# 英文翻译资料

1. 英文原文

**Android: A Programmer’s Guide**

It can be said that, for a while, traditional desktop application developers have been spoiled. This is not to say that traditional desktop application development is easier than other forms of development. However, as traditional desktop application developers, we have had the ability to create almost any kind of application we can imagine. I am including myself in this grouping because I got my start in desktop programming.

One aspect that has made desktop programming more accessible is that we have had the ability to interact with the desktop operating system, and thus interact with any underlying hardware, pretty freely (or at least with minimal exceptions). This kind of freedom to program independently, however, has never really been available to the small group of programmers who dared to venture into the murky waters of cell phone development.

NOTE

I refer to two different kinds of developers in this discussion: traditional desktop application developers, who work in almost any language and whose end product, applications, are built to run on any “desktop” operating system; and Android developers, Java developers who develop for the Android platform. This is not for the purposes of saying one is by any means better or worse than the other. Rather, the distinction is made for purposes of comparing the development styles and tools of desktop operating system environments to the mobile operating system environment, Android.

**Brief History of Embedded Device Programming**

For a long time, cell phone developers comprised a small sect of a slightly larger group of developers known as embedded device developers. Seen as a less “glamorous” sibling to desktop—and later web—development, embedded device development typically got the proverbial short end of the stick as far as hardware and operating system features, because embedded device manufacturers were notoriously stingy on feature support.

Embedded device manufacturers typically needed to guard their hardware secrets closely, so they gave embedded device developers few libraries to call when trying to interact with a specific device. Embedded devices differ from desktops in that an embedded device is typically a “computer on a chip.” For example, consider your standard television remote control; it is not really seen as an overwhelming achievement of technological complexity. When any button is pressed, a chip interprets the signal in a way that has been programmed into the device. This allows the device to know what to expect from the input device (key pad), and how to respond to those commands (for example, turn on the television). This is a simple form of embedded device programming. However, believe it or not, simple devices such as these are definitely related to the roots of early cell phone devices and development.

Most embedded devices ran (and in some cases still run) proprietary operating systems. The reason for choosing to create a proprietary operating system rather than use any consumer system was really a product of necessity. Simple devices did not need very robust and optimized operating systems.

As a product of device evolution, many of the more complex embedded devices, such as early PDAs, household security systems, and GPSs, moved to somewhat standardized operating system platforms about five years ago. Small-footprint operating systems such as Linux, or even an embedded version of Microsoft Windows, have become more prevalent on many embedded devices. Around this time in device evolution, cell phones branched from other embedded devices onto their own path. This branching is evident when you examine their architecture.

Nearly since their inception, cell phones have been fringe devices insofar as they run on proprietary software—software that is owned and controlled by the manufacturer, and is almost always considered to be a “closed” system. The practice of manufacturers using proprietary operating systems began more out of necessity than any other reason. That is, cell phone manufacturers typically used hardware that was completely developed in-house, or at least hardware that was specifically developed for the purposes of running cell phone equipment. As a result, there were no openly available, off-the-shelf software packages or solutions that would reliably interact with their hardware. Since the manufacturers also wanted to guard very closely their hardware trade secrets, some of which could be revealed by allowing access to the software level of the device, the common practice was, and in most cases still is, to use completely proprietary and closed software to run their devices. The downside to this is that anyone who wanted to develop applications for cell phones needed to have intimate knowledge of the proprietary environment within which it was to run. The solution was to purchase expensive development tools directly from the manufacturer. This isolated many of the “homebrew” developers.

A growing culture of homebrew developers has embraced cell phone application development. The term “homebrew” refers to the fact that these developers typically do not work for a cell phone development company and generally produce small, one-off products on their own time.

Another, more compelling “necessity” that kept cell phone development out of the hands of the everyday developer was the hardware manufacturers’ solution to the “memory versus need” dilemma. Until recently, cell phones did little more than execute and receive phone calls, track your contacts, and possibly send and receive short text messages; not really the “Swiss army knives” of technology they are today. Even as late as 2002, cell phones with cameras were not commonly found in the hands of consumers.

By 1997, small applications such as calculators and games (Tetris, for example) crept their way onto cell phones, but the overwhelming function was still that of a phone dialer itself. Cell phones had not yet become the multiuse, multifunction personal tools they are today. No one yet saw the need for Internet browsing, MP3 playing, or any of the multitudes of functions we are accustomed to using today. It is possible that the cell phone manufacturers of 1997 did not fully perceive the need consumers would have for an all-in-one device. However, even if the need was present, a lack of device memory and storage capacity was an even bigger obstacle to overcome. More people may have wanted their devices to be all-in-one tools, but manufacturers still had to climb the memory hurdle.

To put the problem simply, it takes memory to store and run applications on any device, cell phones included. Cell phones, as a device, until recently did not have the amount of memory available to them that would facilitate the inclusion of “extra” programs. Within the last two years, the price of memory has reached very low levels. Device manufacturers now have the ability to include more memory at lower prices. Many cell phones now have more standard memory than the average PC had in the mid-1990s. So, now that we have the need, and the memory, we can all jump in and develop cool applications for cell phones around the world, right? Not exactly.

Device manufacturers still closely guard the operating systems that run on their devices. While a few have opened up to the point where they will allow some Java-based applications to run within a small environment on the phone, many do not allow this. Even the systems that do allow some Java apps to run do not allow the kind of access to the “core” system that standard desktop developers are accustomed to having.

Open Handset Alliance and Android

This barrier to application development began to crumble in November of 2007 when Google, under the Open Handset Alliance, released Android. The Open Handset Alliance is a group of hardware and software developers, including Google, NTT DoCoMo, Sprint Nextel, and HTC, whose goal is to create a more open cell phone environment. The first product to be released under the alliance is the mobile device operating system, Android. (For more information about the Open Handset Alliance, see [www.openhandsetalliance.com](http://www.openhandsetalliance.com/).)

With the release of Android, Google made available a host of development tools and tutorials to aid would-be developers onto the new system. Help files, the platform software development kit (SDK), and even a developers’ community can be found at Google’s Android website, http://code.google.com/android. This site should be your starting point, and I highly encourage you to visit the site.

Google, in promoting the new Android operating system, even went as far as to create a $10 million contest looking for new and exciting Android applications.

While cell phones running Linux, Windows, and even PalmOS are easy to find, as of this writing, no hardware platforms have been announced for Android to run on. HTC, LG Electronics, Motorola, and Samsung are members of the Open Handset Alliance, under which Android has been released, so we can only hope that they have plans for a few Android-based devices in the near future. With its release in November 2007, the system itself is still in a software-only beta. This is good news for developers because it gives us a rare advance look at a future system and a chance to begin developing applications that will run as soon as the hardware is released.

This strategy clearly gives the Open Handset Alliance a big advantage over other cell phone operating system developers, because there could be an uncountable number of applications available immediately for the first devices released to run Android.

Introduction to Android

Android, as a system, is a Java-based operating system that runs on the Linux 2.6 kernel. The system is very lightweight and full featured.

Android applications are developed using Java and can be ported rather easily to the new platform. If you have not yet downloaded Java or are unsure about which version you need, I detail the installation of the development environment in Chapter 2. Other features of Android include an accelerated 3-D graphics engine (based on hardware support), database support powered by SQLite, and an integrated web browser.

If you are familiar with Java programming or are an OOP developer of any sort, you are likely used to programmatic user interface (UI) development—that is, UI placement which is handled directly within the program code. Android, while recognizing and allowing for programmatic UI development, also supports the newer, XML-based UI layout. XML UI layout is a fairly new concept to the average desktop developer. I will cover both the XML UI layout and the programmatic UI development in the supporting chapters of this book.

One of the more exciting and compelling features of Android is that, because of its architecture, third-party applications—including those that are “home grown”—are executed with the same system priority as those that are bundled with the core system. This is a major departure from most systems, which give embedded system apps a greater execution priority than the thread priority available to apps created by third-party developers. Also, each application is executed within its own thread using a very lightweight virtual machine.

Aside from the very generous SDK and the well-formed libraries that are available to us to develop with, the most exciting feature for Android developers is that we now have access to anything the operating system has access to. In other words, if you want to create an application that dials the phone, you have access to the phone’s dialer; if you want to create an application that utilizes the phone’s internal GPS (if equipped), you have access to it. The potential for developers to create dynamic and intriguing applications is now wide open.

On top of all the features that are available from the Android side of the equation, Google has thrown in some very tantalizing features of its own. Developers of Android applications will be able to tie their applications into existing Google offerings such as Google Maps and the omnipresent Google Search. Suppose you want to write an application that pulls up a Google map of where an incoming call is emanating from, or you want to be able to store common search results with your contacts; the doors of possibility have been flung wide open with Android.

**Q**: What is the difference between Google and the Open Handset Alliance?

**A**: Google is a member of the Open Handset Alliance. Google, after purchasing the original developer of Android, released the operating system under the Open Handset Alliance.

**Q**: Is Android capable of running any Linux software?

**A**: Not necessarily. While I am sure that there will be ways to get around most any open source system, applications need to be compiled using the Android SDK to run on Android. The main reason for this is that Android applications execute files in a specific format; this will be discussed in later chapters.

B.原文翻译

Android：一个程序员的指南

可以这么说，暂时，传统的桌面应用程序开发者已经被惯坏了。这个不是说桌面应用程序开发比其他形式的开发很简单。总之，作为传统的桌面应用程序开发者，我们必须有能力创造出各种应用程序凡是我们能想象到的。包括我自己，因为我也是从做桌面程序开始的。

一方面已经使得桌面程序更容易理解就是我们已经有能力去跟桌面操作系统相互作用，因此，任何底部的硬件很自由的相互作用。这种类型独立自主的程序编制，然而，对于很小的开发者团体来说是不敢冒险的去搞手机发展这样浑浊的技术的。

**嵌入式器件编程的简要历史**

有很长一段时间，手机的开发者由大的著名嵌入式的开发团队中的少数人组成，作为嵌入式设备的开发者。相对于桌面开发或者后续的网络开发，被视作更少“魅力”，而且嵌入式设备的开发通常因为硬件和操作系统而处于劣势。因为嵌入式设备的制造商们太小气，他们要保护他们硬件方面的秘密，所以他们给开发者们非常有限的库去运行当他们尝试去让一些特定的设备去相互作用。

嵌入设备与桌面系统显著不同的一部分是嵌入设备是个有特色的“芯片上的电脑”。例如：考虑你的标准电话遥控。这个并不是一个非常强大并且复杂性的技术。当任何的按钮被按下去，一个芯片解释一个信号以一种方式已经被编程进了这个设备。这个允许设备知道什么是从输入设备（键盘）来的需要。并且如何的响应这些命令（比如，打开电视机）。这个是一个简单的嵌入式设备的编程。总之，不管你相不相信，像这样的简单设备绝对的和早期的手机设备开发的根源有着紧密的联系。

大多数的嵌入式设备运行（有些仍然还在运行）在私有的操作系统。原因是选择创建一个私有的操作系统而不是用任何消费系统是产品的需要。简单的设备不需要非常健全和优化的操作系统。

作为一个产品的演化，更多复杂的嵌入式设备，如早期的PDA，家庭安全系统和GPS等。5年前某种程度上都转移标准的操作系统平台上。小的操作系统如Linux,甚至一个微软版本的嵌入式平台，已经在嵌入设备上变得普遍了。设备改革的这段时间里，手机从其他嵌入式设备中分支出去。走上了自己的轨道，这个分支是显而易见的当你去调查他们的体系结构。

在他们最初开始的时候，手机作为一个外围设备并且运行私有软件，而这些软件被制造商们所拥有和控制，而且几乎可以被认为是一个“关闭”的系统。习惯使用私有操作系统主要是制造商自己开发硬件，或者至少定义了开发的目的只是用来运行手机。最终的结果就是使开放成为不可能。现有的软件包或者解决方案会可靠的和他们的硬件交互。而且，制造商想要保护他们硬件的商业秘密。以防允许进入而发现设备软件的水准。所以风尚就是，而且大多数仍然是使用完全私有并且关闭的软件来运行他们的设备。任何人想为手机开发程序必须需要详尽的私有环境来运行软件的知识。而解决方案就是直接从制造商那里购买昂贵的开发工具。这就孤立了很多的“自制软件”的开发者。

让问题变得简单，就要在任何的设备让内存来存储并运行程序，包括手机。手机作为一个设备，直到最近还没有足够多内存来执行“额外”的程序。在最近的两年里，内存的价格已经达到了非常低的水平。设备制造商们有足够的能力压低价格来包含更多的内存。很多的现在的手机标准内存已经超过了90年代中期电脑内存。于是，现在我们有需求，而且有内存。我们可以直接跳到为手机开发酷的应用程序了，对吗？不完全是这样。设备的制造商们仍然紧密的保护他们的操作系统。有一些在手机上开放JAVA为基础的小运行环境。更多的是不允许。即使允许运行JAVA应用程序但还是不允许进入核心的系统。而这些是桌面开发者习惯于拥有的。

**开放手机联盟和Android**

这个对于应用程序开发的障碍开始在2007年的11月份被打破，当Google在开放手机联盟下发布Android。开放手机联盟是一个硬件和软件开发者的集合，包括谷歌，NTT DoCoMo, Sprint Nextel和HTC。他们的目标是创建一个更多的开放手机环境。在开放联盟第

一个被发布的产品就是移动设备操作系统Android。 对于这个Android的发布，谷歌使很多开发工具和向导成为可能来帮助在新

系统上可能的开发者。帮助系统，平台软件开发包(SDK)，甚至一个开发者的论坛，可以在谷歌的Android的网站上找到，这个网站应该是你的起点，而且我极度推荐你去访问。

当手机运行Linux, Windows或者即使Palm OS的手机是很容易找到，如本文所述，没有硬件平台已经宣告可以来运行Android.HTC，LG电子，摩托罗拉和三星都是开发手机成员，在Android的发布下，我们希望在不久的将来有一些Android为基的设备。在2007年11月发布时，系统自身还仍旧是一个测试版的程序。这是个对开发者的好新闻因为它给了我们一个罕见的提前看到将来的设备和有机会来开始开发应用程序，而当硬件发布时就可以运行。

**介绍Android**

Android，作为一个系统，是一个运行在Linux2.6核心上的JAVA基础的操作系统。系统是非常轻量型的而且全特性。

Android应用程序用JAVA开发而且很容易被放置到新的平台上。如果你没有下载JAVA或者不确定那一个版本你需要，我在第二章详细列出了开发环境的

安装。其他Android的特点包括一个加速3-D图形引擎（基于硬件支持），被SQLite推动的数据库支持，和一个完整的网页浏览器。如果你熟悉JAVA编程或者是任何种类的OOP开发者，你可能使用程序用户接口(UI)开发-那就是，UI安置是直接在程序代码中有句柄的。Android，识别并许可UI开发，而且支持新生，XML为基础的UI布局。XMLUI布局对普通桌面开发者是一个非常新的概念。我会在本书的相关章节里描述XMLUI布局和程序化UI开发。

Android另一个更令人激动和关注的特点是因为它的样式，第三方应用程序---包括“自制的”---会和系统捆绑的有着同样的优先权。这是和大多数系统不同之处，但是给了嵌入式系统程序一个比由第三方开发者创建的线性优先权大的优先执行权。而且，每一个应用程序在虚拟计算机上以一个非常轻量的方式按照自己的线路执行。

除了大量的SDK和成型的类库可以用之外，对激动人心的特性对于Android的开发者来说是我们现在可以进入到操作系统可以进入的地方。也就是说，如果你要创建一个应用程序打一个电话，你已经进入到电话的拨号盘。加入你要创建一个应用程序来使用电话内部的GPS（如果安装了），你已经进入了。对于开发者创建动态和令人好奇的程序已经敞开大门。

Android的一些特征和上面这些可用的特点相同，谷歌已经非常迫切的奉送一些特性。Android的开发者可以将自己的应用程序和谷歌提供的如谷歌地图和无所不在的谷歌搜索绑在一起。假设你要写程序在谷歌地图上显示一个来电话者的的位置，或者你要储存一般的搜索结果到你的联系人中。在Android中，这个门已经完全打开。

第二章开始你Android的开发旅程。你会学到如何和为什么使用特定的开发环境或者综合的开发环境（IDE)，而且你会下载并且安装JAVA IDE Eclipse.

问题:谷歌和开放手机联盟的区别在哪里？

答案:谷歌是开放手机联盟的一个成员。谷歌在收购了Android的原开发后，在开放手机联盟发布了操作系统。

问题:Android有能力运行任何的Linux软件吗？

答案:没有必要。但是我坚信会有一种方式绕开大多数的开源系统和应用程序用Android开发包编译而用于Android。主要原因是Android程序执行特定的文件格式，这会在以后的章节中讨论。