第一章 概 述

1.1 Java 嵌入技术的历史

要说 Java 嵌入技术，就要先说一下什么是嵌入技术。嵌入技术是一个简化的说法，严格说应该是嵌入计算技术。现在计算机技术发展很快，很多词的涵义也在变，因而嵌入计算这个词的含义也不止一个，本书所指的是目前最为普遍的含义，即指把计算机制引入各种嵌入设备，如电视、电话、数控机床、PDA(个人数字助理)等，从而使后者具有可编程的特性。因此，嵌入应用需要微处理器，在大多数情况下需要操作系统，并且操作系统一般是实时操作系统 RTOS，如本书将介绍的 CHORUS/OS，这种操作系统与一般计算机常见的操作系统如 Windows95/NT，UNIX 等是很不一样的，关于 CHORUS/OS 本书的第二章还有进一步的介绍。

相信大多数读者在看本书前一定对 Java 已经有了一个直观的了解，并且这种直观的了解一定来自 Java 在 Internet 上的应用。但本书说的是 Java 嵌入技术，这与你们以前所知道的，所理解的 Java 是不太一样的。Java 在 Internet 和 Web 中的地位是显然的，可以说是它们的首选语言，但 Internet 应用环境和嵌入环境是不一样的，为什么 Java 也可以适用于嵌入应用中呢？要说明这一点，需要从 Java 的起源说起。

80 年代末 90 年代初，有一种说法，就是在 90 年代，嵌入计算应用尤其是消费电子设备中的嵌入计算应用，将会有很大的发展。在 80 年代，先是苹果公司的 Apple 微机，后是 IBM 的 PC 及其兼容系列微机取得了相当大的成功。微机的这种成功的根本原因就是随着微电子技术的发展，CPU 的性能越来越高，存储器的容量越来越大，同时它们的价格却越来越低廉。微机的这种成功使人们不禁想到随着技术的进一步发展，还有比微机更小的“计算机”也将获得成功，这就是各种嵌入设备尤其是消费式电子设备。这种说法不见得完全正确(这一点本书的最后一章还会做进一步的讨论)，但在当时却引起了一些公司的注意。Sun 公司就是其中的一个。

在 90 年代初，Sun 在 UNIX 工作站市场已取得了相当大的成功，但 Sun 公司却不满足于此，它把目标放在了其他与计算机技术相关的领域。1991 年，正是看到了嵌入计算应用市场的广阔前景，Sun 公司的由 James Gosling，Bill Joe 等人组成的 Green 小组开发了一个名为 Oak 的软件(这是 Java 的前身)，开发它的目的是用于电视等家用电器的嵌入式应用。但后来，正如我们现在知道的，Java 的发展却出乎它的创造者的预料，它并没有在嵌入应用中大展身手(起码，目前是这样)，却风靡于 WWW 世界。现在，人们提到 Java 总是联想到 Web 和 Internet。

1995 年 5 月，Sun 在 SunWorld 上正式发布了 Java 和 HotJava 浏览器。1996 年 Java 赢得业界的大力支持，Java 应用软件纷纷面世，Java 被评为当年的十大科技成果之一。1997 年 JDK1.1 发布，国际标准化组织批准 Sun 公司作为 Java 公开有效规范(PAS)提案者的申请。Java 的这一系列发展，似乎都只与 Internet 和 Web 相关。但是 Sun 公司在做出了把原先目标在嵌入应用的 Java 转而用于 Internet 和 Web 的决定并取得了巨大的成功之后，并没有放弃

其在嵌入计算中的应用。正好相反，Sun 一直声称 Java 不但要用于 Internet，还可以在嵌入应用中大有作为。笔者现在还记得在自己 1996 年刚接触 Java 时，总是看到 Sun 公布的资料中说什么不但浏览器中有 Java，微波炉中也会有 Java，这让当时的我感到极大的兴趣——微波炉中怎么会有 Java 呢？有了 Java 又会有什么用呢？希望读者也有很大的兴趣关心这样的问题，这将使你饶有兴趣地看这本书。

Sun 公司在大力开发 Java 在 Internet 中的应用技术时，丝毫没有放松 Java 嵌入技术的开发。早在 1996 年初，Sun 公司就公布了 Java 芯片的初步资料，同年底公布了 JavaCard 1.0。1997 年底又正式公布了 PersonalJava 和 JavaCard 2.0 并兼并了从事嵌入式实时操作系统开发的 Chorus 公司。1998 年初公布了 EmbeddedJava 的初步资料。就在本书完稿时，各项 Java 嵌入技术的开发仍在积极进行中：Sun 正在为 1998 年底 MicroJava701 芯片的大批量生产做准备；Gemplus 正在开发采用 picoJava 核心的 32 位新一代智能卡；PersonalJava 正准备运用于 Windows CE 中……

如今，一股关于 Java 应用新的浪潮正悄然兴起。这就是本书将介绍给你的 Java 嵌入技术。图 1.1 是三星公司的与 PersonalJava 平台兼容的 Web 电话。这种电话带了一个简单的显示器和一个键盘，它可以收发电子邮件，浏览万维网等。

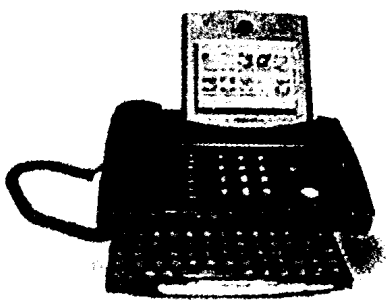

图 1.1　三星公司的 Web 电话

1.2　Java 语言特点

相信读者中有不少人对 Java 语言本身已经相当了解了（如果你就是这种读者，那么可以不用读本节），但笔者希望本书有一个较广的读者范围，所以仍需介绍一下 Java 语言的特点。

Java 如此广受欢迎不是像有些人说的那样纯粹是商业宣传的结果，而是有其内在的原因。下面就是其特点：

1. 简单

有人说从语法形式上讲，Java 语言是一种简化的 C++，这种说法不无道理。Java 略去了 C++ 中的繁琐的运算符重载，多重继承等模糊的概念，并且通过实现自动垃圾回收大大简化了程序设计者的内存管理工作且减少了在内存管理上出错的可能。作为一种面向对象语言，Java 通过提供最基本的方法来完成指定的任务，只需理解一些基本的概念，就可以用它编出适合于各种情况的应用程序。

2. 短小

Java 不但语法形式简单,实现起来也简单。它的基本解释器及其类的支持只有 40KB 左右。加上标准类库和线程的支持也只有 200KB 多。一个同样的程序,用 Java 一般要比 C++等语言短小得多(指编译后)。出现这种情况的原因是 Java 最初准备用于资源极为有限的嵌入设备,开发它时特别注意它的可执行代码的短小,比如 Java 虚拟机以堆栈机为中心,省去了大量的内存操作,同时 Java 的每条字节码都只有一字节,这些都使得 Java 可以在十分有限的系统资源下得以很好地实现,这一点对嵌入应用很有价值。

3. 面向对象

Java 语言的设计集中于对象及其接口,它提供了简单的类机制以及动态的接口模型。对象中封装了它的状态变量以及相应的方法,实现了模块化和信息隐藏。而类提供了相应对象的原形,并且通过继承机制,子类可以使用父类所提供的方法,实现了代码的复用。代码复用是嵌入计算编程一直追求的东西。

4. 取消指针

没有指针的高级语言不只 Java,但 Java 与 FORTRAN77 这样的没有指针的语言不同,它是在可以完成有指针的高级语言所能实现的各种功能的情况下"取消"了指针。Java 中的引用概念使得它可以在没有指针的情况下实现对数据结构的各种复杂操作,Java 的内存垃圾自动回收使编程人员在没有指针的情况下照样可以高效地使用内存。

笔者认为,在面向对象时代,指针已无存在的必要。C++因必须和 C 语言保持兼容不可能取消指针,但 Java 做到了。读者在学习 C 语言或 Pascal 时可能被告知指针是如何的有用,但技术的发展实在是太快了,指针因其太易出错在面向对象时代可以被取消。从没有指针到有指针再到没有指针,经过了一个否定之否定的过程,Java 语言是这一发展过程的结果。

5. 平台无关

关于 Java 的平台无关性也许有的人有一种误解,把 Java 的平台无关性归结于它与 BASIC 等语言一样是解释执行的语言,这种说法并非完全正确。Java 与 BASIC 这样的传统的解释执行语言是不同的。任何 Java 程序都被编译成 Java 字节码,Java 字节码与其他一些程序执行信息组成了 Java 类文件(.class 文件)。字节码可以被看成与汇编语言类似的东西,但是它经过特殊设计特别适用于用软件解释的方式加以执行。

由于 Java 字节码可以被软件解释执行,这样只要安装了 Java 运行的软件系统,Java 程序就可以在任意的处理器上运行。这些字节码指令对应于 Java 虚拟机中的表示,Java 解释器得到字节码后,对它进行转换,使之可以在不同的平台上运行。当 Java 字节码被软件解释器解释执行时,字节码本身携带了许多编译时的信息,使得连接过程更加简单。

对于类似汇编语言的 Java 字节码,只要稍加扩充,就可以成为某种微处理器的指令系统。这种微处理器就是 Java 芯片,本书第五章将详细介绍。

6. 鲁棒性

Java 在编译和运行程序时,都要对可能出现的问题进行检查,以消除错误的产生。它提供自动垃圾收集来进行内存管理,防止程序员在管理内存时容易产生的错误。通过集成的面向对象的例外处理机制,在编译时,Java 提示出可能出现但未被处理的例外,帮助程序员正确地进行选择以防止系统的崩溃。另外,Java 在编译时还可捕获类型声明中的许多常见

错误，防止动态运行时不匹配问题的出现。

7. 分布性

Java是面向网络的语言，通过它提供的类库可以处理TCP/IP协议，用户往往可以通过URL地址在网络上很方便地访问其他对象。读者也许会问，本书说的是Java嵌入技术，还说什么分布性呢？实际上不能把Java在嵌入设备中的应用和其在网络上的应用分割开来，很多嵌入应用都有联网要求，如Web电话，顶置盒等。关于计算网络化和计算嵌入化的关系问题，本书后面的章节尤其是最后一章还有进一步的探讨。

8. 多线程

多线程机制使应用程序能够并行执行，而且同步机制保证了对共享数据的正确操作。通过使用多线程，程序设计者可以分别用不同的线程完成特定的行为，而不需要采用全局的事件循环机制，这样就很容易地实现网络上的实时交互行为，同时对嵌入设备的实时控制也很有好处。

9. 动态性

Java的设计使它适合于一个不断发展的环境。在类库中可以自由地加入新的方法和实例变量而不会影响用户程序的执行。Java通过接口来支持多重继承，这样做确实有些不足，但也使之比严格的类继承具有更灵活的方式和扩展性。

10. 可移植性

与平台无关的特性使Java程序可以方便地运行在网络上的不同计算机和不同的嵌入设备。但这同时也要求Java的运行系统可以移植到各种硬件环境，实际上Java的运行系统和很多开发工具本身的很大一部分是用Java语言写的，剩下的部分也是用标准C语言实现，这使得Java系统本身也具有较强的可移植性。本书中介绍的Java嵌入应用开发工具也有一些自身是用Java写成的。

11. 安全性

大家知道，对于网络环境中的Java来说，安全性是很重要的，必须确保病毒不会通过网络上的Java程序来进行传播。实际上，嵌入应用也要求Java具有很好的安全性，比如智能卡，一旦安全性出了问题，后果不堪设想。

幸运的是Java天生具有完备的安全保证机制。首先，Java语言不支持指针，一切对内存的访问都必须通过对象的实例变量来实现，这样就防止了程序员使用“特洛伊”木马等欺骗手段访问对象的所有成员，同时也避免了指针操作中容易产生的错误对安全的破坏。其次，Java还有严格的安全性检查，一是在编译时就有安全性检查；二是在解释Java字节码时有严格的安全检查，一些完好的通过编译的Java程序如“图谋不轨”照样会被禁止执行。

1.3 Java虚拟机简介

1.3.1 什么是Java虚拟机

读者在本书以后的章节中将大量接触到Java虚拟机的概念。这是因为本书不是仅仅介绍Java嵌入技术的编程，而是想把Java运用到嵌入设备中时的整个系统构成做一些介绍。这就必然要经常讲到Java虚拟机。要想较好地弄明白本书所有内容就起码需要明白

Java 虚拟机是什么。Java 虚拟机是什么？这就是本小节要说的。

Java 虚拟机就是用来运行经过编译的 Java 字节码的计算机实现，这种实现既可以是软件实现也可以是硬件实现。从 Java 应用程序角度来看，Java 虚拟机是一个假想的计算机平台，Java 应用程序运行在 Java 虚拟机这个假想计算机之上并通过它获得所需的本地资源。从计算机或嵌入设备角度来看，Java 虚拟机是建立在处理器等硬件设备基础之上的“上层建筑”，在本设备上运行的所有的 Java 应用都通过它执行。

Java 虚拟机的实现方案有两种。一是纯软件仿真，正是大家目前所接触的形式；其二就是用 Java 芯片了，但用 Java 芯片不是说虚拟机中就没有软件组成了，仍然有一部分需要用软件实现。

要了解 Java 虚拟机，就必须了解 Java 程序生成实际机器可执行代码的过程。许多程序设计语言通过源程序进行编译和链接，直接生成可执行的代码。而各种 Java 程序，无论是标准的 Java 程序，还是本书将介绍的专门用于嵌入应用的 Java 程序则和它们不同，Java 应用程序的这个过程包括程序的编译，字节码的装入，校验，解释或编译。关于 Java 程序执行的这一系列过程见图 1.2。

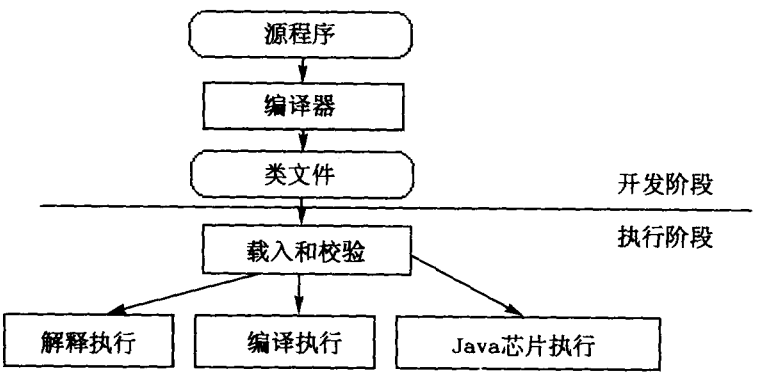

图 1.2　Java 程序生成实际机器可执行代码的过程

首先是对 Java 程序的编译，以生成字节码。这个过程由程序开发人员用的 Java 编译器完成。前面说过，字节码的层次相当于汇编语言一级。但是它经过了专门的设计以有利于软件仿真的方式实现，并不针对某种特定的计算机硬件平台。需要注意的是，在这一步中，对变量和方法的引用，不是确定为数值引用，即通过具体的偏移量的引用，而是将符号引用的信息保存在字节码中。

通过编译后，就意味着完成了 Java 程序的开发，接下来就是 Java 程序的执行阶段了。

在执行阶段中，先是进行字节码的装入，校验，由本地计算环境上的相应软件完成——无论是否用 Java 芯片。类装入器装入程序的所有代码，包括程序中调用，包含和继承的所有类的代码。每个类都被装入一个独立的名字空间内，彼此之间只有通过符号引用才能相互作用。本地的类和外部的类在地址空间上是分开的。所有的类都装入以后，可执行代码的内存布局就被确定。由符号引用到内存地址空间的查询表也建立起来了。然后字节码校验器对装入的字节码进行校验，以排除错误和不安全的因素，如果发现了安全性上的问题，将退出 Java 程序的执行过程。

在通过了装入和校验之后，最后就是 Java 字节码的运行了。这时有三种方式，一是传

统的方法,即由解释器解释执行。但是解释执行速度慢,于是有人发明了第二种方式——编译执行。要说明的是,这时的编译的含义与通常所说的编译不一样,实际上 Java 程序在准备执行前已经通过了编译(通常意义上的编译)成了字节码,但执行时的编译指的是把 Java 字节码一次性地翻译成本地计算环境的机器代码然后执行,而不是像解释器那样边解释边执行。第三种方式也是最快的一种方式就是用 Java 芯片直接执行了,这种方式不但速度快,并且省内存,因为它省去了一个软件实现的解释器或编译器。

1.3.2 Java 虚拟机与平台无关性

Java 虚拟机是 Java 的平台无关特性得以实现的关键。如图 1.3 所示,Java 平台结构中 Java 虚拟机起着承上启下的作用。

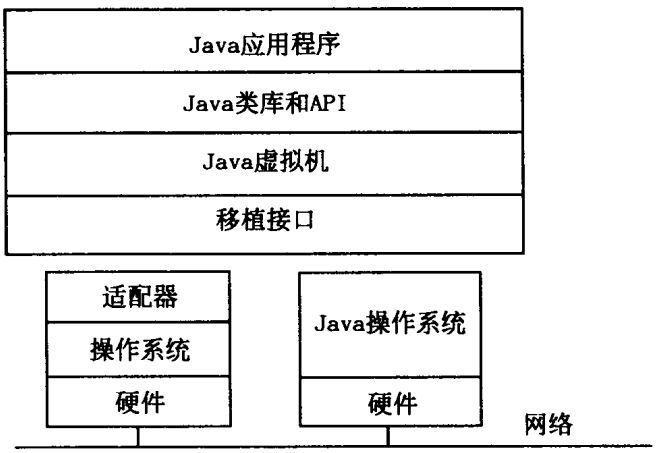

图 1.3 Java 虚拟机与 Java 平台

在 Java 虚拟机的下方是移植接口,移植接口由依赖于平台的和不依赖于平台的两部分组成,其中依赖于平台的那部分为适配器。Java 虚拟机通过移植接口在具体的操作系统上实现。如果是在 Java 操作系统上实现,则不需要依赖于平台的适配器,因为这部分工作已由 Java 操作系统完成。因此,对于 Java 虚拟机来说,操作系统和更底层的硬件是透明的,即对于 Java 虚拟机这一层次来说,操作系统和硬件好像不存在一样,因此也无需考虑。

在 Java 虚拟机上方是 Java 应用程序接口,它由 Java 基本类和 Java 标准扩展类组成。我们编写的 Java 应用程序就运行在 Java 的类库和 API 上,因而对于它们来说,操作系统和硬件就更是透明的了。我们编写的 Java 程序因此可以在任何 Java 平台上运行而无需修改。

Java 虚拟机定义了独立于平台的类文件格式和字节码形式的指令集。在任何 Java 程序的字节码表示中,变量和方法的引用都是使用符号,而不是使用具体的数字。用数字引用替代符号引用是在运行时由解释器完成的。当第一次引用一个变量或方法的时候,用它的名字在查找表中查找,以确定它的数字偏移量。这种查找和替换只在第一次引用时需要,因此对速度影响不是很大。由于存储器的布局要在运行时才确定,所以对类的变量和方法的改变不会影响现存的字节码。例如,某个 Java 程序引用了其他系统中的某个类,该系统中那个类的更新不会使你的程序崩溃。如果是 C++,你必须将你的程序中相关部分再次编译。Java 的这种特点不仅提高了 Java 的平台独立性,同时也使得嵌入应用中的远程控制成为

可能。

1.3.3 Java 虚拟机与安全性

前面说过,无论是对于网络环境,还是嵌入计算环境,安全性是必须要求的。Java 虚拟机在 Java 的安全体系中占有重要地位。至少有以下三项安全性是与 Java 虚拟机相关的。

其一是内存的布局。我们说过,在任何 Java 程序的字节码表示中,变量和方法的引用都是使用符号,而不是使用具体的数字。用数字代替符号引用的工作由解释器或编译器在运行时完成。对于编程人员来说,内存分配是透明的,不可能在编写程序的时候就知道内存的布局。编程人员无法通过伪造指针来访问内存空间。

其二是字节码检查。虽然安全性检查在 Java 源程序编译成字节码时就已经进行过,但这还不够。当从其他地方下载 Java 字节码时,并不能认为它是安全的。因为有可能是生成这些字节码的编译器本身有问题,比如有人故意做了一个不合规范的 Java 编译器,以侵入其他用户的系统。所以必须对字节码进行检验。字节码的一个重要性是它的可解析性很强,能够对指令集进行分析,并对其以后的行为做出一定的推断,这就为检验打下了基础。通过检验的程序,可以确保不存在伪造的指针,不违反访问权限,不非法访问对象,不会导致操作数栈溢出等违反系统安全性的操作。

其三是字节码装载器的安全检查。因为字节码采用符号引用,内存布局在运行时确定,所以可以在字节码装入时将本地的类和外来的类严格区别开来。本地的类共有一个统一的名字空间。这种机制为本地的类建立起一道安全屏障。

虽然标准的 Java 虚拟机已有很高的安全性,但嵌入设备产商还不满足。法国的一家智能卡产商开发了"安全 Java 虚拟机"SJVM。这是一种用于对安全性要求特别高的嵌入设备中的 Java 虚拟机,它在标准的 Java 虚拟机的基础上,又进一步增加了强大的安全保护机制,不但在字节码载入时有安全检查,在执行时还对 Java 字节码边执行边检查,确保不执行破坏操作。

1.4 Java 操作系统

1.4.1 什么是 Java 操作系统?

读者一般以前见过"JavaOS"这个词。首先要说明的是,严格来说"JavaOS"和"Java 操作系统"这两个词并不是完全相同的。顾名思义,Java 操作系统指的是专门为运行 Java 应用程序设计的操作系统。而 JavaOS 则是 Sun 公司的一个产品的名称,是一种 Java 操作系统。除了 Sun 开发的 Java 操作系统之外,其他公司如 IBM,日立等也在开发 Java 操作系统,如 JavaOS 等,你是不是想起了"金庸"和"全庸"?

本书讲的是 Java 嵌入技术,为什么要介绍 Java 操作系统呢? 原因很简单,嵌入应用中用到 Java 操作系统的可能性要比在计算机中用到它的可能性来得大。在计算机领域,没有 JavaOS,我们照样可以在其他操作系统上用软件实现的 Java 虚拟机运行 Java 应用程序。在嵌入计算领域,系统资源十分有限,我们虽然也可以在任一嵌入设备用的操作系统上用软件实现的 Java 虚拟机来运行 Java 应用程序,但这会消耗宝贵的系统资源,这时用 Java 操作系

统就很有必要了。此外,嵌入应用中 Java 操作系统广阔的应用前景也和信息产业的现状有关,这一点本书最后一章再做详细说明。

虽然 Java 操作系统不仅仅指 JavaOS,但是 JavaOS 是目前最有前途的 Java 操作系统。本书以它为例来介绍一下 Java 操作系统。

Java OS™是为使 Java™能够在各种计算机和嵌入设备平台上运行优化而设计的一种新的平台。为使 Java 应用程序能够直接在硬件平台上运行,而不需要一个像 UNIX, Windows 95/NT 这样的主操作系统,JavaOS 特别提供了一种运行时环境(runtime)。JavaOS 提供 Java 平台以自动运行利用 Java 语言设计的功能强大的 Applet 程序及其他应用程序。同样地,它亦提供 Java 虚拟机和低层功能以实现窗口、网络以及文件系统功能,而无需任何主操作系统的支持。

JavaOS 由两部分组成。一是本地代码部分,具有特定的指令集和硬件平台;二是 Java 代码部分。其中 Java 代码具有平台独立性。JavaOS 定义的平台包括 CPU、物理内存、其他附加设备、总线和插槽等构件。操作系统的平台独立性组件被称为 Java 运行时环境,而操作系统中依靠平台的部分则可称为 JavaOS 内核。JavaOS 的组成见图 1.4。

经过设计 JavaOS 可以在一台硬件条件非常有限的平台上运行,这对嵌入应用很重要。例如,运行时本身既不需要内存管理单元 MMU(Memory Management Unit)把虚拟地址映象到物理内存地址,又不需要内存保护。而是由其底层的内核决定是否使用 MMU 或强制执行内存保护。再例如,JavaOS 的内核使用内存管理单元 MMU 来建立一些接近 Java 运行时环境但相互隔离的物理内存区域。未来的 JavaOS 也许还能以一种更为积极的方式来管理 MMU。

JavaOS 被建构成一个层状结构,每一层都以独立的形式存在。这种结构可以满足两种需求:即产品定制工作和一个并行操作系统的模型。

基于产品定制的 JavaOS 就是为了满足产品的限定和要求而集合 JavaOS 的几个层。比如说,运行 JavaOS 的智能型电话需要一个实时内核、Java 虚拟机、最基本的图形能力和一些通讯协议。而一台网络计算机则需要一个没有实时约束但拥有更多也更完整特性的内核、Java 虚拟机、抽象窗口工具集 AWT(包含完整的图形库)以及所有其他的在 Java 开发包 JDK 之内的应用程序接口 API 和 HotJava™浏览器。

图 1.4 展示了在不具备一种主操作系统的前提下运行 Java 平台时所使用的软件体系结构,Java 应用程序接口 API 之上的程序是具有平台独立性的 Java 应用程序和 Applet 程序。

JavaOS 是否真的是一个操作系统? 这完全依赖于用户自己的观点。JavaOS 与传统的操作系统在许多方面截然不同,这些方面包括:

- 需要一个文件系统;
- 需要一个虚拟存储器;
- 需要相互独立的地址空间;
- 支持多种程序语言;
- 有自己的系统调用设置。

JavaOS 与一个传统的操作系统在一些方面亦有类似之处,这些方面是:

- 可以自引导;
- 支持口令保护的登录特性;
- 可以安全地同时运行几个 Applet 程序;

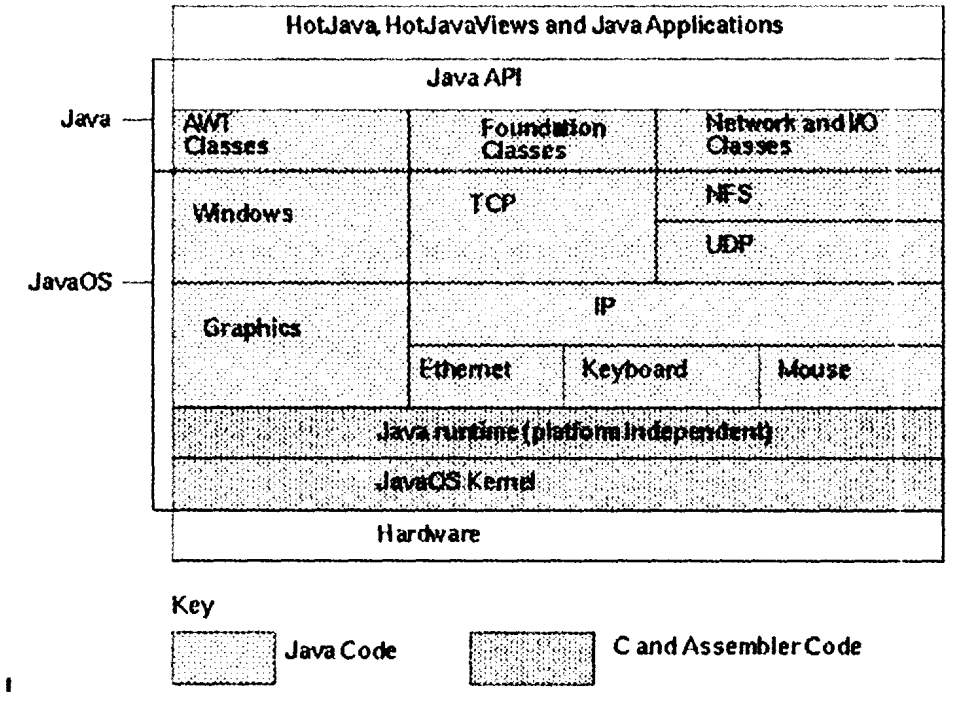

图 1.4　在 JavaOS 上运行的 Java 平台

- 包含一些设备驱动程序；
- 用许多标准网络协议来进行通讯；
- 有自己的窗口系统；
- 有一个应用程序接口(API)；
- 可以运行成千上万的已写好的 Applet 程序和一般的应用程序。

1.4.2　JavaOS 的内存管理机制

JavaOS 并不需要一个内存管理单元 MMU，但它可以通过使用一个 MMU 来让几个在物理地址上相互隔离的内存空间变得相互邻近，以此简化内存分配。Java 程序设计语言把所有的设备访问权封装进对象、类和自动的空闲内存收集器里，由此消除了用户对内存的直接操作。在 Java 语言中不直接使用指针，这使 Java 成为一种比 C 语言更为强健的程序设计语言，并大大降低了与内存有关的错误的数目。

JavaOS 提出了一个可移植的、抽象的内存模型。这个内存模型建立在寻址的概念基础上，而寻址是指确定存储位置的过程。在 JavaOS 中，最基本的寻址单元就是物理内存地址(一个物理地址确定了在内存中一个唯一的物理位置)。物理地址并不受内存管理单元 MMU 的影响；它可以说是个完全意义上的内存位置，其作用是对 ROM，RAM 和输入/输出存储器空间进行定义。

JavaOS 的虚拟地址空间通过 JavaOS 内核层建立。需要注意的是，虚拟这个术语并不一定就意味着页转换。实际上，虚拟这个术语表示这个地址空间被所有软件所使用，并也许与物理地址空间没有什么相似的地方。JavaOS 并不承担在物理地址空间和虚拟地址空间之间

的一一对应关系。而内存管理单元 MMU 的一个作用在于增强对虚拟地址内存空间的支持,对于 JavaOS 来说同样不承担此项任务。

与当今许多正在使用的操作系统不同,JavaOS 并不在假设存在多种虚拟地址空间的环境下运行,而是运行在一个单独的虚拟地址空间中。

1.4.3 设备驱动程序

JavaOS 中所有的设备驱动程序都是用 Java 程序语言编写而成。这对于程序的可移植性是非常重要的。

有些工作是几乎每个驱动程序都需要做的,但却无法用纯 Java 代码写成;这些事情都被抽象化到两个用 C 语言写成的小的支持类中。类 Memory 允许驱动程序访问和修改存储的特定代码和字(word);类 Interrupt 则处理中断调度。任何 Java 应用程序都无法直接调用这些类的方法。

现在,Sun Microsystems 公司已开发了一些不同的 Java 驱动程序类,并正在开发更多的驱动程序以支持 Java 芯片、SPARC™和 X86 的硬件。此外,Sun 的工程师还正在酝酿定义一个 Java 接口以允许第三方的开发者设计可供下载的设备驱动程序。

如果你编写过设备驱动程序,你就会体会到 JavaOS 用 Java 编写设备驱动程序是多么美妙的一件事情。一般编写设备驱动程序是一件麻烦事,而用简单易用的 Java 可以减小工作难度。更重要的是,一般的硬件产商总希望自己的产品可以适用于不同的平台,这在以前就需要编不同的设备驱动程序,用平台无关的 Java 就可大大降低开发强度。

1.4.4 网络协议组

JavaOS 包含了一套网络协议,并全部用 Java 程序语言写成。这些协议包括符合 TCP,UDP,IP 和 ICMP 标准的基本传输和路由选择机制。JavaOS 则用 DNS 和 NIS 来寻找主机名及提供登录时所需的用户名字和口令。

JavaOS 支持反转地址解析协议(Reverse ARP)和 DHCP 以发现一个设备的网络地址。这样便使得我们可以安装 JavaOS 机器,并可以减少对机器的管理工作。

运行 JavaOS 的机器可以以网络文件系统(Network File System)服务器客户机的身份访问文件,还可以通过使用简单网络管理协议(Simple Network Management Protocol)对 JavaOS 进行管理。运行 JavaOS 的机器可以从一台网络服务器中获取时间,这就简化了 JavaOS 的安装和管理工作。

1.4.5 JavaOS 的优点

用 JavaOS 会有许多好处,这些好处体现为在硬件上直接提供 Java 平台。JavaOS 达到了消除一个主操作系统的额外开销的目标。因为 JavaOS 不包含在其他操作系统中所存在的外来特性,它允许建立更小和更为简单的设备,这些设备运行起 Java 程序来要比其他的系统更有效率。

- JavaOS 对内存的需求少。

支持 JavaOS 需要多少内存? 答案是:用 4MB 的 ROM 和 4MB 的 RAM 就可以建立起一个完整的系统。

ROM 中将存放所有 JavaOS 本身所需的所有代码，包括内核代码、驱动程序、Java 虚拟机和标准类，还有 JavaOS 窗口、图形和网络组件，还加上 HotJava [redacted]。ROM 中还包含了将近 1MB 的组合了所有不同种类的字体位图，包括衬线、虚线、打字机字型、一些点的尺寸和不同风格的字体(如黑体和斜体)。

如果 ROM 正常运行，则系统就可以使用 2.5M 的 RAM 以满足 JavaOS 和 HotJava 的动态需求，还有 1.5M 的 RAM 用在下载的 HTML 网页，Applet 和图像上。如果系统使用的 JavaOS 中不包括窗口和 HotJava 代码，则只需少于一半的内存空间。

- JavaOS 可被存在 ROM 中，并允许简单而低消耗的系统快速地启动。
- JavaOS 用 Java 语言编写，这样，新的组件就可以快速地开发出来，因为 Java 代码更易于调试，有本身就固有的可移植性，并具有动态扩展性。
- JavaOS 允许系统的安装和维护既能和终端的安装维护同样简单，又有传统的桌面机器一样的强大功能。一般说来拥有一台计算机所需的耗费中最昂贵的部分大概就要算是计算机的配置和维护了。而与一台典型的个人计算机相比，JavaOS 能够戏剧性地降低这方面的开销。
- JavaOS 高度的可配置性。

如果一台设备只有 1 或 2M RAM 和 1 或 2M 的 ROM，也许连图形显示功能都没有，但你还想在这台设备上加载和运行 Java 应用程序，该怎么办？JavaOS 可被'量身定做'以适合特定的设备使用，像顶置盒(set - top boxes)，个人数字助理(PDA)和没有任何图形显示功能的电子设备等。需要注意的是这些并不表示我们可以以语言本身作为子设备或是删除语言、工具类中的某些特性。但是，如果这台设备确实没有显示功能，我们就不仅可以删除抽象窗口工具集(AWT)，还可以把窗口和图形代码也从 JavaOS 中删掉。

类似地，如果一台设备不需要某些网络协议，这些协议就可以被剔除。为了满足一些嵌入式设备的需求，我们仍然要保留 Java 虚拟机和垃圾收集器以支持一些软实时功能。

总之，JavaOS 是一个新的软件平台，它允许 Java 应用程序在硬件设备上直接运行而无需一个主操作系统。

JavaOS 包括 Java 虚拟机、类的标准包和正好足够的操作系统代码作支持。而 OS 代码又包括用 C 语言和汇编语言写成的低级代码，还有大量用 Java 程序语言写成的设备驱动程序，网络、窗口和图形变换的代码。

JavaOS 的主要优势在于消除了主操作系统的额外开销和复杂性，它使各种新的简单、智能化、动态和低消耗的网络设备的实现成为可能。JavaOS 将被应用在许多系统之中，这些系统包括作为企业桌面的 Intranet 终端，适宜作网络冲浪的客户端 Internet 计算机和硬件资源更为苛刻的嵌入式设备等。

第二章　Personal Java

2.1　概　述

2.1.1　什么是 Personal Java?

Personal Java 是一个新的 Java API 和 Java 程序运行环境。它主要用于有联网要求的消费式电子系统，如顶置盒、专用游戏机、手持电脑、Web 电话、PDA(Personal Digital Assistants)等，这些消费式电子系统在今后几年中将有很大需求。

和标准 Java 类似，Personal Java 也由一个 Java 核心和扩展类库组成。它运行在 Java 虚拟机上，可用于各种不同的操作系统和处理器。Personal Java 的虚拟机与一般的 Java 虚拟机完全相同，没有删节。Personal Java 支持一个高级的图形用户接口，支持下载和执行 Java Applet。

从图 2.1 和图 2.2 中可以看到 Personal Java、Embedded Java 的系统结构和标准 Java 的系统结构的区别。

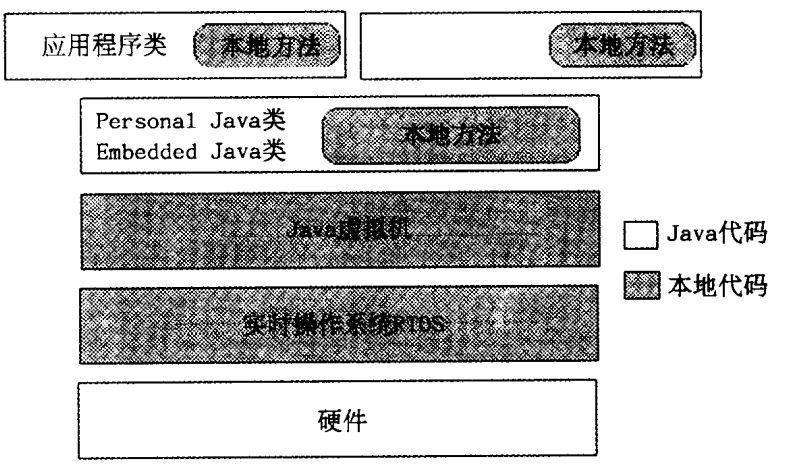

图 2.1　采用 Personal Java 或 Embedded Java 的系统结构

Personal Java 使得那些为专用游戏机那样的消费式电子设备编写嵌入式应用程序的程序员也可以享有 Java 的种种优点：平台无关，安全性，面向对象，多线程等。它使得为消费式电子设备编程不再需要专门的程序员，一般的 Java 程序员经简单的学习(例如阅读一下本章内容)就可以胜任。

1997 年发布了 Personal Java 1.0 规范的草案，并在该年底推出正式版本。本书以此为标准介绍 Personal Java。

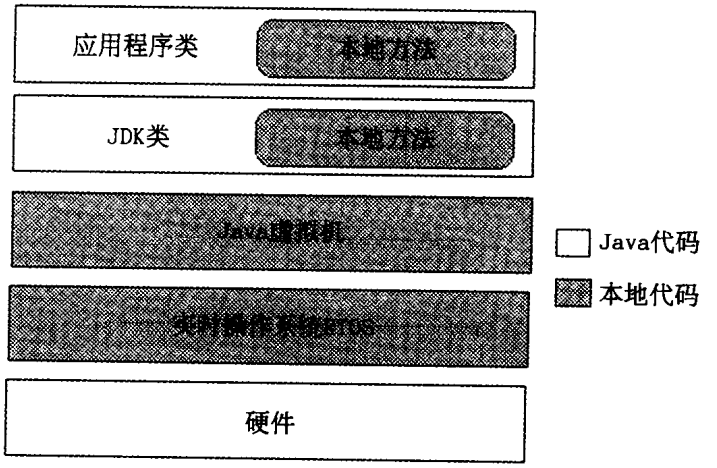

图 2.2　采用标准 Java 的系统结构

2.1.2　Personal Java 的设计目标

Personal Java 1.0 达到以下目标：

1. 对硬件环境的要求比较宽松。我们知道，像顶置盒、智能电话等那样的嵌入式设备对软件的要求一般不像 PC 那样高。同时，它们所能提供的硬件环境也不比 PC 来得强大。而 Personal Java 比标准 Java 小得多，对硬件的要求也少得多。

Personal Java 1.0 对硬件的要求如下：

- 0.5 ~ 1M 的 RAM
- 2M 的 ROM
- 一个主频大于 50MHz 的 32 位处理器

2. 对标准 Java 保持向上兼容，即为 Personal Java 编的 Applet 可用支持 JDK1.1 的浏览器浏览。
3. 可与 JDK1.1 的程序包保持动态编联兼容。即一旦运行在支持 JDK1.1 的运行环境中，PersonJava1.0 的 Applet 可拥有 JDK1.1 的 Applet 所特有的特征。
4. 为适应各种嵌入设备的要求，Personal Java1.0 可处理多种输入输出方式。如游戏杆、触摸屏、电视输出、远程控制等。
5. 用 Personal Java 开发的产品要能让那些没有计算机使用经验的人方便使用。
6. 在 Personal Java1.0 环境中能运行的 JDK1.0.2 和 JDK1.1 程序要有相当大的比例。
7. Personal Java 是可选配的，它的很多模块可根据嵌入设备的不同要求选择使用。
8. 用 Personal Java 开发程序将是简单的。这里的简单包括两方面，一是开发者可用现在众多的简单易用的 Java 开发工具，如用 Visual J + + , Visual Cafe Pro 等为 Personal Java 写程序。二是一般的 Java 程序员经过简单学习后就能掌握 Personal Java 的编程。

2.1.3　Personal Java API

Personal Java API 1.0 基本上是 JDK1.1 的一个子集，但仍然有一些 Personal Java 的 API

是 JDK1.1 中没有的，还有一些 API 在 JDK1.1 中虽有，但在 Peraonal Java 中的功能已发生变化。1.1 以后版本的 JDK 中新的 API 只有在经过考虑后才会被加入到 Personal Java 中。

JDK1.1 的 API 可根据其与 Personal Java 的关系分为四种：

1. 需要的（“Required”），即完全支持。
2. 不被支持的（“Unsupported”）。
3. 可选配的（“Optional”）。
4. 已改变的（“Modified”），即该 API 在 Personal Java 中虽有但功能发生了改变。

在 JDK 的 24 个程序包中，在 Personal Java 中有下列 13 个：

- java.Applet
- java.awt
- java.awt.datatransfer
- java.awt.event
- java.awt.image
- java.awt.peer
- java.beans
- java.io
- java.lang
- java.lang.reflect
- java.net
- java.util
- java.util.zip

在上述 13 个程序包中，java.awt，java.net 和 java.util，java.util.zip 四个程序包中的一些类在 Personal Java 中已有不同——或者不被支持，或者是可选配的，或者是被改变的。而剩下的七个程序包中的所有类在 Personal Java 中都被支持。

下述三个程序包是可选配的：

- java.math
- java.security
- java.security.interfaces

下述 8 个程序包不被支持：

- java.rmi
- java.rmi.dgc
- java.rmi.registry
- java.rmi.server
- java.security.acl
- java.sql
- java.text
- java.text.resources

去掉这些程序包可使 Personal Java 节省大量内存。

表 2.1 中列出的是 Personal Java API，这个表只列出了 Personal Java 和标准 Java 不同的部

分。大家在阅读本章第二、第三、第四、第五节内容时应该参考本表。

表 2.1 Personal Java API(部分)

函 数	状 态	备 注
CheckboxMenuItem( )	可选配	该构造方法是不要求的,假如调用也许引起 java.lang.UnsupportedOperationException 例外。但如 Menu 类的构造方法不会引起该例外的话则此方法也不会引起,反之亦然。
CheckboxMenuItem(String)	可选配	同上。
CheckboxMenuItem(String,boolean)	可选配	同上。
Component.setCursor(Cursor)	已改变	指定的光标也许会被忽略,许多消费式电子设备都没有光标,或者其拥有的光标的种类不全。
Dialog(Frame)	已改变	该构造方法将产生一无模式(Modalless)对话框,但在 Personal Java 中,假如 Frame 类不被支持的话(见 Frame 备注),该方法将引起 java.lang.UnsupportedOperationException 例外。反之亦然。
Dialog(Frame,boolean)	已改变	该构造方法可产生一个无模式对话框或者一个模式(Modal)对话框,这决定于 boolean 型参数(详见 JDK 1.1 API 说明)。如是无模式的话,同上备注。如是模式对话框,则不会引起 java.lang.UnsupportedOperationException 例外。 但需注意 Personal Java 一次最多只允许一个模式对话框显示,假如 Applet 在已有模式对话框存在的情况下再打开一个,则原先的对话框将变为不可见,直到新对话框消失,原对话框才重新可见。
Dialog(Frame,String)	已改变	同 Dialog (Frame)。
Dialog(Frame,String,boolean)	已改变	同 Dialog (Frame,boolean),但要注意的是,String 型参数原先是对话框的标题,在 Personal Java 中将被忽略。
Dialog.setResizable(boolean)	已改变	指定值也许会被忽略。
FileDialog(Frame)	可选配	该构造方法是不要求的,假如调用也许引起 java.lang.UnsupportedOperationException 例外。很多消费式电子设备不需要有可让使用者访问的文件系统。只有在那些有可让用户访问的文件系统的设备平台上该方式才可正常工作。
FileDialog(Frame,String)	可选配	同上。
FileDialog(Frame,String,int)	可选配	同上。

续表

函 数	状 态	备 注
Frame()	可选配	该构造方法是不要求的，假如调用也许引起 java.lang.UnsupportedOperationException 例外。但假如设备平台允许重叠窗口的话，该方法可正常运行。 需要注意的是，无论任何 Personal Java1.0 的应用，都需有一个根 Frame 实例，在一些 Personal Java 的方法中需要有 Frame 类的对象做参数，这时可将该根 Frame 做这个参数。
Frame(String)	可选配	同上。
Graphics.setXORMode()	已改变	有些应用的显示，尤其是图形保真(anti - aliased)的，不允许以异或模式显示，这时，该方法的调用将引起 java.lang.UnsupportedOperationException 例外。
Menu()	可选配	该构造方法是不要求的，假如调用也许引起 java.lang.UnsupportedOperationException 例外。但如 Frame 类的构造方法不会引起该例外的话则此方法也不会，并且 Frame 类还必须有 Menu 类，反之亦然。
Menu(String)	可选配	同上。
Menu(String,boolean)	可选配	同上。
MenuBar()	可选配	该构造方法是不要求的，假如调用也许引起 java.lang.UnsupportedOperationException 例外。但如 Frame 类的构造方法不会引起该例外的话则此方法也不会，并且 Frame 类还必须有 Menu 类，反之亦然。
MenuShortcut(int)	可选配	该构造方法是不要求的，假如调用也许引起 java.lang.UnsupportedOperationException 例外。但如 MenuBar 类的构造方法不会引起该例外的话则此方法也不会，并且 MenuBar 类还必须有 MenuShortcut 类，反之亦然。
MenuShortcut(int,boolean)	可选配	同上。
PopupMenu.add(MenuItem)	已改变	MenuItem 在 Personal Java 中是被支持的，但假如有它的不被支持的子类如 Menu 被当作参数传递给该方法的话，则会引起 java.lang.UnsupportedOperationException 例外。
Scrollbar()	不被支持	该构造方法是不要求的，假如调用将引起 java.lang.UnsupportedOperationException 例外。
Scrollbar(int)	不被支持	同上。
Scrollbar(int,int,int,int,int)	不被支持	同上。

续表

函 数	状 态	备 注
ScrollPane( )	已改变	ScrollPane 类一般需要滚动,但 Scrollbar 又不被支持。所以各种不同的平台会用其他机制来代替 Scrollbar。
ScrollPane(int)	已改变	同上
TextArea( )	已改变	TtexArea 类一般需要滚动,但 Scrollbar 又不被支持。所以各种不同的平台会用其他机制来代替 Scrollbar。
TextArea(String)	已改变	同上。
TextArea(int,int)	已改变	同上。
TextArea(String,int,int)	已改变	同上。
TextArea(String,int,int,int)	已改变	同上。
Toolkit. getPrintJob (Frame, String, Properties)	已改变	在一个不支持打印的设备平台,该方法将引起 java.lang. UnsupportedOperationException 例外。
Window(Frame)	可选配	该构造方法是不要求的,假如调用也许引起 java.lang. UnsupportedOperationException 例外。但如 Frame 类的构造方法不会引起该例外的话则此方法也不会,反之亦然。

2.1.4 Personal Java 编程

前面已经说过,Personal Java 的设计目标之一就是一般的 Java 程序员经过简单的学习就可以掌握。但要想编出高效的 Personal Java 程序,仍有不少地方需要注意。

一是 Personal Java 工作的硬件环境比 PC 更有限得多。

二是 Personal Java 的使用者大多是没有计算机使用经验的消费者。这一点也许会被一般的 Personal Java 程序员忽视。

更容易被忽视的是,与为计算机编程不同,不应要求用户为掌握用 Personal Java 编写的软件而作专门的学习。当一位消费者(他甚至有可能从没见过电脑)买回装有 Personal Java 的家用电器时,他和任何消费者一样都要求电器买回就能用。

本书的读者想必都能较熟练的操作计算机,在用 Personal Java 编程时更要留神,不要一不小心就写出一个复杂的用户界面。记住,简单易用永远是 Personal Java 的主要要求之一。

2.2 Personal Java AWT

Personal Java AWT 是 Personal Java 与一般 Java 的最大区别。

2.2.1 设计简单易用的交互界面

前面曾提到过 Personal Java 的程序设计必须要保证用户界面简单易用。无论是程序的