第 3 章

开源跨平台的设备端开发

移动开发并不是一个非常新的命题。但就微软来说，早在 20 年前微软公司就推出了 Windows CE。用户真正熟悉的是 Windows CE 4.0 和 6.0 等版本，10 年前的多普达 686 和 696 是资深手机玩家的必备产品。在 Linux 阵营，基于 ARM 架构的版本 ARM Linux 的发展也有将近 20 年的历史了。本章将向读者介绍移动开发的历史、特点及主流技术。

3.1 移动开发简史

既然要讲到移动端开发，那么就需要简单了解一下移动开发的历史。本节将简要介绍 Windows CE、Symbian、Android 三款著名的移动端操作系统。

3.1.1 Windows 的精简版 Windows CE

Windows CE 是 Windows Compact Edition 的缩写，顾名思义，是一个经过精简的 Windows 操作系统。Windows CE 为了适应移动设备的小屏幕、低功耗等需求，对操作系统中不必要的功能和服务进行了删减，形成了 Windows 的一个分支。

第一个版本的 Windows CE 诞生于 1997 年，最早被应用于卡西欧的个人数字助理设备中。Windows CE 经过长时间的发展，在 4.0 版本时日臻成熟。并在 4.0 版本中首次区分了带有硬件键盘的 SmartPhone 版本，以及大屏但不带有硬件键盘的 Pocket PC 版本。微软当时与多家硬件厂商进行深度合作，将 Palm OS 视为最大的竞争对手。Windows CE 后来又出现了多个分支版本，比较著名的有 Zune 版本。这款名为 Zune 的硬件是微软用来与 iPod 一较高下的播放器。之所以要提到 Zune，是因为它是第一款应用了 Modern App 界面（又称 Metro 界面）的微软产品，后来 Xbox 和 Windows Phone 也相继应用了这套界面规范。

Windows CE 给这个社会带来的影响是非常深远的，很多早期的 ATM 自动柜员机都是使用 Windows CE 版本作为操作系统，直到最近十年才被 Windows XP Embedded 和 Windows 7 Embedded 版本逐渐替代。请注意，Windows CE 与 Windows XP Embedded 没有继承关系，它们是不同类型的操作系统。另一个广泛应用的场景是微软与福特合作，在福

特全系列车型中植入 Windows CE 作为汽车的车载系统，至今很多行驶在美国马路上的福特汽车仍然运行以 Windows CE 为内核的车载终端。

Windows CE 的最后一个高潮版本是 Windows Phone 7.0 和 7.5。微软在这个版本上将 Modern 风格的界面作为应用程序的设计标准加以规范并推广。

后来，随着 Windows 8 内核开始直接可以运行在 ARM 架构的 CPU 上面，微软停止了对 Windows CE 系列操作系统的支持和更新。

Windows CE 的开发早期以 Visual C++和 Visual Basic 两种开发工具为主，微软甚至专门推出了针对 Windows CE 的开发 IDE EVC(Embedded Visual C++)和 EVB(Embedded Visual Basic)。在 20 年前，用 VC++给 Windows CE 编写应用程序是很先进的。微软甚至允许程序员在 Windows CE 上面有限地使用庞大的 MFC 类库。在 Windows CE 4.0 时代，随着.NET 逐渐流行，Windows CE 也支持了.NET 的一个精简版本.NET CF(Compact Framework)。不过由于这个.NET 版本太过精简功能极度受限，并没受到广大程序员的欢迎。

3.1.2 工控起家的 Symbian

在 20 世纪 90 年代末期，诺基亚手机以其耐用性和经济性迅速占领了移动电话市场。在 21 世纪初，诺基亚甚至打出了“你每眨一次眼，诺基亚就卖出四部手机”的口号。可见市场占用率之高。作为功能机的老大，诺基亚也觊觎智能机市场。于是在 2000 年前后拉着爱立信、索爱等几家手机生产商共同注资成立了 Symbian 公司。Symbian 早期并没有发明 Symbian OS，而是从一家英国做工业控制和自动化的公司买下了它们的操作系统 EPOC。

EPOC 操作系统体系结构完善，架构优良，最重要的是能耗非常低。Symbian 在 EPOC 的基础上加入了 WAP 浏览器并集成了音乐、图像等系统服务，从而开发了 SDK 形成 Symbian OS 操作系统。这款操作系统的 EPOC 印记明显，在它的 SDK 中有很多文件夹仍然以 EPOC 命名。

Symbian OS SDK 采用 C++作为编程语言，并支持 Visual C++6.0 和 Code Warrior 作为 IDE。在 SDK 中使用 Perl 脚本语言生成自动化脚本来创建项目文件，同时也通过 Perl 脚本语言生成自动化构建脚本。这意味着开发者在 Windows 上面不能使用带有空格和中文字符的路径。Perl 脚本无法支持这种路径字符串。

Symbian OS 是一款专门针对 ARM 4、ARM7 等 ARM 架构 CPU 优化的操作系统。因此，这款操作系统是一款纯 32 位的操作系统。由于其内核精巧，使得应用程序可以占用较低的内存高效地运行在操作系统之上。作者依稀记得，在 Symbian OS 的开发教材上，一开始就讨论了一个问题：Boolean Int 两种数据类型中的哪一种在 Symbian OS 上运行更快？由于前一种类型只用一位来表示，因此在 ARM CPU 处理时要加入补位指令补齐 31 个零，所以后者更快。

针对所有应用程序都要共同使用到的基础功能比如文件读写、音频播放、网络访问等 Symbian OS 都将其逐个封装成服务。应用程序作为服务的客户端可以创建一个指向服务

的指针。当服务使用结束，再将指向服务的指针释放，从而实现了对高资源占用功能的调度。对于开发者来说，这非常像使用 COM 组件，因此没有什么障碍。同时，SDK 中也提供了对应的智能指针的封装类。

Symbian OS 根据当时智能机硬件规格分为 S40、S60、S80/90 等几个分支。Symbian 公司的投资商如索爱等公司再在 Symbian OS 的基础上根据各自对用户使用习惯的理解设计了界面程序。而诺基亚自己则选择使用了原生的 Symbian OS 界面。

总之，在当时硬件功能有限，电池容量不高的背景条件下 Symbian OS 是一款出色的操作系统。低功耗、支持 WAP 浏览的特性都引领了那个时代的潮流。只是 SDK 复杂，开发者学习成本高昂，同时用户界面简陋，后期无法快速地支持更多硬件规格，这为 Symbian 的没落埋下了伏笔。

3.1.3　Android 的革命性崛起

Android 是机器人的意思，中文发音安卓。这款操作系统也不是谷歌自己发明的，而是由一位硅谷的创业大师 Andy Rubin 在 2004 年发明的。

最早 Andy 的公司设计了一款智能手机操作系统，这款操作系统在初期被微软看中以 2 亿美元购买了它的知识产权，并以此推出一款名为 Kin 的针对儿童市场的智能手机。这款手机支持儿童社交等功能，但市场反应平平，仅卖出 400 多部。成为微软 CEO Steve Ballmer 执政期间的一大败笔。现在想想，如果把之前那 2 亿美元算进来 Kin 手机可能现在仍是世界上成本最高昂的手机，任何镶钻的奢侈手机也无法与它的单机成本抗衡。

Andy Rubin 在卖掉第一款智能手机操作系统后，专心研发他的第二款智能手机操作系统，也就是现在的 Android。而当时，谷歌已经在搜索领域大获成功，有大笔的资金可以对未来进行投资。谷歌敏锐地意识到，未来将是移动设备的天下，用户在移动设备上的使用时间将大大地超过 PC 上的使用时间。而此时不抓住机会占领移动设备市场，将会丢掉大部分市场份额。

于是，谷歌开发了针对以 ARM 的 Linux 为核心的 Android 系统，非常有超前意识地广泛支持了各种设备规格。

3.2　移动开发的显著特性

相较于服务器端应用开发和桌面应用程序开发，移动应用开发具有几个鲜明的特点。一款移动应用程序是否很好地体现了以下的特点，决定了这款应用的成败。

3.2.1　用户体验是第一位的

与桌面应用程序不同，移动端应用程序的主要输入设备只有键盘（无论物理键盘和虚拟键盘），并且键盘的面积很小，无法提供给用户全尺寸键盘。而桌面应用程序不同，用户常用的输入设备是鼠标和全尺寸物理键盘。因此，合理设计用户界面让用户方便地在移动设备

上输入就成了决定移动应用开发的成败。图 3.1 展示的是桌面上用于数学建模计算的 MATLAB 软件界面。由于可以使用鼠标，这种界面在 PC 上使用毫无难度。但是在移动设备上就非常难以操作。

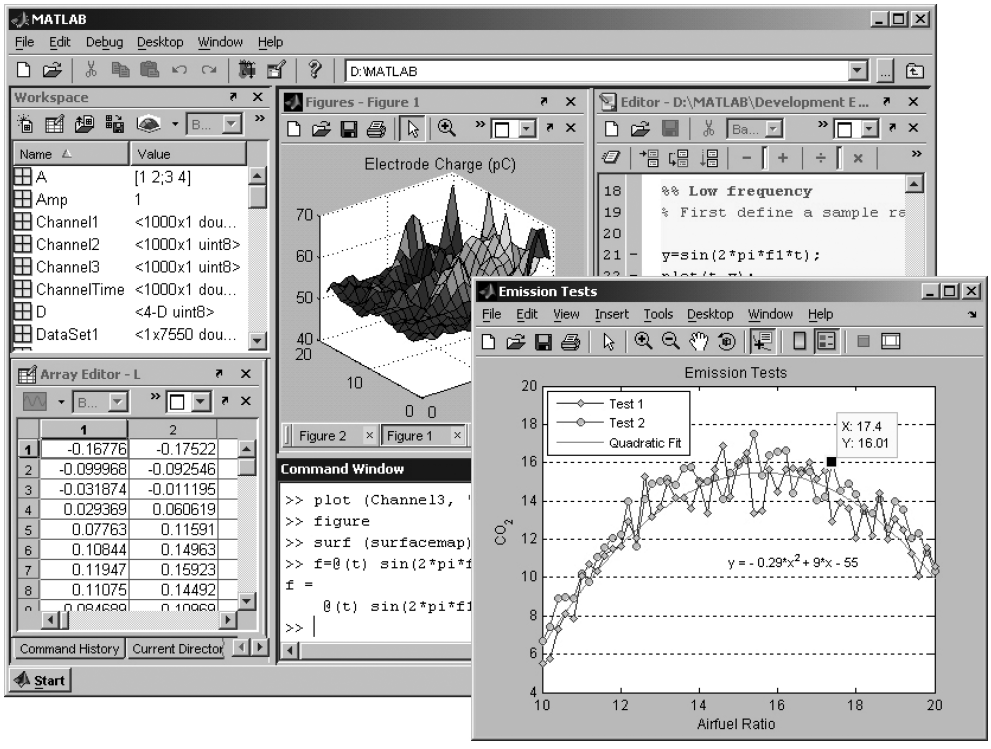


图 3.1　MATLAB 软件界面

因此，需要在开发移动应用时充分考虑用户的使用场景，数据能查询或者推算的就尽量不要让用户输入；可以使用选择器控件的（如 Dropdown List）就尽量不要使用文本框；非敏感信息，能帮助用户缓存的就一定要缓存，避免同一信息多次输入。

在这方面一个极为优秀的例子是微信。微信率先使用移动设备的麦克风作为用户输入设备，把用户读出的语音记录下来以极高的压缩率制成音频文件传给对话的另外一方。这一改变可以说是革命性的，它在方便了用户交流的同时，提高了微信的使用率，也增加了用户的黏度。同时，微信在播放语音时，还可以判断目前移动设备的姿态。如果移动设备在用户耳边，就使用听筒播放语音；如果移动设备是其他姿态就使用外放播放语音。既照顾到用户通信的私密性又照顾到使用的便利。这样的应用在设计之初，就把用户的体验放在了应用开发需求的最高优先级。

另一个提升输入体验的典型例子是二维码。以前用户如果需要用移动设备访问某个网页，就必须要手工用移动设备的简化键盘输入网址。有了二维码，通过移动设备的镜头进行扫描，马上就可以访问网页，大大方便了信息的交流。

3.2.2　移动设备的资源限制颇多

通常来说，移动设备的软硬件资源较 PC 和服务器来说极为有限。比如 iPhone 一直只有 1GB 内存，这一情况直到 iPhone 6S 才有所改观变成 2GB。而在 PC 和服务器领域，4GB 内存在多年前早已是标配。这就意味着在移动设备上做开发的程序员必须对申请的资源精打细算，不能无节制地申请和释放内存。

同样的问题还反映在电池上。由于通常情况下 PC 和服务器都是连接在不间断电源上工作的。在高性能 CPU 的支持下，算法的效率其实并不是最高优先级考虑的事情。换句话说，在 PC 上对一万个无序数字组成的数组使用冒泡排序还是快速排序往往只是让用户感觉到程序运行时快慢有所不同，而不会考虑两个算法在功耗上的差距。但是对于移动设备就截然不同了，因为移动设备往往是依赖电池工作的。一个高功耗的缓慢算法，不但会让用户等待的时间显著延长（由于移动是设备 CPU 运算能力普遍低于 PC，会让算法的效率差异更加的显著），还会大大影响移动设备单次充电后的使用时长，甚至由于电池长时间高负荷运转导致高发热量造成整个移动设备运行不稳定等状况。因此在移动设备编程领域，开发人员在编码时必须要时刻注意代码对移动设备资源的使用影响，尽量降低软件功耗。

其他的资源方面限制还体现在网络带宽和存储空间上。虽然 iPhone 的最高存储容量已经提高到了 128GB，但是这并不是强制性的。因此，开发者在编写移动设备应用程序时既要尽量使用本地缓存减少网络带宽占用，又要顾及本地存储空间的使用量。举例来说，几乎所有市面上流行的新闻客户端都提供了本地缓存清理功能，让用户在移动设备存储空间较为窘迫时可以释放缓存数据占用的空间。

3.2.3　移动应用程序需要处理移动设备的特殊事件

通常，在桌面应用程序中往往只有运行和停止两种状态，而在移动设备中，往往还多了一个挂起的状态。所谓挂起状态（Suspended），是指用户在移动设备商将当前运行应用切换成另一个应用程序，但是老的应用程序并没有被关闭仍然驻留内存的一种状态。当应用程序由运行状态切换为挂起状态时，需要程序员保存当前操作会话的临时数据，以便当程序从挂起状态切换回运行状态时将临时数据恢复。移动应用程序生命周期的三种状态如图 3.2 所示。

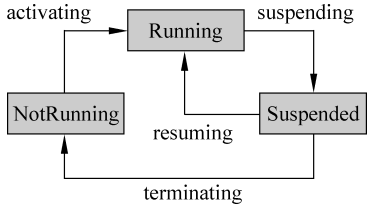


图 3.2　移动应用程序生命周期的三种状态

除了需要比桌面应用程序多处理生命周期状态，移动应用程序还需要及时处理诸如电池事件、网络连接事件以及蓝牙连接事件等。因为移动设备普遍使用电量有限的电池作为能源。因此，需要在设备发出低电量警告通知时保存当前用户操作的临时数据到设备的存储上。又因为移动设备经常因地理位置、海拔等因素导致网络连接频繁地在连接/断开状态之间切换，所以在应用程序中非常有必要处理好在网络不同状态下应用程序功能的行为。这些都是作为一个移动应用开发人员编写用户体验良好的应用程序时必须要考虑到的问题。

3.2.4 移动应用程序需要处理好数据同步和数据冲突的问题

在移动应用开发领域中，由于网络条件和网络带宽的限制，往往需要缓存一部分数据到移动应用程序中，待网络条件允许后再把数据批量同步到服务端。一个典型的例子是铁路12306手机客户端应用程序，这个程序每次启动时都要检测服务器上有没有车站和时刻表的更新数据，如果有就先下载到本地进行缓存，如图3.3所示。

图3.3 12306手机客户端下载缓存数据

缓存数据对于开发人员来说，其实是增大了应用开发的逻辑难度。因为这涉及数据同步和数据冲突的处理问题。对于数据同步，可以使用微软Sync Framework等组件帮助实现，也可以由开发人员自己编写代码实现。但是考虑到数据同步将对业务数据产生重大影响，建议在程序编写完成后充分地进行数据同步测试。

3.3 主流跨平台移动开发技术介绍

各大移动开发平台都有自己专有的开发工具、SDK乃至开发语言。iOS系统使用XCode集成开发工具通过Object-C或者Swift语言进行应用程序的开发；Android系统使用Android Studio作为集成开发环境，开发者可以使用Java或者C来进行编程。

每种平台使用各自的开发工具和开发语言都有各自的考虑，但这样一来给开发商也带

来了极大的困扰。使用各个平台的原生开发环境和开发语言使得开发商无法使用一套codebase生成针对各个平台的应用程序。这使得移动应用的开发成本和代码维护工作量成倍增加。在这个互联网创业大爆炸的年代，如果一个重要的产品功能不能先于竞争对手推出，那么很可能就意味着抢占市场的努力就白费了。这是广大创业者所不能容忍的。

于是，一些可以使用一个统一 codebase 开发横跨各种流行移动平台的技术就应运而生了。下面将介绍几款跨平台移动开发技术的杰出代表。

3.3.1　QT Library 介绍

QT Library 是挪威奇趣(TrollTech)公司在 1994 年创立的一套跨操作系统平台开发类库。这套类库基于 C/C++编写，实现了完整的消息处理机制和桌面 UI 元素如按钮、下拉框等。QT Library 创立之初其实不是专门为了移动开发而创建的，而主要是为了解决 OS X、Linux、Windows 图形用户界面(GUI)应用程序跨平台而创立的。Linux 上著名的 KDE 就是基于 QT 二次开发形成的。

QT Library 为了掩盖各个操作系统平台之间的差异，做了大量的工作。QT Library 自己实现了线程类，封装了对各个操作系统平台原生线程 API。这样一来，基于 QT Library 开发的桌面程序通过针对不同操作系统平台的编译，就可以实现一套代码 UNIX、Linux、Windows 三个操作系统运行。

QT Library 甚至还开发了自己的 IDE，帮助开发者可视化编程。当然 IDE 本身也是用 QT Library 开发的。

进入 21 世纪以后，随着移动开发领域的逐渐热络，QT Library 也渐渐地开始支持各种移动操作系统平台。其实这也很自然，像 Android 系统本身也是由 Linux 发展而来，在初期兼容 Android 没有太多的困难。2008 年，还处在巅峰时期的诺基亚收购了奇趣公司，想让 QT Library 更好地在 Symbian 系统上发展。不过后来随着诺基亚的江河日下，QT Library 也开始了一段多舛的命运。先是支持了 MeeGo 系统但 MeeGo 仅出了一款手机后官方就停止发展了。后来诺基亚将 QT Library 逐渐(多次交易)卖给了芬兰的 Digia 公司，整个 QT Library 于 2012 年 8 月整体移交给 Digia 公司。自这时起，QT Library 开始完善地支持所有流行的移动开发平台。QT Library 目前支持的移动平台主要有：

(1) Windows Phone-arm；

(2) Windows RT-64bit；

(3) iOS 8.1；

(4) Android 4.4；

(5) Tizen OS。

QT Library 目前有两个主力版本：QT5.5 和 QT 4.8。QT Library 是开放源代码的，获取源代码请参考：http://wiki.qt.io/Get_The_Source。但是请注意，QT Library 的商业授权仍然需要付费，每个开发者授权大概是 350 美元/月。QT Library 的首页请访问：http://www.qt.io/。

3.3.2 Xamarin 介绍

相较于 QT Library 的奇趣公司几十年的历史，Xamarin 实在是年轻。这家成立于 2011 年的公司，正是看中了当前移动开发平台诸侯争霸中的商机。虽说公司创立才刚刚 5 年，但是 Xamarin 公司的技术已非常成熟。Xamarin 公司的技术负责人 Miguel de Icaza 是个大有来头的人物。1972 年出生于墨西哥城的他，从 20 岁起就开始了他自由软件程序员的生涯，1997 年创建了 Linux 上著名的图形桌面环境 Gnome，2001 年与 Xamarin CEO Nat Friedman 一起创立了另一个著名的开源项目. NET Mono。

读者也一定想到了，Xamarin 能够迅速地被开发出来，主要是因为 Xamarin 是构建在 Mono Framework 上的缘故。有了 Mono Framework 的支持，程序界面和代码很多都可以被复用，Xamarin 的开发速度大大加快。所不足的是，这样一个模式下，需要为移动应用植入一个运行时来支持应用程序的运行。

Xamarin 使用清爽简洁、功能强大的 C# 作为开发语言，提供了一套对移动开发所需要的界面、I/O 等功能进行了封装的. NET 类库。使得开发人员可以像开发 WinForm 或者 UWP 应用程序一样开发运行在 iOS 和 Android 操作系统上的应用程序。

与 QT Library 比较，Xamarin 没有以前桌面操作系统的历史包袱，而是专司 iOS、Windows、Android 三个平台的跨平台实现。Xamarin 同时基于 Mono Framework 创建了自己跨平台的 IDE，称为 Xamarin Studio。同时也提供 Visual Studio 扩展插件，将自己植入 Visual Studio 中。

经过数年的发展，Xamarin 已经从一个工程支持一个平台项目的方式，发展到了一个共享工程可以支持多个平台运行的方式，Xamarin 将这个大一统的界面库叫做 Xamarin. Forms。Xamarin. Forms 显著的特点是它的统一界面库既保持了相对统一的风格，又在细节之处符合各自平台的设计规范。如图 3.4 所示。

图 3.4 Xamarin Master Detail 视图在 iOS、Android、Windows Phone 三个平台上的展现

虽然使用托管语言开发，但是经过编译、生成的应用程序其实是原生代码的。因此，Xamarin 开发的应用程序具有开发方式统一、运行效率高等特点，并逐渐受到了森海塞尔、JetBlue 和博世、西门子等著名客户的青睐。

由于 Xamarin 的诸多特点，与微软的开发技术非常契合。这引起了微软的持续关注，在 2016 年 2 月 24 日，微软与 Xamarin 签订了收购协议，以 5 亿美元的价格全资收购了 Xamarin 公司。这一交易在 2016 年 3 月得到政府监管部门的批准，2016 年 4 月微软在 BUILD 大会上正式宣布了这一消息。图 3.5 展示了 Scott Guthrie 与 Miguel De Icaza 的合影，中间的那位是微软执行副总裁人称红衣教主 Scott Guthire，右边那位就是 Gnome 和 Mono 项目的创始人 Miguel De Icaza。

图 3.5　Scott Guthrie 与 Miguel De Icaza 合影

随着微软对 Xamarin 公司收购的完成，Xamarin 已经对 Visual Studio 用户免费开放使用。是的，即使是免费版的 Visual Studio 2015 Community，用户仍然可以用到完全功能版本的 Xamarin。

Xamarin 的另一重大改变就是 Xamarin 代码也开源了！Xamarin 是构建在 Mono Framework 之上的，Mono Framework 是开源的，但是 Xamarin 作为一款商业软件是不开源的。现在微软把 Xamarin. Forms 全套代码也贡献了出来，放在了 GitHub 网站上。读者有兴趣可以访问：https://github. com/xamarin/Xamarin. Forms 来查看。

3.3.3　Cordova 介绍

Apache Cordova 简称 Cordova，是由 Nitobi 小组创建一个一套横跨多个移动平台的移动开发框架。2011 年 Adobe 公司收购了 Nitobi 小组和知识产权，再加入 Adobe 自己的一些云服务组件构建了一个大名鼎鼎的产品 PhoneGap。随后 Adobe 将 Nitobi 小组的代码捐

献给了 Apache 组织，成立了开源项目 Apache Cordova。之所以被叫做 Cordova，是因为 Nitobi 小组当初办公室所在的大街叫做 Cordova 大街（位于加拿大温哥华）。

与前两个跨平台开发框架不同，Cordova 不是架构在原生界面库上的技术。Cordova 是一种寄宿在浏览器上的一种跨平台开发技术。Cordova 支持 JavaScript 和微软 TypeScript 两种脚本语言编程（一个在 JavaScript 基础上的强类型扩展，从根本上说是同一种语言），使用 HTML 5 标准渲染界面并支持与移动设备资源交互。从本质上来说，Cordova 技术就是用 HTML 页面与用户交互，用脚本语言表达逻辑运行在移动设备浏览器上的一组动态页面。

Cordova 的出现可谓是恰逢其时。这里有几个原因。第一，当前主流移动开发平台的浏览器都已经更换为现代浏览器了。微软的 Trident，Mozilla 的 Gecko，谷歌的 Chrome 和苹果的 Safari 等主流的渲染引擎都已经大幅度地提升了在移动设备上的渲染性能，并且对 HTML 5 给予良好的支持。第二，当今是前端技术大爆发的时代。JavaScript 和 TypeScript 有着庞大的用户群体，并且用 HTML＋CSS 渲染一个界面比用代码控制容易多了（iOS 原生开发，界面渲染是基于代码的），再加上不断推陈出新的前端框架（如 AngularJs），更加方便了开发者。第三，Cordova 是开源社区在维护的一个项目，全球的开发者都可以贡献自己的代码给 Cordova，大大地丰富了 Cordova 的功能。

Cordova 的优势在于可以快速低成本地开发一款横跨多种移动平台的应用，但是由于界面和代码严重依赖浏览器渲染，性能上与原生应用尚存一定的差距。因此比较适合信息展示类的应用程序开发，如商店、新闻、RSS 阅读等，不适合游戏开发场景。Cordova 对于一般的商业应用来说，已经足够了。它最大的优势就是可以快速开发和部署一个移动应用。这对于创业者来说弥足珍贵，因为创业本身是一个试错的过程。能用最小的时间和金钱代价进行试错，才是创业者的最优技术方案。

3.4 移动应用开发方式的选择

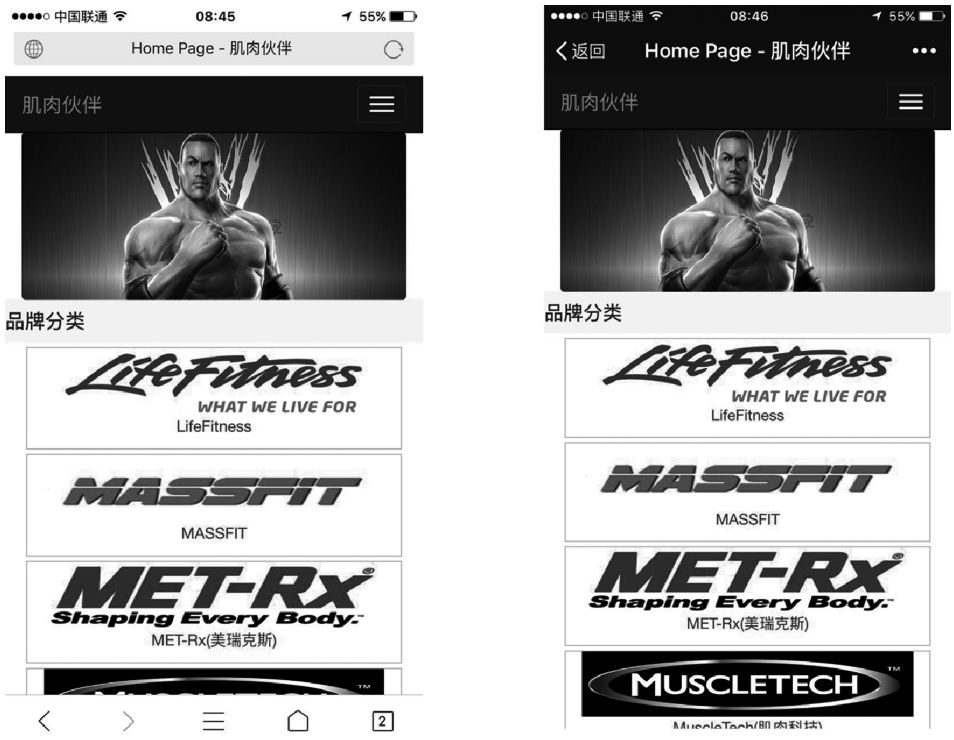
在了解了众多的支持移动应用开发的平台和界面库之后，怎么选择移动应用的开发方式就是一个摆在面前的问题了。本节将讨论如何选择一种恰当的移动应用开发方式。

3.4.1 Web 页面还是 App

其实即使是一个采用了响应式界面设计（Responsive Design）的 Web 网站也可以在移动设备的浏览器中进行完美的展示。比如本书中的示例 Muscle Fellow 网站，图 3.6 展示了 Muscle Fellow 网站在 QQ 浏览器和微信浏览器中的样子。

既然支持响应式设计的 Web 页面可以在移动浏览器内正常展示，那么为什么还需要开发众多的移动端应用呢？下面来看一个市场调查的结果，如图 3.7 所示。

在图 3.7 中展示了 2013 年和 2014 年手机用户在移动端浏览器和移动端应用上花费的时间比例。在图 3.7 中，我们可以了解到用户在浏览器上面花费的时间很少，且一直呈现递



QQ 浏览器截屏　　微信浏览器截屏

图 3.6　Muscle Fellow 页面在移动浏览器中的展示

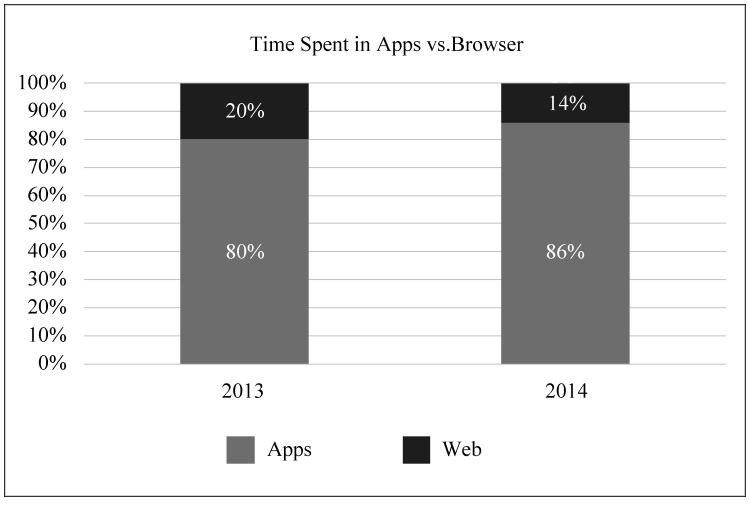


图 3.7　用户在浏览器和 App 上的使用时间

减的趋势。其中重要的原因是，用户在小屏设备上输入网站地址是一件非常费力的事情，且有那么多的域名需要记忆。而移动端的应用的易用性就在这里体现出来了，只要单击一下图标就可以访问功能了。因此，从用户的使用行为决定了必须要开发移动端应用，才能获得用户更多的使用机会和时间。

有了这个结论，下面就是要讨论怎么开发移动应用了。

3.4.2 移动应用开发方式的选择

其实从微软技术层面来看，总结起来移动应用的开发方式有三种：HTML＋JavaScript、Java 和 C＃等高级语言开发、C/C++开发原生程序，如图 3.8 所示。在这三种方式中，位于上层的 HTML＋JavaScript 虽然最简单，但是对于移动设备访问能力是受到一定限制的，同时性能上也没法与原生的 C/C++编写的应用相比。位于底层的 C/C++编写的移动应用虽然执行效率好，但是适配能力和移植很弱，一旦有新规格的设备出现，就需要通过修改程序来进行兼容。

图 3.8 开发方式的选择

那么在这三种方式里面应该如何选择呢？其实发挥决定性因素的不是技术本身，而是开发的成本。开发移动端应用的成本不仅包括财务成本，还有时间成本。尤其是互联网应用，在一个创业想法有多家对标公司在同时抢占市场的情况下，谁能最快地推出应用，谁就能占得市场的先机。

从开发成本上看，居于最上层的 HTML＋JavaScript 的方式可能是最低的，而位于最底层的 C++方式是最高的。首先，从人力成本上说，一个懂得 HTML＋JavaScript 的前端程序员要比会写 C/C++的程序员好招聘得多。HTML＋JavaScript 需求远大于 C/C++。其次，从开发方式和 API 来说，用脚本化的语言描述一段逻辑本来就要比 C/C++简单很多。这就导致了图 3.8 中三种模式开发成本自上而下地增加。

而出于对性能的一定需求和对硬件访问的需求，位于中间层次的.NET/C#方式是更加均衡的一种方式。本书既然讨论开源和跨平台，那么位于底层的C/C++就不在讨论范围之内了，剩下的就是Cordova和Xamarin两种开发方式。它们都开源，也都可以横跨多个移动平台。

3.4.3　Cordova还是Xamarin

这个问题就比较难回答了。因为这两种方式各有优长。另一个市场调查数据告诉你Cordova程序员为什么当初会选择Cordova，如图3.9所示。位居前三的理由分别是：使用了它们熟悉的编程语言HTML+JavaScript，可以跨平台，性能还可以接受。

也就是说，程序员选择Cordova的最根本原因是技术背景。用Cordova开发移动应用和做网页差不多，其学习曲线平滑，很快就能掌握，并且运行性能也还可以接受。

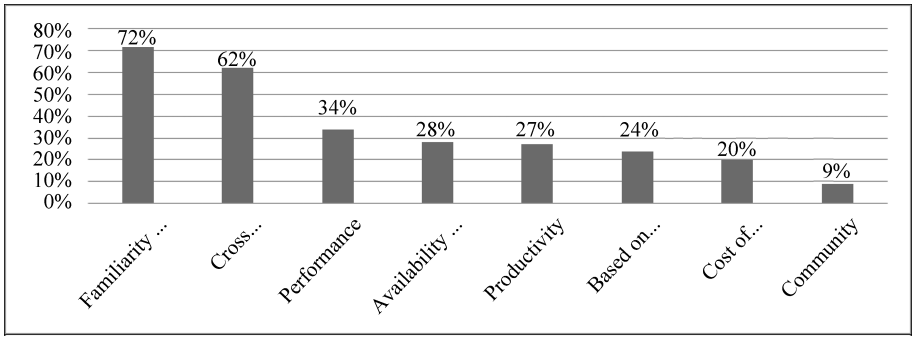


图3.9　Cordova市场调查数据

那么为什么会选择Xamarin呢？总结起来，Xamarin吸引程序员的理由有以下几条：

(1) 可高性能地访问本机API；

(2) 支持强类型和面向对象的开发；

(3) 用户界面设计支持控件拖曳；对于Windows开发者，没有必要学习JS或HTML；

(4) iOS和Android界面设计也支持控件拖曳；

(5) 使用XAML定义界面布局和自适应布局；

(6) 基于Mono，完全开放源代码。

那么什么时候适合使用Cordova而什么时候适合使用Xamarin开发移动应用程序呢？总结起来可以用下面几个条件来帮助你在Cordova和Xamarin中如何选择。

Cordova技术通常适应的场景：

(1) 移动应用在不同的平台之间界面保持较高的一致性，不需要为特定的移动平台定制界面；

(2) 与移动应用对应的服务器端Web网站可以共享界面元素资源；

(3) 开发人员有HTML5+JavaScript开发背景。

Xamarin 技术通常适应的场景：

（1）Xamarin.Forms 可以帮助程序员实现特定的应用界面；

（2）开发人员背景是.NET 技术背景，尤其是进行 UWP 应用开发；

（3）应用运行需要较高的性能，比如要开发一款游戏。

不管开发者在 Cordova 和 Xamarin 之间怎样选择，它们现在都是开源且免费的！本书的最后三章将介绍如何使用 Cordova 和 Xamarin 来开发移动端的应用。