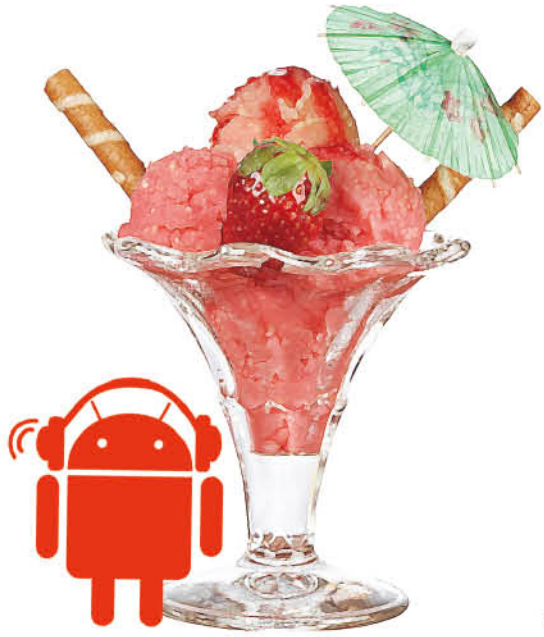
HZ BOOKS
华章科技

系统讲解Android网络编程的各项核心技术和功能模块，通过多个案例解读Android网络编程的方法和技巧

从源码角度深入解析Android核心网络处理方法和关键应用的实现原理，包含大量最佳实践

移动开发



陈文　郭依正◎著

Understanding Android Network Programming: Core Technology and Best Practice

深入理解Android网络编程

技术详解与最佳实践

机械工业出版社
China Machine Press

深入理解 Android 网络编程：技术详解与最佳实践

陈　文　郭依正　著

HZ BOOKS
华章图书

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

深入理解 Android 网络编程：技术详解与最佳实践 / 陈文，郭依正著 . —北京：机械工业出版社，2013.8

ISBN 978-7-111-43502-0

Ⅰ. 深… Ⅱ. ①陈… ②郭… Ⅲ. 移动终端－应用程序－程序设计 Ⅳ. TN929.53

中国版本图书馆 CIP 数据核字（2013）第 171759 号

版权所有 · 侵权必究

封底无防伪标均为盗版

本书法律顾问 北京市展达律师事务所

这是一本 Android 网络编程方面的专著，旨在帮助开发者们高效地编写出高质量的 Android 网络应用。不仅系统讲解了 Android 网络编程的各项核心技术和功能模块，通过多个案例解读了 Android 网络编程的方法和技巧，而且从源码角度深入解析了 Android 核心网络处理方法和关键应用的实现原理，包含大量最佳实践。

全书共 11 章，分为三个部分：概述篇（第 1 章）在介绍了 Android 开发平台后，重点讲解了 Android 网络程序的功能及开发环境的配置，引领读者走进 Android 网络编程的大门；实战篇（第 2~8 章），详细讲解了 TCP、UDP、HTTP 等基本网络协议在 Android 上的应用，展示了使用 Android 处理 JSON、SOAP、HTML、XML 等数据的方法，讲解了 Android 中的 RSS 编程、Email 编程、OAuth 认证等，解析了 Android 中 Locations、Maps、USB、Wi-Fi、Bluetooth、NFC 等网络模块的编程，讨论了 Android 中线程、数据存取、消息缓存、界面更新等的处理方法，探讨了 SIP、XMPP 等协议在 Android 上的应用；源码分析篇（第 9~11 章）分析了 Android 中与 HTML 处理、网络处理以及部分网络应用相关的源代码，帮助读者从底层原理上加深对相关知识点的理解。

HZ BOOKS
华章图书

机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码 100037）

责任编辑：孙海亮

印刷

2013 年 8 月第 1 版第 1 次印刷

186mm×240mm • 23.75 印张

标准书号：ISBN 978-7-111-43502-0

定 价：69.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991 88361066 投稿热线：（010）88379604

购书热线：（010）68326294 88379649 68995259 读者信箱：hzjsj@hzbook.com

序

移动互联网发展迅速，既有机遇又充满危机。善于学习、勇于挑战者总能在机遇来临时抓住机会迈上一个新的台阶，达到一个新的高度，面对危机时也能从中看到希望从而破茧而出，实现危与机的转换。

和陈文的相识纯属偶然，当时创业不久，没有多少钱可以雇佣高薪员工，所以希望能从应届毕业生中淘些璞玉出来。仅是一个电话，我意识到这就是我要找的人才，思路清晰、有冲劲、知识面开阔，而且对新事物有强烈的兴趣和参与的欲望。

以后的合作如预料中一样愉快。为实现“每个人都有平等学习的机会”的美好愿景，大家通力协作，每一个团队成员都能以主人翁的心态去面对一些挫折并积极考虑解决方案，希望能在移动教育领域书写属于我们自己的传奇。虽然暂时并未如愿，但我相信通过创业实现人生价值和为社会做出贡献的理想已经融入到团队每一位成员的日常行动中。所以陈文后来跟我说他再次创业了，我丝毫不觉得意外，并乐于为其在创业之余挤出时间完成的这部著作作序。

一个好的行知者总喜欢探索新事物，并在实践中不断修炼、感悟，实现理论和实践的良性互动，不断在失败和成功中总结经验教训，只为下一步能走得更好。这也是德鲁克的自我管理精神和当前热门行业敏捷开发理念的内涵所在。纵观陈文的这本书，覆盖了 Android 乃至整个移动互联网领域中网络编程的方方面面，从基础的 HTTP 协议、多线程开发到即时通信、VOIP 均有涉及，而且行文之处不时闪现出自己的理解和独到见解，足见作者百忙之余的用心和用功。

人生的起步总是从站在前人的肩膀上开始，愿本书的出版能够对渴望迈入移动互联网行业的读者提供一些帮助，并为其下一步的成长提供知识上的助力和理想的传递，如果有机会，就开始属于自己的传奇！

苗忠良

于福州机场

前　言

为什么要写这本书

随着 Android 平台市场占有率的稳步上升，Android 应用的数量和种类越来越多，涉及的范围也越来越大。从单机应用发展到联网应用，再到云端体系，其发挥的作用越来越重要。

Android 移动开发领域正处在一个转折点：移动用户的需求日益增长，他们希望在移动终端上拥有一个永远在线的个人管理中心；开发者希望手机成为互联网移动终端，以扩展互联网应用的广度和深度；企业希望在手机平台上实现更多的管理和应用，随时随地保持沟通，进而使企业低成本、高效率地运营。这些需求更多地表现在 Android 的互联网应用方面，其技术核心正是 Android 网络编程的相关知识。

不断扩展网络相关应用是 Android 平台的主要方向，Android 网络编程不但能够实现信息的实时交互、在线存储和计算等基础应用，甚至可以实现移动办公、电子商务等复杂逻辑，进而实现无处不在的移动应用。Android 开发者们了解 Android 网络编程相关的知识，将能更加快速高效地编写 Android 网络应用。通过本书能和大家一起来分享和探讨这些内容，笔者自觉是一件非常有意义的事情。

读者对象

本书适合于以下读者阅读：

- ❑ Android 移动互联网开发者；
- ❑ 希望了解 Android 网络编程，利用 Android 平台实现网络应用的个人；
- ❑ 开设相关课程的大专院校师生及科研工作者。

如何阅读本书

全书共分为三个部分。

第一部分为概述篇，即第 1 章，这部分在简介了 Android 开发平台后，重点讲解了 Android 网络程序的功能及开发环境的配置，引导读者走进 Android 网络编程的大门。

第二部分为实战篇，包括第 2 章到第 8 章，这部分介绍了基本的网络协议 TCP、UDP、HTTP 等在 Android 上的应用；讲解了使用 Android 处理 JSON、SOAP、HTML、XML 等数据的方法，同时讲解了 Android 中的 RSS 编程、Email 编程、OAuth 认证等；解析了 Android 中 Locations、Maps、USB、Wi-Fi、Bluetooth、NFC 等网络模块的编程；讨论了 Android 中线程、数据存取、消息缓存、界面更新等的处理方法；探讨了 SIP、XMPP 等协议在 Android 上的应用。

第三部分为源码分析篇，包括第 9 章到第 11 章，这部分分析了 Android 中与 HTML 处理、网络处理以及部分网络应用相关的源码。读者通过阅读这部分内容，可以进一步加深对相关知识点的理解。

以下是各章内容的分述：

第 1 章：认识 Android 和 Android 网络程序的工作原理，简单介绍 Android 的发展、功能特性、系统构架，重点讲解 Android 网络编程和设置 Android 开发环境，探讨用 Android 编写网络程序的一般步骤并给出简单的 Android 网络编程的实践案例。

第 2 章：介绍支撑 Internet 的 TCP/IP 网络模型，重点讨论其中的 TCP、UDP 协议，讲解基于 TCP 及 UDP 协议的 Socket 编程，并通过聊天程序、FTP 客户端及 Telnet 客户端等案例讲解 Socket 编程的实践方法。

第 3 章：介绍 HTTP 协议，讨论如何使用 Android 处理 JSON、SOAP、HTML。案例部分给出了基于 HTTP 协议的文件上传、使用 HttpClient 和 URLConnection 访问维基百科、JSON 解析 wikipedia 内容、SOAP 解析天气服务及 Android 自定义打开 HTML 页面等内容。

第 4 章：介绍 Android 解析 XML 文件的三种方式，即 DOM、SAX、PULL，讨论 Android 中 RSS 编程、Email 编程等，同时还介绍与 Android 安全相关的知识，重点讲解 Android 加密解密及 OAuth 认证。

第 5 章：介绍 Android 中常用的网络编程组件，包括解析 Locations、Maps、USB、Wi-Fi、Bluetooth 等重要通信接口在 Android 上的使用方法，并结合具体案例介绍这些接口的一般使用方法。

第 6 章：介绍 Android 线程机制以及数据存取、消息缓存、UI 同步的方法。在线程部分重点讲解利用 AsyncTask 实现 Android 多线程应用开发；在数据存取部分介绍 Shared Preferences、Internal Storage、External Storage、SQLite Database 及 Network Connection 等 5 种存储永久性应用程序数据的方法；在消息缓存部分介绍 Android 本地存储的缓存机制；在 UI 同步部分讲解了在加载数据前、刷新数据时、完成任务时更新界面的方法。

第 7 章：介绍如何使用 SIP 协议构建 VoIP 应用。详细阐述了使用 Brekeke SIP Server 搭

建 SIP 服务器的方法，从设置应用程序的权限到初始化和监听 SIP 通话，逐步介绍 Android 中实现 SIP 通话的一般步骤。

第 8 章：介绍如何使用 XMPP 协议在 Android 上建立即时通信应用。包括如何使用 Openfire 搭建 XMPP 服务器，客户端如何使用 Asmack 登录服务器。

第 9 章：分析 Android 中与 HTML 处理相关的源码。重点讲解 WebView 对 HTML 文档的处理，涉及 WebView 加载入口的分析及在 WebView 中如何调用 JavaScript 等。同时解析 WebKit 内核，分析 WebKit 下一些比较重要的类。

第 10 章：分析 Android 中与网络处理相关的源码。重点分析 Android 网络处理的流程，包括网络状态监控、网络认证、DHCP 处理、网络代理等相关内容。此外，介绍 AndroidHttpClient 及 SSL 认证，分析与 RTP 协议和 SIP 协议相关的源码。

第 11 章：分析 Android 中部分与网络应用相关的源码。重点分析 Android 中使用 SAX 方式解析 XML 时如何发现 XML 根元素与子元素，讲解 Android 中如何实现基于位置的服务，同时简介媒体传输协议（MTP）的概念，对 MTP 设备、MTP 设备上的对象与存储单元等进行具体分析。

本书内容涉及面广、知识点多，案例部分包括现实中网络开发经常遇到的问题。我们不期望读者通过这一本书就能全面掌握 Android 的网络编程，但是如果读者能够耐心地从头到尾多读几遍，相信您一定有所收获。如果我们的书能对您有所启迪，我们再辛苦地写作也是值得的。

勘误和支持

除了陈文和郭依正之外，翟旭军、郭里城、潘道远、周巧扣、章莉、陈海光等也参与了资料的整理和示例的收集等工作。书中源代码大多都是一些代码片段，笔者认为提供电子版下载意义不大，故未专门设置下载链接，若有读者需要可发邮件至 book@chenwen.org，笔者会逐一回复，提供所需相应电子版源代码。由于作者的水平有限，加之编写时间仓促，书中难免会出现不足和错误之处，恳请广大读者批评指正。如果您有什么宝贵意见，欢迎您发送邮件至我的邮箱或者到我的博客 (http://chenwen.org) 上和我一起探讨，期待能够得到你们的真挚反馈。

致谢

首先要感谢 Android，没有开源的 Android 系统就没有现在如此丰富的移动网络生活，当然也就没有这本书。

感谢和我一起编写本书的南京师范大学泰州学院的郭依正老师，全书内容是我们共同的劳动结晶，他丰富的教学经验和严谨的写作风格使我受益匪浅。

感谢在我编程成长中每一位给我力量的朋友：陈亚必、张怀锋、程晓节、李国财、曾旭、

孙明坤、孙正然、朱鹏飞、张家荣、曹文、何伟伟、高志立、罗衍、于勇、刘伟、耿飙、梁成全、王超、王军、顾同跃、贺强、周亮、尤慧丽等，以及名单之外的更多热爱 Android 的朋友们，感谢你们对我的启发和帮助。感谢苗忠良老师的引荐，在您的努力下才促成了这本书的合作与出版。

我要特别感谢机械工业出版社华章公司的编辑杨福川老师和孙亮海老师，在这一年多的时间中是你们始终支持着我们的写作，你们的鼓励和帮助引导我们顺利完成了全部书稿。

感谢我亲爱的兄弟姐妹：郑琴、郑邮生、万骞谦、陈莉、戴亚，感谢你们陪伴我一同成长，为我平淡的生活增添了无尽的色彩。

最后感谢我的父亲陈新明、母亲郑秀兰，感谢你们赋予我生命，将我培养成人，并给我最好的帮助。

谨以此书献给我最亲爱的家人，以及众多热爱 Android 的朋友们！

陈　文

HZ BOOKS
华章图书

目　录

第二篇 实战篇

HZ BOOKS 华章图书

第三篇 源码分析篇

HZ BOOKS
华章图书

第一篇

概　述　篇

HZ BOOKS
华章图书

本部分内容：

- Android 网络编程概要

第 1 章　Android 网络编程概要

今天，Android 上使用网络的应用越来越多，如电子邮件、Web 浏览器和 IM 等传统的应用都是基于网络的程序；微博、微信等大量的新兴应用都是在网络的基础上开发的；音乐播放器、词典等传统的本地应用，在加入在线存储功能、在线推荐、分享等功能后也成为网络应用。

随着 Android 的发展，其对网络编程的支持也日益强大。Android 系统的功能已经远远超过了普通通信手机的功能，更像是有手机功能的 PC。Android 网络编程将会变得更加简洁和广泛：一方面 Android 的开源和强大的开发框架大大简化了网络应用的编程；另一方面众多网络服务提供商的开放 API 也对网络编程提供了极大的便利。

在用 Android 编写网络程序的时候，需要了解一些 Android 开发的基础知识。本章将概述 Android 的发展，讨论 Android 网络程序的功能，设置 Android 开发环境。本章最后将用实战案例来具体分析 Android 网络编程的步骤。

1.1　Android 简介

1.1.1　Android 的发展

Android 纪元正式开始于 2008 年 10 月 22 日。这天，T-Mobile G1 正式在美国公开发售。时至今日，Android 平台集成了操作系统、中间件、用户界面和应用软件，已经成为开放和完整的移动生态系统，可谓发展飞速。

目前移动终端市场上，随着 Android 平台的发展以及不断完善，越来越多的厂商开始选择 Android 系统作为其主要发展方向，自 2008 年 9 月 Android 系统的第一个版本发布至今，Android 系统在手机市场大放异彩，已经长期占据市场份额第一的位置。就目前来说，Android 手机的统治地位还是无可动摇的。Android 4.0 版本发布以来已渐成主力，推动 Android 手机和平板的份额不断提高，同时也为 Android 系统“碎片化”的问题提供了可靠的解决方案。

Android 系统能够取得今天的成功，最主要的应归功于其开源及免费性。正是在其开源和免费的基础上，各大厂商纷纷在原生系统的基础上进行定制和扩展，植入自身的应用，开发出更多有特色的产品，来满足市场的需求。这种情况在促进全球智能手机产业发展的同时，也使得 Android 系统的覆盖面积更为广阔。

Android 的开源，对于厂家来说可以更好地集成自己的产品和服务；开发者更可以在其开源的基础上进行进一步开发，提供更好的应用；用户能用到免费的 Android 系统和众多

的应用。

目前采用了 Android 系统的主要的大手机厂商包括：HTC、联想、华为、中兴、魅族、小米、酷派、天语、华硕、OPPO、三星、摩托罗拉、索尼、LG 和夏普等。Android 已经成为互联网的重要入口和载体。很多互联网企业开始在 Android 系统上发力，360、盛大、百度、阿里巴巴和网易等互联网巨头，均开始致力于千元左右的智能手机的开发。

Android 的未来充满了活力，将给人们的生活带来更加深刻的变革。Google 在 Android 移动平台的基础上推出云音乐服务和电影服务，并与电子书服务相结合，提供更为全面的内容资源。Android TV 借助各种应用和游戏，变身成为客厅多媒体娱乐中心的理想将成为现实。Android 开放式配件标准包括第三方配件的硬件设计和系统 API。第三方配件将会层出不穷，届时这些配件均可得到 Android 设备的兼容支持。未来将会有更多的智能设备出现，比如 Android 音箱、闹钟，甚至电饭锅、电冰箱等。如果有大量的 Android 第三方配件出现，基于 Android 的家庭自动化则可以让整个家庭生活都会更方便、更欢乐。

1.1.2　Android 功能特性

Android 系统在其开放性的基础上，引入了很多由软件和硬件实现的实用功能，在方便人们使用的同时，也给了开发者广阔的空间。下面是其中的一些重要的功能特性。

- 数据存储。Android 提供了 SharedPreferences、ContentProvider、文件、SQLite 数据库和网络等多种方式来存储数据。
- 通信网络。Android 操作系统支持所有的网络格式，包括 GSM/EDGE、IDEN、CDMA、EV-DO、UMTS、Bluetooth、Wi-Fi、LTE、NFC 和 WiMAX 等。
- 信息。Android 操作系统原生支持短信和邮件，并且支持所有的云端信息和服务器信息。
- 语言。Android 操作系统支持多语言。
- 浏览器。Android 操作系统中内置的网页浏览器基于 WebKit 内核，并且采用了 Chrome V8 引擎。在 Android 4.0 内置的浏览器测试中，HTML5 和 Acid3 故障处理中均获得了满分。
- 支持 Java。虽然 Android 操作系统中的应用程序大部分都是由 Java 编写的，但是 Android 却需要转换为 Dalvik 执行文件，在 Dalvik 虚拟机上运行。由于 Android 中并不自带 Java 虚拟机，因此无法直接运行 Java 程序。不过 Android 平台上提供了多个 Java 虚拟机供用户下载使用，安装了 Java 虚拟机的 Android 系统可以运行 J2ME 的程序。
- 多媒体。Android 操作系统本身支持以下格式的音频 / 视频 / 图片媒体：WebM、H.263、H.264（in 3GP or MP4 container）、MPEG-4 SP、AMR，AMR-WB（in 3GP container）、AAC、HE-AAC（in MP4 or 3GP container）、MP3、MIDI、Ogg Vorbis、FLAC、WAV、JPEG、PNG、GIF、BMP。
- 流媒体。Android 操作系统支持 RTP/RTSP（3GPP PSS，ISMA）的流媒体以及（HTML5<video>）的流媒体，在安装了 RealPlayer 之后，还支持苹果公司的流媒体。

- 外围设备。Android 操作系统支持识别并且使用视频 / 照片摄像头、多点电容 / 电阻触摸屏、GPS、加速计、陀螺仪、气压计、磁强计、键盘、鼠标、U 盘、专用的游戏控制器、体感控制器、游戏手柄、蓝牙设备、无线设备、感应和压力传感器、温度计、2D 和 3D 图形加速等。
- 多点触控。Android 内核支持原生的多点触摸。
- 多任务处理。Android 操作系统支持原生的多任务处理。
- 语音功能。除了支持普通的电话通话之外，Android 操作系统从最初版本就支持使用语音进行网页搜索等功能。而从 Android 2.2 开始，语音还可以用来输入文本、实现语音导航等功能。
- 无线共享功能。Android 操作系统支持用户使用本机充当无线路由器，并且将本机的网络共享给其他手机，其他机器只需要通过 WiFi 寻找到共享的无线热点，就可以上网。
- 截图功能。从 Android 4.0 版本开始，Android 操作系统便支持截图功能，该功能允许用户直接抓取手机屏幕上的任何画面，用户可以通过编辑功能对截图进行处理，还可以通过蓝牙、Email、微博等方式共享给其他用户或者上传到网络上，也可以复制到计算机中。
- Google Now。Google Now 是 Android 4.1 的一个新功能，这个功能可以根据搜索历史或者日历以及其他更多数据来预测出用户想要的到底是什么，并在指定的时间或者地点进行搜索并提出反馈建议。比如当用户有一个新的日历预约，Google 将利用各种信息（交通数据、地图、公交换乘）来帮助用户准时到达预约地点；如果用户搜索了一个航班信息，Google 将会持续通知这个航班的动态更新；甚至还可以跟踪一个球队的表现情况。
- Android Beam 功能。Android Beam 优化了近场通信以及蓝牙分享功能。
- Smart App Updates。Smart App Updates 是一种智能型的应用更新模式，应用程序在更新时不需要下载整个 APK，只需要下载修改的部分即可，这样更节省流量。

1.1.3 Android 系统构架

Android 不仅仅局限于操作系统，Android 平台由操作系统、中间件、用户友好的界面和应用软件组成。Android 核心是经过 Google 剪裁和调优的 Linux Kernel，对于掌上设备的硬件提供了优良的支持；在 Dalvik 虚拟机上，大部分 Java 核心类库都已经可以直接运行；拥有大量立即可用的类库和应用软件，可以轻易开发出可媲美桌面应用复杂度的手机软件；基于 Android，Google 已经开发大量好的应用软件，同时可以直接使用 Google 很多的在线服务；Google 还提供了基于 Eclipse 的完整开发环境、模拟器、文档、帮助和示例。

Android 系统框图如图 1-1 所示。可以看出 Android 分为 5 层，从低到高分别是 Linux Kernel 内核层、Android 系统库、Android 运行时、应用程序框架层和应用层。

- Linux Kernel 内核层。Linux 内核层是硬件和软件层之间的抽象层。其包含了显示驱动、摄像头驱动、蓝牙驱动、闪存驱动、IPC 管道通讯驱动、USB 驱动、键盘驱动、Wi-Fi 无线驱动、音频驱动和电源管理驱动。最下层是 Linux 系统核心驱动，主要用

于协调 CPU 处理和内存管理。

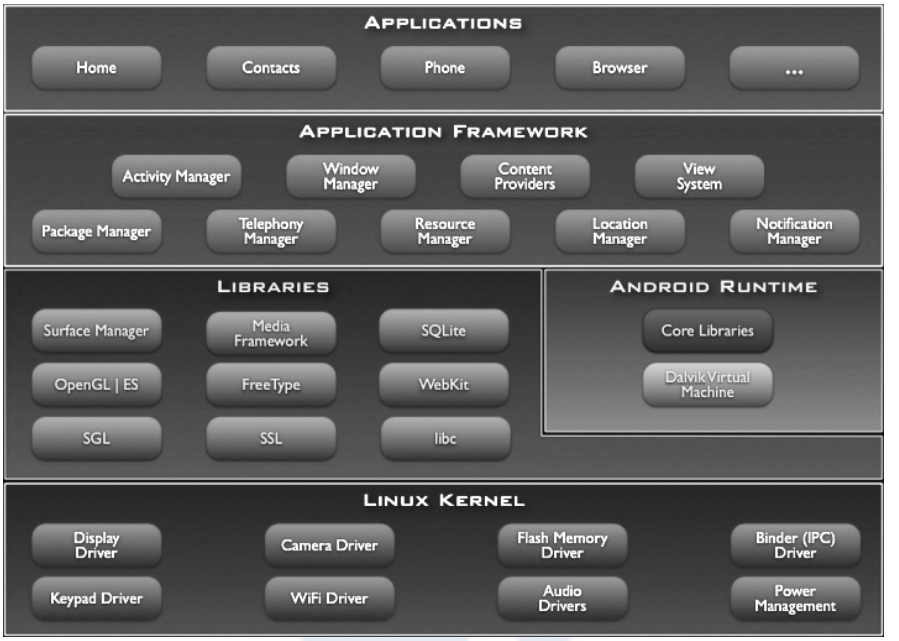


图 1-1　Android 系统框图

- Android 系统库。Android 包含一个 C/C++ 库的集合，供 Android 系统的各个组件使用。这些功能通过 Android 的应用程序框架提供给开发者。其核心库包含：SurfaceManager 显示系统管理库，负责把 2D 或 3D 内容显示到屏幕；Media Framework 媒体库，负责支持图像，支持多种视频和音频的录制和回放；SQLite 数据库引擎、OpenGL ES 图形引擎、FreeType 位图和矢量字体渲染引擎、Webkit 浏览器引擎、SGL 基本的 2D 图形引擎、SSL 安全套接字层引擎、Libc 库以及 Android Dalvik 虚拟机运行库。
- Android 运行时。Android 包含一个核心库的集合，提供大部分在 Java 编程语言核心类库中可用的功能。Dalvik 被设计成在一个设备可以高效地运行的多个虚拟机，每一个 Android 应用程序都在它自己的进程中运行，也就是都有一个属于自己的 Dalvik 虚拟机。这可以让系统在运行时可以优化，从而使程序间的影响大大降低。Dalvik 虚拟机并非运行 Java 字节码，而是运行自己的字节码。Dalvik 虚拟机依赖于 Linux 内核提供基本功能，如线程和底层内存管理。
- 应用程序框架层。应用程序框架层简化了程序开发的架构设计，开发者可以完全使用核心应用程序所使用的框架接口，任何应用程序都能发布它的功能，且任何其他应用程序可以使用这些功能（需要服从框架执行的安全限制）。应用程序框架层主要是系统管理类库，包括 Activity 管理、窗口管理、内容提供、显示系统基类、消息通知管理、程序包管理、电话管理、资源管理和定位管理。
- 应用层。Android 应用层包含核心应用程序，如 Home 桌面、Contacts 联系人、Phone

拨打电话、Browser 浏览器等，开发者的大部分应用也在这一层。

1.2 Android 网络程序的功能

Android 的网络程序大大增强了简单程序的功能。通过网络，一个程序可以和成千上万的人进行通信；可以获取世界上联网计算机中存储的信息；可以利用许多计算机的能力来解决一个问题。

Android 网络应用程序最基本的形式是作为应用客户端。Android 客户端获取服务器的数据并显示。比较复杂的 Android 网络应用还会对获取的数据进行处理，不断更新数据，向他人和计算机发送数据以实现实时交互。较少的 Android 网络程序将 Android 作为服务器来使用，这是 Android 网络应用程序的另外一种形式，比如作为家庭多媒体中心，为其他设备提供信息服务。Android 平台具有的开放特点和其高度集成的开发框架，使其成为个人移动中心、家庭媒体中心，也必将使其在企业应用上大展身手。

1.2.1 通信功能

电话、短信等传统应用在 Android 系统上得到了加强，Google 为电话、短信和联系人提供了强大的管理功能和智能云备份的能力，一些应用也为电话提供了归属地查询和垃圾短信过滤等功能。电话和短信构成了 Android 最基本的通信应用，满足用户最基本的通信需要。借助于 IM、KIK 等应用，不仅可以用文本来传输信息，还可以使用图像、语音、视频等方式交互；可以将通信内容完全保存在服务器上，方便随时查看。

Android 网络应用程序丰富了通信的方式，提高了通信的质量，只要在联网的情况下就可以使用，不必通过电信运营商，大大降低了通信的成本。

1.2.2 及时分享

Android 应用的社会化已经成为了不可或缺的基本功能，很多应用程序都增加了分享的功能以实现其社会化。通过社会化应用，用户可以将自己的心情、想法随时随地发到微博上；看到有趣的图片也可以直接上传到分享网站；有需要帮助的问题，立刻就可以使用问答应用提问，等待大家的回答。这些分享的信息，不但能充分利用移动的优势，将用户的信息及时发布到互联网上，分享给好友，还能充分利用终端的硬件优势，比如 GPS、摄像头等，多维度地分享自己的生活和见闻。用户在分享的过程中，大量的经验和知识被保存起来，为知识化的互联网提供了更多的素材。

Android 网络应用程序丰富了社会化交流，用户可以及时发布信息、提供帮助、交流看法，分享身边的一切。

1.2.3 个人管理

个人事务管理应用将 Android 变成随身的智能管理工具。印象笔记、有道笔记和麦库

等云笔记应用以知识管理见长，帮助用户收集随手得来的内容，把碎片知识整理起来，存储到云端，用户可以在多个设备上方便地查看。Doit.im、Remember The Milk 和 toodledo 等时间管理应用重造了任务列表，最大程度地帮助用户管理时间。挖财、随手记等应用帮助用户对个人财务进行分析。

Android 网络应用程序加强了用户管理时间、知识、财务的能力，用户把以前记到本子上的内容，通过 Android 记到云端，可以随时随地查看、学习和管理，用起来更加得心应手。

1.2.4　娱乐游戏

近年来，Android 系统软硬件的快速发展（软件方面，Android 系统原生支持更多的游戏外设；硬件方面，CUP、内存和屏幕等硬件变得更加强大），使得游戏和娱乐应用不断发展。Android 应用商店里面的游戏娱乐的比重在不断增加，更多的大型游戏出现在 Android 上。同时使用 Android 进行在线支付、购物变得更加方便。

在线的游戏、视频、音乐、广播已经成为人们生活的一部分，Android 已变为娱乐游戏的强大载体。

1.2.5　企业应用

Android 作为免费的系统，可以有效降低企业应用的成本；而开放的源代码又为企业应用提供了更多稳定性的保证。众多硬件厂商也根据不同的应用环境开发了不同特性的硬件，比如医疗、军事、运动等不同方面的 Android 硬件产品。企业软件也在不断发展，比如 Epocrates 应用可帮助医疗护理专业人员快速和方便地获取可靠的药物信息；SAP 的 SkyMobile 应用可帮助销售和服务人员访问 SAP 的 CRM 系统，进而访问客户数据。

Android 企业应用将进一步提高企业的生产效率，为企业移动办公提供更加方便、可靠的平台。

1.3　设置 Android 开发环境

Android 应用的开发需要建立一个开发环境，并需要进行一些设置。本节将以 Windows 平台为例来一步一步地讲解如何设置 Android 开发环境。

1.3.1　相关下载

1. JDK 下载

首先需要下载 JDK6。Android SDK 需要 JDK5 或者 JDK6，我们使用 JDK6 来开发本书中的案例。可以从 Oracele 的官方网站（http://www.oracle.com/technetwork/java/javase/downloads/index.html）下载 JDK6，然后安装。

安装 JDK 之后需要设置环境变量，设置 JAVA_HOME 环境变量指向 JDK 安装文件

夹。在 Windows XP 系统中，可以右击“我的电脑”图标，在弹出的快捷菜单中选择“属性”命令，然后在弹出的对话框中选择“高级”选项卡，然后单击“环境变量”按钮。“新建”或者“编辑”JAVA_HOME 变量，将其设置为上面 JDK 的安装目录。在 Windows Vista 和 Window 7 中，可以选择“开始”→“计算机”，右击选择“属性”，依次单击“高级系统设置”→“环境变量”，就可以更改环境变量了，将 JAVA_HOME 变量设置为 JDK 的安装目录。

通过在命令行里面输入 Java-c 来验证设置是否正确，设置正确之后会出现图 1-2 所示的提示（其中版本号可能不同）。

```
java version "1.6.0_30"
Java(TM) SE Runtime Environment (build 1.6.0_30-b12)
Java HotSpot(TM) 64-Bit Server VM (build 20.5-b03, mixed mode)
```

图 1-2　Java-c 提示

2. Eclipse 下载

安装 JDK 之后，可以下载 Eclipse IDE for JAVA Developer。本书中使用 Eclipse 3.7 进行示例开发。其下载网址为 http://www.eclipse.org/downloads/。下载之后为一个 .zip 文件，可以解压缩到合适的目录下，目录中的启动图标为 eclipse.exe。

3. 下载 Android SDK

需要使用 Android SDK 来开发 Android 应用程序，其下载网址为 http://developer.android.com/sdk/index.html，根据不同的系统选择相应的版本下载并安装。图 1-3 所示是 SDK 安装过程中的一个截图。安装之后启动 Android SDK Manager，根据开发需要，选择安装平台版本。本书中如无特别指明，均是使用 Android 4.1 的平台版本。

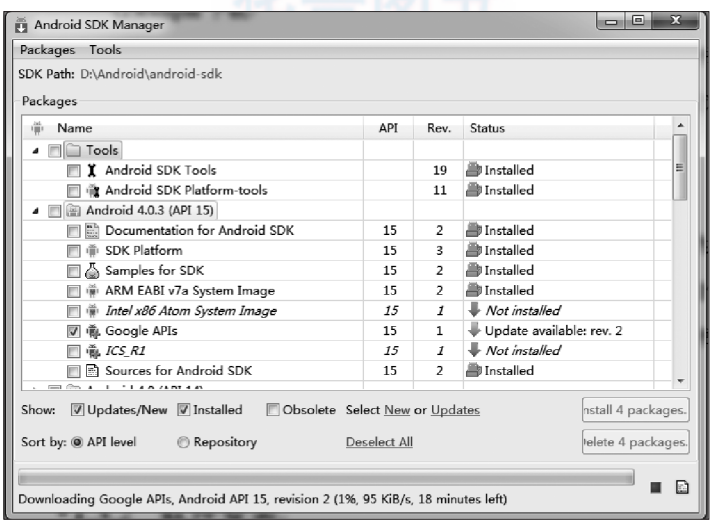


图 1-3　SDK 安装过程截图

1.3.2　安装 ADT

ADT 插件对于开发 Android 应用程序有非常重要的作用，ADT 和 Eclipse 集成之后，提供了一些工具来自动地创建、检测、测试和调试 Android 应用程序。可以在 Eclipse 中的 Install New Software 处进行安装。启动 Eclipse IDE 之后，其步骤如下：

步骤 1　在 Eclipse 菜单栏上选择 Help，单击下拉菜单 Install New Software 选项。

步骤 2　在 Work with 字段中，输入 https://dl-ssl.google.com/android/eclipse/，回车后 Eclipse 会链接到该网站，并生成可以下载插件的列表，如图 1-4 所示。

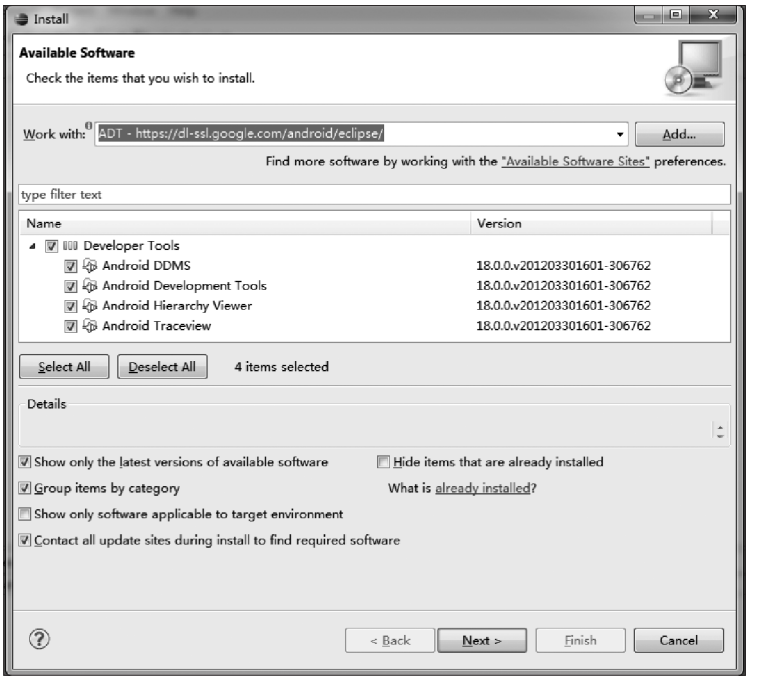


图 1-4　ADT 下载插件列表

注意　如果已经安装过会显示需要更新的项目。如输入的链接不能使用，可以使用 http://dl-ssl.google.com/android/eclipse/ 链接，或者下载离线安装包，单击图中的 Add 按钮，从 Archive 里面选择离线安装包的位置。

步骤 3　选择 Developer Tools，将全部选中其子选项，然后单击 Next 按钮。

步骤 4　Eclipse 将要查看 ADT 和 ADT 安装所需要工具的许可协议。查看许可协议，单击“I accept…”，然后单击 Finish 按钮。

步骤 5　Eclipse 将下载 Developer Tools 并安装，安装过程中 Eclipse 需要重新启动才能使用 ADT。

步骤 6 在 Eclipse 中安装 ADT 之后需要配置 Android SDK，单击菜单 Window–>Preferences，进入图 1-5 所示界面，选择你的 Android SDK 解压后的目录或者安装目录。

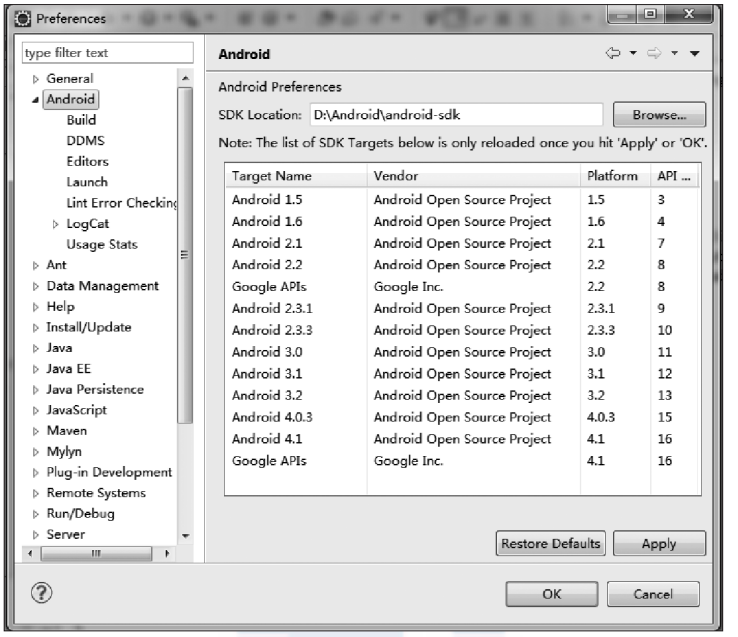


图 1-5 选择目录

步骤 7 新建 AVD（Android Vitural Device）。单击菜单 Window–>AVD Manager，进入图 1-6 所示的 AVD 管理器。

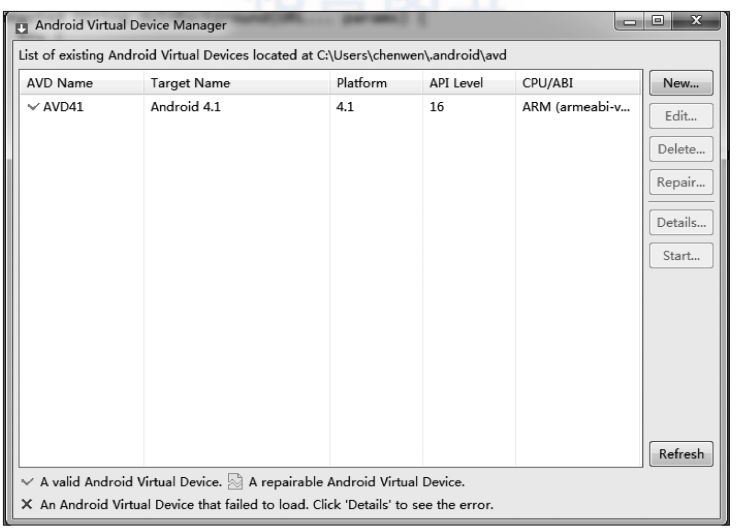


图 1-6 AVD 管理器

步骤 8　单击 New 按钮后，进入如图 1-7 所示的新建 AVD 详细界面。名称可以随便取，在 Target 下拉列表中选择你需要的 SDK 版本，SD 卡大小自定义，单击 Create AVD 按钮，创建 AVD 完毕。

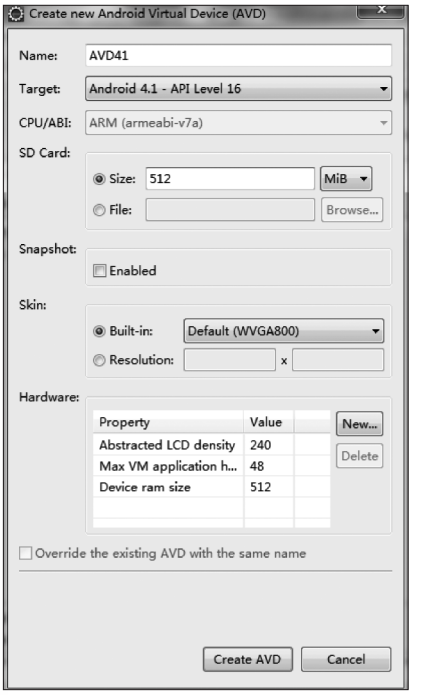


图 1-7　新建 AVD 详细界面

注意　如果需要添加（如 GPS 等）其他 AVD 选项，可以单击 New 按钮，从下拉菜单里面选择相应选项。

1.3.3　Hello World！

本节通过 Hello World 程序，介绍创建 Android 程序的过程。这段程序甚至不需要写一行代码，就可以全自动地创建好。

步骤 1　打开 Eclipse，选择菜单 File–>New–>Android Application Project，显示如图 1-8 所示的创建 Android 程序界面。

在图 1-8 所示的 Application Name 文本框中输入应用名称，在 Project Name 文本框中输入项目名称，在 Package Name 文本框中输入包名，在 Build SDK 下拉列表中选择项目目标 SDK 版本，在 Minimum Required SDK 下拉列表中选择项目所需要的最小 SDK 版本。

图 1-8　新建 Android 界面

注意 如果该项目为其他项目的库，应勾选 Mark this project as a library。

步骤 2 单击 Next 按钮之后，显示的界面如图 1-9 所示，用户可以定义应用显示的图标。Image 选择背景图片；Clipart 选择系统的切图；Text 中输入文字。Trim Surrounding Blank Space 选项设置前景图是否自动充满背景图；Additional Padding 设置前景图和背景图的显示比例；Foreground Scaling 设定背景的尺寸，Crop 为拉伸，Center 为居中；Shape 设定背景图的形状，None 设置背景为空，Square 设为方形背景，Circle 设为圆形背景；Backgroud Color 和 Foreground Color 分别设置显示图标的背景色和前景色。

图 1-9 选择应用图标

步骤 3 单击 Next 按钮之后，进入的界面如图 1-10 所示。选择 Activity 的显示方式：BlankActivity 创建空的 Acitivity，这个和传统的创建方式类似；MasterDetailFlow 将创建带“碎片”的程序，可以重复利用较大的屏幕。

步骤 4 单击 Next 按钮，填写 Activity 信息，进入的界面如图 1-11 所示。在 Activity Name 文本框中输入该 Acitivity 的名称；在 Layout Name 文本框中输入其对应的布局名称；在 Navigation Type 下拉列表中选择导航类型，该下拉列表包括的选项有 None、Tabs、Tabs+Swipes、Swipe Views+Title Strip 和 Dropdown，可以根据需要进行选择，本例子中选择 None；在 Hierarchical Parent 中选择父节点，用以指定向上按键指向的界面；在 Title 文本框中指定显示的名称。

步骤 5 单击 Finish 按钮进入如图 1-12 所示的布局编辑界面。

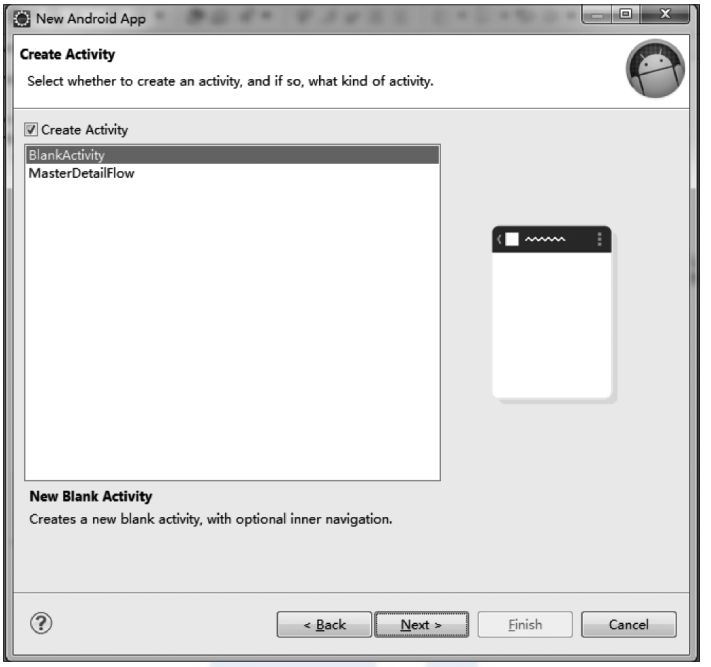


图 1-10　选择 Activity 类型

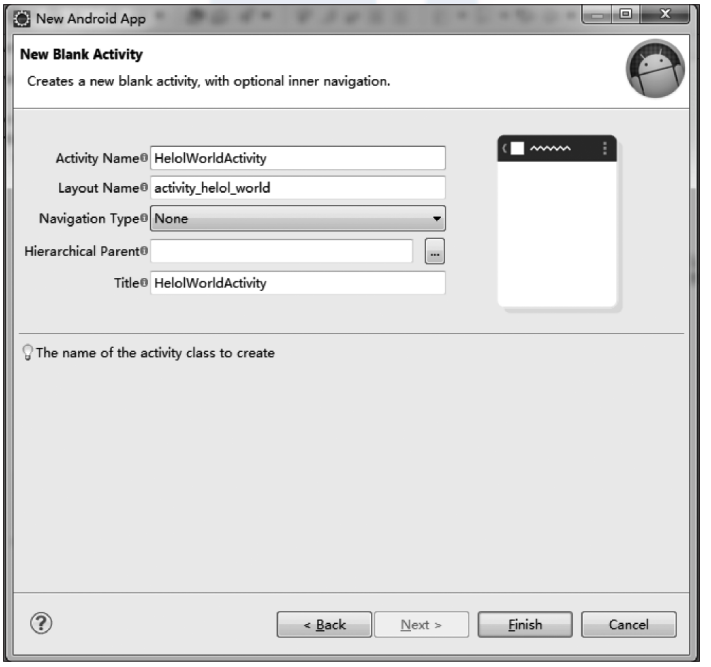


图 1-11　填写 Acitivity 信息

步骤 6　切换到代码编辑界面，如图 1-13 所示。

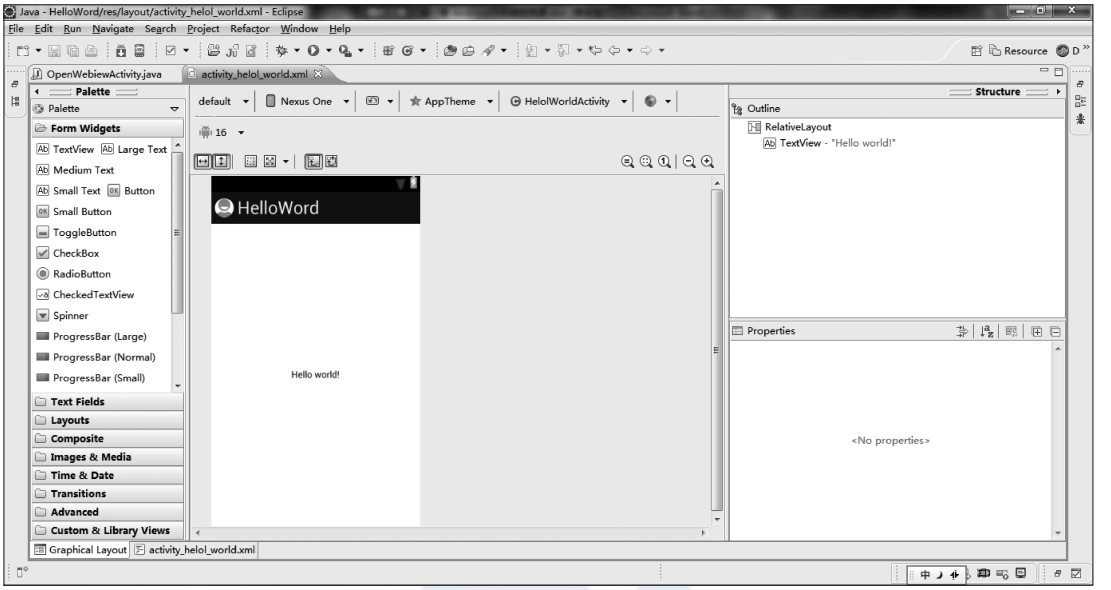


图 1-12 界面编辑

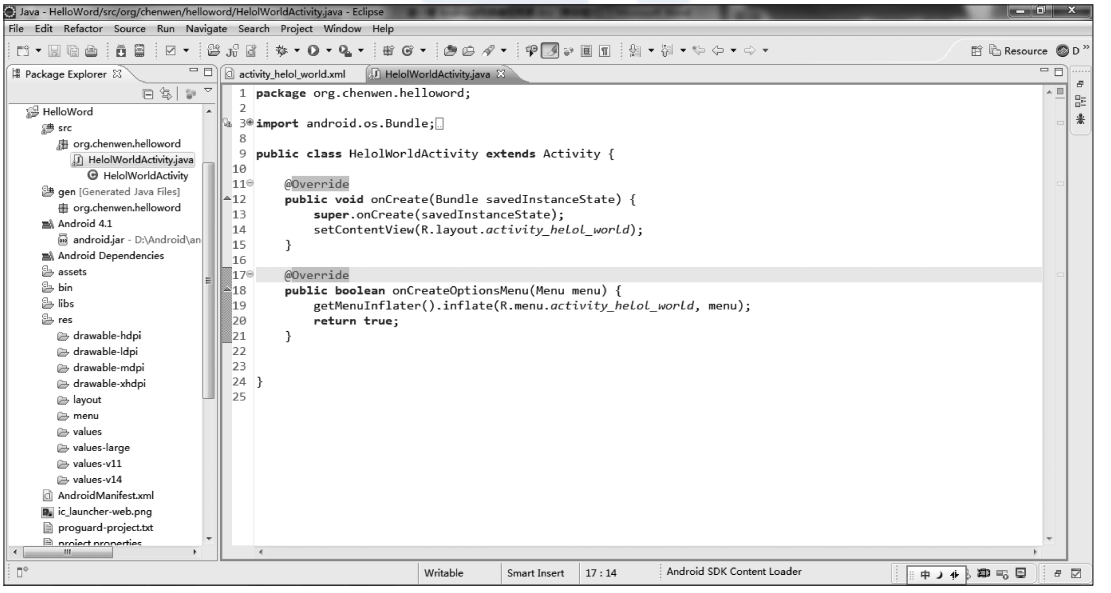


图 1-13 代码编辑界面

注意 若有错误，如 Project...is missing required source folder: 'gen'，则将 gen–>Android.Test–>R.java 这个文件删掉，Eclipse 会重新生成这个文件，并且不会再报错。

步骤 7 运行该项目前，需要进行一些设置，右击项目，在弹出的快捷菜单中依次单

击 Run as–>Run Configuration，进入如图 1-14 所示界面，选择需要运行的项目。

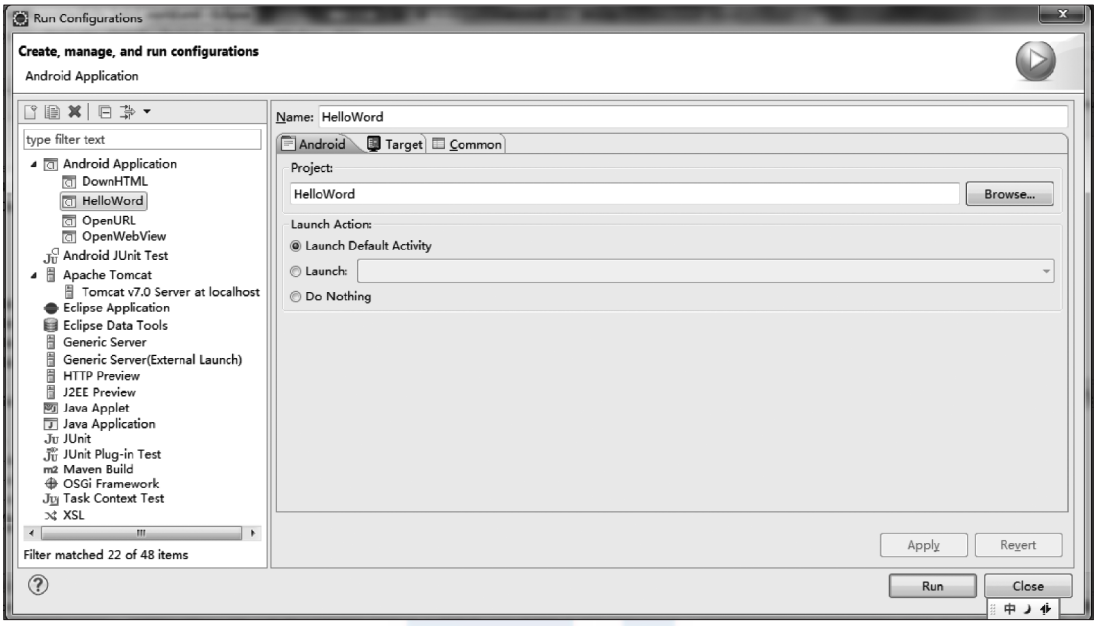


图 1-14　选择项目配置运行

选择 Target，切换到如图 1-15 所示的界面，在对应的复选框中打钩，以选择希望运行的 AVD。

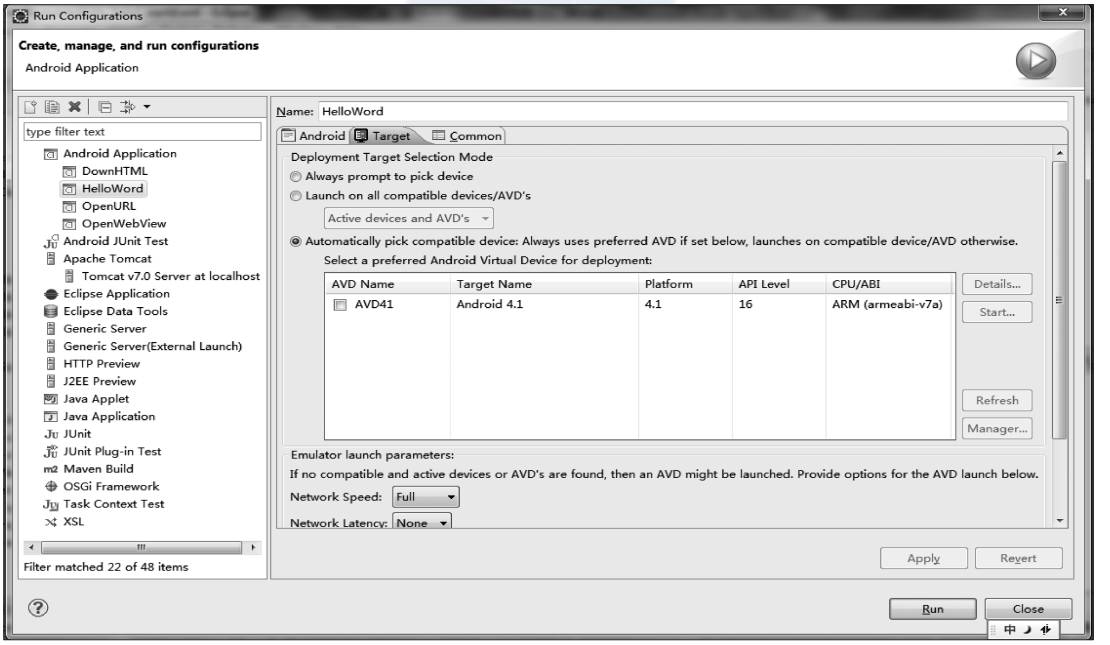


图 1-15　选择 AVD 运行目标

步骤 8 单击 Run 按钮，就可以运行该项目了。图 1-16 所示为 AVD 正在加载的界面，模拟器启动后如图 1-17 所示。

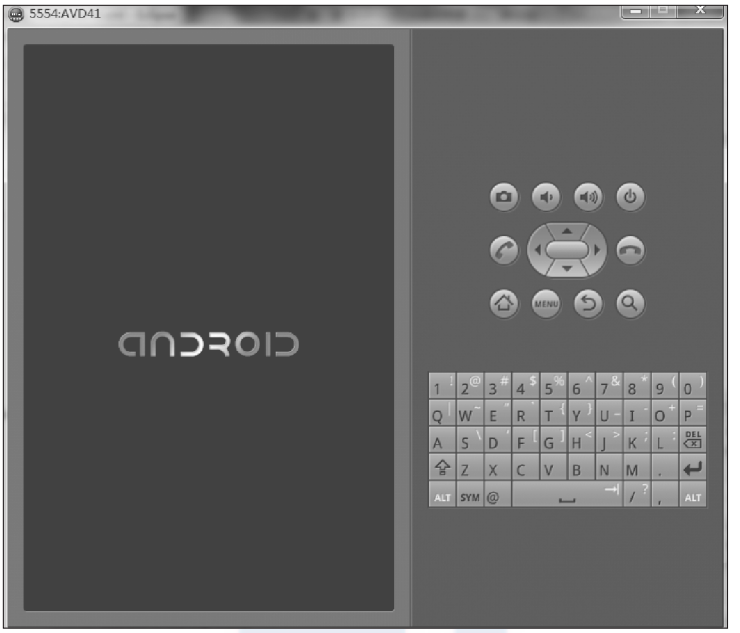


图 1-16 正在加载 AVD

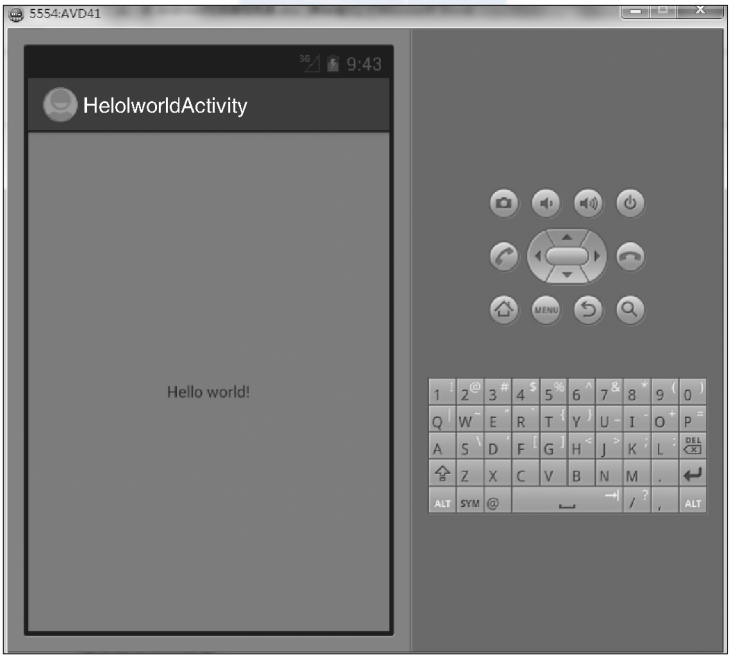


图 1-17 启动成功

1.4　网络应用实战案例

通过加载现有的 Web 页面来实现 Android 应用的功能是一种常见的方法。本节通过一个实际案例来展示如何加载一个现有页面，并在此页面的基础上增加需要的功能。

1.4.1　加载一个页面

如需在应用里面提供在线翻译的功能，最简单的方法就是给用户加载一个翻译的网站，用户进入该网站后自行输入想要翻译的单词，并查看翻译的结果。在界面上添加一个按钮，使得用户单击之后自动打开谷歌翻译网站。

步骤 1　在 Eclipse 里面创建项目。

步骤 2　添加字符串。在字符串文件 \res\values\strings.xml 里面（如下代码所示）添加需要的字符串，并指定显示的内容。

```
<string name="open_ulr">OpenURL</string>
```

步骤 3　添加 Button 控件。打开 res\layout\main.xml 文件，在布局文件中添加 Button 按钮。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/hello" />
   <Button
      android:id="@+id/open_ulr"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/open_ulr" />
</LinearLayout>
```

步骤 4　最简单的加载情况，调用系统的浏览器来打开一个页面，代码如下：

```
package org.chenwen.openurl;
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.Button;
public class OpenURL extends Activity {
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
                Button bt1 = (Button)findViewById(R.id.button1);
                bt1.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View arg0) {
                                Uri uri = Uri.parse("http://translate.google.cn/");
                                Intent intent = new Intent(Intent.ACTION_VIEW, uri);
                                startActivity(intent);
                        }
                });
        }
}
```

Uri.parse() 方法返回的是一个 URI 类型，通过这个 URI 可以访问一个网络上的或者本地的资源，Intent() 方法告诉系统调用哪个组件来打开这个 URI。这里指明是使用 Intent.ACTION_VIEW，这将调用系统里面的浏览器来打开指定的网页，如图 1-18 所示。这个网页会显示一个翻译网页。

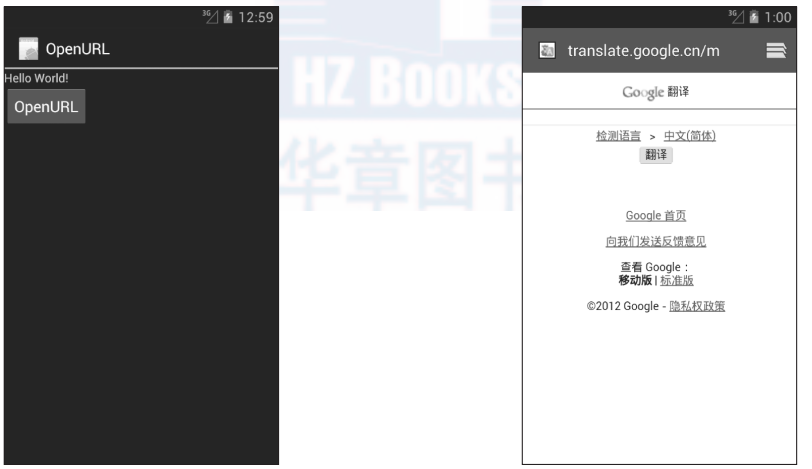


图 1-18 运行 OpenURL 应用和单击按钮 OpenURL 之后的效果

有时候，希望页面能够在应用内部打开，以方便添加一些需要的功能，这时可以在应用里面使用 WebView 控件。WebView 是 Android 里面的浏览器组件，负责打开 HTML 文件，setContent() 方法动态的添加布局，loadUrl() 方法从网址加载一个页面，loadDate() 和 loadDataWithBaseURL() 方法都是从字符串来加载一个页面。

重新建立一个项目，添加 WebView 控件到项目的主界面。

WebView 打开页面的方法如下：

```
WebView wv = (WebView)findViewById(R.id.webView1);
wv.loadUrl("http://translate.google.cn/m");
```

loadUrl 的原型如下：

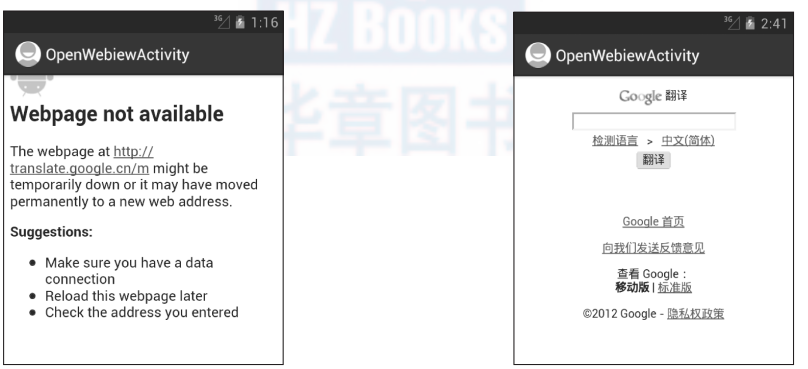
```
void loadData(String data, String mimeType, String encoding)
```

如果参数 encoding 的值为 base64，则必须使用 base64 编码之后的数据。其各参数含义如下。

- data 参数：数据字符串。
- mimeType 参数：表明数据的 MIME 类型，如 text/html。
- Encoding 参数：数据的编码。

注意　loadUrl 方法在遇到错误网页的时候不会报出异常，且 loadData 方法不能处理 js、https 等格式的页面特效。如果需要检测页面异常，可以先对页面进行判定，使用 loadDataWithBaseURL 可以加载 https 等特殊页面。

使用 WebView 控件后，运行应用，其效果如图 1-19a 所示。这是因为没有添加网络权限，在配置文件中添加权限后运行效果如图 1-19b 所示。



a）使用 WebView 控件后　　b）添加权限后

图 1-19　WebView 打开页面

实现上述功能的主要代码如下所示：

```
<manifest xmlns:android="http:// schemas.android.com/apk/res/android"
   package="org.chenwen.openwebview"
   android:versionCode="1"
   android:versionName="1.0" >
   <uses-sdk
```

```
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".OpenWebiewActivity"
            android:label="@string/title_activity_open_webiew" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

注意 添加权限的位置，要在 application 标签之前，否则在某些高版本的编译器中不能通过，或者没有效果。

在翻译输入框里面，输入 Android 单击“翻译”按钮，其翻译结果如图 1-20a 所示。其翻译结果显示时，打开的页面已经超出了应用的控制范围了，如果希望其翻译结果仍然显示在该应用内部，可以使用 WebView 的 setWebViewClient() 方法来解决这个问题。修改后翻译效果如图 1-20b 所示。

a）翻译界面　　b）修改后的翻译界面

图 1-20　翻译效果

实现上述功能的主要代码如下所示：

```
WebView wv = (WebView)findViewById(R.id.webView1);
```

```
wv.loadUrl("http://translate.google.cn/m");
wv.setWebViewClient(new WebViewClient(){
     public boolean shouldOverrideUrlLoading(WebView view, String url) {
               view.loadUrl(url);
               return true;
     }
});
```

1.4.2　下载一个页面

为了更好地服务使用者，在很多情况下需要保存用户搜索过的翻译页面。要实现这个功能，需要保存加载的页面。首先通过网址生成 URL 对象，然后打开链接，写入 Buffer 中，最后写入字符串中，方便进一步的处理。实现相应功能的主要代码如下所示：

```
try {
     URL newUrl = new URL("http:// translate.google.cn/m");
     URLConnection connect = newUrl.openConnection();
     DataInputStream dis = new DataInputStream(connect.getInputStream());
     BufferedReader in = new BufferedReader(new InputStreamReader(dis,"UTF-8"));
     String html = "";
     String readLine = null;
     while ((readLine = in.readLine()) != null) {
               html = html + readLine;
               Log.d("OpenWebiewActivity", readLine);
     }
     in.close();
} catch (MalformedURLException me) {
} catch (IOException ioe) {
}
```

1.5　小结

本章介绍 Android 的发展、Android 在网络方面的应用、Android 开发的设置、如何对现有的页面进行简单的加载下载等内容。接下来我们会围绕 Android 网络编程从核心概念到实战案例，一步一步地来深入理解 Android 网络编程。

HZ BOOKS
华章图书

第二篇

实 战 篇

本部分内容：

- Android 基本网络技术和编程实践
- Android 基本 Web 技术和编程实践
- Android 常见网络接口编程
- Android 网络模块编程
- Android 线程、数据存取、缓存和 UI 同步
- 基于 SIP 协议的 VoIP 应用
- 基于 XMPP 协议的即时通信应用

HZ BOOKS 华章图书

第 2 章　Android 基本网络技术和编程实践

本章主要介绍计算机网络的相关概念，包括 TCP/IP 分层模型及 IP、TCP、UDP 等主要协议。在此基础上，重点阐述在 Android 中如何使用 TCP 和 UDP 进行通信。最后，给出 Android Socket 编程的一个实际案例。

2.1　计算机网络及其协议

网络原指用一个巨大的虚拟画面，把所有东西连接起来。计算机网络最初的目的也只是将各个独立的计算机连接起来，但是现在的计算机网络所能实现的已远远超出了人们早期的构想。在人们生活中，计算机网络已经无处不在，如网络视频、网络购物、网络教育等。

2.1.1　计算机网络概述

计算机网络就是用物理链路将各个孤立的工作站或主机连接在一起，组成数据链路，从而达到资源共享和通信的目的。凡将地理位置不同且具有独立功能的多个计算机系统通过通信设备和线路连接起来，并以功能完善的网络软件（网络协议、信息交换方式及网络操作系统等）实现网络资源共享的系统，均可称为计算机网络。简单地说，计算机网络即连接两台或多台计算机进行通信的系统。

计算机网络的功能主要表现在硬件资源共享、软件资源共享和用户间信息交换等方面。

- 硬件资源共享：可以在全网范围内提供对处理资源、存储资源、输入输出资源等设备的共享，使用户节省投资，也便于集中管理和均衡分担负荷。
- 软件资源共享：允许互联网上的用户远程获得各类服务，如网络文件传送服务、远地进程管理服务和远程文件访问服务，从而避免软件研制上的重复劳动及数据资源的重复存储，也便于集中管理。
- 用户间信息交换：计算机网络为分布在各地的用户提供了强有力的通信手段。用户可以通过计算机网络传送电子邮件、发布新闻消息和进行电子商务活动。
- 提高计算机的可靠性和可用性：网络中的每台计算机都可通过网络相互成为后备机。一旦某台计算机出现故障，它的任务就可由其他的计算机代为完成，这样可以避免在单机情况下，一台计算机发生故障引起整个系统瘫痪的现象，从而提高系统的可靠性。而当网络中的某台计算机负担过重时，网络又可以将新的任务交给较空闲的计算机完成，均衡负载，从而提高了每台计算机的可用性。
- 分布式处理：通过算法将大型的综合性问题交给不同的计算机同时进行处理。用户

可以根据需要合理选择网络资源，就近快速地进行处理。

计算机网络中用于规定信息的格式以及如何发送和接收信息的一套规则称为网络协议或通信协议。网络协议是为计算机网络中进行数据交换而建立的规则、标准或约定的集合。不同的计算机之间必须使用相同的网络协议才能进行通信。

2.1.2　网络协议概述

大多数网络都采用分层的体系结构，每一层都建立在它的下层之上，同时向它的上一层提供一定的服务，而把如何实现这一服务的细节对上一层加以屏蔽。一台设备上的第 *n* 层与另一台设备上的第 *n* 层进行通信的规则就是第 *n* 层协议。在网络的各层中存在着许多协议，接收方和发送方同层的协议必须一致，否则一方将无法识别另一方发出的信息。网络协议是网络上所有设备（网络服务器、计算机及交换机、路由器、防火墙等）之间通信规则的集合，它规定了通信时信息必须采用的格式和这些格式的意义。通过网络协议，网络上各种设备才能够相互交换信息。

由于网络节点之间联系的复杂性，在制定协议时，通常把复杂成分分解成一些简单成分，然后再将它们复合起来。最常用的复合技术就是采用层次的方法，网络协议的层次结构如下：

- 结构中的每一层都规定有明确的任务及接口标准。
- 把用户的应用程序作为最高层。
- 除了最高层外，中间的每一层都向上一层提供服务，同时又是下一层的用户。
- 把物理通信线路作为最低层，它使用从最高层传送来的参数，是提供服务的基础。

为了使不同厂家生产的计算机能够相互通信，以便在更大的范围内建立计算机网络，国际标准化组织（ISO）在 1978 年提出了“开放系统互连参考模型”，即著名的 OSI/RM 模型（Open System Interconnection/Reference Model）。它将计算机网络体系结构的通信协议划分为七层，自下而上依次为：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。

其中低 4 层完成数据传送服务，上面 3 层面向用户。对于每一层，至少制定两项标准：服务定义和协议规范。前者给出了该层所提供的服务的准确定义，后者详细描述了该协议的动作和各种有关规程，以保证服务的提供。

- 应用层：是开放系统互连环境的最高层。应用层为操作系统或网络应用程序提供访问网络服务的接口。
- 表示层：为上层用户提供共同的数据或信息的语法表示变换。为了让采用不同编码方法的计算机在通信中能相互理解数据的内容，可以采用抽象的标准方法来定义数据结构，并采用标准的编码表示形式。表示层管理这些抽象的数据结构，并将计算机内部的表示形式转换成网络通信中采用的标准表示形式。数据压缩和加密也是表示层可提供的表示变换功能。
- 会话层：也称会晤层，主要功能是组织和同步不同的主机上各种进程间的通信（称为对话），负责在两个会话层实体之间进行对话连接的建立和拆除。会话层还提供在

数据流中插入同步点的机制，使得数据传输因网络故障而中断后，可以不必从头开始而是仅重传最近一个同步点以后的数据。

- 传输层：负责数据传送的最高层次。传输层完成同处于资源子网中的两个主机（即源主机和目的主机）间的连接和数据传输，也称为端到端的数据传输。
- 网络层：主要任务就是要选择合适的路由，使网络层的数据传输单元——分组能够正确无误地按照地址找到目的站。
- 数据链路层：负责在两个相邻的节点间的线路上无差错地传送以帧为单位的数据，每一帧包括一定的数据和必要的控制信息，在接收点接收到数据出错时要通知发送方重发，直到这一帧无误地到达接收节点。
- 物理层：定义了为建立、维护和拆除物理链路所需的机械的、电气的、功能的和规程的特性，其作用是使原始的数据比特流能在物理介质上传输。具体涉及接插件的规格、“0”或“1”信号的电平表示、收发双方的协调等内容。物理层为上一层的数据链路层提供一个物理连接，通过物理连接透明地传输比特流。所谓透明传输是指经实际电路传送后的比特流没有变化，任意组合的比特流都可以在这个电路上传输，物理层并不知道比特的含义。

2.1.3 IP、TCP 和 UDP 协议

由于 OSI/RM 模型过于复杂也难以实现，现实中广泛应用的是 TCP/IP 模型。TCP/IP 是一个协议集，是由 ARPA 于 1977 年到 1979 年推出的一种网络体系结构和协议规范。随着 Internet 的发展，TCP/IP 也得到进一步的研究开发和推广应用，成为 Internet 上的“通用语言”。

TCP/IP 模型也是分层模型，分为 4 层。OSI/RM 模型与 TCP/IP 模型的参考层次如图 2-1 所示。

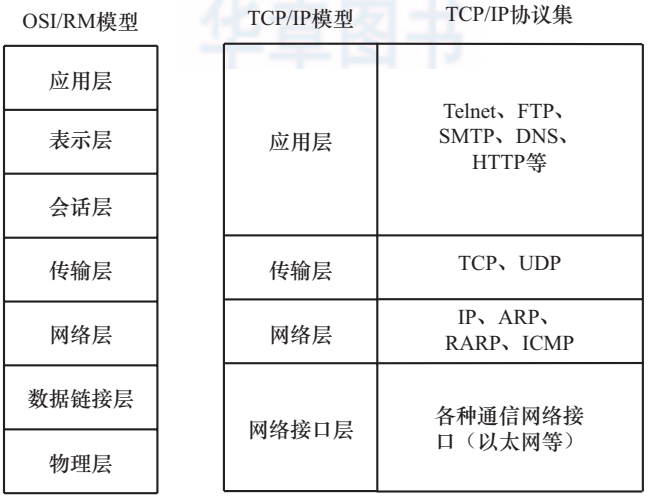


图 2-1 OSI/RM 模型与 TCP/IP 模型的参考层次

TCP/IP 模型 4 层分述如下。

- 应用层：应用层是大多数普通与网络相关的程序为了通过网络与其他程序通信所使用的层。在应用层中，数据以应用内部使用的格式进行传送，然后被编码成标准协议的格式。重要的例子如万维网使用的 HTTP 协议、文件传输使用的 FTP 协议、接收电子邮件使用的 POP3 和 IMAP 协议、发送邮件使用的 SMTP 协议，以及远程登录使用的 SSH 和 Telnet 等。所以用户通常是与应用层进行交互。
- 传输层：传输层响应来自应用层的服务请求，并向网络层发出服务请求。传输层提供两台主机之间透明的数据传输，通常用于端到端连接、流量控制或错误恢复。这一层的两个最重要的协议是 TCP（Transmission Control Protocol，传输控制协议）和 UDP（User Datagram Protocol，用户数据报协议）。
- 网络层：网络层提供端到端的数据包交付，换言之，它负责数据包从源发送到目的地，任务包括网络路由、差错控制和 IP 编址等。这一层包括的重要协议有 IP（版本 4 和版本 6）、ICMP（Internet Control Message Protocol，Internet 控制报文协议）和 IPSec（Internet Protocol Security，Internet 协议安全）。
- 网络接口层：是 TCP/IP 参考模型的最底层，负责通过网络发送和接收 IP 数据报；允许主机连入网络时使用多种现成的与流行的技术，如以太网、令牌网、帧中继、ATM、X.25、DDN、SDH、WDM 等。

一个应用层应用一般都会使用到两个传输层协议之一：面向连接的 TCP 传输控制协议和面向无连接的 UDP 用户数据报协议。下面分析 TCP/IP 协议栈中常用的 IP、TCP 和 UDP 协议。

1. IP 协议

互联网协议（Internet Protocol，IP）是用于报文交换网络的一种面向数据的协议。IP 是在 TCP/IP 协议中网络层的主要协议，任务是根据源主机和目的主机的地址传送数据。为达到此目的，IP 定义了寻址方法和数据报的封装结构。第一个架构的主要版本，现在称为 IPv4，仍然是最主要的互联网协议，如图 2-2 所示。当前世界各地正在积极部署 IPv6。

<table>
<tr><td colspan="5">32位</td><td rowspan="7">20
字
节</td></tr>
<tr><td>4位版本</td><td>4位首部长度</td><td>8位服务类型（TOS）</td><td colspan="2">16位总长度（字节数）</td></tr>
<tr><td colspan="3">16位标识</td><td>3位标志</td><td>13位片偏移</td></tr>
<tr><td>8位生存时间（TTL）</td><td colspan="2">8位协议</td><td colspan="2">16位首部检验和</td></tr>
<tr><td colspan="5">32位源IP地址</td></tr>
<tr><td colspan="5">32位目的IP地址</td></tr>
<tr><td colspan="5">选项（如果有）</td></tr>
<tr><td colspan="5">数据</td></tr>
</table>

图 2-2　IPv4 封装结构

下面对 IPv4 协议包的结构进行介绍，包含多个数据域。各个数据域的含义如下。

- 4 位版本：表示目前的协议版本号，数值是 4 表示版本为 4，因现在主要使用的还是版本为 4 的 IP 协议，所以 IP 有时也称为 IPv4。
- 4 位首部长度：头部的长度，它的单位是 32 位（4 字节），数值为 5 表示 IP 头部长度为 20 字节。
- 8 位服务类型（TOS）：这个 8 位字段由 3 位的优先权子字段（现在已经被忽略）、4 位的 TOS 子字段以及 1 位的未用字段（现在为 0）构成。4 位的 TOS 子字段包含最小延时、最大吞吐量、最高可靠性以及最小费用构成，对应位为 1 时指出上层协议对处理当前数据报所期望的服务质量。如果都为 0，则表示是一般服务。
- 16 位总长度（字节数）：总长度字段是指整个 IP 数据报的长度，以字节为单位。如数值为 00 30，换算成十进制为 48 字节，48 字节 =20 字节的 IP 头 +28 字节的 TCP 头。这个数据报只是传送的控制信息，还没有传送真正的数据，所以目前看到的总长度就是报头的长度。
- 16 位标识：标识字段唯一标识主机发送的每一份数据报。
- 3 位标志：该字段用于标记该报文是否分片，后面是否还有分片。
- 13 位片偏移：指当前分片在原数据报中相对于用户数据字段的偏移量，即在原数据报中的相对位置。
- 8 位生存时间（TTL）：TTL（Time-To-Live）生存时间字段设置了数据报可以经过的最多路由器数目。它指定了数据报的生存时间。TTL 的初始值由源主机设置，一旦经过一个处理它的路由器，它的值就减去 1。可根据 TTL 值判断服务器是什么系统和经过的路由器。举个例子，TTL 的十六进制初始值为 80H，换算成十进制为 128，Windows 操作系统的 TTL 初始值一般为 80H，UNIX 操作系统初始值为 FFH。
- 8 位协议：表示协议类型，6 表示传输层是 TCP 协议。
- 16 位首部检验和：当收到一份 IP 数据报后，同样对首部中的每个 16 位进行二进制反码的求和。由于接收方在计算过程中包含了发送方存在首部中的检验和，因此，如果首部在传输过程中没有发生任何差错，那么接收方计算的结果应该为全 1。如果结果不是全 1，即检验和错误，那么 IP 就丢弃收到的数据报，但是不生成差错报文，而是由上层发现丢失的数据报并进行重传。
- 32 位源 IP 地址和 32 位目的 IP 地址：实际这是 IPv4 协议中核心的部分。32 位的 IP 地址由一个网络 ID 和一个主机 ID 组成。源地址是指发送数据的源主机的 IP 地址，目的地址是指接收数据的目的主机的 IP 地址。
- 选项：长度不定，如果没有选项就表示这个字节的域等于 0。
- 数据：该 IPv4 协议包负载的数据。

2. TCP 协议

传输控制协议（Transmission Control Protocol，TCP）是一种面向连接的、可靠的、基

于字节流的传输层通信协议。

注意　IETF RFC 793 是 TCP 的正式规范。其网址为 http://tools.ietf.org/html/rfc793。

在 Internet 协议族中，传输层是位于网络层之上、应用层之下的中间层。不同主机的应用层之间经常需要可靠的、像管道一样的连接，但是网络层不提供这样的流机制，其只能提供不可靠的包交换，所以传输层就自然出现了。

应用层向传输层发送用于网间传输的、用 8 位字节表示的数据流，然后 TCP 协议把数据流分成适当长度的报文段（通常受该计算机连接的网络的数据链路层的最大传送单元 MTU 的限制）。之后 TCP 协议把结果包传给网络层，由它来通过网络将包传送给接收端实体的传输层。TCP 为了保证不发生丢包，就给每个包一个序号，同时序号也保证了传送到接收端实体的包能按序接收。然后接收端实体为已成功收到的包发回一个相应的确认（ACK）；如果发送端实体在合理的往返时延（RTT）内未收到确认，那么对应的数据包（假设丢失了）将会被重传。TCP 协议用一个校验和（Checksum）函数来检验数据是否有错误，在发送和接收时都要计算校验和。

TCP 协议头最少长度为 20 个字节，其协议包结构及所包含的字段如图 2-3 所示。

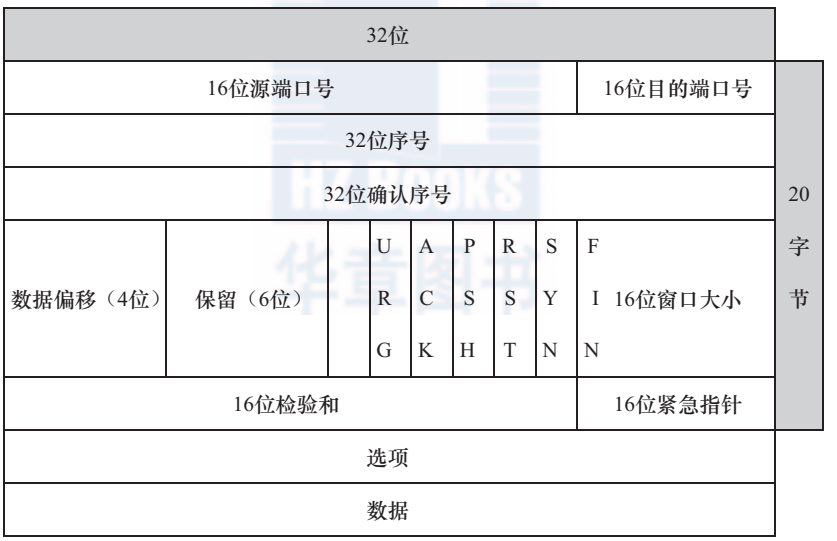


图 2-3　TCP 数据包格式

各个字段的含义如下：

- 16 位源端口号：源端口号是指发送数据的源主机的端口号，16 位的源端口中包含初始化通信的端口。源端口和源 IP 地址的作用是标识报文的返回地址。
- 16 位目的端口号：目的端口号是指接收数据的目的主机的端口号，16 位的目的端口域定义传输的目的地。这个端口指明报文接收计算机上的应用程序地址端口。
- 32 位序号：TCP 是面向字节流的，在一个 TCP 连接中传送的字节流中的每一个字

节都按顺序编号。整个要传送的字节流的起始序号必须在连接建立时设置。首部中的序号字段值则指的是本报文段所发送的数据的第一个字节的序号。

- 32 位确认序号：是期望收到对方下一个报文段的第一个数据字节的序号，若确认号为 N，则表明到序号 N-1 为止的所有数据都已正确收到。
- 4 位数据偏移：指出 TCP 报文段的数据起始处距离 TCP 报文段的起始处有多远，整个字段实际上指明了 TCP 报文段的首部长度。
- 保留（6 位）：为了将来定义新的用途而保留的位，但目前应置为 0。
- URG、ACK、PSH、RST、SYN、FIN：6 位标志域，依次对应为紧急标志、确认标志、推送标志、复位标志、同步标志、终止标志。
- 16 位窗口大小：指的是发送本报文段的一方的接收窗口，以便告诉对方，从本报文段首部中的确认号算起，接收方目前允许对方发送的数据量。因为接收方的数据缓存空间有限，该窗口值可作为接收方让发送方设置其发送窗口的依据。
- 16 位校验和：源机器基于数据内容计算一个数值，目的机器根据所接收到的数据内容也要计算出一个数值，这个数值要与源机器数值完全一样，从而证明数据的有效性。检验和字段检验的范围包括首部和数据两部分。这是一个强制性的字段，一定是由发送端计算和存储，并由接收端进行验证的。
- 16 位紧急指针：在 URG 标志为 1 时其才有效，指出了本报文段中的紧急数据的字节数。
- 选项：长度可变，最长可达 40 字节。当没有使用选项时，TCP 首部长度是 20 字节。
- 数据：该 TCP 协议包负载的数据。

在上述字段中，6 位标志域中各标志的功能如下。

- URG：紧急标志。该位为 1 表示该位有效。
- ACK：确认标志。该位被置位时表示确认序号栏有效。大多数情况下该标志位是置位的。
- PSH：推送标志。该标志位置位时，接收端不将该数据进行队列处理，而是尽可能快地将数据转由应用处理。在处理 Telnet 或 rlogin 等交互模式的连接时，该标志位总是置位的。
- RST：复位标志。该位被置位时表示复位相应的 TCP 连接。
- SYN：同步标志。在连接建立时用来同步序号。当 SYN=1 时而 ACK=0 时，表明这是一个连接请求报文段。对方若同意建立连接，则应在响应的报文段中是 SYN=1 和 ACK=1。即 SYN 置位时表示这是一个连接请求或者连接接受报文。
- FIN：结束标志，用来释放一个连接。

3. UDP 协议

用户数据报协议（UDP）是 TCP/IP 模型中一种面向无连接的传输层协议，提供面向事务的简单不可靠信息传送服务。UDP 协议基本上是 IP 协议与上层协议的接口。UDP 协议适用于端口分别运行在同一台设备上的多个应用程序中。

注意　IETF RFC 768 是 UDP 的正式规范，其网址为 http://tools.ietf.org/html/rfc768。

与 TCP 不同，UDP 并不提供对 IP 协议的可靠机制、流控制以及错误恢复功能等，在数据传输之前不需要建立连接。由于 UDP 比较简单，UDP 头包含很少的字节，所以比 TCP 负载消耗少。

UDP 适用于不需要 TCP 可靠机制的情形，比如，当高层协议或应用程序提供错误和流控制功能的时候。UDP 服务于很多知名应用层协议，包括网络文件系统（Network File System，NFS）、简单网络管理协议（Simple Network Management Protocol，SNMP）、域名系统（Domain Name System，DNS）以及简单文件传输系统（Trivial File Transfer Protocol，TFTP）。

UDP 数据报格式包含的字段如图 2-4 所示。

- 源端口 ：16 位。源端口是可选字段。当使用时，它表示发送程序的端口，同时它还被认为是没有其他信息的情况下需要被寻址的答复端口。如果不使用，设置其值为 0。
- 目的端口 ：16 位。目标端口在特殊互联网目标地址的情况下具有意义。
- 长度：16 位。UDP 用户数据报的总长度。
- 校验和：16 位。用于校验 UDP 数据报的 UDP 首部和 UDP 数据。
- 数据：包含上层数据信息。

16位	16位
源端口	目的端口
长度	校验和
数据	

图 2-4　UDP 数据报格式

2.2　在 Android 中使用 TCP、UDP 协议

上面介绍了 TCP 和 UDP 的报文结构，每段报文里面除了数据本身，还包含了包的目的地址和端口、包的源地址和端口以及其他各种附加的校验信息。这些包的长度是有限的，传输的时候需要将其分解为多个包，在到达传输的目的地址后再组合还原。如包有丢失或者破坏需要重传时，则乱序发送的包在达到时需要重新排序。处理这些过程是一项繁杂的工作，需要大量可靠的代码来完成。为了使程序员不必费心于上述这些底层具体细节，人们通过 Socket 对网络纠错、包大小、包重传等进行了封装。

2.2.1　Socket 基础

Socket 通常称为“套接字”。Socket 字面上的中文意思为“插座”。一台服务器可能会提供很多服务，每种服务对应一个 Socket（也可以这么说，每个 Socket 就是一个插座，客户若是需要哪种服务，就将插头插到相应的插座上面），而客户的“插头”也是一个

Socket。Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层，它是一组接口。Socket 把复杂的 TCP/IP 协议族隐藏在 Socket 接口后面，对用户来说，一组简单的接口就是全部，让 Socket 去组织数据，以符合指定的协议。Socket 用于描述 IP 地址和端口，是一个通信链的句柄。应用程序通常通过套接字向网络发出请求或者应答网络请求。

Socket 的基本操作包括：

- 连接远程机器
- 发送数据
- 接收数据
- 关闭连接
- 绑定端口
- 监听到达数据
- 在绑定的端口上接受来自远程机器的连接

服务器要和客户端通信，两者都要实例化一个 Socket。服务器和客户端的 Socket 是不一样的，客户端可以实现连接远程机器、发送数据、接收数据、关闭连接等，服务器还需要实现绑定端口，监听到达的数据，接受来自远程机器的连接。Android 在包 java.net 里面提供了两个类：ServerSocket 和 Socket，前者用于实例化服务器的 Socket，后者用于实例化客户端的 Socket。在连接成功时，应用程序两端都会产生一个 Socket 实例，操作这个实例，完成客户端到服务器所需的会话。

接下来分析一些重要的 Socket 编程接口。首先是如何构造一个 Socket，常用的构造客户端 Socket 的方法有以下几种。

- Socket()：创建一个新的未连接的 Socket。
- Socket(Proxy proxy)：使用指定的代理类型创建一个新的未连接的 Socket。
- Socket(String dstName,int dstPort)：使用指定的目标服务器的 IP 地址和目标服务器的端口号，创建一个新的 Socket。
- Socket(String dstName,int dstPort,InetAddress localAddress,int localPort)：使用指定的目标主机、目标端口、本地地址和本地端口，创建一个新的 Socket。
- Socket(InetAddress dstAddress,int dstPort)：使用指定的本地地址和本地端口，创建一个新的 Socket。
- Socket(InetAddress dstAddress,int dstPort,InetAddress localAddress,int localPort)：使用指定的目标主机、目标端口、本地地址和本地端口，创建一个新的 Socket。

其中，proxy 为代理服务器地址，dstAddress 为目标服务器 IP 地址，dstPort 为目标服务器的端口号（因为服务器的每种服务都会绑定在一个端口上面），dstName 为目标服务器的主机名。Socket 构造函数代码举例如下所示：

```
Socket client=new Socket("192.168.1.23", 2012);
// 第一个参数是目标服务器的 IP 地址，2012 是目标服务器的端口号
```

```
Socket sock = new Socket(new Proxy(Proxy.Type.SOCKS,
new InetSocketAddress("test.domain.org", 2130)));
// 实例化一个 Proxy，以该 Proxy 为参数，创建一个新的 Socket
```

注意　0~1023 端口为系统保留，用户设定的端口号应该大于 1023。

Socket 类的重要方法如表 2-1 所示。

表 2-1　Socket 类重要方法

方 法 原 型	功 能 描 述
Public InputStream getInputStream()	读出该 Socket 中的数据
public OutputStream getOutputStream()	向该 Socket 中写入数据
public synchronized void close()	关闭该 Socket

上面是构造 Socket 客户端的几种常用的方法，构造服务器端的 ServerSocket 的方法有以下几种。

- ServerSocket()：构造一个新的未绑定的 ServerSocket。
- ServerSocket(int port)：构造一个新的 ServerSocket 实例并绑定到指定端口。如果 port 参数为 0，端口将由操作系统自动分配，此时进入队列的数目将被设置为 50。
- ServerSocket(int port,int backlog)：构造一个新的 ServerSocket 实例并绑定到指定端口，并且指定进入队列的数目。如果 port 参数为 0，端口将由操作系统自动分配。
- ServerSocket(int port,int backlog,InetAddress localAddress)：构造一个新的 ServerSocket 实例并绑定到指定端口和指定的地址。如果 localAddress 参数为 null，则可以使用任意地址，如果 port 参数为 0，端口将由操作系统自动分配。

下面举例说明 ServerSocket 的构建方法，代码如下所示：

```
ServerSocket socketserver=new ServerSocket(2012);
// 2012 表示服务器要监听的端口号
```

构造完 ServerSocket 之后，需要调用 ServerSocket.accept() 方法来等待客户端的请求（因为 Socket 都是绑定在端口上面的，所以知道是哪个客户端请求的）。accept() 方法就会返回请求这个服务的客户端的 Socket 实例，然后通过返回的这个 Socket 实例的方法，操作传输过来的信息。当 Socket 对象操作完毕之后，使用 close() 方法将其关闭。

ServerSocket 类的重要方法如表 2-2 所示。

表 2-2　ServerSocket 类重要方法

方 法 原 型	功 能 描 述
public Socket accept ()	等待 Socket 请求，直到连接被打开，该方法返回一个刚刚打开的连接 Socket 对象
public void close()	关闭该服务器 Socket

Socket 一般有两种类型：TCP 套接字和 UDP 套接字。

TCP 和 UDP 在传输过程中的具体实现方法不同。两者都接收传输协议数据包并将其内容向前传送到应用层。TCP 把消息分解成数据包（数据报，datagrams）并在接收端以正确的顺序把它们重新装配起来，TCP 还处理对遗失数据包的重传请求，位于上层的应用层要处理的事情就少多了。UDP 不提供装配和重传请求这些功能，它只是向前传送信息包。位于上层的应用层必须确保消息是完整的，并且是以正确的顺序装配的。

接下来我们分别详细讨论使用 TCP 和 UDP 通信的一些细节。

2.2.2 使用 TCP 通信

TCP 建立连接之后，通信双方都同时可以进行数据的传输；在保证可靠性上，采用超时重传和捎带确认机制；在流量控制上，采用滑动窗口协议，协议中规定，对于窗口内未经确认的分组需要重传；在拥塞控制上，采用慢启动算法。TCP 通信的原理示意图如图 2-5 所示。

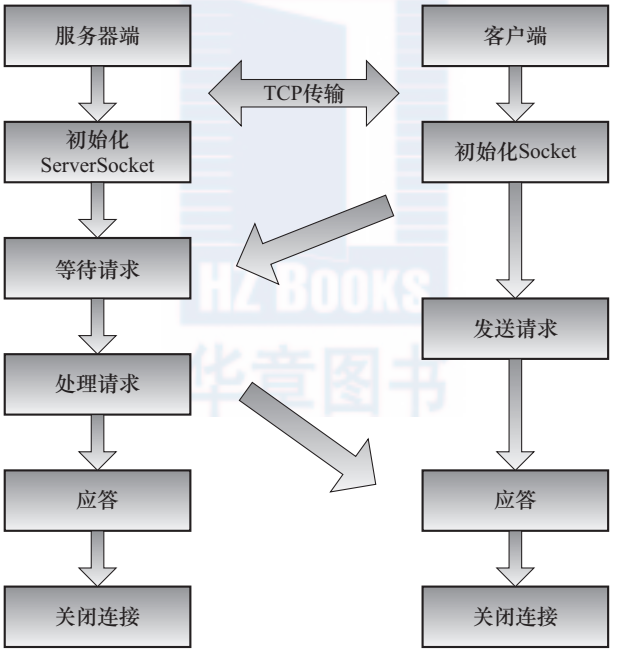


图 2-5 TCP 通信原理示意图

TCP 服务器端工作的主要步骤如下。

步骤 1 调用 ServerSocket（int port）创建一个 ServerSocket，并绑定到指定端口上。

步骤 2 调用 accept()，监听连接请求，如果客户端请求连接，则接受连接，返回通信套接字。

步骤 3 调用 Socket 类的 getOutputStream() 和 getInputStream() 获取输出和输入流，开始网络数据的发送和接收。

步骤 4　关闭通信套接字。

示例代码如下所示：

```
// 创建一个 ServerSocket 对象
ServerSocket serverSocket = null;
try {
      // TCP_SERVER_PORT 为指定的绑定端口，为 int 类型
      serverSocket = new ServerSocket(TCP_SERVER_PORT);
      // 监听连接请求
      Socket socket = serverSocket.accept();
      // 写入读 Buffer 中
      BufferedReader in = new BufferedReader(new
      // 获取输入流
      InputStreamReader(socket.getInputStream()));
      // 放到写 Buffer 中
      BufferedWriter out = new BufferedWriter(new
      // 获取输出流
      OutputStreamWriter(socket.getOutputStream()));
      // 读取接收信息，转换为字符串
      String incomingMsg = in.readLine() + System.getProperty("line.separator");
      // 生成发送字符串
      String outgoingMsg = "goodbye from port " + TCP_SERVER_PORT +
      System.getProperty("line.separator");
      // 将发送字符串写入上面定义的 BufferedWriter 中
      out.write(outgoingMsg);
      // 刷新，发送
      out.flush();
      // 关闭
      socket.close();
} catch (InterruptedIOException e) {
      // 超时错误
      e.printStackTrace();
      // IO 异常
} catch (IOException e) {
      // 打印错误
      e.printStackTrace();
} finally {
      // 判定是否初始化 ServerSocket 对象，如果初始化则关闭 serverSocket
      if (serverSocket != null) {
                try {
                    serverSocket.close();
            } catch (IOException e) {
                    e.printStackTrace();
            }
      }
}
```

TCP 客户端工作的主要步骤如下。

步骤 1 调用 Socket() 创建一个流套接字，并连接到服务器端。

步骤 2 调用 Socket 类的 getOutputStream() 和 getInputStream() 方法获取输出和输入流，开始网络数据的发送和接收。

步骤 3 关闭通信套接字。

编写 TCP 客户端代码如下所示：

```
try {
      //初始化 Socket，TCP_SERVER_PORT 为指定的端口，int 类型
      Socket socket = new Socket("localhost", TCP_SERVER_PORT);
      //获取输入流
      BufferedReader in = new BufferedReader(new
      InputStreamReader(socket.getInputStream()));
      //生成输出流
      BufferedWriter out = new BufferedWriter(new
      OutputStreamWriter(socket.getOutputStream()));
      //生成输出内容
      String outMsg = "TCP connecting to " + TCP_SERVER_PORT +
      System.getProperty("line.separator");
      //写入
      out.write(outMsg);
      //刷新，发送
      out.flush();
      //获取输入流
      String inMsg = in.readLine() + System.getProperty("line.separator");
      Log.i("TcpClient", "received: " + inMsg);
      //关闭连接
      socket.close();
      } catch (UnknownHostException e) {
                e.printStackTrace();
      } catch (IOException e) {
                e.printStackTrace();
}
```

注意 无论是客户端还是服务器端，都要添加 UnknownHostException（处理网络异常）和 IOException（处理 I/O 异常）异常处理。

在 Android 配置文件中需要添加下面的权限：

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

2.2.3 使用 UDP 通信

UDP 有不提供数据报分组、组装和不能对数据包排序的缺点，也就是说，当报文发送

之后，是无法得知其是否安全完整到达的。UDP 用来支持那些需要在计算机之间传输数据的网络应用，包括网络视频会议系统在内的众多的客户端 / 服务器模式的网络应用都需要使用 UDP 协议。UDP 协议的主要作用是将网络数据流量压缩成数据报的形式。一个典型的数据报就是一个二进制数据的传输单位。UDP 传输原理示意图如图 2-6 所示。

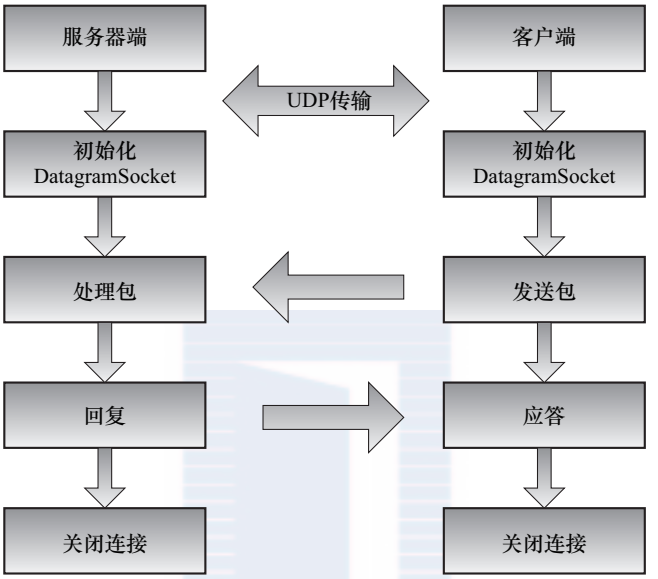


图 2-6　UDP 传输原理示意图

UDP 服务器端工作的主要步骤如下。

步骤 1　调用 DatagramSocket(int port) 创建一个数据报套接字，并绑定到指定端口上。

步骤 2　调用 DatagramPacket（byte[]buf,int length），建立一个字节数组以接收 UDP 包。

步骤 3　调用 DatagramSocket 类的 receive()，接受 UDP 包。

步骤 4　关闭数据报套接字。

示例代码如下所示：

```
// 接收的字节大小，客户端发送的数据不能超过 MAX_UDP_DATAGRAM_LEN
byte[] lMsg = new byte[MAX_UDP_DATAGRAM_LEN];
// 实例化一个 DatagramPacket 类
DatagramPacket dp = new DatagramPacket(lMsg, lMsg.length);
// 新建一个 DatagramSocket 类
DatagramSocket ds = null;
try {
      // UDP 服务器监听的端口
      ds = new DatagramSocket(UDP_SERVER_PORT);
      // 准备接收数据
      ds.receive(dp);
} catch (SocketException e) {
      e.printStackTrace();
```

```
} catch (IOException e) {
      e.printStackTrace();
} finally {
      // 如果 ds 对象不为空，则关闭 ds 对象
      if (ds != null) {
                ds.close();
      }
}
```

UDP 客户端工作的主要步骤如下。

步骤 1 调用 DatagramSocket() 创建一个数据包套接字。

步骤 2 调用 DatagramPacket（byte[]buf,int offset,int length,InetAddress address,int port），建立要发送的 UDP 包。

步骤 3 调用 DatagramSocket 类的 send() 发送 UDP 包。

步骤 4 关闭数据报套接字。

示例代码如下所示：

```
// 定义需要发送的信息
String udpMsg = "hello world from UDP client " + UDP_SERVER_PORT;
// 新建一个 DatagramSocket 对象
DatagramSocket ds = null;
try {
      // 初始化 DatagramSocket 对象
      ds = new DatagramSocket();
      // 初始化 InetAddress 对象
      InetAddress serverAddr = InetAddress.getByName("127.0.0.1");
      DatagramPacket dp;
      // 初始化 DatagramPacket 对象
      dp = new Datagram
      Packet(udpMsg.getBytes(), udpMsg.length(), serverAddr, UDP_SERVER_PORT);
      // 发送
      ds.send(dp);
}
// 异常处理
// Socket 连接异常
catch (SocketException e) {
      e.printStackTrace();
// 不能连接到主机
}catch (UnknownHostException e) {
      e.printStackTrace();
// 数据流异常
} catch (IOException e) {
      e.printStackTrace();
// 其他异常
} catch (Exception e) {
      e.printStackTrace();
} finally {
      // 如果 DatagramSocket 已经实例化，则需要将其关闭
```

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

2.3　Socket 实战案例

2.3.1　Socket 聊天举例

本节以 Socket 聊天为例，实现了从服务器端到客户端的聊天功能。

服务器端任务如下：ChatServer 类负责开启 Server 端服务；ReceiveMsg 负责接收消息；SendMsg 负责发送消息；Server 端响应请求，向 Client 端返回数据。其主要代码如下所示：

```
// Socket 聊天服务器端
// 继承 Thread 类，创建一个线程
public class ChatServer extends Thread {
      private ChatServer() throws IOException {
                // 创建 Socket 服务器
                CreateSocket();
      }
      // 重写 run() 方法
      public void run() {
                Socket client;
                // 定义接收的文本
                String txt;
                try {
                          // 始终在监听
                          while (true) {
                          // 开启线程，实时监听 socket 端口
                          // 获取应答消息
                                    client=ResponseSocket();
                                    // 响应客户端连接请求
                                    while(true) {
                                              // 接收客户端消息
                                              txt=ReceiveMsg(client);
                                              System.out.println(txt);
                                              // 连接获得客户端发来消息
                                              // 并将其显示在 Server 端的屏幕上
                                              SendMsg(client,txt);
                                              // 向客户端返回消息
                                              if(true)break;
                                              // 中断，继续等待连接请求
                                    }
                                    // 关闭此次连接
                                    CloseSocket(client);
```

```
                    }
                }
                catch (IOException e) {
                    System.out.println(e);
                }
        }
        //定义一个 ServerSocket 对象
        private ServerSocket server = null;
        //定义端口号
        private static final int PORT = 5000;
        //定义写 Buffer
        private BufferedWriter writer;
        //定义读 Buffer
        private BufferedReader reader;
        //实例化 ServerSocket
        private void CreateSocket() throws IOException{
                server = new ServerSocket(PORT, 100);
        }
        //接收消息
        private Socket ResponseSocket() throws IOException{
                Socket client = server.accept();
                return client;
        }
        //关闭打开的连接和缓存
        private void CloseSocket(Socket socket) throws IOException{
                reader.close();
                writer.close();
                socket.close();
        }
        //封装发送消息的方法
        private void SendMsg(Socket socket,String Msg) throws IOException{
                //要发送的消息写入 BufferedWriter 中
                writer = new BufferedWriter(
                        new OutputStreamWriter(socket.getOutputStream()));
                //添加空行分隔符
                writer.write(Msg+"\n");
                //刷新，发送
                writer.flush();
        }
        //接收消息的方法
        private String ReceiveMsg(Socket socket) throws IOException{
                //保存到读 Buffer
                reader = new BufferedReader(
                //将接收到的信息写入缓冲区
                new InputStreamReader(socket.getInputStream()));
                //将接收到的信息保存到字符串中
                String txt="Sever send:"+reader.readLine();
                //以字符串的方式返回接收到的信息
                return txt;
        }
}
```

启动服务器的代码如下所示：

```
//创建聊天服务器
ChatServer chatserver = new ChatServer ();
//检测服务器是否已经启动，如果没有则启动服务器
if (chatserver != null) {
      chatserver.start();
}
```

客户端的任务如下：

客户端发起连接请求，并向服务器端发送数据；客户端接收服务器端的数据。其主要代码如下所示：

```
    //客户端获取消息类
    private Socket RequestSocket(String host,int port)throws UnknownHostException,
IOException{
          //新建 Socket 类
          Socket socket = new Socket(host, port);
          return socket;
    }
    //客户端发送消息类
    private void SendMsg(Socket socket,String msg) throws IOException{
          //将要发送的消息写入 Buffer 中
          BufferedWriter writer = new BufferedWriter(new
                    OutputStreamWriter(socket.getOutputStream()));
          //格式转换
          writer.write(msg.replace("\n", " ")+"\n");
          //刷新，发送
          writer.flush();
    }
    //客户端接收消息
    private String ReceiveMsg(Socket socket) throws IOException{
          //写入读 Buffer 中
          BufferedReader reader = new BufferedReader(new
          //获取接收的消息到数据流中
                    InputStreamReader(socket.getInputStream()));
          //读取消息到字符串中
          String Msg=reader.readLine();
          //以字符串的方式返回消息
          return Msg;
    }
```

2.3.2　FTP 客户端

文件传输协议（File Transfer Protocol，FTP）是用于在网络上进行文件传输的一套标准协议。它属于网络传输协议的应用层。

FTP 是一个 8 位的客户端 - 服务器协议，能操作任何类型的文件。FTP 服务一般运行在 20 和 21 两个端口。端口 20 用于在客户端和服务器之间传输数据流，而端口 21 用于传输控制流，并且是命令通向 FTP 服务器的进口。当数据通过数据流传输时，控制流处于空闲状态。运行 FTP 服务的许多站点都开放匿名服务，在这种设置下，用户不需要账号就可以登录服务器，默认匿名用户的用户名是 anonymous，这个账号不需要密码；不开放匿名服务的则要求输入用户名和密码。

在 Android 中可以使用第三方的库来操作 FTP，这里使用 Apache 的包，其下载地址为 http://commons.apache.org/net/download_net.cgi。其文件名为 commons-net-3.2-bin.zip，解压之后得到需要使用的包 commons-net-3.2.jar。

步骤 1 在项目引入包 commons-net-3.2.jar 之后，导入需要其中的 FTPClient 类，代码如下：

```
import org.apache.commons.net.ftp.FTPClient;
```

步骤 2 初始化 FTPClient，代码如下：

```
mFtp=new FTPClient();
```

步骤 3 设置登录地址和端口号，代码如下：

```
mFtp.connect(FTP_URL,21);
```

步骤 4 设置登录用户名和密码，代码如下：

```
mFtp.login(Name, Password);
```

步骤 5 设置文件类型和采用被动传输方式，代码如下：

```
mFtp.setFileType(FTP.BINARY_FILE_TYPE);
mFtp.enterLocalPassiveMode();
```

步骤 6 传输文件，代码如下：

```
boolean aRtn=mFtp.storeFile(remoteFileName, aInputStream);
// 成功返回 true
aInputStream.close();
```

步骤 7 关闭连接，代码如下：

```
mFtp.disconnect();
```

HZ BOOKS 华章图书

注意　FTP 支持以下两种模式。

主动模式：FTP 客户端向服务器的 FTP 控制端口（默认是 21 端口）发送请求，服务器接受连接，建立一条命令链路，当需要传送数据时候，客户端在命令链路上用 PORT 命令通知服务器，服务器从 20 端口向客户端的该端口发送连接请求，建立一条数据链路来传送数据。在数据链路建立的过程中因为是服务器主动请求的，所以称为主动模式。

被动模式：FTP 客户端向服务器的 FTP 控制端口发送连接请求，服务器接收连接，建立一条命令链路，当需要传送数据时候，服务器在命令链路上用 PASV 命令通知客户端；客户端向服务器的该端口发送连接请求，建立一条数据链路来传送数据。在数据链路建立的过程中因为是服务器被动等待客户端请求的，所以称为被动模式。

Android FTP 客户端核心代码如下：

```
// 导入需要的 FTPClient 类
import org.apache.commons.net.ftp.FTPClient;
// 初始化 FTPClient 对象
FTPClient ftpClient = new FTPClient();
try {
      // 连接到指定的 FTP 服务器
      ftpClient.connect(InetAddress.getByName(SERVER));
      // 使用用户名和密码登录 FTP 服务器
      ftpClient.login(USERNAME, PASSWORD);
      // 检测返回的字符串里面是否包含 250
      // 250 响应代码表示“行为完成”
      if (ftpClient.getReplyString().contains("250")) {
                // 设置文件类型
                ftpClient.setFileType(
                // 默认使用的是 ASCII 编码的，这里设置为二进制文件
                        org.apache.commons.net.ftp.FTP.BINARY_FILE_TYPE);
                // 定义一个输入缓冲区
                BufferedInputStream buffIn = null;
                // 将文件加载到缓冲区中
                buffIn = new BufferedInputStream(new
                        FileInputStream(FULL_PATH_TO_LOCAL_FILE));
                // 设置客户端 PASV 模式 （客户端主动连服务器端；端口用 20）
                ftpClient.enterLocalPassiveMode();
                // 存储文件，返回是否成功
                boolean result = ftpClient.storeFile(localAsset.
                        getFileName(), progressInput);
                // 关闭缓冲区
                buffIn.close();
                // 登出
                ftpClient.logout();
                // 断开连接
```

```
                ftpClient.disconnect();
            }
        } catch (SocketException e) {
        } catch (UnknownHostException e) {
        } catch (IOException e) {
        }
```

2.3.3 Telnet 客户端

Telnet 协议是 TCP/IP 协议族中的一员，是 Internet 远程登录服务的标准协议和主要方式。它为用户提供了在本地计算机上完成远程主机上的工作的能力。在终端使用者的计算机上使用 Telnet 程序，用它连接到服务器。终端使用者可以在 Telnet 程序中输入命令，这些命令会在服务器上运行，就像直接在服务器的控制台上输入一样。通过 Telnet 在本地就能控制服务器。要开始一个 Telnet 会话，必须输入用户名和密码来登录服务器。Telnet 是常用的远程控制 Web 服务器的方法。

在 Android 中可以使用第三方的库来操作 Telnet，这里使用 Apache 的包，其下载地址为 http://commons.apache.org/net/download_net.cgi。其文件名为 commons-net-3.2-bin.zip，解压之后得到需要使用的包 commons-net-3.2.jar。

步骤 1 在项目引入包 commons-net-3.2.jar 之后，导入其中的 TelnetClient 类，代码如下：

```
import org.apache.commons.net.telnet.TelnetClient;
```

步骤 2 初始化 TelnetClient，代码如下：

```
TelnetClient tc = new TelnetClient();
```

步骤 3 打开连接，代码如下：

```
tc.connect(remoteip, remoteport);
```

步骤 4 读写数据，代码如下：

```
tc.getInputStream()   tc.getOutputStream()
```

步骤 5 断开 Telnet 连接，代码如下：

```
telnet.disconnect();
```

Android Telnet 客户端核心代码如下：

```
//定义一个 TelnetClient
TelnetClient telnet;
telnet = new TelnetClient();
      try{
                //建立连接
                telnet.connect(remoteip, remoteport);
      }
      catch (IOException e) {
                e.printStackTrace();
                //捕获到异常，停止程序
                System.exit(1);
      }
      IOUtil.readWrite(telnet.getInputStream(), telnet.getOutputStream(),
                System.in, System.out);
      try{
                //断开连接
                telnet.disconnect();
      }
      catch (IOException e) {
                e.printStackTrace();
      }
}
```

其中调用的 IOUtil 类封装了一些读写操作，其代码如下：

```
//IOUtil 类封装了一些读写操作
public final class IOUtil{
      public static final void readWrite(final InputStream remoteInput,
      final OutputStream remoteOutput,
      final InputStream localInput,
      final OutputStream localOutput){
                //定义读写的线程
                Thread reader, writer;
                //定义读线程的具体操作
                reader = new Thread(){
                        @Override
                        public void run(){
                                    int ch;
                                    try{
                                              //判断没有被中断的时候
                                              while (!interrupted()
                                              //不是结束标志
                                               && (ch = localInput.read()) != -1) {
                                                        //写字节到远程输入里面
                                                        remoteOutput.write(ch);
                                                        //刷新，发送
                                                        remoteOutput.flush();
                                               }
                                    }
```

```
                                catch (IOException e){
                                        e.printStackTrace();
                                }
                        }
                };
                //定义写线程的具体操作
                writer = new Thread(){
                        @Override
                        public void run(){
                                try{
                                        //把数据从输入流复制到输出流
                                        Util.copyStream(remoteInput,
                                                        localOutput);
                                }
                                catch (IOException e){
                                        e.printStackTrace();
                                        //发生异常的时候退出该操作
                                        System.exit(1);
                                }
                        }
                };
                //设置 writer 线程
                writer.setPriority(Thread.currentThread().getPriority() + 1);
                //启动 writer 线程
                writer.start();
                //设置 reader 为后台运行
                reader.setDaemon(true);
                //启动 reader 线程
                reader.start();
                try{
                        //使得 writer 线程完成 run() 方法后，再执行 join() 方法后的代码
                        writer.join();
                        //中断 reader 线程
                        reader.interrupt();
                }
                catch (InterruptedException e){
                }
        }
}
```

2.4 小结

本章主要讲解了 Android 的基本网络概念，包括 IP、TCP、UDP 等相关协议，特别阐述了 Android Socket 编程的原理和方法，并给出了相关的 Android Socket 实战案例。

第 3 章　Android 基本 Web 技术和编程实践

本章主要介绍 Android 的 Web 编程，包括 Android 中的 HTTP 编程，以及使用 Android 处理 JSON、SOAP、HTML 等。本章中每节在详尽而重点地介绍相关概念的基础上，均给出了实战案例，以便读者更好地理解相关概念。

3.1　HTTP 协议

HTTP（HyperText Transport Protocol）协议是互联网上应用最为广泛的一种网络协议标准。所有的 WWW 文件都必须遵守这个标准。本节简要介绍 HTTP 协议的历史、特点，重点介绍 HTTP 报文结构，最后给出 Android 中基于 HTTP 协议的文件上传方法。

3.1.1　HTTP 简介

HTTP 是一个适用于分布式超媒体信息系统的应用层协议。它于 1990 年被提出，现已得到广泛使用，并得到不断完善和扩展。HTTP 是万维网协会（World Wide Web Consortium）和 Internet 工作小组（Internet Engineering Task Force）合作的结果，他们最终发布了一系列的 RFC：RFC 1945 定义了 HTTP/1.0 版本；RFC 2616 定义了今天普遍使用的一个版本——HTTP 1.1。

HTTP 协议的主要特点如下：

- 支持 C/S（客户端 / 服务器）模式。
- 简单快速。客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有 GET、HEAD、POST。每种方法规定了客户与服务器联系的不同类型。因为 HTTP 协议比较简单，HTTP 服务器的程序规模较小，因而其通信速度很快。
- 灵活。HTTP 允许传输任意类型的数据对象。正在传输的类型由 Content-Type 加以标记。
- 无连接。无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。
- 无状态。HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

绝大多数的 Web 开发都是构建在 HTTP 协议之上的 Web 应用。要想了解 Web 开发，先要了解 HTTP 的 URL，其一般形式如下：

```
http:// host[":"port][abs_path]
```

http 表示要通过 HTTP 协议来定位网络资源的；host 表示合法的 Internet 主机域名或者 IP 地址；port 指定一个端口号，为空则使用默认端口 80；abs_path 指定请求资源的 URI（Uniform Resource Identifier，通用资源标志符，指 Web 上任意的可用资源）。

HTTP 报文是面向文本的，报文中的每一个字段都是一些 ASCII 码串，各个字段的长度是不确定的。

HTTP 有两类报文：请求报文和响应报文。

1. HTTP 请求报文

一个 HTTP 请求报文由请求行、请求报头、空行和请求数据 4 个部分组成，请求报文的一般格式如图 3-1 所示。

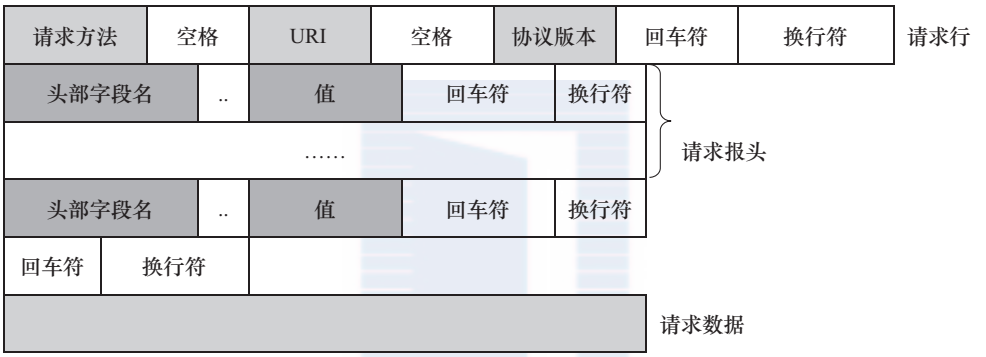


图 3-1 HTTP 请求报文的一般格式

（1）请求行

请求行由请求方法字段、URI 字段和 HTTP 协议版本字段组成，它们之间用空格分隔。格式如下：

```
Method Request-URI HTTP-Version CRLF
例如：GET /form.html HTTP/1.1 (CRLF)
```

其中 Method 表示请求方法；Request-URI 是一个统一资源标识符；HTTP-Version 表示请求的 HTTP 协议版本；CRLF 表示回车和换行（除了作为结尾的 CRLF 外，不允许出现单独的 CR 或 LF 字符）。

请求方法有多种，各种方法的解释如表 3-1 所示。

表 3-1 HTTP 请求报文中请求方法及含义

请 求 方 法	含 义
GET	请求获取 Request-URI 所标识的资源
POST	在 Request-URI 所标识的资源后附加新的数据
HEAD	请求获取由 Request-URI 所标识的资源的响应消息报头
PUT	请求服务器存储一个资源，并用 Request-URI 作为其标识

（续）

请 求 方 法	含　义
DELETE	请求服务器删除 Request-URI 所标识的资源
TRACE	请求服务器回送收到的请求信息，主要用于测试或诊断
CONNECT	保留将来使用
OPTIONS	请求查询服务器的性能，或者查询与资源相关的选项和需求

（2）请求报头

这部分内容后边会有详细介绍，这里就不赘述了。

（3）空行

最后一个请求头之后是一个空行，发送回车符和换行符，通知服务器以下不再有请求头。

（4）请求数据

请求数据不在 GET 方法中使用，而是在 POST 方法中使用。POST 方法适用于需要客户填写表单的场合。与请求数据相关的、最常使用的请求头是 Content-Type 和 Content-Length。

在接收和解释请求消息后，服务器会返回一个 HTTP 响应消息。

2. HTTP 响应报文

HTTP 响应报文也是由 4 个部分组成的，分别是：状态行、消息报头、空行、响应正文，如图 3-2 所示。

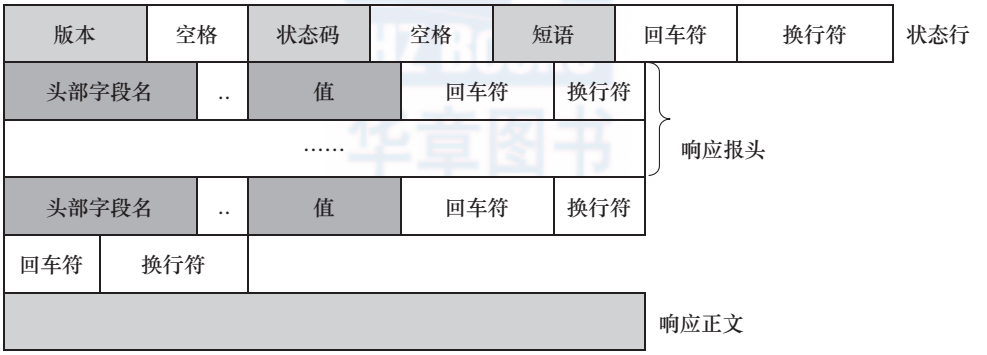


图 3-2　HTTP 响应报文

（1）状态行

状态行的格式如下：

```
HTTP-Version Status-Code Reason-Phrase CRLF
例如：HTTP/1.1 200 OK (CRLF)
```

其中，HTTP-Version 表示服务器 HTTP 协议的版本；Status-Code 表示服务器发回的响应状态代码；Reason-Phrase 表示状态代码的文本描述。

状态代码由 3 位数字组成，第一个数字定义了响应的类别，且有 5 种可能取值，如表 3-2 所示。

表 3-2 状态代码可能取值及其含义

状 态 代 码	含 义
1xx	指示信息，表示请求已接受，继续处理
2xx	成功，表示请求已被成功接收、理解、接受
3xx	重定向，要完成请求必须进行进一步的操作
4xx	客户端错误，请求有语法错误或请求无法实现
5xx	服务器端错误，服务器未能实现合法的请求

状态代码具体种类很多，可查阅相关文档，这里列出最常见的状态代码及状态描述，如表 3-3 所示。

表 3-3 常见状态代码及状态描述

状 态 代 码	状 态 描 述
200 OK	客户端请求成功
400 Bad Request	客户端请求有语法错误，不能被服务器所理解
401 Unauthorized	请求未经授权
403 Forbidden	服务器收到请求，但是拒绝提供服务
404 Not Found	请求资源不存在，比如输入了错误的 URL
500 Internal Server Error	服务器发生不可预期的错误
503 Server Unavailable	服务器当前不能处理客户端的请求，一段时间后可能恢复正常

（2）响应报头

这部分内容后边会有详细介绍，这里就不赘述了。

（3）空行

这行是空的。

（4）响应正文

响应正文就是服务器返回的资源的内容。

3. HTTP 消息报头

如前所述，HTTP 消息由客户端到服务器的请求和服务器到客户端的响应组成。请求消息和响应消息都是由开始行（对于请求消息，开始行就是请求行；对于响应消息，开始行就是状态行）、消息报头（可选）、空行（只有 CRLF 的行）、消息正文（可选）4 部分组成。

其中消息报头包括普通报头、请求报头、响应报头、实体报头。消息报头由“头部字段名 / 值”对组成，每行一对，头部字段名和值用英文冒号“:”分隔，即形式如下：

```
头部字段名 +":"+ 空格 + 值
```

其中消息报头头部字段名是大小写无关的。报头描述了客户端或者服务器的属性、被

传输的资源以及应该实现的连接。

4 种不同类型的消息报头分述如下：

- 普通报头：即可用于请求，也可用于响应，不用于被传输的实体，只用于传输消息，是作为一个整体而不是特定资源与事务相关联。比如，Date 普通报头表示消息产生的日期和时间；Connection 普通报头允许发送指定连接的选项，例如指定连接是连续，或者指定 close 选项，通知服务器，在响应完成后，关闭连接。
- 请求报头：允许客户端传递关于自身的信息和希望的响应形式。请求报头通知服务器有关客户端请求的信息，典型的请求报头有如下几种。
  - User-Agent：包含产生请求的操作系统、浏览器类型等信息。
  - Accept：客户端可识别的内容类型列表，用于指定客户端接受哪些类型的信息。
  - Host：请求的主机名，允许多个域名同处一个 IP 地址，即虚拟主机。
- 响应报头：服务器用于传递自身信息的响应。典型的响应报头有如下两种。
  - Location：用于重定向接收者到一个新的位置。Location 响应报头常用在更换域名的时候。
  - Server：包含了服务器用来处理请求的系统信息，与 User-Agent 请求报头是相对应的。
- 实体报头：定义被传送资源的信息。既可用于请求，也可用于响应。请求和响应消息都可以传送一个实体。典型的实体报头如下。
  - Content-Encoding：被用作媒体类型的修饰符，它的值指示了已经被应用到实体正文的附加内容的编码。因而要获得 Content-Type 报头中所引用的媒体类型，必须采用相应的解码机制。
  - Content-Language：描述了资源所用的自然语言。没有设置该选项则认为实体内容将提供给所有的语言阅读。
  - Content-Length：用于指明实体正文的长度，以字节方式存储的十进制数字来表示。
  - Last-Modified：用于指示资源的最后修改日期和时间。

注意　这里只是列出了各种消息报头的典型形式，更多信息大家可以查阅 RFC 文档。官方的文档地址是 http://www.ietf.org/rfc.html，用户可以在主页通过 RFC 文档号或者协议名找到自己想要的 RFC 文档。如可以输入 2616 或者 Hypertext Transfer Protocol--HTTP/1.1 找到 HTTP 1.1 的 RFC 文档。

3.1.2　实战案例：基于 HTTP 协议的文件上传

在 Android 中很多时候需要上传文件，这里介绍一种利用 HTTP 协议的文件上传方法。HTTP 协议文件上传的功能是通过 RFC1867 规范来描述的，Chrome、IE、Mozila 和 Opera 等浏览器都支持此功能。本案例从搭建服务器开始，分析了文件上传的内容及上传的过程，

最后给出 Android 中实现文件上传的核心代码。具体步骤如下。

步骤 1 搭建 PHP 环境，提供接收上传文件的服务器。

本案例在 PHP 环境中完成，为此需要搭建 PHP 环境。为了快速配置 PHP 环境，这里选用的 PHP 套件包为 PHPnow（流行的 PHP 套件有多种，读者可以任意选择）。

首先从 http://www.phpnow.org/download.html 下载最新版 PHPnow。当前的最新版本为 1.5.6，其下载文件名为 PHPnow-1.5.6.zip，该包包含的文件如图 3-3 所示。双击 Setup.cmd 安装。

安装完成之后其目录下所含的文件如图 3-4 所示。后面编写的 PHP 代码要放到 htdocs 目录下面。

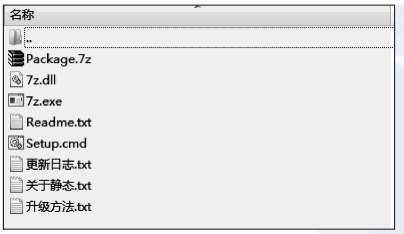


图 3-3 PHPnow 安装文件

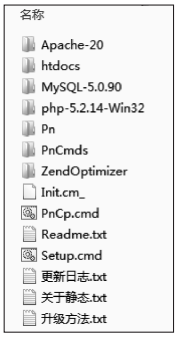


图 3-4 PHPnow 安装目录下文件列表

接着编写 PHP 上传文件的页面。在 HTML 网页中，写一个如下的 form，<form> 标签用于为用户输入创建 HTML 表单。

```
<html>
<body>
      <form action="upload_file.php" method="post" enctype="multipart/form-data">
      <label for="file">文件名 :</label>
      <input type="file" name="file" id="file" />
      <br/>
      <input type="submit" name="submit" value=" 提交 " />
      </form>
</body>
</html>
```

代码中 <input type="file"/> 标签在被浏览器解析后会产生一个文本框和一个“浏览”按钮。单击“浏览”按钮会出现系统的文件选择框，效果如图 3-5 所示。

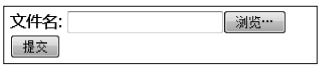


图 3-5 标签效果

HTML 中 <form> 标签的属性、取值及其含义如表 3-4 所示。

表 3-4　HTML 中 <form> 标签的属性

属　性	值	描　述
action	URL	规定当提交表单时，向何处发送表单数据
accept	MIME_type	规定通过文件上传来提交的文件的类型
enctype	MIME_type	规定表单数据在发送到服务器之前应该如何编码
method	get、post	规定如何发送表单数据
name	name	规定表单的名称
target	_blank、_parent、_self、_top	规定在何处打开 action URL

代码中 action="upload_file.php" 表示文件提交的时候会执行 upload_file.php 中的代码。upload_file.php 主要实现了对上传文件进行检查、复制等处理。其代码如下：

```
<?php
// 文件大小小于 20000 个字节
if ($_FILES["file"]["size"] < 20000) {
      // 文件上传相关的错误代码大于 0
      if ($_FILES["file"]["error"] > 0) {
                // 输出文件上传的错误代码信息
                echo "Return Code: " . $_FILES["file"]["error"] . "<br />";
      }
      else{
                // 输出文件名
                echo "Upload: " . $_FILES["file"]["name"] . "<br />";
                // 输出文件类型
                echo "Type: " . $_FILES["file"]["type"] . "<br />";
                // 输出文件大小，单位为 Kb
                echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";
                // 输出文件缓存的文件信息
                echo "Temp file: " . $_FILES["file"]["tmp_name"] . "<br />";
                // 判断文件是否存在
                if (file_exists("upload/" . $_FILES["file"]["name"])){
                        // 输出提示文件已经存在
                        echo $_FILES["file"]["name"] . " already exists. ";
                }
                else{
                        // 将上传的文件移动到新位置
                        move_uploaded_file($_FILES["file"]["tmp_name"],
                                "upload/" . $_FILES["file"]["name"]);
                        // 输出提示文件存储的位置
                        echo "Stored in: " . "upload/" . $_FILES["file"]["name"];
                }
      }
}
else{
      // 文件大于 20000 个字节，提示不合法
      echo "Invalid file";
}
?>
```

为了测试效果，这里建立一个文本文件，文件名为 test.txt，文件内容为“我是上传测试正文”。单击“浏览”按钮选中该文件，然后单击“提交”按钮。成功之后会显示如下信息：

```
Upload: test.txt
Type: text/plain
Size: 0.0263671875 Kb
Temp file: C:\Windows\Temp\php6E50.tmp
Stored in: upload/test.txt
```

注意 根据 RFC1867 规范，enctype="multipart/form-data",method=post,type="file"。这 3 个属性是必需的。multipart/form-data 用于设置上传文件的编码类型，确保匿名上传文件的正确编码。具体的解释请参阅 RFC1867 规范。

步骤 2 分析 HTTP 上传过程。

这里使用 HTTP 协议的分析工具 fiddler2 来分析文件上传的过程，其下载地址为 http://www.fiddler2.com/fiddler2/。安装后运行界面如图 3-6 所示。

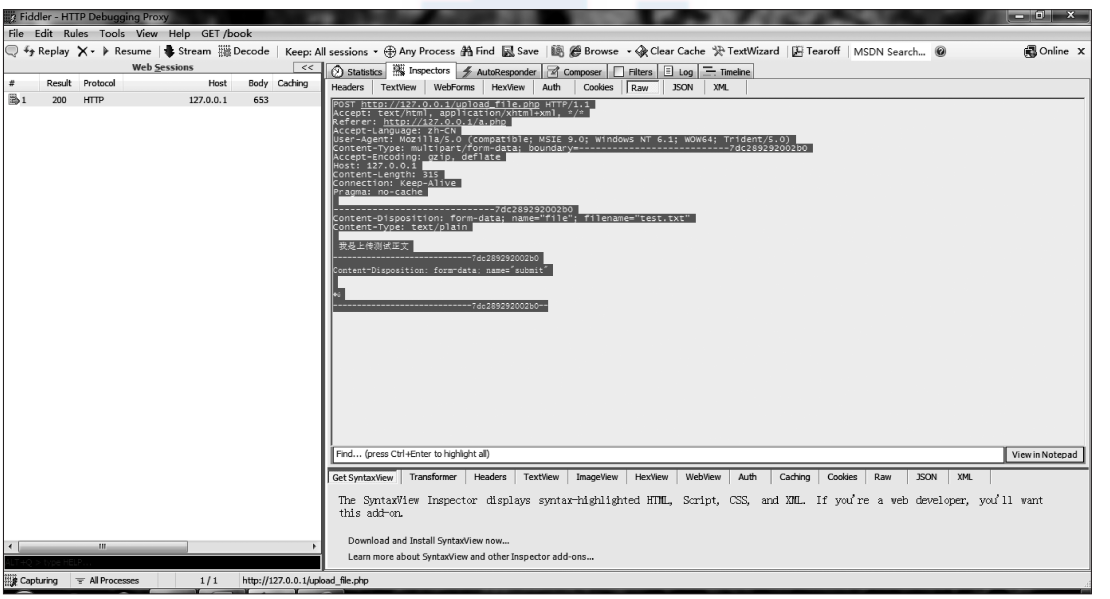


图 3-6 fiddler2 运行界面

在 fiddler2 左侧选中上传文件的一行，右侧单击 Inspectors->Raw 可以看到如下内容：

```
POST http://127.0.0.1/upload_file.php HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Referer: http://127.0.0.1/a.php
Accept-Language: zh-CN
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64;
```

```
Trident/5.0)
    Content-Type: multipart/form-data; boundary=----------------------------
7dc289292002b0
    Accept-Encoding: gzip, deflate
    Host: 127.0.0.1
    Content-Length: 315
    Connection: Keep-Alive
    Pragma: no-cache

    ------------------------------7dc289292002b0
    Content-Disposition: form-data; name="file"; filename="test.txt"
    Content-Type: text/plain

    我是上传测试正文
    ------------------------------7dc289292002b0
    Content-Disposition: form-data; name="submit"


    ------------------------------7dc289292002b0--
```

对照先前对 HTTP 协议的介绍，分析一下上文传输的原始内容。RFC1867 对 HTTP 头做了适当的变更，将 content-type 头由以前的：

```
    content-type: application/x-www-form-urlencoded
```

变为：

```
    content-type: multipart/form-data; +空格+boundary=----------------------------
7dc289292002b0
```

即增加了 boundary，其实就是分隔线。RFC1867 利用 boundary 分隔 HTTP 实体数据。boundary 中数字字符区是随机生成的。因为 RFC1867 增加了文件上传的功能，而上传文件内容自然也会被加入到 HTTP 的实体中。因为既有 HTTP 一般的参数实体，又有上传文件的实体，所以用 boundary 把两种实体进行了分隔。HTTP 的实体内容如下：

```
    ------------------------------7dc289292002b0
    Content-Disposition: form-data; name="file"; filename="test.txt"
    Content-Type: text/plain

    我是上传测试正文
    ------------------------------7dc289292002b0
    Content-Disposition: form-data; name="submit"


    ------------------------------7dc289292002b0--
```

根据 RFC1867 协议，在 HTTP 实体对每个上传的文件必须有说明头，如：

```
Content-Disposition: form-data; name="file"; filename="test.txt"
Content-Type: text/plain
```

其中 Content-Disposition 指明内容类型是 form-data ； name="file" 指明页面上 <input> 标签的名字是 file；filename="test.txt" 指明上传文件在客户端上的路径。其后为空行，文件头说明完毕后，要加一空行，以表示后面的数据是文件的内容。最后是文件内容。

步骤 3 编写 Android 代码实现文件上传。

下面是 Android 中以 HTTP 方式上传文件的核心代码。

```
    public static String post(String actionUrl, String FileName)throws
IOException {
          //产生随机分隔内容
          String BOUNDARY = java.util.UUID.randomUUID().toString();
          String PREFIX = "--", LINEND = "\r\n";
          String MULTIPART_FROM_DATA = "multipart/form-data";
          String CHARSET = "UTF-8";
          //定义 URL 实例
          URL uri = new URL(actionUrl);
          //定义 HttpURLConnection 实例，打开连接
          HttpURLConnection conn = (HttpURLConnection)uri.openConnection();
          //设置从主机读取数据超时（单位：毫秒）
          conn.setReadTimeout(5 * 1000);
          //设置允许输入
          conn.setDoInput(true);
          //设置允许输出
          conn.setDoOutput(true);
          //设置不允许使用缓存
          conn.setUseCaches(false);
          //设置为 POST 发送方法
          conn.setRequestMethod("POST");
          //设置维持长连接
          conn.setRequestProperty("connection", "keep-alive");
          //设置文件字符集为 UTF-8
          conn.setRequestProperty("Charsert", "UTF-8");
          //设置文件类型
          conn.setRequestProperty("Content-Type", MULTIPART_FROM_DATA
                    + ";boundary=" + BOUNDARY);
          //创建一个新的数据输出流，将数据写入指定基础输出流
          DataOutputStream outStream = new DataOutputStream(conn.getOutputStream());
          //发送文件数据
          if (FileName != "") {
                    //定义 StringBuilder 对象，构建发送字符串数据
                    StringBuilder sb1 = new StringBuilder();
                    sb1.append(PREFIX);
                    sb1.append(BOUNDARY);
                    sb1.append(LINEND);
                    sb1.append("Content-Disposition: form-data;
```

```
                name=\"file1\"; filename=\"\"+ FileName + \"\""+ LINEND);
                sb1.append("Content-Type: application/octet-stream; charset="+
                        CHARSET + LINEND);
                sb1.append(LINEND);
                //写入输出流中
                outStream.write(sb1.toString().getBytes());
                //将文件读到输入流中
                InputStream is = new FileInputStream(FileName);
                byte[] buffer = new byte[1024];
                int len = 0;
                //写入输出流中
                while ((len = is.read(buffer)) != -1) {
                        outStream.write(buffer, 0, len);
                }
                //关闭输入流
                is.close();
                //添加换行标志
                outStream.write(LINEND.getBytes());
        }
        //请求结束标志
        byte[] end_data = (PREFIX + BOUNDARY + PREFIX + LINEND).getBytes();
        outStream.write(end_data);
        //刷新，发送数据
        outStream.flush();
        //得到响应码
        int res = conn.getResponseCode();
        InputStream in = null;
        //上传成功则会返回响应码 200
        if (res == 200) {
                //读取数据
                in = conn.getInputStream();
                int ch;
                //定义 StringBuilder 字符串
                StringBuilder sb2 = new StringBuilder();
                //保存数据
                while ((ch = in.read()) != -1) {
                        sb2.append((char) ch);
                }
        }
        //如果数据不为空，则以字符串方式返回数据；否则返回 null
        return in == null ? null : in.toString();
}
```

3.2　Android 中的 HTTP 编程

3.2.1　HttpClient 和 URLConnection

HTTP 协议是现在 Internet 上使用得最多、也是最重要的协议之一，越来越多的

Android 应用程序需要直接通过 HTTP 协议来访问网络资源。虽然在 Android 的 java.net 包中已经提供了访问 HTTP 协议的基本功能，但是对于大部分应用程序来说，Android 原生提供的功能还不够丰富和灵活。HttpClient 是 Apache Jakarta Common 下的子项目，用来提供高效的、最新的、功能丰富的支持 HTTP 协议的客户端编程工具包，并且支持 HTTP 协议最新的版本和建议。

一般情况下我们使用浏览器来访问一个 Web 服务器，用来浏览页面查看信息或者提交一些数据等。所访问的这些页面有的仅仅是一些普通的页面，有的需要用户登录后方可使用，或者需要认证以及通过加密方式传输，如 HTTPS。目前我们使用的浏览器处理这些情况都不会构成问题。不过用户可能在某些时候想通过自己的程序来使用别人所提供的服务页面，比如从网页中“抓取”一些数据；利用某些站点提供的页面来完成某种功能等，例如通过已有网站提供的服务去查看某个手机号码的归属地。考虑到一些服务授权的问题，很多公司提供的页面往往并不是可以通过一个简单的 URL 就可以访问的，而必须经过注册然后登录后方可使用提供服务的页面，这个时候就涉及 COOKIE 的处理问题。这些问题有了 HttpClient 就很容易解决了！ HttpClient 就是专门设计用来简化 HTTP 客户端与服务器间各种通信编程的。通过它可以让原来很复杂的事情现在轻松解决。

URL（Uniform Resource Locator）代表统一资源定位符，Internet 上的每个资源都具有一个唯一的名称标识，通常称为 URL 地址，这种地址可以是本地磁盘，也可以是局域网上的某一台计算机，更多的是 Internet 上的站点，因此 URL 是指向互联网“资源”的指针。URLConnection 则代表了应用程序和 URL 之间的通信链接，通过 URLConnection 类的实例可以读取和写入此 URL 应用的资源。本节后面会给出使用 URLConnection 获取网络信息的实例。

3.2.2 Post 和 Get 在 HttpClient 的使用

HttpClient 提供的主要的功能如下：

- ❑ 实现了所有 HTTP 的方法（GET、POST、PUT、HEAD 等）
- ❑ 支持自动转向
- ❑ 支持 HTTPS 协议
- ❑ 支持代理服务器

HTTP 请求方法中最常用的是 GET 方法和 POST 方法。

1）GET 方法

GET 方法要求服务器将 URL 定位的资源放在响应报文的数据部分，回送给客户端。使用 GET 方法时，请求参数和对应的值附加在 URL 后面，利用一个问号（“?”）代表 URL 的结尾与请求参数的开始。

```
// 通过 GET 方法获取页面信息
// 参数为对应页面的 URL
```

```
public static InputStream getInputStreamFromUrl(String url) {
//定义输出流变量
InputStream content = null;
try {
      //取得默认的 HttpClient 实例
      HttpClient httpclient = new DefaultHttpClient();
      //连接到服务器
      HttpResponse response = httpclient.execute(
      //创建 HttpGet 实例
      new HttpGet(url));
      //获取数据内容
      content = response.getEntity().getContent();
      } catch (Exception e) {
      }
      //以 InputStream 形式返回页面信息
      return content;
}
```

上面的 GET 方法是以 InputStream 的形式返回页面的信息，很多情况下需要以 StringBuilder、String 等字符串的格式。下面的方法把 InputStream 格式转为 StringBuilder 和 String 格式。

```
//将 InputStream 格式转化为 StringBuilder 格式
private StringBuilder inputStreamToStringBuilder (InputStream is) {
      //定义空字符串
      String line = "";
      //定义 StringBuilder 的实例 total
      StringBuilder total = new StringBuilder();
      //定义 BufferedReader，载入 InputStreamReader
      BufferedReader rd = new BufferedReader(new InputStreamReader(is));
      //readLine 是一个阻塞的方法，当没有断开连接的时候就会一直等待，直到有数据返回
      //  返回 null 表示读到数据流最末尾
      while ((line = rd.readLine()) != null) {
                total.append(line);
      }
      //以 StringBuilder 形式返回数据内容
      return total;
}
//将 InputStream 格式数据流转换为 String 类型
private String inputStreamToString(InputStream is) {
      //定义空字符串
      String s = "";
      String line = "";
      //定义 BufferedReader，载入 InputStreamReader
      BufferedReader rd = new BufferedReader(new InputStreamReader(is));
      //读取到字符串中
      while ((line = rd.readLine()) != null) {
                s += line;
      }
```

```
        //以字符串方式返回信息
        return s;
    }
```

2）POST 方法

POST 方法要求被请求服务器接收附在请求后面的数据，常用于提交表单。当客户端给服务器提供信息较多时可以使用 POST 方法。POST 方法将请求参数封装在 HTTP 请求数据中，以名称值的形式出现，可以传输大量数据。

```
public void postData() {
        //创建一个新的 HttpClient Post 头
        HttpClient httpclient = new DefaultHttpClient();
        HttpPost httppost = new HttpPost("http://www.google.com");
        try {
                    //添加数据
                    List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(2);
                    nameValuePairs.add(new BasicNameValuePair("id", "12345"));
                    nameValuePairs.add(new BasicNameValuePair("stringdata", "myString"));
                    //使用 utf-8 格式对数据进行编码
                    httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs ,"UTF-8"));
                    //执行 HTTP Post 请求
                    HttpResponse response = httpclient.execute(httppost);
        } catch (ClientProtocolException e) {
        } catch (IOException e) {
        }
}
```

使用 HttpClient 需要以下 6 个步骤：

步骤 1 创建 HttpClient 的实例。

步骤 2 创建某种连接方法的实例，对于 get 方法是 GetMethod，而对于 post 方法是 PostMethod。

步骤 3 调用步骤 1 中创建好的实例的 execute 方法来执行步骤 2 中创建好的 method 实例。

步骤 4 读 response。

步骤 5 释放连接。

步骤 6 对得到的内容进行处理。

3.2.3 实战案例：使用 HttpClient 和 URLConnection 访问维基百科

这里介绍使用 URLConnection 对象和 HttpClient 组件访问维基百科，本案例要在维基百科里面搜索有关 Android 的信息。先了解一下维基百科 API 的构成。

维基百科是一个内容开放、多语言的百科全书协作计划。其内容除了浏览之外，还给出了其自身的 API 访问接口。

注意　具体的 API 可以参考 http://zh.wikipedia.org/w/api.php。

案例中使用的 API 的格式如下：

```
http:// zh.wikipedia.org/w/api.php?action=opensearch&search=Android
```

其中 action=opensearch 表示使用 OpenSearch 协议来访问该 API。

注意　OpenSearch 是一套基于 XML 的开放网站搜索协议。OpenSearch 其实是一个简单的 XML 格式，用以分享搜索的结果，或定义该网站搜索的方法，支持 OpenSearch 的 OpenSearch search clients 即可使用。目前支持的浏览器有 Internet Explorer 7、Firefox 2.0、Chrome 等。

search=Android 表示搜索的关键词为 Android；前面的 & 为连接符号。

注意　URL 中的一些字符有特殊含义，基本编码规则如下：
空格换成加号（+）；正斜杠（/）分隔目录和子目录；问号（?）分隔 URL 和查询；百分号（%）指定特殊字符；# 号指定书签；& 号分隔参数，有时也作为连接符号。

维基百科的 API 访问接口中常用的参数及含义如表 3-5 所示。

表 3-5　参数及含义

参　数	含　义
search	搜索字符串
limit	返回结果的最大值，默认是 10，最大不超过 100
namespace	搜索的命名空间，最大值取 50，默认取 0
suggest	是否打开搜索建议
format	输出格式，可选 json、jsonfm、xml、xmlfm，默认为 json

（1）使用 URLConnection 访问维基百科

在项目布局文件里面定义一个 TextView 用来显示访问的数据内容，代码如下：

```
<RelativeLayout xmlns:android="http:// schemas.android.com/apk/res/android"
   xmlns:tools="http:// schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent" >
   <TextView
      android:id="@+id/showWiki"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:layout_alignParentLeft="true"
      android:layout_alignParentTop="true"
```

```
        android:layout_marginTop="14dp"
        android:text="TextView" />
</RelativeLayout>
```

使用 URLConnection 访问维基百科的主要代码如下：

```
// 给出访问的 URL
String wikiSearchURL =
        "http:// zh.wikipedia.org/w/api.php?action=opensearch&search=Android";
try {
        // 初始化 URL
        URL url = new URL(wikiSearchURL);
        // 创建 HttpURLConnection，并打开连接
        HttpURLConnection httpconn = (HttpURLConnection) url.openConnection();
        // 判断获取的应答码是否正常
        if (httpconn.getResponseCode() == HttpURLConnection.HTTP_OK) {
                // 给出连接成功的提示
                Toast.makeText(getApplicationContext(), " 连接维基百科成功 !",
                        Toast.LENGTH_SHORT).show();
                // 创建 InputStreamReader
                InputStreamReader isr =
                // 设置字符编码为 utf-8
                        new InputStreamReader(httpconn.getInputStream(), "utf-8");
                int i;
                String content = "";
                // 读取消息到 content 中
                while ((i = isr.read()) != -1) {
                        content = content + (char) i;
                }
                isr.close();
                // 将获取的内容显示到界面上
                showWiki.setText(content);
        }
        // 断开连接
        httpconn.disconnect();
} catch (Exception e) {
        // 提示连接失败
        // Toast.LENGTH_SHORT 设置显示较短的时间
        Toast.makeText(getApplicationContext(),
                " 连接维基百科失败 ", Toast.LENGTH_SHORT).show();
        e.printStackTrace();
}
```

（2）使用 HttpCient 访问维基百科

```
// 给出访问的 URL
String wikiSearchURL =
        "http:// zh.wikipedia.org/w/api.php?action=opensearch&search=Android";
```

```
// 初始化 DefaultHttpClient
DefaultHttpClient httpclient = new DefaultHttpClient();
// 创建 HttpGet
HttpGet httpget = new HttpGet(wikiSearchURL);
// 创建 ResponseHandler
ResponseHandler<string> responseHandler = new BasicResponseHandler();
try {
      // 获取返回的内容
      String content = httpclient.execute(httpget, responseHandler);
      // 提示连接成功
      Toast.makeText(getApplicationContext(), "连接维基百科成功!",
                Toast.LENGTH_SHORT).show();
      // 显示到应用界面上
      showWiki.setText(content);
} catch (Exception e) {
      // 提示连接失败
      Toast.makeText(getApplicationContext(), "连接维基百科失败",
                 Toast.LENGTH_SHORT).show();
      e.printStackTrace();
      }
      // 关闭连接
      httpclient.getConnectionManager().shutdown();
}
```

运行效果如图 3-7 所示。

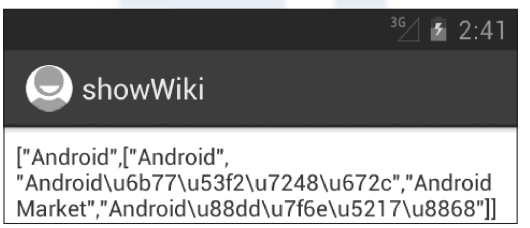


图 3-7　访问维基百科运行效果

图 3-7 中的 \u6b77、\u53f2、\u7248、\u672c、\u88dd、\u7f6e、\u5217、\u8868 为转义字符，使用 log 查看之后，可以得到如下的内容：

```
[
"Android",
[
"Android",
"Android 歷史版本",
"Android Market",
"Android 装置列表"
]
]
```

3.3 Android 处理 JSON

3.3.1 JSON 简介

JSON 指的是 JavaScript 对象表示法（JavaScript Object Notation），它是一种轻量级的文本数据交换格式，类似于 XML，但比 XML 更小、更快、更易解析。

JSON 是基于 JavaScript 的一个子集，它使用 JavaScript 语法来描述数据对象，但是 JSON 仍然独立于语言和平台。JSON 解析器和 JSON 库支持许多不同的编程语言。这些特性都使得 JSON 成为理想的数据交换语言，使其易于人们阅读和编写，同时也易于机器解析和生成。

JSON 的优点如下：

- 数据格式比较简单，易于读写，格式都是压缩的，占用带宽小。
- 易于解析语言，客户端 JavaScript 可以简单地通过 eval() 进行 JSON 数据的读取。
- 支持多种语言，包括 ActionScript、C、C#、ColdFusion、Java、JavaScript、Perl、PHP、Python、Ruby 等服务器端语言，便于服务器端的解析。
- 众多服务器端的对象、数组等能够直接生成 JSON 格式，便于客户端的访问提取。
- 因为 JSON 格式能够直接为服务器端代码使用，大大简化了服务器端和客户端的代码开发量，但是完成的任务不变，且易于维护。

JSON 用于描述数据结构，有以下两种形式。

（1）“名称 / 值”对的集合（A collection of name/value pairs）

“名称 / 值”对的集合形式又称为 JSON Object，其名称和值之间使用“：”隔开，一般的形式如下：

```
{name:value}
例如：{ "Width":"800" , "Height":"600" }
```

其中名称是字符串；值可以是字符串、数值、对象、布尔值、有序列表或者 null 值。字符串是以 "" 括起来的一串字符；数值是一系列 0 ～ 9 的数字组合，可以为负数或者小数，还可以用 e 或者 E 表示为指数形式；布尔值表示为 true 或者 false。

上述是以“{”开始，并以“}”结束的一系列非排序的“名称 / 值”对（每个“名称 / 值”对之间使用“，”分隔）。不同的语言中，这种“名称 / 值”可以理解为对象（object）、记录（record）、结构（struct）、字典（dictionary）、哈希表（hash table）、有键列表（keyed list）或者关联数组（associative array）等。

（2）值的有序列表（An ordered list of values）

值的有序列表形式又称为 JSON Array。在大部分语言中，值的有序列表被理解为数组（array），一个或者多个值用“，”分隔后，使用“[”和“]”括起来就形成了这样的列表，如下所示：

```
[collection, collection]
```

例如：

```
{
"employees": [
{ "Width":"800" , "Height":"600" },
{ "Width":"700" , "Height":"800" },
{ "Width":"900" , "Height":"900" }
]
}
```

3.3.2　JSON 数据解析

解析 JSON 数据时，首先需要明确待解析的是 JSON Object 还是 JSON Array，然后再解析。举例如下。

（1）解析 JSON Object 之一

下面是一个简单的 JSON Object，name 为名称，Lili 是 name 的值，将 name 和 Lili 用“：”隔开，其文本如下。

```
{"name":"Lili"}
```

JSONObject.getString（"String"）方法可以得到 JSON 对象中 String 名称对应的值。下面是对上面 JSON 对象的解析方法：

```
// 新建 JSONObject，jsonString 字符串中为上面的 JSON 对象的文本
JSONObject demoJson = new JSONObject(jsonString);
// 获取 name 名称对应的值
String s = demoJson.getString("name");
```

（2）解析 JSON Object 之二

下面是一个包含两个“名称 / 值”对的 JSON 对象，两个“名称 / 值”对分别是 "name1":"android" 和 "name2":"iphone"，中间使用“,”隔开，其文本如下：

```
{"name1":"android","name2":"iphone"}
```

上面 JSON 对象的解析方法如下：

```
// 新建 JSONObject 对象，将 jsonString 字符串转换为 JSONObject 对象
// jsonString 字符串为上面的文本
JSONObject demoJson = new JSONObject(jsonString);
// 获取名称为 name1 对应的值
```

```
String name1= demoJson.getString("name1");
//获取名称为 name2 对应的值
String name2 = demoJson.getString("name2");
```

（3）解析 JSON Array

下面是一个简单的 JSONArray，number 为数组名称，[1,2,3] 为数组的内容，其 JSON 文本表示如下：

```
{"number":[1,2,3]}
```

上面的 JSON Array 解析方法如下：

```
//新建 JSONObject 对象，将 jsonString 字符串转换为 JSONObject 对象
//jsonString 字符串为上面的文本
JSONObject demoJson = new JSONObject(jsonString);
//获取 number 对应的数组
JSONArray numberList = demoJson.getJSONArray("number");
//分别获取 numberList 中的每个值
for(int i=0; i<numberList.length(); i++){
        //因为数组中的类型为 int，所以为 getInt，其他 getString、getLong 具有类似的用法
        System.out.println(numberList.getInt(i));
}
```

（4）解析 JSON Object 和 JSON Array 混合对象

下面是一个 JSON Object 和 JSON Array 的混合文本，mobile 为 JSON Object 名称，其对应的值为 JSON Array，JSON Array 中包含的对象为 JSON Object，其文本表示如下：

```
{"mobile":[{"name":"android"},{"name":"iphone"}]}
```

上面文本的解析方法如下：

```
//新建 JSONObject 对象，将 jsonString 字符串转换为 JSONObject 对象
//jsonString 字符串为上面的文本
JSONObject demoJson = new JSONObject(jsonString);
//首先获取名为 mobile 的对象对应的值
//该值为 JSONArray，这里创建一个 JSONArray 对象
JSONArray numberList = demoJson.getJSONArray("mobile");
//依次取出 JSONArray 中的值
for(int i=0; i<numberList.length(); i++){
       //从第 i 个取出 JSONArray 中的值为 JSON Object“名称 / 值”对
       //通过 getString("name") 获取对应的值
       System.out.println(numberList.getJSONObject(i).getString("name"));
}
```

3.3.3　JSON 打包

要想在客户端通过JSON传送对象，需要在JSON把信息全部“打包”之后将JSONObject转换为String。这样JSON就会将“打包”的信息按照特定标准的格式进行“压缩”，之后在服务端进行解析，读取通过JSON传送来的信息。

Android提供的JSON解析类都在包org.json下，主要有以下几个。

- JSONObject：可以看作是一个JSON对象，这是系统中有关JSON定义的基本单元，即前面提到的“名称/值”对。
- JSONStringer：JSON文本构建类，这个类可以帮助快速和方便地创建JSON文本。其最大的优点在于可以减少由于格式的错误导致的程序异常，引用这个类可以自动严格按照JSON语法规则创建JSON文本。每个JSONStringer实体只能对应创建一个JSON文本。
- JSONArray：它代表一组有序的数值。将其转换为String输出(toString)所表现的形式是用方括号包裹，数值以逗号“,”分隔，即前面提到的值的有序列表。
- JSONTokener：JSON解析类。
- JSONException：JSON中涉及的异常。

下面看一个用JSONObject、JSONArray来构建JSON文本的例子，其需要构建的文本如下：

```
{
      "Strings" : { "Strings1" : "MyStrings", "Strings2" : "MyStrings" },
      "Number" : ["987654321", "123456789","456789123"],
      "String" : "good",
      "Int" : 100,
      "Boolean" : false
}
```

上面的JSON对象中包含了5个“名称/值”对，其中名称为Strings的对应的值本身也是一个包含2个“名称/值”对的JSON对象；名称为Number的对应的值为一个JSON Array；名称为String的对应的值为一个字符串；名称为Int的对应的值为一个整型；名称为Boolean的对应的值为布尔型。

构建上述文本的主要代码如下：

```
try {
      // 创建 JSONObject 对象
      JSONObject mJSONObject = new JSONObject();
      // 为 Strings 创建 JSONObject 对象
      JSONObject Strings = new JSONObject();
      // 为 Strings JSONObject 对象添加第一个“名称 / 值”对
      Strings.put("Strings1", " MyStrings");
      // 为 Strings JSONObject 对象添加第二个“名称 / 值”对
```

```
            Strings.put("Strings2", " MyStrings");
            //将 Strings 添加到 mJSONObject 中
            mJSONObject.put("Strings", Strings);
            //为 Number 创建 JSONArray 对象
            JSONArray Number = new JSONArray();
            //将有序列表添加到 Number 中
            Number.put("987654321").put("123456789").put("456789123");
            //将 Number 添加到 mJSONObject 中
            mJSONObject.put("Number", Number);
            //将 Int“名称 / 值”对添加到 mJSONObject 中
            mJSONObject.put("Int", 100);
            //将 Boolean“名称 / 值”对添加到 mJSONObject 中
            mJSONObject.put("Boolean", false);
        } catch (JSONException ex) {
            //进行异常处理
            throw new RuntimeException(ex);
        }
```

3.3.4 实战案例：JSON 解析 wikipedia 内容

下面的实例将解析以 JSON 表现的 wikipedia 的内容。wikipedia 的 API 网址是 http://en.wikipedia.org/w/api.php。如果需要查询 Android 的相关内容，并用 JSON 格式显示出来，可以使用如下的 API：

```
http://en.wikipedia.org/w/api.php?action=query&prop=revisions&rvprop=content&titles=Android&format=json
```

其显示的内容为：

```
{
    "query": {
       "pages": {
          "8325880": {
             "pageid": 8325880,
             "ns": 0,
             "title": "Android",
             "revisions": [
                {
                "*": "{{wiktionary|Android|android}}\n'''Android''' commonly-refers to:\n* [[Android (robot)]], designed to resemble a human\n* [[Android (operating system)]], for mobile devices, produced by Google\n'''Android''' may also refer to:\n* [[Android (board game)|''Android'' (board game)]], published by Fantasy Flight Games\n* [[Android (drug)]], brand name for the anabolic steroid methyltestosterone\n* [[Android (film)|''Android'' (film)]], directed by Aaron Lipstadt\n* [[Android (song)|\"Android\" (song)]], by The Prodigy\n\n==See also==\n* [[The Androids]], Australian rock band\n* [[Droid (disambiguation)]]\n*
```

```
{{Lookfrom}}\n* {{Intitle}}\n\n{{disambiguation}}\n\n[[cs:Android (rozcestník)]]\
n[[de:Android]]\n[[fa:يدانـدرو (يـيزدا ابهـام)]]\n[[ko:안드로이드]]\n[[id:Android]]\
n[[he:אנדרואיד]]\n[[hu:Android (egyértelmű sítő lap)]]\n[[nl:Android]]\n[[ru:Андроїд
(значения)]]\n[[sk:Android (rozlišovacia stránka)]]\n[[sl:Android (razločitev)]]\
n[[th:แอนดรอยด์ (แก้ความกำกวม)]]\n[[tr:Android]]\n[[uk:Андроїд (значення)]]\n[[zh-
yue:Android (搞清楚)]]"
                    }
                ]
            }
        }
    }
```

此处以考察 Android Dome 里面的 simplewiktionary 为例，其核心代码如下：

```
    // 从 wiki 服务器获取相关页面信息
    protected static synchronized String getUrlContent(String url) throws
ApiException {
    // 检测是否设置了 UserAgent 属性
    /*User Agent 中文名为用户代理，简称 UA，
    UserAgent 是一个特殊字符串头，使得服务器能够识别客户使用的操作系统及版本、
    CPU 类型、浏览器及版本、浏览器渲染引擎、浏览器语言、浏览器插件等 */
        if (sUserAgent == null) {
                // 抛出自定义异常，提示没有设置 UserAgent 属性
                throw new ApiException("User-Agent string must be prepared");
        }
        // 新建 HttpClient 对象
        HttpClient client = new DefaultHttpClient();
        // 新建 HttpGet 对象
        HttpGet request = new HttpGet(url);
        // 添加用户代理
        request.setHeader("User-Agent", sUserAgent);
        try {
                // 打开连接，获取返回信息
                HttpResponse response = client.execute(request);
                // 检测服务器是否可用
                StatusLine status = response.getStatusLine();
                // 检测服务器返回的状态码
                if (status.getStatusCode() != HTTP_STATUS_OK) {
                        // 如果服务状态码返回不正常，这里就抛出自定义异常
                        throw new ApiException("Invalid response from server: " +
                                        status.toString());
                }
                // 取回服务器返回的内容
                HttpEntity entity = response.getEntity();
                // 将返回的内容保存到 InputStream 对象中
                InputStream inputStream = entity.getContent();
                // 创建 ByteArrayOutputStream 对象
                // ByteArrayOutputStream 捕获内存缓存中的数据
```

```
                //然后可以将其转换成字节数组或者字符型
                ByteArrayOutputStream content = new ByteArrayOutputStream();
                int readBytes = 0;
                //将缓存中的数据保存到 ByteArrayOutputStream 对象中
                while ((readBytes = inputStream.read(sBuffer)) != -1) {
                        content.write(sBuffer, 0, readBytes);
                }
                //将内容转化为字符串
                return new String(content.toByteArray());
        } catch (IOException e) {
                //检测到 IO 异常的时候，抛出自定义的异常
                throw new ApiException("API 返回错误 ", e);
        }
}
//错误检测，自定义异常
public static class ApiException extends Exception {
        //含有 2 个参数的构造函数
        public ApiException(String detailMessage, Throwable throwable) {
                super(detailMessage, throwable);
        }
        //含有 1 个参数的构造函数
        public ApiException(String detailMessage) {
                super(detailMessage);
        }
}
```

下面的代码调用了上文定义 getUrlContent 方法，解析本节开头的那段 JSON 信息。

```
/* @param title Wiktionary 页面返回的标题
 @param expandTemplates 设置模板是否可用
 @返回解析之后的页面内容 */
public static String getPageContent(String title, boolean expandTemplates)
                throws ApiException, ParseException {
        //如有需要对标题和模板进行编码
        String encodedTitle = Uri.encode(title);
        String expandClause = expandTemplates ? WIKTIONARY_EXPAND_TEMPLATES : "";
        //查询 API 获取内容
        String content = getUrlContent(String.format(WIKTIONARY_PAGE,
                encodedTitle, expandClause));
        try {
                //解析返回的 JSON 内容
                JSONObject response = new JSONObject(content);
                //获取 query 对应的 JSON 对象
                JSONObject query = response.getJSONObject("query");
                //获取 pages 对应的 JSON 对象
                JSONObject pages = query.getJSONObject("pages");
                //获取 page 对应的 JSON 对象
                JSONObject page = pages.getJSONObject((String) pages.keys().next());
```

```
                // 获取 revisions 对应的 JSON 数组
                JSONArray revisions = page.getJSONArray("revisions");
                // 获取版本 JSON 对象，为 revisions 对应的 JSON 数组包含的第一个 JSON 对象
                JSONObject revision = revisions.getJSONObject(0);
                // 获取 revision JSON 对象的值
                return revision.getString("*");
        } catch (JSONException e) {
        // 错误处理
        throw new ParseException("API 返回错误 ", e);
        }
}
```

3.4　Android 处理 SOAP

3.4.1　SOAP 简介

简单对象访问协议（Simple Object Access Protocol，SOAP）是一种标准化的通信规范，主要用于 Web 服务（Web service）。SOAP 的出现可以使网页服务器（Web Server）从 XML 数据库中提取数据时，无需花时间去格式化页面，并能够让不同应用程序之间通过 HTTP 协议，以 XML 格式互相交换彼此的数据，使这个交换过程与编程语言、平台和硬件无关。此标准由 IBM、Microsoft、UserLand 和 DevelopMentor 在 1998 年共同提出，并得到 IBM、Lotus（莲花）、Compaq（康柏）等公司的支持，于 2000 年提交给万维网联盟（World Wide Web Consortium，W3C）。目前 SOAP 1.1 版是业界共同的标准。

SOAP 基于 XML 标准，用于在分布式环境中发送消息，并执行远程过程调用。使用 SOAP，不用考虑任何特定的传输协议（尽管通常选用 HTTP 协议），就能使数据序列化。

SOAP 的优点如下：

- SOAP 是可扩展的。SOAP 无需中断已有的应用程序，SOAP 客户端、服务器和协议自身都能发展。而且 SOAP 能极好地支持中间介质和层次化的体系结构。
- SOAP 是简单的。客户端发送一个请求，调用相应的对象，然后服务器返回结果。这些消息是 XML 格式的，并且封装成符合 HTTP 协议的消息。因此，它符合任何路由器、防火墙或代理服务器的要求。
- SOAP 是完全和厂商无关的。SOAP 可以相对于平台、操作系统、目标模型和编程语言独立实现。另外，传输和语言绑定以及数据编码的参数选择都是由具体的实现决定的。
- SOAP 与编程语言无关。SOAP 可以使用任何语言来完成，只要客户端发送正确 SOAP 请求（也就是说，传递一个合适的参数给一个实际的远端服务器）。SOAP 没有对象模型，应用程序可以捆绑在任何对象模型中。

3.4.2 SOAP 消息

1. SOAP 消息简介

SOAP 使用 Internet 应用层协议作为其传输协议。SMTP 以及 HTTP 协议都可以用来传输 SOAP 消息，SOAP 亦可以通过 HTTPS 传输。一条 SOAP 消息就是一个普通的 XML 文档，包含下列元素：

- 必需的 Envelope 元素，可把此 XML 文档标识为一条 SOAP 消息。
- 可选的 Header 元素，包含头部信息。
- 必需的 Body 元素，包含所有的调用和响应信息。
- 可选的 Fault 元素，提供有关在处理此消息时发生错误的信息。

SOAP 消息的重要的语法规则如下：

- SOAP 消息必须使用 XML 来编码。
- SOAP 消息必须使用 SOAP Envelope 命名空间。
- SOAP 消息必须使用 SOAP Encoding 命名空间。
- SOAP 消息不能包含 DTD 引用。
- SOAP 消息不能包含 XML 处理指令。

SOAP 消息格式如图 3-8 所示。

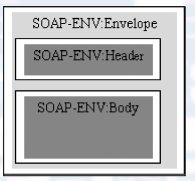


图 3-8 SOAP 消息格式

2. SOAP 消息实例

请求时候发送的消息内容如下：

```
<soapenv:Envelope
      xmlns:soapenv="http:// schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsd="http:// www.w3.org/2001/XMLSchema"
      xmlns:xsi="http:// www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
                <req:echo xmlns:req="http:// localhost:8080/axis2/services/MySer-vice/">
                <req:category>classifieds</req:category>
                </req:echo>
      </soapenv:Body>
</soapenv:Envelope>
```

响应时候发送的消息内容如下：

```
<soapenv:Envelope
      xmlns:soapenv="http:// schemas.xmlsoap.org/soap/envelope/"
      xmlns:wsa="http:// schemas.xmlsoap.org/ws/2004/08/addressing">
<soapenv:Header>
      <wsa:ReplyTo><wsa:Address>http:// schemas.xmlsoap.org/ws/2004/08/
                addressing/role/anonymous</wsa:Address>
      </wsa:ReplyTo>
      <wsa:From><wsa:Address>http:// localhost:8080/axis2/
                services/MyService</wsa:Address>
      </wsa:From>
      <wsa:MessageID>ECE5B3F187F29D28BC11433905662036</wsa:MessageID>
</soapenv:Header>
<soapenv:Body>
      <req:echo xmlns:req="http:// localhost:8080/axis2/services/MyService/">
      <req:category>classifieds</req:category>
      </req:echo>
</soapenv:Body>
</soapenv:Envelope>
```

3. 调用 WebService

SOAP 调用 WebService 的具体步骤如下。

步骤 1　添加 ksoap2 包。WebService 是一种基于 SOAP 协议的远程调用标准，通过 WebService 可以将不同操作系统平台、不同语言、不同技术整合到一块。在 Android SDK 中并没有提供调用 WebService 的库，因此，需要使用第三方的 SDK 来调用 WebService。PC 版本的 WebService 客户端库非常丰富，例如 Axis2、CXF 等，但这些开发包对于 Android 系统来说过于庞大，也未必很容易移植到 Android 系统中。因此，这些开发包并不在我们考虑范围内。适合手机的 WebService 客户端的 SDK 有一些，比较常用的是 Ksoap2，可以从网址 http://code.google.com/p/ksoap2-android/ 下载，然后将下载的 ksoap2-android-assembly-2.4-jar-with-dependencies.jar 包复制到 Eclipse 工程的 lib 目录中，当然也可以放在其他的目录里。在 Eclipse 工程中引用这个 jar 包。

步骤 2　指定 WebService 的命名空间和调用的方法名，如：

```
SoapObject request =new SoapObject(http:// service,"getName");
```

SoapObject 类的第一个参数表示 WebService 的命名空间，可以从 WSDL 文档中找到 WebService 的命名空间；第二个参数表示要调用的 WebService 方法名。

步骤 3　设置调用方法的参数值，如果没有参数，可以省略。设置方法的参数值的代码如下：

```
Request.addProperty("param1","value");
Request.addProperty("param2","value");
```

要注意的是，addProperty 方法的第 1 个参数虽然表示调用方法的参数名，但该参数值并不一定与服务端的 WebService 类中的方法参数名一致，只要设置参数的顺序一致即可。

步骤 4 生成调用 WebService 方法的 SOAP 请求信息。该信息由 SoapSerializationEnvelope 对象描述，代码如下：

```
SoapSerializationEnvelope envelope=
      new SoapSerializationEnvelope(SoapEnvelope.VER11);
Envelope.bodyOut = request;
```

创建 SoapSerializationEnvelope 对象时需要通过 SoapSerializationEnvelope 类的构造方法设置 SOAP 协议的版本号。该版本号需要根据服务端 WebService 的版本号设置。在创建 SoapSerializationEnvelope 对象后，不要忘了设置 SOAPSoapSerializationEnvelope 类的 bodyOut 属性，该属性的值就是在步骤 2 创建的 SoapObject 对象。

步骤 5 创建 HttpTransportsSE 对象。通过 HttpTransportsSE 类的构造方法可以指定 WebService 的 WSDL 文档的 URL。

```
HttpTransportSE ht=new HttpTransportSE
      ("http:// fy.webxml.com.cn/webservices/EnglishChinese.asmx?wsdl");
```

步骤 6 使用 call 方法调用 WebService 方法，代码如下：

```
ht.call(null, envelope);
```

call 方法的第一个参数一般为 null，第 2 个参数就是在步骤 4 创建的 SoapSerialization-Envelope 对象。

步骤 7 使用 getResponse 方法获得 WebService 方法的返回结果，代码如下：

```
SoapObject soapObject =(SoapObject)envelope.getResponse();
```

步骤 8 解析返回的内容。

3.4.3 实战案例：SOAP 解析天气服务

以下为一个简单的实现天气查看功能的例子。在这个例子中，用户在文本框中输入城市名之后单击“查询”按钮，查询成功后，会在应用界面上显示所查询城市的天气信息。

其实现的具体过程为：从客户端获取用户输入的城市名称，将城市名称打包成符合 SOAP 协议的查询消息，把查询信息发送给提供 SOAP 天气服务的服务器；服务器内部进

行操作之后，返回给客户端查询城市的天气信息，该信息以 SOAP 格式返回，客户端对其进行解析之后显示给用户。

下面是案例的布局文件，给出了使用的控件：

```
<RelativeLayout xmlns:android="http:// schemas.android.com/apk/res/android"
      xmlns:tools="http:// schemas.android.com/tools"
      android:layout_width="match_parent"
      android:layout_height="match_parent" >
<!-- 显示控件，用于显示天气情况 -->
    <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:layout_centerVertical="true"
      android:padding="@dimen/padding_medium"
      tools:context=".AndroidSoapActivity" />
<!-- 按钮，用户提交城市名称时候单击该按钮 -->
    <Button
      android:id="@+id/ok"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_toRightOf="@+id/textView1"
      android:text="@string/search" />
<!-- 输入控件，用户输入城市名称 -->
    <EditText
      android:id="@+id/cityName"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_alignParentTop="true"
      android:text="@string/cityName" />
</RelativeLayout>
```

应用内部对查询处理的主要代码如下：

```
import java.io.UnsupportedEncodingException;
// 加入需要使用的 ksoap2 包中的类
import org.ksoap2.SoapEnvelope;
import org.ksoap2.serialization.SoapObject;
import org.ksoap2.serialization.SoapSerializationEnvelope;
import org.ksoap2.transport.HttpTransportSE;
import android.os.AsyncTask;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
```

```
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
// SOAP 方式查询天气情况
public class AndroidSoapActivity extends Activity {
      //指定命名空间
      private static final String NAMESPACE = "http:// WebXml.com.cn/";
      //给出接口地址
      private static String URL =
                "http:// www.webxml.com.cn/webservices/weatherwebservice.asmx";
      //设置方法名
      private static final String METHOD_NAME = "getWeatherbyCityName";
      //设置查询接口参数
      private static String SOAP_ACTION =
                "http:// WebXml.com.cn/getWeatherbyC-ityName";
      //定义字符串，保存天气信息
      private String weatherToday;
      //定义按钮
      private Button okButton;
      //定义 SoapObject 对象
      private SoapObject detail;
      //定义输入控件
      private EditText cityNameText;
      //定义显示控件，显示天气信息
      private TextView cityMsgView;
      @Override
      public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                //加载布局文件
                setContentView(R.layout.activity_android_soap);
                //获取控件
                cityNameText  =(EditText)findViewById(R.id.cityName);
                cityMsgView = (TextView)findViewById(R.id.textView1);
                okButton = (Button) findViewById(R.id.ok);
                //为按钮添加事件监听
                okButton.setOnClickListener(new Button.OnClickListener() {
                        public void onClick(View v) {
                                //执行获取天气信息的操作
                                new showWeatherAsyncTask().execute();
                        }
                });
      }
// 使用 AsyncTask 异步方式获取并显示天气信息
private class showWeatherAsyncTask extends AsyncTask<String, Integer, String> {
      @Override
      protected String doInBackground(String... Urls) {
                //获取并显示天气信息
                showWeather();
                return null;
      }
```

```
        protected void onPostExecute(String result) {
        }
};
// 获取并显示天气信息
private void showWeather() {
        // 获取需要查询的城市名称
        String city = cityNameText.getText().toString().trim();
        // 检测城市名称是否为空
        if(!city.isEmpty()){
                // 获取指定城市的天气信息
                getWeather(city);
        }
}
// 获取指定城市的天气信息，参数 cityName 为指定的城市名称
public void getWeather(String cityName) {
        try {
                // 新建 SoapObject 对象
                SoapObject rpc = new SoapObject(NAMESPACE, METHOD_NAME);
                // 给 SoapObject 对象添加属性
                rpc.addProperty("theCityName", cityName);
                // 创建 HttpTransportSE 对象，并指定 WebService 的 WSDL 文档的 URL
                HttpTransportSE  ht = new HttpTransportSE(URL);
                // 设置 debug 模式
                ht.debug = true;
                // 获得序列化的 envelope
                SoapSerializationEnvelope envelope =
                        new SoapSerializationEnvelope(SoapEnvelope.VER11);
                // 设置 bodyOut 属性的值为 SoapObject 对象 rpc
                envelope.bodyOut = rpc;
                // 指定 webservice 的类型为 dotNet
                envelope.dotNet = true;
                envelope.setOutputSoapObject(rpc);
                // 使用 call 方法调用 WebService 方法
                ht.call(SOAP_ACTION, envelope);
                // 获取返回结果
                SoapObject result = (SoapObject) envelope.bodyIn;
                // 使用 getResponse 方法获得 WebService 方法的返回结果
                detail = (SoapObject)result.getProperty("getWeatherbyCityNameResult");
                System.out.println("detail" + detail);
                // 解析返回的数据信息为 SoapObject 对象，对其进行解析
                parseWeather(detail);
                return;
        } catch (Exception e) {
                e.printStackTrace();
        }
}
// 解析 SoapObject 对象
private void parseWeather(SoapObject detail)throws UnsupportedEncodingException {
```

```
        //获取日期
        String date = detail.getProperty(6).toString();
        //获取天气信息
        weatherToday = "今天: " + date.split(" ")[0];
        weatherToday = weatherToday + " 天气: " + date.split(" ")[1];
        weatherToday = weatherToday + " 气温: " + detail.getProperty(5).toString() ;
        weatherToday = weatherToday + " 风力:" + detail.getProperty(7).toString()+ " ";
        System.out.println("weatherToday is " + weatherToday);
        //显示到 cityMsgView 控件上
        cityMsgView.setText(weatherToday);
}
//创建 Menu 菜单
@Override
public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_android_soap, menu);
        return true;
        }
}
```

运行效果如图 3-9 所示。

图 3-9 SOAP 解析天气服务

3.5　Android 对 HTML 的处理

3.5.1　解析 HTML

在 Android 应用程序开发过程中，经常需要解析 HTML 文档，特别是那类通过“爬网站”抓取数据的应用，比如天气预报。Java 常用的解析 HTML 文档的方法有以下几种：

- 使用正则表达式来抽取数据。
- 以纯字符串查找定位来实现。
- 使用 HTML Parser 解析器。
- 使用 Jsoup 解析器。

在 Android 平台上推荐使用 Jsoup 解析器来解析 HTML 文档。Jsoup 既可以通过一个 URL 网址，也可以通过存储 HTML 脚本的文件或者存储 HTML 脚本的字符串作为数据源，然后通过 DOM、CSS 选择器来查找、抽取数据。

使用 Jsoup 解析字符串形式的 HTML 文件的方法如下：

```
// 定义需要解析的 HTML 字符串
String html = "<html><head><title>First parse</title></head>"
      + "<body><p>Parsed HTML into a doc.</p></body></html>";
// 将字符串解析之后放到 Document 对象中
Document doc = Jsoup.parse(html);
}
```

下面是一个具体的解析列子，使用 Jsoup 从 HTML 文件中提取出超链接、超链接文本、页面描述等内容。

```
// 需要解析的 HTML 字符串
String html = "<p>An <a href='http://example.com/'><b>example</b></a> link.</p>";
// 保存到 Document 对象中
Document doc = Jsoup.parse(html);
// 得到第一个 a 标签的超链接
Element link = doc.select("a").first();
// 取出 HTML 字符串中的文本内容
// 这里 test 的值为 An example link
String text = doc.body().text();
// 获取属性为 href 的字符串
// 这里 linkHref 的值为 "http://example.com/"
String linkHref = link.attr("href");
// 获取 a 标签内部的纯文本
// linkText 为 "example"
String linkText = link.text();
// 获取整个 a 标签里面的字符串
// 这里 linkOuterH 的值为 <a href="http://example.com"><b>example</b></a>
String linkOuterH = link.outerHtml();
```

```
// 获取 a 标签内部（不包含 a 标签）的全部字符串
// 这里 linkInnerH 的值为 <b>example</b>
String linkInnerH = link.html();
```

Jsoup 还可以使用 Whitelist() 方法把不规范的 HTML 格式整理为规范格式，Whitelist 方法定义了哪些 HTML 的元素和属性可以保留，其他的全部会被删除掉。Whitelist.basic() 方法允许通过的文本节点为：a、b、blockquote、br、cite、code、dd、dl、dt、em、i、li、ol、p、pre、q、small、strike、strong、sub、sup、u、ul，以及相应的属性，不允许图片通过。

具体的使用方法如下：

```
String unsafe =
      "<p><a href='http:// example.com/' onclick='stealCookies()'>Link</a></p>";
// 调用 clean 方法整理不标准的代码
String safe = Jsoup.clean(unsafe, Whitelist.basic());
// safe 为 <p><a href="http:// example.com/" rel="nofollow">Link</a></p>
```

3.5.2 HTML 适配屏幕

Android 终端的物理尺寸、分辨率的类别众多，可能你为一种终端设计了一套 UI 符合要求，但是在另一类大小的物理终端上显示的就完全不是你想要的。如何将一个应用程序适配在不同的手机上，这就是本节要讲的 Android 适配屏幕问题。

这里面涉及几个概念，关于像素（pixel）、分辨率（Resolution，width×height）大家已经很清楚了，这里不再强调。要强调的是如下几个：

- 屏幕的尺寸（Screen size），指屏幕对角线长度（单位 inch，即英寸）。
- 屏幕密度（Screen density），指单位长度上的像素点数（dots per inches，dpi）。
- 独立像素密度或密度无关像素（Density-independent pixel，dp），标准是 160dpi，此时 1dp 对应 1 个 pixel。dp 转换为屏幕像素的计算公式是：px=dp×（dpi/160）。例如根据公式，可知 240dpi 的屏幕，1dp 对应 1.5 个 pixel。屏幕密度越大，1dp 对应的像素点数越多。

Android 屏幕有两种分类方式，具体如下。

（1）以总像素数分

每种屏幕都有其最小分辨率：xlarge 屏幕最小分辨率为 960×720dp；large 屏幕最小分辨率为 640×480dp；normal 屏幕最小分辨率为 470×320dp；small 屏幕最小分辨率为 426×320dp。这样通过计算就可以根据分辨率划分种类，如图 3-10 所示。

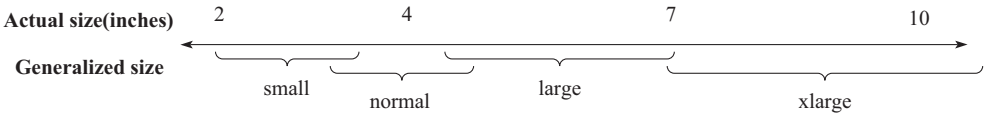


图 3-10 Android 屏幕支持的范围（以总像素数分）

（2）以屏幕密度分

Android 屏幕按密度分类，具体如图 3-11 所示。

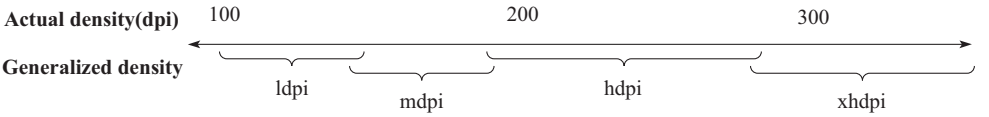


图 3-11　Android 屏幕支持的范围（以屏幕密度分）

屏幕的匹配要以上面两种方式为参考，开发过程中需要比较两图的对应关系。此外，有时还要考虑屏幕是水平（landscape）的还是竖直（portrait）的。

当新建一个 Android 工程后，系统会自动创建 3 个存放不同分辨率、不同密度 UI 的文件夹，如图 3-12 所示。

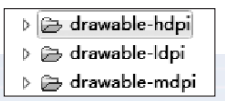


图 3-12　Android 工程自动创建的文件夹

3 个文件夹下的 UI 对应不同分辨率及密度的屏幕，对应关系如表 3-6 所示。

表 3-6　UI 与分辨率等的对应关系

文　件　夹	屏 幕 类 型	分　辨　率	密　　度	尺　　寸
drawable-hdpi	WVGA	480×800	240	大
drawable-ldpi	QVGA	240×320	120	小
drawable-mdpi	HVGA	320×480	160	中

这样，3 个文件夹中的 UI 就是针对特征屏幕分辨率及密度进行设计的，Android 终端会在打开应用的时候自动根据终端类型匹配与文件夹里提供的分辨率及密度相近的 UI。

3.5.3　JavaScript 混合编程

本节主要介绍在 Android 中显示 HTML 代码、添加 JavaScript 支持，以及通过 JavaScript 调用 Acitiviy。

1. 在 Android 中显示 HTML 代码

首先定义布局文件，添加 WebView 的组件。下面这段代码，展示了如何在布局文件中添加 WebView。

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http:// schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="vertical">
      <!-- 定义线性布局 -->
```

```
        <LinearLayout
                  <!-- 定义线性布局为垂直方向 -->
                  android:orientation="vertical"
                  android:layout_width="match_parent"
                  android:layout_height="wrap_content">
                  <!-- 添加第一个 WebView 控件 -->
                  <WebView android:id="@+id/wv1"
                  android:layout_height="wrap_content"
                  android:layout_width="match_parent" />
                  <!-- 添加第二个 WebView 控件 -->
                  <WebView android:id="@+id/wv2"
                  android:layout_height="wrap_content"
                  android:layout_width="match_parent"/>
                  <!-- 添加第三个 WebView 控件 -->
                  <WebView android:id="@+id/wv3"
                  android:layout_height="wrap_content"
                  android:layout_width="match_parent" />
        </LinearLayout>
</ScrollView>
```

下面介绍如何在代码中为 WebView 加载数据：

```
public class WebView1 extends Activity {
        @Override
        public void onCreate(Bundle icicle) {
                  super.onCreate(icicle);
                  setContentView(R.layout.webview_1);
                  //定义数据类型
                  final String mimeType = "text/html";
                  //定义编码类型
                  final String encoding = "utf-8";
                  WebView wv;
                  wv = (WebView) findViewById(R.id.wv1);
                  //使用 loadData 方法来加载 HTML 数据
                  //其第一个参数表示需要加载的数据
                  //mimeType 表示数据类型
                  //encoding 表示编码类型
                  wv.loadData("<a href='x'>Hello World! - 1</a>", mimeType, encoding);
                  wv = (WebView) findViewById(R.id.wv2);
                  //加载到第二个 WebView 控件
                  wv.loadData("<a href='x'>Hello World! - 2</a>", mimeType, encoding);
                  wv = (WebView) findViewById(R.id.wv3);
                  //加载到第三个 WebView 控件
                  wv.loadData("<a href='x'>Hello World! - 3</a>", mimeType, encoding);
        }
}
```

需要添加允许访问网络的权限，如下所示：

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

在 Android 的安全模型中，每一个应用都有自己的 Linux 用户和群组，在单独的进程和 VM 上运行，不能影响到其他应用。Android 同时也限定了系统资源的使用，像网络设备、SD 卡、录音设备等。如果应用希望去使用指定的系统资源，就必须去申请 Android 的权限，这就是 <uses-permission> 元素的作用。

一个权限通常有以下格式，用一个名字为 name 的字符串去表示希望使用的权限。

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

这里使用 android.permission.INTERNET 表示访问网络的权限。全部的权限列表可以参考网页 http://developer.android.com/reference/android/Manifest.permission.html 的内容。根据 Android 版本的不同，其列表各有不同。

2. WebView 中添加 JavaScript 支持

WebView 是 Android 中 View 的扩展，可以使用 WebView 作为客户端的一部分，能将 Web 页面作为显示界面的一部分。WebView 组件不能实现一个浏览器的完整功能，比如不能实现导航控制或者地址栏，其默认仅是展现一个 Web 页面。

在展示终端用户协议或者用户指南等内容的时候，使用 WebView 可以方便地在应用中提供一些需要及时更新的信息。此时在 Android 应用中，需要创建包含 WebView 的 Activity，然后利用它来展现网上的文档。

当展示的数据需要连接网络来获取的时候，也可以在 Android 应用中构建一个 WebView 实现。应用这种方法可以更方便地提供相关数据的 Web 页面，WebView 通过解析页面数据并将这些数据重新处理之后，再以更加适合用户使用的方式放到 Android 应用中。此外，也可以设计一个专供 Android 设备使用的 Web 页面，并在 Android 中通过一个 WebView 来加载这个页面。

要在应用中加入 WebView，只需要在布局文件中加入 WebView 控件即可。例如，下面是一个布局文件，在这个文件中，通过设置属性使得 WebView 控件充满屏幕。

```
<?xml version="1.0" encoding="utf-8"?>
<WebView  xmlns:android="http:// schemas.android.com/apk/res/android"
      android:id="@+id/webview"
      // 定义 WebView 在屏幕上可以使用的宽度，fill_parent 即填充整个屏幕
      android:layout_width="fill_parent"
      // 定义 WebView 在屏幕上可以使用的高度，fill_parent 即填充整个屏幕
      android:layout_height="fill_parent"
/>
```

通过使用 loadUrl()，在 WebView 中加载页面。

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.loadUrl("http:// www.example.com");
```

3. 在 WebView 中使用 JavaScript

如果需要在加载 WebView 中的 Web 页面中使用 JavaScript，就要在 WebView 中启用 JavaScript。可以通过 WebView 的 WebSettings 属性来启用它。先通过 getSettings() 来获取 WebSettings 的值，然后通过 setJavaScriptEnabled() 来启用 JavaScript。例如：

```
WebView myWebView = (WebView) findViewById(R.id.webview);
WebSettings webSettings = myWebView.getSettings();
webSettings.setJavaScriptEnabled(true);
```

4. JavaScript 和 Android 代码相互调用

通过 WebView，可以在 JavaScript 代码和客户端的 Android 代码间创建接口。例如，JavaScript 代码可以调用 Android 代码中的方法来展示一个 Dialog，而不需要使用 JavaScript 中的 alert() 函数。为了在你的 JavaScript 和 Android 代码间绑定一个新的接口，需要调用 addJavascriptInterface()，传给它一个类实例来绑定到 JavaScript，还需要一个接口名让 JavaScript 可以调用，以便来访问类。

例如，在 Android 应用中包括如下类：

```
public class JavaScriptInterface {
      Context mContext;
      /* 实例化接口 */
      JavaScriptInterface(Context c) {
                mContext = c;
      }
      public void showToast(String toast) {
                Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
      }
}
```

在这个例子中，JavaScriptInterface 类让 Web 页面可以使用 showToast() 方法来创建一个 Toast 消息。

可以通过 addJavascriptInterface() 绑定 JavaScriptInterface 类到正在 WebView 运行的 JavaScript，并将接口命名为 Android。例如：

```
WebView webView = (WebView) findViewById(R.id.webview);
webView.addJavascriptInterface(new JavaScriptInterface(this), "Android");
```

这段代码为在 WebView 上运行的 JavaScript 创建了一个名为 Android 的接口。这

时候，你的 Web app 就能访问 JavaScriptInterface 类了。例如，下面是一些 HTML 以及 JavaScript，在用户单击按钮的时候，它们使用这个新接口创建一个 Toast 消息。

```
    <input type="button" value="Say hello" onClick="showAndroidToast('Hello
Android!')" />
    <script type="text/javascript">
    function showAndroidToast(toast) {
          Android.showToast(toast);
    }
    </script>
```

不需要从 JavaScript 初始化 Android 接口，WebView 会自动让它为你的 Web 页面所用。所以，在单击按钮的时候，showAndroidToast() 函数会用这个 Android 接口来调用 JavaScriptInterface.showToast() 方法。

注意　绑定到你的 JavaScript 的对象在另一个线程中运行，而不是在创建它的线程中运行。使用 addJavascriptInterface() 可以让 JavaScript 控制你的 Android 应用。这是一把双刃剑，既有用同时也可能带来安全威胁。当 WebView 中的 HTML 不可信时（例如，HTML 的部分或者全部都是由一个未知的人或者进程提供的），那么一个攻击者就可能使用 HTML 来执行客户端的任何他想要的代码。因此，不应该使用 addJavascriptInterface()，除非 WebView 中的所有 HTML 以及 JavaScript 都是你自己写的。同样不应该让用户将你的 WebView 定向到另外一个不是你自己的 Web 页面上去（相反，让用户的默认浏览器应用打开外部链接——用户浏览器默认打开所有 URL 链接，因此一定要小心处理页面导航，像下面所描述的那样）。

5. 处理页面导航

当用户单击一个 WebView 中的页面链接时，默认是让 Android 启动一个可以处理 URL 的应用。通常，是由默认的浏览器打开并加载目标 URL 的。然而，你可以在 WebView 中覆盖这一行为，那么链接就会在 WebView 中打开。这样，你就可以让用户通过保存在 WebView 中的浏览记录执行前进或者后退操作了。

要想让用户可以通过单击打开链接，只需要使用 setWebViewClient() 为 WebView 提供一个 WebViewClient 即可。例如：

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.setWebViewClient(new WebViewClient());
```

这样，现在所有用户单击的链接都会直接在 WebView 中加载了。

如果想要对于加载的链接的位置有更多控制，你可以创建自己的 WebViewClient，并覆盖 shouldOverrideUrlLoading() 方法。例如：

```
private class MyWebViewClient extends WebViewClient {
      @Override
      public boolean shouldOverrideUrlLoading(WebView view, String url) {
                if (Uri.parse(url).getHost().equals("www.example.com")) {
                        //让 WebView 加载该页面
                        return false;
                }
                //设置为在新的页面中打开
                Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
                startActivity(intent);
                return true;
      }
}
```

然后为 WebView 创建一个新的 WebViewClient 的实例。

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.setWebViewClient(new MyWebViewClient());
```

现在当用户单击链接的时候，系统会调用 shouldOverrideUrlLoading()，通过 if(Uri.parse(url).getHost().equals("www.example.com")) 来检查 URL host 是否和某个特定的域匹配（如上面定义的 "www.example.com"）。如果匹配，那么该方法就返回 false，不去覆盖 URL 加载（它仍然让 WebView 像往常一样加载 URL）。如果不匹配，那么就会创建一个 Intent 来加载默认活动（default Activity）来处理 URL（通过用户默认的 Web 浏览器解析）。

6. 历史记录导航

当 WebView 覆盖了 URL 加载时，它会自动生成历史访问记录。可以通过 goBack() 或 goForward() 向前或向后访问已访问过的站点。

例如，下面的代码实现了通过 Activity 来利用设备的后退按钮来向后导航。

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
      //检测是否是返回键，并且有可以返回的历史记录
      if ((keyCode == KeyEvent.KEYCODE_BACK) && myWebView.canGoBack() {
                myWebView.goBack();
                return true;
      }
      //如果没有可以返回的历史记录则返回给系统处理，此时有可能是退出该界面
      return super.onKeyDown(keyCode, event);
}
```

如果有历史访问记录可供访问，canGoBack() 方法会返回 true。类似地，可以使用 canGoForward() 来检查是否有向前访问记录。如果不做这个检查，那么一旦用户访问到历

史记录最后一项，goBack() 或 goForward() 就什么都不做。

3.5.4　实战案例：Android 自定义打开 HTML 页面

案例解析 http://translate.google.cn/m 里面的语言列表页面，取出其文字，同时过滤不需要的部分，即去除“反馈意见”等相关说明，其标识为如下的加粗字体。

```
<html>
<body dir="ltr">
<form class="" action="/m?hl=zh-CN">
<div class="small center">
      <br><br>
      <a href="http:// www.google.cn/m?hl=zh-CN">Google 首页 </a>
      <br><br>
      <a href="http:// www.google.cn/m/support?hl=zh-CN"> 向我们发送反馈意见 </a>
      <br><br>
      查看 Google:
      <br>
      <b> 移动版 </b>
      | <a href="http:// translate.google.cn/?hl=zh-CN&vi=c"> 标准版 </a>
      <br><br>
      ©2012 Google -
      <a href="http:// m.google.cn/static/zh-CN/privacy.html"> 隐私权政策 </a>
</div>
</body>
</html>
```

解析该翻译页面的主要代码如下：

```
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
// 引入 Jsoup 解析包
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
import android.os.AsyncTask;
import android.os.Bundle;
import android.app.Activity;
import android.util.Log;
import android.view.Menu;
import android.webkit.WebSettings;
```

```
import android.webkit.WebView;
import android.webkit.WebViewClient;
public class MainActivity extends Activity {
      //定义调试的标签
      private static final String TAG = "ParseHtml";
      WebView wv ;
      //设置获取页面的地址，使用移动版的地址可以减少流量，加快速度
      String url = "http://translate.google.cn/m";
      //设置语言选择列表页面
      String langListUrl
                ="http://translate.google.cn/m?sl=auto&tl=zh-CN&hl=zh-CN&mui=sl";
      @Override
      public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                //加载布局文件
                setContentView(R.layout.activity_main);
                //获取 WebView 控件
                wv = (WebView)findViewById(R.id.webView1);
                //获取 WebView 属性
                WebSettings webSettings = wv.getSettings();
                //允许在 WebView 里面运行 JavaScript
                webSettings.setJavaScriptEnabled(true);
                wv.setWebViewClient(new WebViewClient(){
                        //在 WebView 加载的时候运行
                public boolean shouldOverrideUrlLoading(WebView  view, String url) {
                        //获取翻译页面
                        new HTMLAsyncTask().execute(url);
                        return true;
                }
                });
                //获取语言列表页面
                new HTMLAsyncTask().execute(langListUrl);
      }
      //创建 Menu 菜单
      @Override
      public boolean onCreateOptionsMenu(Menu menu) {
                getMenuInflater().inflate(R.menu.activity_main, menu);
                return true;
      }
      //使用异步方法
      private class HTMLAsyncTask extends AsyncTask<String, Integer, String> {
      @Override
      protected String doInBackground(String... Urls) {
                String html = "";
                try {
                        //使用参数 Url，新建 URL 对象
                        URL newUrl = new URL(Urls[0]);
                        //新建 URLConnection 对象，打开连接
                        URLConnection connect = newUrl.openConnection();
```

```
                    //设置 User-Agent 的值
                    connect.setRequestProperty("User-Agent","Mozilla/4.0
                                (compatible; MSIE 5.0; Windows NT; DigExt)");
                    //保存到数据流中
                    DataInputStream dis = new DataInputStream(
                    //获取数据流
                    connect.getInputStream());
                    //把数据放到 Buffer 中
                    BufferedReader in = new BufferedReader(
                    //根据实际情况指定编码，防止乱码
                    new InputStreamReader(dis,"GB2312"));
                    //定义字符串
                    String readLine = null;
                    //将数据读到字符串中
                    while ((readLine = in.readLine()) != null) {
                                html = html + readLine;
                    }
                    //关闭 Buffer
                    in.close();
                    //以字符串方式返回页面信息
                    return html;
            } catch (MalformedURLException me) {
            } catch (IOException ioe) {
            }
  return null;
  }
  //异步方法调用对返回的字符串进行处理
  protected void onPostExecute(String result) {
            //对字符串进行处理
            String html = ModifyingHtml(result);
            //重新加载页面
            wv.loadDataWithBaseURL("http://translate.google.cn/m",
                    result,"text/html", "UTF-8","");
     }
};
//处理页面过滤掉不需要的部分
String ModifyingHtml(String Html) {
            //定义字符串
            String html = Html;
            //新建 Jsoup 的 Document 对象
            Document doc = Jsoup.parse(html);
            //取出字符串中第 3 个 div 标签的内容
            Element body = doc.select("div").get(2);
            //将 body 中的字符串替换为 "" 空内容字符串
            body.text("");
            //解析语言列表页面
            ParseLanguageList(doc.html());
            //返回解析之后的内容
```

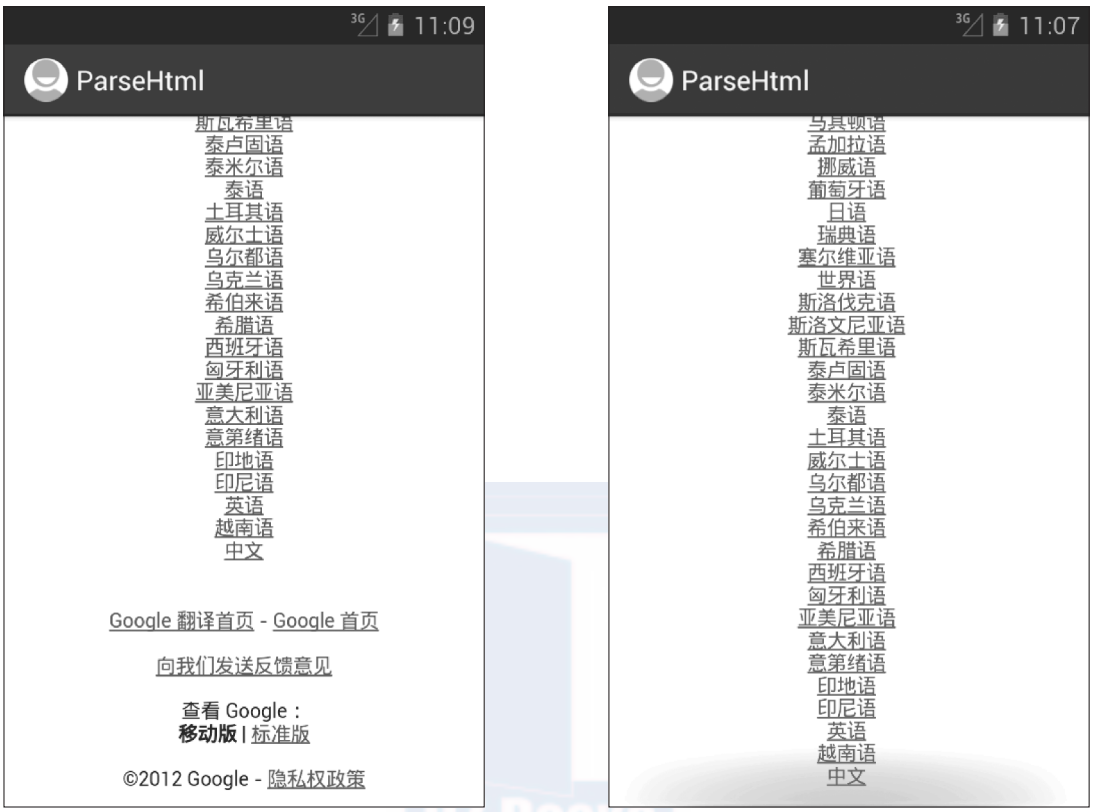
```
            return doc.html();
    }
    // 解析语言列表
    void ParseLanguageList(String Html) {
            // 创建 Jsoup 中的 Document 对象
            Document doc = Jsoup.parse(Html);
            // 找出其中的超链接
            Elements links = doc.select("a");
            // 对超链接元素的组合进行操作
            for(int i = 0; i<links.size();i++){
                    // 获取第 i 个超链接元素
                     Element link = links.get(i);
                    // 得到其中的文本
                    String text = link.text();
                    // 得到其中的链接地址
                    String linkHref = link.attr("href");
                    // 得到其中的文本
                    String linkText = link.text();
                    // 打印得到的内容
                    Log.d(TAG, "text      " + text);
                    Log.d(TAG, "linkHref " + linkHref);
                    Log.d(TAG, "linkText " + linkText);
            }
    }
}
```

从 HTML 中解析出的语言如图 3-13 所示。

```
ParseHtml       text     德语
ParseHtml       linkHref http://translate.google.cn/m?sl=de&tl=zh-CN&hl=zh-CN
ParseHtml       linkText 德语
ParseHtml       text     俄语
ParseHtml       linkHref http://translate.google.cn/m?sl=ru&tl=zh-CN&hl=zh-CN
ParseHtml       linkText 俄语
ParseHtml       text     法语
ParseHtml       linkHref http://translate.google.cn/m?sl=fr&tl=zh-CN&hl=zh-CN
ParseHtml       linkText 法语
ParseHtml       text     菲律宾语
ParseHtml       linkHref http://translate.google.cn/m?sl=tl&tl=zh-CN&hl=zh-CN
ParseHtml       linkText 菲律宾语
ParseHtml       text     芬兰语
ParseHtml       linkHref http://translate.google.cn/m?sl=fi&tl=zh-CN&hl=zh-CN
ParseHtml       linkText 芬兰语
ParseHtml       text     格鲁吉亚语
ParseHtml       linkHref http://translate.google.cn/m?sl=ka&tl=zh-CN&hl=zh-CN
ParseHtml       linkText 格鲁吉亚语
ParseHtml       text     古吉拉特语
ParseHtml       linkHref http://translate.google.cn/m?sl=gu&tl=zh-CN&hl=zh-CN
```

图 3-13 Android 从 Google 翻译页面解析出的语言

解析前和解析后的效果图比较如图 3-14 所示。



a）解析前　　b）解析后

图 3-14　Android 解析 Google 翻译页面

3.6　小结

本章详细介绍了 HTTP 协议，在此基础上，给出了 Android 中基于 HTTP 协议的文件上传案例。随后介绍了 HttpClient 和 URLConnection 等相关概念，并使用其访问维基百科。接着介绍了 JSON 和 SOAP 等相关概念，分别给出了 JSON 解析 wikipedia 及 SOAP 解析天气的案例。最后介绍了 Android 解析 HTML 的相关知识点。本章内容涉及面广，读者需要耐心研读并多加以实践。