
    PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);
    int wakeFlags;
    if (mPedometerSettings.wakeAggressively()) {
        wakeFlags = PowerManager.SCREEN_DIM_WAKE_LOCK | PowerManager.
            ACQUIRE_CAUSES_WAKEUP;
    }
    else if (mPedometerSettings.keepScreenOn()) {
        wakeFlags = PowerManager.SCREEN_DIM_WAKE_LOCK;
    }
    else {
        wakeFlags = PowerManager.PARTIAL_WAKE_LOCK;
    }
    wakeLock = pm.newWakeLock(wakeFlags, TAG);
    wakeLock.acquire();
}
```

14.3.4 获取并显示热量

编写文件 `CaloriesNotifier.java`，功能是获取并显示行走过程中所消耗的近似热量值，热量的单位是卡路里。文件 `CaloriesNotifier.java` 的具体实现代码如下。

```
public class CaloriesNotifier implements StepListener, SpeakingTimer.Listener {

    public interface Listener {
        public void valueChanged(float value);
        public void passValue();
    }
    private Listener mListener;

    private static double METRIC_RUNNING_FACTOR = 1.02784823;
    private static double IMPERIAL_RUNNING_FACTOR = 0.75031498;

    private static double METRIC_WALKING_FACTOR = 0.708;
    private static double IMPERIAL_WALKING_FACTOR = 0.517;
```

```

private double mCalories = 0;

PedometerSettings mSettings;
Utils mUtils;

boolean mIsMetric;
boolean mIsRunning;
float mStepLength;
float mBodyWeight;

public CaloriesNotifier(Listener listener, PedometerSettings settings,
Utils utils) {
    mListener = listener;
    mUtils = utils;
    mSettings = settings;
    reloadSettings();
}

public void setCalories(float calories) {
    mCalories = calories;
    notifyListener();
}

public void reloadSettings() {
    mIsMetric = mSettings.isMetric();
    mIsRunning = mSettings.isRunning();
    mStepLength = mSettings.getStepLength();
    mBodyWeight = mSettings.getBodyWeight();
    notifyListener();
}

public void resetValues() {
    mCalories = 0;
}

public void isMetric(boolean isMetric) {
    mIsMetric = isMetric;
}

public void setStepLength(float stepLength) {
    mStepLength = stepLength;
}

public void onStep() {

    if (mIsMetric) {
        mCalories +=
            (mBodyWeight * (mIsRunning ? METRIC_RUNNING_FACTOR : METRIC_
WALKING_FACTOR))
        // Distance:
        * mStepLength          // centimeters
        / 100000.0;           // centimeters/kilometer
    }
    else {
        mCalories +=

```

```

        (mBodyWeight * (mIsRunning ? IMPERIAL_RUNNING_FACTOR :
        IMPERIAL_WALKING_FACTOR))
        // Distance:
        * mStepLength          // inches
        / 63360.0;             // inches/mile
    }

    notifyListener();
}

private void notifyListener() {
    mListener.valueChanged((float)mCalories);
}

public void passValue() {
}

public void speak() {
    if (mSettings.shouldTellCalories()) {
        if (mCalories > 0) {
            mUtils.say("" + (int)mCalories + " calories burned");
        }
    }
}
}
}

```

14.3.5 显示行走距离

编写文件 DistanceNotifier.java，功能是获取并显示行走的距离，具体实现代码如下。

```

public class DistanceNotifier implements StepListener, SpeakingTimer.Listener {

    public interface Listener {
        public void valueChanged(float value);
        public void passValue();
    }

    private Listener mListener;

    float mDistance = 0;

    PedometerSettings mSettings;
    Utils mUtils;

    boolean mIsMetric;
    float mStepLength;

    public DistanceNotifier(Listener listener, PedometerSettings settings,
        Utils utils) {
        mListener = listener;
    }
}

```

```

        mUtils = utils;
        mSettings = settings;
        reloadSettings();
    }

    public void setDistance(float distance) {
        mDistance = distance;
        notifyListener();
    }

    public void reloadSettings() {
        mIsMetric = mSettings.isMetric();
        mStepLength = mSettings.getStepLength();
        notifyListener();
    }

    public void onStep() {

        if (mIsMetric) {
            mDistance += (float)(           // kilometers
                mStepLength                 // centimeters
                / 100000.0);                 // centimeters/kilometer
        }
        else {
            mDistance += (float)(           // miles
                mStepLength                 // inches
                / 63360.0);                 // inches/mile
        }

        notifyListener();
    }

    private void notifyListener() {
        mListener.valueChanged(mDistance);
    }

    public void passValue() {
        // Callback of StepListener - Not implemented
    }

    public void speak() {
        if (mSettings.shouldTellDistance()) {
            if (mDistance >= .001f) {
                mUtils.say(("" + (mDistance + 0.000001f)).substring(0, 4) +
                    (mIsMetric ? " kilometers" : " miles"));
                // TODO: format numbers (no "." at the end)
            }
        }
    }
}

```

14.3.6 获取并显示步伐速率

编写文件 `PaceNotifier.java`，功能是获取并显示步伐速率，就是每分钟走多少步。我们可以为系统设置步伐速率，系统会根据用户设置的值提示快还是慢。文件 `PaceNotifier.java` 的具体实现代码如下。

```
public class PaceNotifier implements StepListener, SpeakingTimer.Listener {

    public interface Listener {
        public void paceChanged(int value);
        public void passValue();
    }

    private ArrayList<Listener> mListeners = new ArrayList<Listener>();

    int mCounter = 0;

    private long mLastStepTime = 0;
    private long[] mLastStepDeltas = {-1, -1, -1, -1};
    private int mLastStepDeltasIndex = 0;
    private long mPace = 0;

    PedometerSettings mSettings;
    Utils mUtils;

    /** Desired pace, adjusted by the user */
    int mDesiredPace;

    /** Should we speak? */
    boolean mShouldTellFasterslower;

    /** When did the TTS speak last time */
    private long mSpokenAt = 0;

    public PaceNotifier(PedometerSettings settings, Utils utils) {
        mUtils = utils;
        mSettings = settings;
        mDesiredPace = mSettings.getDesiredPace();
        reloadSettings();
    }

    public void setPace(int pace) {
        mPace = pace;
        int avg = (int) (60*1000.0 / mPace);
        for (int i = 0; i < mLastStepDeltas.length; i++) {
            mLastStepDeltas[i] = avg;
        }
        notifyListener();
    }

    public void reloadSettings() {
        mShouldTellFasterslower =
            mSettings.shouldTellFasterslower()
    }
}
```



```

        && mSettings.getMaintainOption() == PedometerSettings.M_PACE;
        notifyListener();
    }

    public void addListener(Listener l) {
        mListeners.add(l);
    }

    public void setDesiredPace(int desiredPace) {
        mDesiredPace = desiredPace;
    }

    public void onStep() {
        long thisStepTime = System.currentTimeMillis();
        mCounter ++;

        // Calculate pace based on last x steps
        if (mLastStepTime > 0) {
            long delta = thisStepTime - mLastStepTime;

            mLastStepDeltas[mLastStepDeltasIndex] = delta;
            mLastStepDeltasIndex = (mLastStepDeltasIndex + 1) % mLastStepDeltas.
                length;

            long sum = 0;
            boolean isMeaningfull = true;
            for (int i = 0; i < mLastStepDeltas.length; i++) {
                if (mLastStepDeltas[i] < 0) {
                    isMeaningfull = false;
                    break;
                }
                sum += mLastStepDeltas[i];
            }
            if (isMeaningfull && sum > 0) {
                long avg = sum / mLastStepDeltas.length;
                mPace = 60*1000 / avg;

                // TODO: remove duplication. This also exists in SpeedNotifier
                if (mShouldTellFasterSlower && !mUtils.isSpeakingEnabled()) {
                    if (thisStepTime - mSpokenAt > 3000 && !mUtils.isSpeakingNow()) {
                        float little = 0.10f;
                        float normal = 0.30f;
                        float much = 0.50f;

                        boolean spoken = true;
                        if (mPace < mDesiredPace * (1 - much)) {
                            mUtils.say("much faster!");
                        }
                        else
                            if (mPace > mDesiredPace * (1 + much)) {

```

```

        mUtils.say("much slower!");
    }
    else
    if (mPace < mDesiredPace * (1 - normal)) {
        mUtils.say("faster!");
    }
    else
    if (mPace > mDesiredPace * (1 + normal)) {
        mUtils.say("slower!");
    }
    else
    if (mPace < mDesiredPace * (1 - little)) {
        mUtils.say("a little faster!");
    }
    else
    if (mPace > mDesiredPace * (1 + little)) {
        mUtils.say("a little slower!");
    }
    else {
        spoken = false;
    }
    if (spoken) {
        mSpokenAt = thisStepTime;
    }
}
}
}
else {
    mPace = -1;
}
}
mLastStepTime = thisStepTime;
notifyListener();
}

private void notifyListener() {
    for (Listener listener : mListeners) {
        listener.paceChanged((int)mPace);
    }
}

public void passValue() {
    // Not used
}

//-----
// Speaking

public void speak() {
    if (mSettings.shouldTellPace()) {

```

```

        if (mPace > 0) {
            mUtils.say(mPace + " steps per minute");
        }
    }
}

```

14.3.7 获取并显示行走速率

编写文件 `SpeedNotifier.java`，功能是获取并显示行走速率，就是每小时走多少公里。我们可以为系统设置行走速率，系统会根据用户设置的值提示快还是慢。文件 `SpeedNotifier.java` 的具体实现代码如下。

```

public class SpeedNotifier implements PaceNotifier.Listener, SpeakingTimer.
Listener {
    public interface Listener {
        public void valueChanged(float value);
        public void passValue();
    }
    private Listener mListener;

    int mCounter = 0;
    float mSpeed = 0;

    boolean mIsMetric;
    float mStepLength;

    PedometerSettings mSettings;
    Utils mUtils;

    /** Desired speed, adjusted by the user */
    float mDesiredSpeed;

    /** Should we speak? */
    boolean mShouldTellFasterSlower;
    boolean mShouldTellSpeed;

    /** When did the TTS speak last time */
    private long mSpokenAt = 0;

    public SpeedNotifier(Listener listener, PedometerSettings settings, Utils
utils) {
        mListener = listener;
        mUtils = utils;
        mSettings = settings;
        mDesiredSpeed = mSettings.getDesiredSpeed();
        reloadSettings();
    }

    public void setSpeed(float speed) {
        mSpeed = speed;
        notifyListener();
    }

```

```

    }
    public void reloadSettings() {
        mIsMetric = mSettings.isMetric();
        mStepLength = mSettings.getStepLength();
        mShouldTellSpeed = mSettings.shouldTellSpeed();
        mShouldTellFasterslower =
            mSettings.shouldTellFasterslower()
            && mSettings.getMaintainOption() == PedometerSettings.M_SPEED;
        notifyListener();
    }
    public void setDesiredSpeed(float desiredSpeed) {
        mDesiredSpeed = desiredSpeed;
    }

    private void notifyListener() {
        mListener.valueChanged(mSpeed);
    }

    public void paceChanged(int value) {
        if (mIsMetric) {
            mSpeed = // kilometers / hour
                value * mStepLength // centimeters / minute
                / 100000f * 60f; // centimeters/kilometer
        }
        else {
            mSpeed = // miles / hour
                value * mStepLength // inches / minute
                / 63360f * 60f; // inches/mile
        }
        tellFasterSlower();
        notifyListener();
    }

    /**
     * Say slower/faster, if needed.
     */
    private void tellFasterSlower() {
        if (mShouldTellFasterslower && mUtils.isSpeakingEnabled()) {
            long now = System.currentTimeMillis();
            if (now - mSpokenAt > 3000 && !mUtils.isSpeakingNow()) {
                float little = 0.10f;
                float normal = 0.30f;
                float much = 0.50f;

                boolean spoken = true;
                if (mSpeed < mDesiredSpeed * (1 - much)) {
                    mUtils.say("much faster!");
                }
                else
                    if (mSpeed > mDesiredSpeed * (1 + much)) {

```

```

        mUtils.say("much slower!");
    }
    else
    if (mSpeed < mDesiredSpeed * (1 - normal)) {
        mUtils.say("faster!");
    }
    else
    if (mSpeed > mDesiredSpeed * (1 + normal)) {
        mUtils.say("slower!");
    }
    else
    if (mSpeed < mDesiredSpeed * (1 - little)) {
        mUtils.say("a little faster!");
    }
    else
    if (mSpeed > mDesiredSpeed * (1 + little)) {
        mUtils.say("a little slower!");
    }
    else {
        spoken = false;
    }
    if (spoken) {
        mSpokenAt = now;
    }
}
}

}

public void passValue() {
    // Not used
}

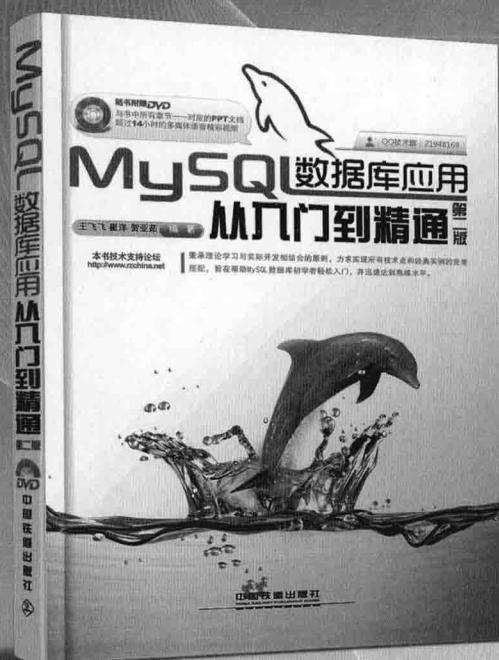
public void speak() {
    if (mSettings.shouldTellSpeed()) {
        if (mSpeed >= .01f) {
            mUtils.say(("" + (mSpeed + 0.000001f)).substring(0, 4) +
                (mIsMetric ? " kilometers per hour" : " miles per hour"));
        }
    }
}
}
}

```

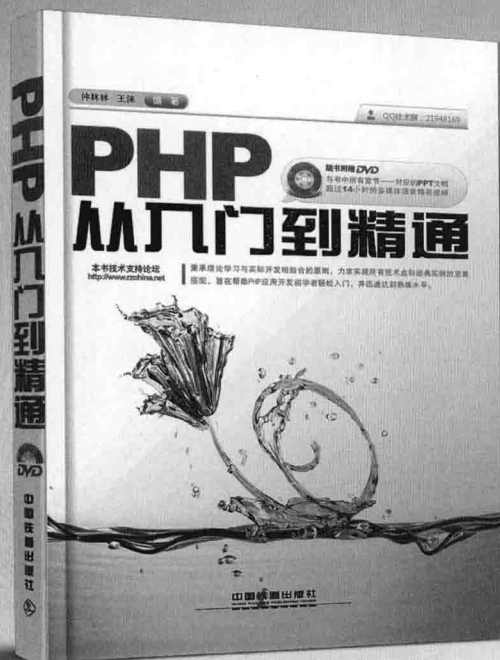
至此，本系统的主要功能模块全部介绍完毕。本书只对重点和难点内容进行了剖析，至于其他部分的具体实现，读者可参阅本书光盘中附带的源码。

PHP·MySQL入门到精通

秉承理论学习与实际开发相结合的原则，力求实现所有技术点和经典实例的完美搭配，旨在帮助PHP和MySQL数据库初学者轻松入门，并迅速达到熟练水平。



MySQL 数据库应用从入门到精通(第2版)
ISBN: 978-7-113-15131-7

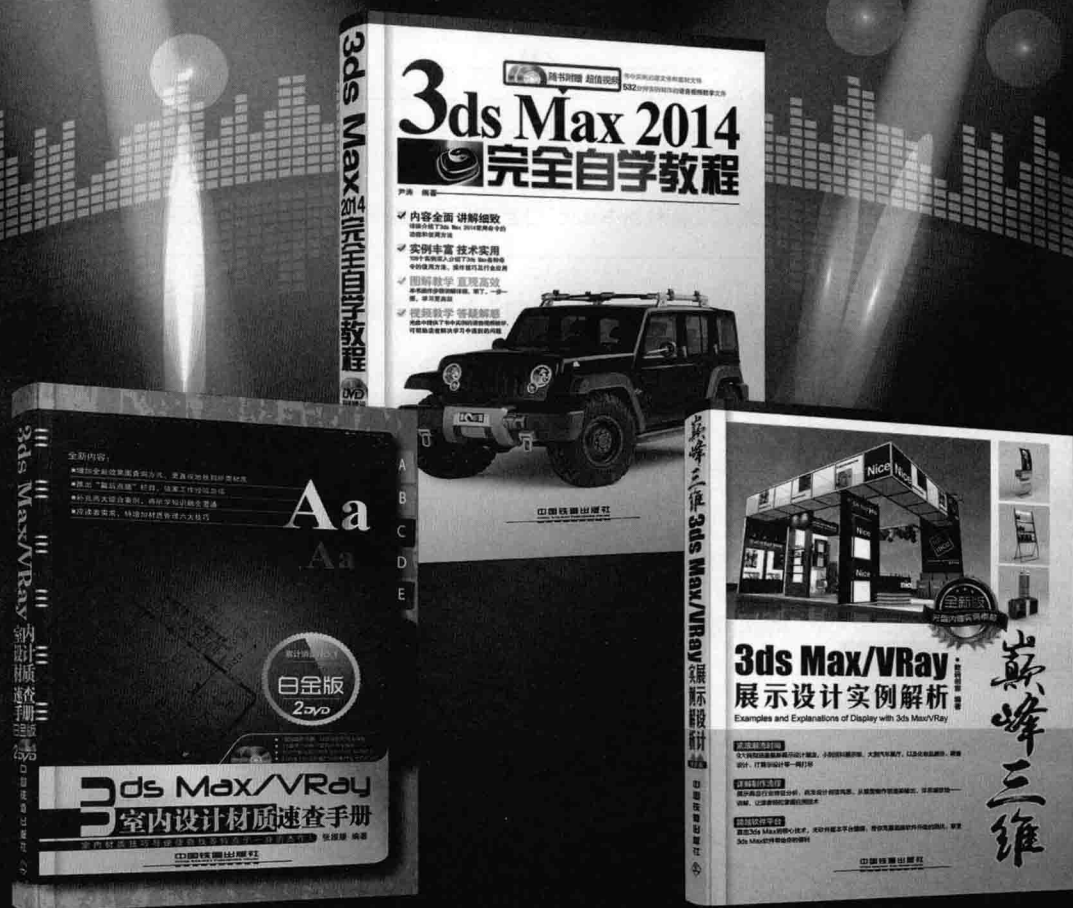


PHP从入门到精通
ISBN: 978-7-113-18011-9



中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

中国铁道出版社图书推荐!



《3ds Max 2014完全自学教程》
ISBN: 978-7-113-17570-2
定价: 108.00元 (附赠光盘)

《3ds Max/VRay室内设计材质速查手册 (白金版)》
ISBN: 978-7-113-16721-9
定价: 49.00元 (附赠光盘)

《巅峰三维 3ds Max/VRay展示设计实例解析 (全新版)》
ISBN: 978-7-113-15651-0
定价: 69.00元 (附赠光盘)

读者意见反馈表

亲爱的读者：

感谢您对中国铁道出版社的支持，您的建议是我们不断改进工作的信息来源，您的需求是我们不断开拓创新的基础。为了更好地服务读者，出版更多的精品图书，希望您能在百忙之中抽出时间填写这份意见反馈表发给我们。随书纸制表格请在填好后剪下寄到：北京市西城区右安门西街8号中国铁道出版社综合编辑部 于先军 收（邮编：100054）。或者采用传真（010-63549458）方式发送。此外，读者也可以直接通过电子邮件把意见反馈给我们，E-mail地址是：46768089@qq.com。我们将选出意见中肯的热心读者，赠送本社的其他图书作为奖励。同时，我们将充分考虑您的意见和建议，并尽可能地给您满意的答复。谢谢！

所购书名：_____

个人资料：

姓名：_____性别：_____年龄：_____文化程度：_____

职业：_____电话：_____E-mail：_____

通信地址：_____邮编：_____

您是如何得知本书的：

☐书店宣传 ☐网络宣传 ☐展会促销 ☐出版社图书目录 ☐老师指定 ☐杂志、报纸等的介绍 ☐别人推荐

☐其他（请注明）_____

您从何处得到本书的：

☐书店 ☐邮购 ☐商场、超市等卖场 ☐图书销售的网站 ☐培训学校 ☐其他

影响您购买本书的因素（可多选）：

☐内容实用 ☐价格合理 ☐装帧设计精美 ☐带多媒体教学光盘 ☐优惠促销 ☐书评广告 ☐出版社知名度

☐作者名气 ☐工作、生活和学习的需要 ☐其他

您对本书封面设计的满意程度：

☐很满意 ☐比较满意 ☐一般 ☐不满意 ☐改进建议

您对本书的总体满意程度：

从文字的角度 ☐很满意 ☐比较满意 ☐一般 ☐不满意

从技术的角度 ☐很满意 ☐比较满意 ☐一般 ☐不满意

您希望书中图的比例是多少：

☐少量的图片辅以大量的文字 ☐图文比例相当 ☐大量的图片辅以少量的文字

您希望本书的定价是多少：

本书最令您满意的是：

1.

2.

您在使用本书时遇到哪些困难：

1.

2.

您希望本书在哪些方面进行改进：

1.

2.

您需要购买哪些方面的图书？对我社现有图书有什么好的建议？

您更喜欢阅读哪些类型和层次的计算机书籍（可多选）？

☐入门类 ☐精通类 ☐综合类 ☐问答类 ☐图解类 ☐查询手册类 ☐实例教程类

您在学习计算机的过程中有什么困难？

您的其他要求：

Android

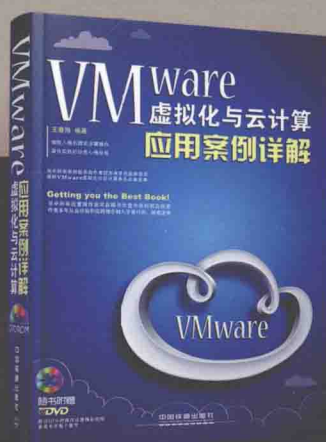


智能穿戴设备开发 从入门到精通

中国铁道出版社开发类重点图书推荐：



书名：Microsoft虚拟化与云计算应用案例详解
ISBN：978-7-113-17177-3
定价：79.00元



VMware虚拟化与云计算应用案例详解
ISBN：978-7-113-17176-6
定价：89.00元

上架建议：计算机/程序设计/系统开发



中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

地址：北京市西城区右安门西街8号
邮编：100054
网址：<http://www.tdpress.com>

ISBN 978-7-113-19212-9



定价：59.80元（附赠光盘）