走进Java

- 走进Java
  - Java发展史
  - Java虚拟机发展史
    - Sun Classic/Exact VM
    - Sun HotSpot VM
    - Sun Mobile-Embedded VM/Meta-Circular VM
      - KVM
      - CDC/CLDC HotSpot Implementation
      - Squawk VM
      - JavaInJava
      - Maxine VM
    - BEA JRockit/IBM J9 VM
    - Azul VM/BEA Liquid VM
    - Apache Harmony/Google Android Dalvik VM
    - Microsoft JVM
    - 其他虚拟机
  - 编译JDK
    - Mac OS
    - Windows
  - 参考

Java发展史

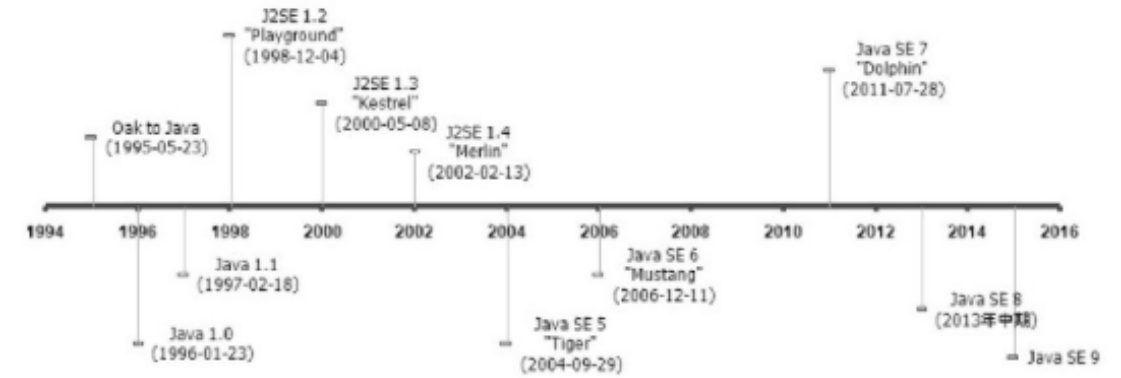

图 1-3 Java技术发展的时间线

“1991年4月，由James Gosling博士领导的绿色计划（Green Project）开始启动，此计划的目的是开发一种能够在各种消费性电子产品（如机顶盒、冰箱、收音机等）上运行的程序架构。这个计划的产品就是Java语言的前身：Oak（橡树）。Oak当时在消费品市场上并不算成功，但随着1995年互联网潮流的兴起，Oak迅速找到了最适合自己发展的市场定位并蜕变成为Java语言。”

“1995年5月23日，Oak语言改名为Java，并且在SunWorld大会上正式发布Java 1.0版本。Java语言第一次提出了"Write Once,Run Anywhere"的口号。”

“1996年1月23日，JDK 1.0发布，Java语言有了第一个正式版本的运行环境。JDK 1.0提供了一个纯解释执行的Java虚拟机实现（Sun Classic VM）。JDK 1.0版本的代表技术包括：Java虚拟机、Applet、AWT等。”

“1997年2月19日，Sun公司发布了JDK 1.1，Java技术的一些最基础的支撑点（如JDBC等）都是在JDK 1.1版本中发布的，JDK 1.1版的技术代表有：JAR文件格式、JDBC、JavaBeans、RMI。Java语法也有了一定的发展，如内部类（Inner Class）和反射（Reflection）都是在这个时候出现的。

直到1999年4月8日，JDK 1.1一共发布了1.1.0~1.1.8九个版本。从1.1.4之后，每个JDK版本都有一个自己的名字（工程代号），分别为：JDK 1.1.4-Sparkler（宝石）、JDK 1.1.5-Pumpkin（南瓜）、JDK 1.1.6-Abigail（阿比盖尔，女子名）、JDK 1.1.7-Brutus（布鲁图，古罗马政治家和将军）和JDK 1.1.8-Chelsea（切尔西，城市名）。”

“1998年12月4日，JDK迎来了一个里程碑式的版本JDK 1.2，工程代号为Playground（竞技场），Sun在这个版本中把Java技术体系拆分为3个方向，分别是面向桌面应用开发的J2SE（Java 2 Platform,Standard Edition）、面向企业级开发的J2EE（Java 2 Platform,Enterprise Edition）和面向手机等移动终端开发的J2ME（Java 2 Platform,Micro Edition）。在这个版本中出现的代表性技术非常多，如EJB、Java Plug-in、Java IDL、Swing等，并且这个版本中Java虚拟机第一次内置了JIT（Just In Time）编译器（JDK 1.2中曾并存过3个虚拟机，Classic VM、HotSpot VM和Exact VM，其中Exact VM只在Solaris平台出现过；后面两个虚拟机都是内置JIT编译器的，而之前版本所带的Classic VM只能以外挂的形式使用JIT编译器）。在语言和API级别上Java添加了strictfp关键字与现在Java编码之中极为常用的一系列Collections集合类。在1999年3月和7月，分别有JDK 1.2.1和JDK 1.2.2两个小版本发布。”

“1999年4月27日，HotSpot虚拟机发布，HotSpot最初由一家名为"Longview Technologies"的小公司开发，因为HotSpot的优异表现，这家公司在1997年被Sun公司收购了。HotSpot虚拟机发布时是作为JDK 1.2的附加程序提供的，后来它成为了JDK 1.3及之后所有版本的Sun JDK的默认虚拟机。”

“2000年5月8日，工程代号为Kestrel（美洲红隼）的JDK 1.3发布，JDK 1.3相对于JDK 1.2的改进主要表现在一些类库上（如数学运算和新的Timer API等），JNDI服务从JDK 1.3开始被作为一项平台级服务提供（以前JNDI仅仅是一项扩展），使用CORBA IIOP来实现RMI的通信协议，等等。这个版本还对Java 2D做了很多改进，提供了大量新的Java 2D API，并且新添加了JavaSound类库。JDK 1.3有1个修正版本JDK 1.3.1，工程代号为Ladybird（瓢虫），于2001年5月17日发布。”

“自从JDK 1.3开始，Sun维持了一个习惯：大约每隔两年发布一个JDK的主版本，以动物命名，期间发布的各个修正版本则以昆虫作为工程名称。”

“2002年2月13日，JDK 1.4发布，工程代号为Merlin（灰背隼）。JDK 1.4是Java真正走向成熟的一个版本，Compaq、Fujitsu、SAS、Symbian、IBM等著名公司都有参与甚至实现自己独立的JDK 1.4。哪怕是在十多年后的今天，仍然有许多主流应用（Spring、Hibernate、Struts等）能直接运行在JDK 1.4之上，或者继续发布能运行在JDK 1.4上的版本。JDK 1.4同样发布了很多新的技术特性，如正则表达式、异常链、NIO、日志类、XML解析器和XSLT转换器等。JDK 1.4有两个后续修正版：2002年9月16日发布的工程代号为Grasshopper（蚱蜢）的JDK 1.4.1与2003年6月26日发布的工程代号为Mantis（螳螂）的JDK 1.4.2。”

“2002年前后还发生了一件与Java没有直接关系，但事实上对Java的发展进程影响很大的事件，那就是 微软公司的.NET Framework发布了。这个无论是技术实现上还是目标用户上都与Java有很多相近之处的技术平台给Java带来了很多讨论、比较和竞争，.NET平台和Java平台之间声势浩大的孰优孰劣的论战到目前为止都在继续。”

“2004年9月30日，JDK 1.5[1]发布，工程代号Tiger（老虎）。从JDK 1.2以来，Java在语法层面上的变换一直很小，而JDK 1.5在Java语法易用性上做出了非常大的改进。例如，自动装箱、泛型、动态注解、枚举、可变长参数、遍历循环（foreach循环）等语法特性都是在JDK 1.5中加入的。在虚拟机和API层面上，这个版本改进了Java的内存模型（Java Memory Model,JMM）、提供了java.util.concurrent并发包等。另外，JDK 1.5是官方声明可以支持Windows 9x平台的最后一个JDK版本。”

“2006年12月11日，JDK 1.6发布，工程代号Mustang（野马）。在这个版本中，Sun终结了从JDK 1.2开始已经有8年历史的J2EE、J2SE、J2ME的命名方式，启用Java SE 6、Java EE 6、Java ME 6的命名方式。JDK 1.6的改进包括：提供动态语言支持（通过内置Mozilla JavaScript Rhino引擎实现）、提供编译API和微型HTTP服务器API等。同时，这个版本对Java虚拟机内部做了大量改进，包括锁与同步、垃圾收集、类加载等方面的算法都有相当多的改动。”

“在2006年11月13日的JavaOne大会上，Sun公司宣布最终会将Java开源，并在随后的一年多时间内，陆续将JDK的各个部分在GPL v2（GNU General Public License v2）协议下公开了源码，并建立了OpenJDK组织对这些源码进行独立管理。除了极少量的产权代码（Encumbered Code，这部分代码大多是Sun本身也无权限进行开源处理的）外，OpenJDK几乎包括了Sun JDK的全部代码，OpenJDK的质量主管曾经表示，在JDK 1.7中，Sun JDK和OpenJDK除了代码文件头的版权注释之外，代码基本上完全一样，所以OpenJDK 7与Sun JDK 1.7本质上就是同一套代码库开发的产品。”

“JDK 1.6发布以后，由于代码复杂性的增加、JDK开源、开发JavaFX、经济危机及Sun收购案等原因，Sun在JDK发展以外的事情上耗费了很多资源，JDK的更新没有再维持两年发布一个主版本的发展速度。JDK 1.6到目前为止一共发布了37个Update版本，最新的版本为Java SE 6 Update 37，于2012年10月16日发布。”

“2009年2月19日，工程代号为Dolphin（海豚）的JDK 1.7完成了其第一个里程碑版本。根据JDK 1.7的功能规划，一共设置了10个里程碑。最后一个里程碑版本原计划于2010年9月9日结束，但由于各种原因，JDK 1.7最终无法按计划完成。”

“从JDK 1.7最开始的功能规划来看，它本应是一个包含许多重要改进的JDK版本，其中的Lambda项目（Lambda表达式、函数式编程）、Jigsaw项目（虚拟机模块化支持）、动态语言支持、GarbageFirst收集器和Coin项目（语言细节进化）等子项目对于Java业界都会产生深远的影响。在JDK 1.7开发期间，Sun公司由于相继在技术竞争和商业竞争中都陷入泥潭，公司的股票市值跌至仅有高峰时期的3%，已无力推动JDK 1.7的研发工作按正常计划进行。为了尽快结束JDK 1.7长期“跳票”的问题，Oracle公司收购Sun公司后不久便宣布将实行“B计划”，大幅裁剪了JDK 1.7预定目标，以便保证JDK 1.7的正式版能够于2011年7月28日准时发布。“B计划”把不能按时完成的Lambda项目、Jigsaw项目和Coin项目的部分改进延迟到JDK 1.8之中。最终，JDK 1.7的主要改进包括：“提供新的G1收集器（G1在发布时依然处于Experimental状态，直至2012年4月的Update 4中才正式“转正”）、加强对非Java语言的调用支持（JSR-292，这项特性到目前为止依然没有完全实现定型）、升级类加载架构等。”

“到目前为止，JDK 1.7已经发布了9个Update版本，最新的Java SE 7 Update 9于2012年10月16日发布。从Java SE 7 Update 4起，Oracle开始支持Mac OS X操作系统，并在Update 6中达到完全支持的程度，同时，在Update 6中还对ARM指令集架构提供”“了支持。至此，官方提供的JDK可以运行于Windows（不含Windows 9x）、Linux、Solaris和Mac OS平台上，支持ARM、x86、x64和Sparc指令集架构类型。”

“2009年4月20日，Oracle公司宣布正式以74亿美元的价格收购Sun公司，Java商标从此正式归Oracle所有（Java语言本身并不属于哪间公司所有，它由JCP组织进行管理，尽管JCP主要是由Sun公司或者说Oracle公司所领导的）。由于此前Oracle公司已经收购了另外一家大型的中间件企业BEA公司，在完成对Sun公司的收购之后，Oracle公司分别从BEA和Sun中取得了目前三大商业虚拟机的其中两个：JRockit和HotSpot,Oracle公司宣布在未来1~2年的时间内，将把这两个优秀的虚拟机互相取长补短，最终合二为一[2]。可以预见在不久的将来，Java虚拟机技术将会产生相当巨大的变化。”

“根据Oracle官方提供的信息，JDK 1.8的第一个正式版本将于2013年9月发布，JDK 1.8将会提供在JDK 1.7中规划过，但最终未能在JDK 1.7中发布的特性，即Lambda表达式、Jigsaw（很不幸，随后Oracle公司又宣布Jigsaw在JDK 1.8中依然无法完成，需要延至JDK 1.9）和JDK 1.7中未实现的一部分Coin等。”

“在2011年的JavaOne大会上，Oracle公司还提到了JDK 1.9的长远规划，希望未来的Java虚拟机能够管理数以GB计的Java堆，能够更高效地与本地代码集成，并且令Java虚拟机运行时尽可能少人工干预，能够自动调节。”

摘录：《周志明.深入理解Java虚拟机：JVM高级特性与最佳实践》

Java虚拟机发展史

“从1996年初Sun公司发布的JDK 1.0中所包含的Sun Classic VM到今天，曾经涌现、湮灭过许多或经典或优秀或有特色的虚拟机实现，在这一节中，我们先暂且把代码与技术放下，一起来回顾一下Java虚拟机家族的发展轨迹和历史变迁。”

Sun Classic/Exact VM

“以今天的视角来看，Sun Classic VM的技术可能很原始，这款虚拟机的使命也早已终结。但仅凭它“世界上第一款商用Java虚拟机”的头衔，就足够有让历史记住它的理由。

1996年1月23日，Sun公司发布JDK 1.0，Java语言首次拥有了商用的正式运行环境，这个JDK中所带的虚拟机就是Classic VM。这款虚拟机只能使用纯解释器方式来执行Java代码，如果要使用JIT编译器，就必须进行外挂。但是假如外挂了JIT编译器，JIT编译器就完全接管了虚拟机的执行系统，解释器便不再工作了。”

Sun HotSpot VM

“它是Sun JDK和OpenJDK中所带的虚拟机，也是目前使用范围最广的Java虚拟机。但不一定所有人都知道的是，这个目前看起来“血统纯正”的虚拟机在最初并非由Sun公司开发，而是由一家名为"Longview Technologies"的小公司设计的；甚至这个虚拟机最初并非是为Java语言而开发的，它来源于Strongtalk VM，而这款虚拟机中相当多的技术又是来源于一款支持Self语言实现“达到C语言50%以上的执行效率”的目标而设计的虚拟机，Sun公司注意到了这款虚拟机在JIT编译上有许多优秀的理念和实际效果，在1997年收购了Longview Technologies公司，从而获得了HotSpot VM。”

“HotSpot VM既继承了Sun之前两款商用虚拟机的优点（如前面提到的准确式内存管理），也有许多自己新的技术优势，如它名称中的HotSpot指的就是它的热点代码探测技术（其实两个VM基本上是同时期的独立产品，HotSpot还稍早一些，HotSpot一开始就是准确式GC，而Exact VM之中也有与HotSpot几乎一样的热点探测。为了Exact VM和HotSpot VM哪个成为Sun主要支持的VM产品，在Sun公司内部还有过争论，HotSpot打败Exact并不能算技术上的胜利），HotSpot VM的热点代码探测能力可以通过执行计数器找出最具有编译价值的代码，然后通知JIT编译器以方法为单位进行编译。如果一个方法被频繁调用，或方法中有效循环次数很多，将会分别触发标准编译和OSR（栈上替换）编译动作。通过编译器与解释器恰当地协同工作，“可以在最优化的程序响应时间与最佳执行性能中取得平衡，而且无须等待本地代码输出才能执行程序，即时编译的时间压力也相对减小，这样有助于引入更多的代码优化技术，输出质量更高的本地代码。”

“在2006年的JavaOne大会上，Sun公司宣布最终会把Java开源，并在随后的一年，陆续将JDK的各个部分（其中当然也包括了HotSpot VM）在GPL协议下公开了源码，并在此基础上建立了OpenJDK。这样，HotSpot VM便成为了Sun JDK和OpenJDK两个实现极度接近的JDK项目的共同虚拟机。”

“在2008年和2009年，Oracle公司分别收购了BEA公司和Sun公司，这样Oracle就同时拥有了两款优秀的Java虚拟机：JRockit VM和HotSpot VM。Oracle公司宣布在不久的将来（大约应在发布JDK 8的时候）会完成这两款虚拟机的整合工作，使之优势互补。整合的方式大致上是在HotSpot的基础上，移植JRockit的优秀特性，譬如使用JRockit的垃圾回收器与MissionControl服务，使用HotSpot的JIT编译器与混合的运行时系统。”

Sun Mobile-Embedded VM/Meta-Circular VM

“Sun公司所研发的虚拟机可不仅有前面介绍的服务器、桌面领域的商用虚拟机，除此之外，Sun公司面对移动和嵌入式市场，也发布过虚拟机产品，另外还有一类虚拟机，在设计之初就没抱有商用的目的，仅仅是用于研究、验证某种技术和观点，又或者是作为一些规范的标准实现。这些虚拟机对于大部分不从事相关领域开发的Java程序员来说可能比较陌生。Sun公司发布的其他Java虚拟机有：”

KVM

KVM中的K是"Kilobyte"的意思，它强调简单、轻量、高度可移植，但是运行速度比较慢。在Android、iOS等智能手机操作系统出现前曾经在手机平台上得到非常广泛的应用。

CDC/CLDC HotSpot Implementation

“CDC/CLDC全称是Connected（Limited）Device Configuration，在JSR-139/JSR-218规范中进行定义，它希望在手机、电子书、PDA等设备上建立统一的Java编程接口，而CDC-HI VM和CLDC-HI VM则是它们的一组参考实现。CDC/CLDC是整个Java ME的重要支柱，但从目前Android和iOS二分天下的移动数字设备市场看来，在这个领域中，Sun的虚拟机所面临的局面远不如服务器和桌面领域乐观。”

Squawk VM

“Squawk VM由Sun公司开发，运行于Sun SPOT（Sun Small Programmable Object Technology，一种手持的WiFi设备），也曾经运用于Java Card。这是一个Java代码比重很高的嵌入式虚拟机实现，其中诸如类加载器、字节码验证器、垃圾收集器、解释器、编译器和线程调度都是Java语言本身完成的，仅仅靠C语言来编写设备I/O和必要的本地代码。”

JavaInJava

“JavaInJava是Sun公司于1997年~1998年间研发的一个实验室性质的虚拟机，从名字就可以看出，它试图以Java语言来实现Java语言本身的运行环境，既所谓的“元循环”（Meta-Circular，是指使用语言自身来实现其运行环境）。它必须运行在另外一个宿主虚拟机之上，内部没有JIT编译器，代码只能以解释模式执行。在20世纪末主流Java虚拟机都未能很好解决性能问题的时代，开发这种项目，其执行速度可想而知。”

Maxine VM

“Maxine VM和上面的JavaInJava非常相似，它也是一个几乎全部以Java代码实现（只有用于启动JVM的加载器使用C语言编写）的元循环Java虚拟机。这个项目于2005年开始，到现在仍然在发展之中，比起JavaInJava,Maxine VM就显得“靠谱”很多，它有先进的JIT编译器和垃圾收集器（但没有解释器），可在宿主模式或独立模式下执行，其执行效率已经接近了HotSpot Client VM的水平。”

BEA JRockit/IBM J9 VM

“前面介绍了Sun公司的各种虚拟机，除了Sun公司以外，其他组织、公司也研发过不少虚拟机实现，其中规模最大、最著名的就是BEA和IBM公司了。

JRockit VM曾经号称“世界上速度最快的Java虚拟机”（广告词，貌似J9 VM也这样说过），它是BEA公司在2002年从Appeal Virtual Machines公司收购的虚拟机。BEA公司将其发展为一款专门为服务器硬件和服务器端应用场景高度优化的虚拟机，由于专注于服务器端应用，它可以不太关注程序启动速度，因此JRockit内部不包含解析器实现，全部代码都靠即时编译器编译后执行。除此之外，JRockit的垃圾收集器和MissionControl服务套件等部分的实现，在众多Java虚拟机中也一直处于领先水平。”

“IBM J9 VM并不是IBM公司唯一的Java虚拟机，不过是目前其主力发展的Java虚拟机。IBM J9 VM原本是内部开发代号，正式名称是"IBM Technology for Java Virtual Machine"，简称IT4J，只是这个名字太拗口了一点，普及程度不如J9。J9 VM最初是由IBM Ottawa实验室一个名为SmallTalk的虚拟机扩展而来的，当时这个虚拟机有一个bug是由8k值定义错误引起的，工程师花了很长时间终于发现并解决了这个错误，此后这个版本的虚拟机就称为K8了，后来扩展出支持Java的虚拟机就被称为J9了。与BEA JRockit专注于服务器端应用不同，IBM J9的市场定位与Sun HotSpot比较接近，它是一款设计上从服务器端到桌面应用再到嵌入式都全面考虑的多用途虚拟机，“J9的开发目的是作为IBM公司各种Java产品的执行平台，它的主要市场是和IBM产品（如IBM WebSphere等）搭配以及在IBM AIX和z/OS这些平台上部署Java应用。”

Azul VM/BEA Liquid VM

“我们平时所提及的“高性能Java虚拟机”一般是指HotSpot、JRockit、J9这类在通用平台上运行的商用虚拟机，但其实Azul VM和BEA Liquid VM这类特定硬件平台专有的虚拟机才是“高性能”的武器。”

“Azul VM是Azul Systems公司在HotSpot基础上进行大量改进，运行于Azul Systems公司的专有硬件Vega系统上的Java虚拟机，每个Azul VM实例都可以管理至少数十个CPU和数百GB内存的硬件资源，并提供在巨大内存范围内实现可控的GC时间的垃圾收集器、为专有硬件优化的线程调度等优秀特性。在2010年，Azul Systems公司开始从硬件转向软件，发布了自己的Zing JVM，可以在通用x86平台上提供接近于Vega系统的特性。”

“Liquid VM即是现在的JRockit VE（Virtual Edition），它是BEA公司开发的，可以直接运行在自家Hypervisor系统上的JRockit VM的虚拟化版本，Liquid VM不需要操作系统的支持，或者说它自己本身实现了一个专用操作系统的必要功能，如文件系统、网络支持等。由虚拟机越过通用操作系统直接控制硬件可以获得很多好处，如在线程调度时，不需要再进行内核态/用户态的切换等，这样可以最大限度地发挥硬件的能力，提升Java程序的执行性能。”

Apache Harmony/Google Android Dalvik VM

“这节介绍的Harmony VM和Dalvik VM只能称做“虚拟机”，而不能称做“Java虚拟机”，但是这两款虚拟机（以及所代表的技术体系）对最近几年的Java世界产生了非常大的影响和挑战，甚至有些悲观的评论家认为成熟的Java生态系统有崩溃的可能。”

“Apache Harmony是一个Apache软件基金会旗下以Apache License协议开源的实际兼容于JDK 1.5和JDK 1.6的Java程序运行平台，这个介绍相当拗口。它包含自己的虚拟机和Java库，用户可以在上面运行Eclipse、Tomcat、Maven等常见的Java程序，但是它没有通过TCK认证，所以我们不得不用那么一长串拗口的语言来介绍它，而不能用一句“Apache的JDK”来说明。如果一个公司要宣布自己的运行平台“兼容于Java语言”，那就必须要通过TCK（Technology Compatibility Kit）的兼容性测试。Apache基金会曾要求Sun公司提供TCK的使用授权，但是一直遭到拒绝，直到Oracle公司收购了Sun公司之后，双方关系越闹越僵，最终导致Apache愤然退出JCP（Java Community Process）组织，这是目前为止Java社区最严重的一次“分裂”。”

“在Sun将JDK开源形成OpenJDK之后，Apache Harmony开源的优势被极大地削弱，甚至连Harmony项目的最大参与者IBM公司也宣布辞去Harmony项目管理主席的职位，并参与OpenJDK项目的开发。虽然Harmony没有经过真正大规模的商业运用，但是它的许多代码（基本上是Java库部分的代码）被吸纳进IBM的JDK 7实现及Google Android SDK之中，尤其是对Android的发展起到了很大的推动作用。”

“说到Android，这个时下最热门的移动数码设备平台在最近几年间的发展过程中所取得的成果已经远远超越了Java ME在过去十多年所获得的成果，Android让Java语言真正走进了移动数码设备领域，只是走的并非Sun公司原本想象的那一条路。”

“Dalvik VM是Android平台的核心组成部分之一，它的名字来源于冰岛一个名为Dalvik的小渔村。Dalvik VM并不是一个Java虚拟机，它没有遵循Java虚拟机规范，不能直接执行Java的Class文件，使用的是寄存器架构而不是JVM中常见的栈架构。但是它与Java又有着千丝万缕的联系，它执行的dex（Dalvik Executable）文件可以通过Class文件转化而来，使用Java语法编写应用程序，可以直接使用大部分的Java API等。目前Dalvik VM随着Android一起处于迅猛发展阶段，在Android 2.2中已提供即时编译器实现，在执行性能上有了很大的提高。”

Microsoft JVM

“在十几年的Java虚拟机发展过程中，除去上面介绍的那些被大规模商业应用过的Java虚拟机外，还有许多虚拟机是不为人知的或者曾经“绚丽”过但最终湮灭的。我们以其中微软公司的JVM为例来介绍一下。”

“也许Java程序员听起来可能会觉得惊讶，微软公司曾经是Java技术的铁杆支持者（也必须承认，与Sun公司争夺Java的控制权，令Java从跨平台技术变为绑定在Windows上的技术是微软公司的主要目的）。在Java语言诞生的初期（1996年~1998年，以JDK 1.2发布为分界），它的主要应用之一是在浏览器中运行Java Applets程序，微软公司为了在IE3中支持Java Applets应用而开发了自己的Java虚拟机，虽然这款虚拟机只有Windows平台的版本，却是当时Windows下性能最好的Java虚拟机，它在1997年和1998年连续两年获得了《PC Magazine》杂志的“编辑选择奖”。但好景不长，在1997年10月，Sun公司正式以侵犯商标、不正当竞争等罪名控告微软公司，在随后对微软公司的垄断调查之中，这款虚拟机也曾作为证据之一被呈送法庭。这场官司的结果是微软公司赔偿2000万美金给Sun公司（最终微软公司因垄断赔偿给Sun公司的总金额高达10亿美元），承诺终止其Java虚拟机的发展，并逐步在产品中移除Java虚拟机相关功能。具有讽刺意味的是，到最后在Windows XP SP3中Java虚拟机被完全抹去的时候，Sun公司却又到处登报希望微软公司不要这样做[1]。Windows XP高级产品经理Jim Cullinan称：“我们花费了3年的时间和Sun打官司，当时他们试图阻止我们在Windows中支持Java，现在我们这样做了，可他们又在抱怨，这太具有讽刺意味了。”

“我们试想一下，如果当年Sun公司没有起诉微软公司，微软公司继续保持着对Java技术的热情，那Java的世界会变得怎么样呢？.NET技术是否会发展起来？但历史是没有假设的。其他在本节中没有介绍到的Java虚拟机还有（当然，应该还有很多笔者所不知道的）：”

其他虚拟机

以下是其他虚拟机实现

JamVM

cacaovm

SableVM

Kaffe

Jelatine JVM

NanoVM

MRP

Moxie JVM

Jikes RVM

编译JDK

Mac OS

参考：

https://www.jianshu.com/p/b8177780c939

https://blog.csdn.net/u010416101/article/details/84839103

Windows

参考：

https://my.oschina.net/langxSpirit/blog/1624428

参考

《周志明.深入理解Java虚拟机：JVM高级特性与最佳实践》第一章