# JVM与Java体系结构

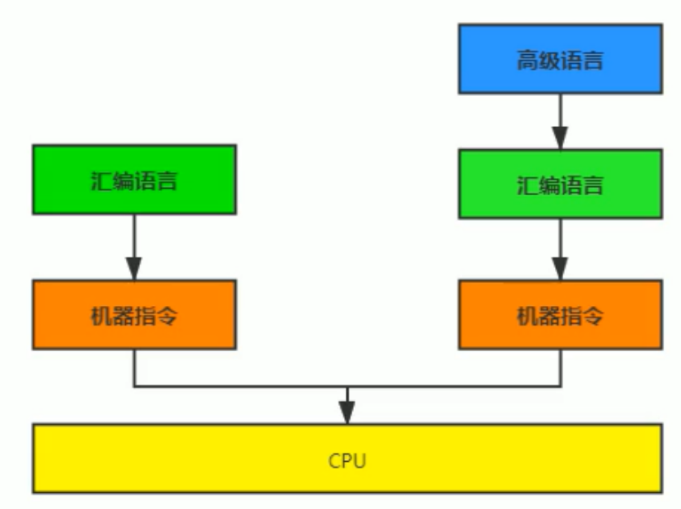
## 前言



大部分 Java 开发人员，除会在项目中使用到与 Java 平台相关的各种高精尖技术，对于 Java 技术的核心 Java 虚拟机了解甚少。

**开发人员如何看待上层框架：**

* 一些有一定工作经验的开发人员，打心眼儿里觉得 SSM 、微服务等上层技术才是重点，基础技术并不重要，这其实是一种本末倒置的“病态”。
* 如果我们把核心类库的 AP エ比做数学公式的话，那么 Java 虚拟机的知识就好比公式的推导过程。



计算机系统体系对我们来说越来越远，在不了解底层实现方式的前提下，通过高级语言很容易编写程序代码。但事实上计算机并不认识高级语言。

**我们为什么要学习 JVM ？**

1. 面试的需要（ BATJ 、 TMD 、 PKQ 等面试都爱问）
2. 中高级程序员必备技能

项目管理、调优的需要

1. 追求极客精神

比如：垃圾回收算法、 JIT 、底层原理

## 2. 面向人群及参考书目

官方参考书目：

1. The Java Virtual Machine Specification-JavaSE7
2. **The Java Virtual Machine Specification-JavaSE8**
3. **The Java Virtual Machine Specification-JavaSE11**

Java 8 和 11 是长期支持版本，目前国内主流是 8，之后会是 11。

**中文参考书目：**

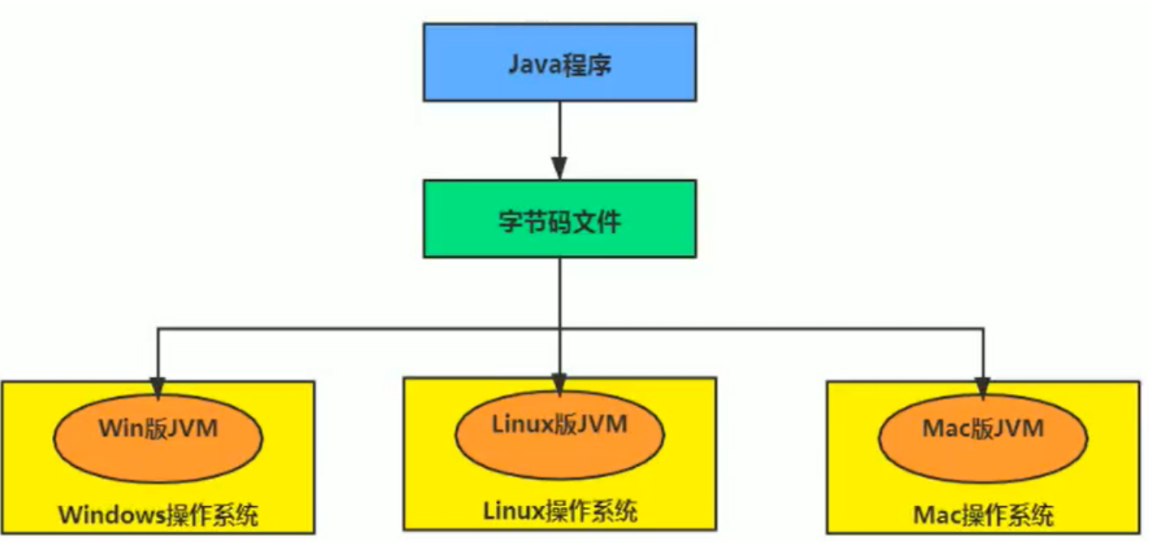
* Java 虚拟机规范（Java SE 8版）
* 官方文档汉化，仅适合平时参考
* **深入理解 Java 虚拟机**
* 深入理解 JVM & G1 GC
* 揭秘 Java 虚拟机
* Java 虚拟机基础教程
* 实战 Java 虚拟机
* Java 虚拟机精讲
* 码出高效 Java 开发手册

## 3 - Java 及 JVM 简介

Java 长期占据 TIOBE 语言热度排行榜第一（现在已被 C 反超），不是说明 Java 语言多么优越，只是因为庞大的生态、够用的功能、没有致命的缺陷。

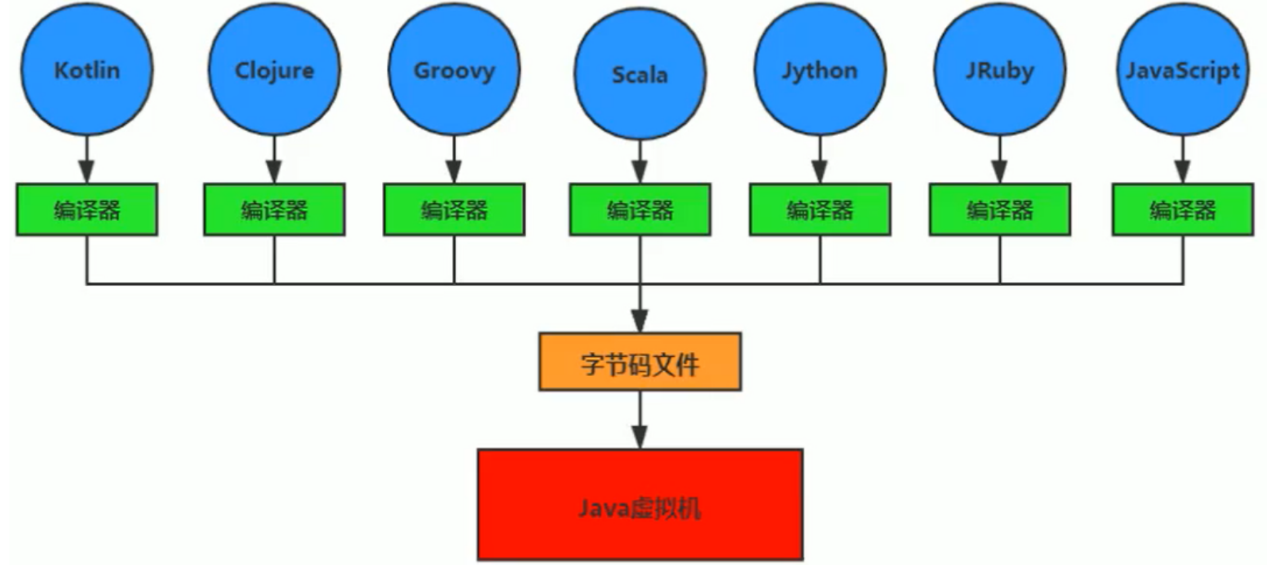
**世界上没有最好的编程语言，只有最适用于具体应用场景的编程语言。**

Java : 跨平台的语言

****

**Write once, run anywhere.**

JVM : 跨语言的平台

****

随着 Java 7 的正式发布，Java 虚拟机的设计者们通过 **JSR-292** 规范基本实现。在 **Java 虚拟机平台上运行非 Java 语言编写的程序**。

Java 虚拟机根本不关心运行在其内部的程序到底是使用何种编程语言编写的，它**只关心 “字节码” 文件**。也就是说 **Java 虚拟机拥有语言无关性**，并不会单纯地与 Java 语言 “终身绑定”，只要其他编程语言的编译结果满足并包含 Java 虚拟机的内部指令集、符号表以及其他的辅助信息，它就是一个有效的字节码文件，就能够被虚拟机所识别并装载运行。

**Java 不是最强大的语言，但是 JVM 是最强大的虚拟机。**

### 字节码

* 我们平时说的 java 字节码，指的是用 java 语言编译成的字节码。准确的说任何能在 JVM 平台上执行的字节码格式都是一样的。所以应该统称为 **JVM字节码。**
* 不同的编译器，可以编译出相同的字节码文件，字节码文件也可以在不同的 JVM 上运行。
* Java 虚拟机与 Java 语言并没有必然的联系，它只与特定的二进制文件格式 —— Class 文件格式所关联， Class 文件中包含了 Java 虚拟机指令集（或者称为字节码、 Bytecodes ）和符号表，还有一些其他辅助信息。

### 多语言混合编程

**Java 平台上的多语言混合编程正成为主流，通过特定领域的语言去解决特定领域的问题是当前软件开发应对日趋复杂的项目需求的一个方向。**

试想一下，在一个项目之中，并行处理用 Clojure 语言编写，展示层使用 JRuby/Rails ，中间层则是 Java ，每个应用层都将使用不同的编程语言来完成，而且，接口对每一层的开发者都是透明的，**各种语言之间的交互不存在任何困难，就像使用自己语言的原生 API 一样方便，因为它们最终都运行在一个虚拟机之上。**

对这些运行于 Java 虚拟机之上、 Java 之外的语言，来自系统级的、底层的支持正在迅速增强，以 JSR-292 为核心的一系列项目和功能改进（如 DaVinci Machine 项目、 Nasho引擎、 InvokeDynamic 指令、 java.lang.invoke 包等），推动 Java 虚拟机从 “Java 语言的虚拟机” 向 “多语言虚拟机” 的方向发展。

## - Java 发展的重大事件

* 1990 年，在 Sun 计算机公司中，由 Patrick Naughton 、 Mike Sheridan 及 James Gosling 领导的小组 Green Team ，开发出的新的程序语言，命名为 Oak ，后期命名为 Java
* 1995 年，Sun 正式发布 Java 和 HotJava 产品，Java 首次公开亮相。
* 1996 年 1 月 23 日 Sun Microsystems 发布了 JDK 1.0 。
* 1998 年， JDK 1.2 版本发布。同时，Sun发布了 JSP/Servlet 、 EJB 规范，以及将 Java 分成了 J2EE 、 J2SE 和 J2ME 。这表明了 Java 开始向企业、桌面应用和移动设备应用 3 大领域挺进。
* 2000 年，DK 1.3 发布， **Java HotSpot Virtual Machine 正式发布，成为 Java 的默认虚拟机**。
* 2002 年，JDK 1.4 发布，古老的 Classic 虚拟机退出历史舞台。
* 2003 年年底，**Java 平台的 Scala 正式发布，同年 Groovy 也加入了 Java 阵营。**
* 2004 年， JDK 1.5 发布。同时 JDK 1.5 改名为 JavaSE 5.0 。
* 2006 年， JDK 6 发布。同年，**Java 开源并建立了 OpenJDK 。顺理成章， Hotspot 虚拟机也成为了 OpenJDK 中的默认虚拟机。**
* 2007年， **Java 平台迎来了新伙伴 Clojure**
* 2008年， Oracle 收购了 BEA ，**得到了 JRockit 虚拟机。**
* 2009 年， Twitter 宣布把后台大部分程序从 Ruby 迁移到 Scala ，这是 Java 平台的又ー次大规模应用。
* 2010 年， Oracle 收购了 Sun ，**获得 Java 商标和最具价值的 HotSpot 虚拟机**。此时， Oracle 拥有市场占用率最高的两款虚拟机 HotSpot 和 JRockit ，并计划在未来对它们进行整合：HotRockit
* 2011 年，JDK 7发布。在 JDK 1.7u4 中，**正式启用了新的垃圾回收器 G1** 。
* 2017 年，JDK 9发布。**将 G1 设置为默认 GC ，替代 CMS**
* 同年，**IBM的J9开源**，形成了现在的 Open J9 社区
* 2018 年，Android 的 Java 侵权案判决， Goggle 赔偿 Oracle 88 亿美元
* 同年，Oracle 宣告 JavaEE 成为历史名词， JDBC 、 JMS 、 Servlet 赠予 Eclipse 基金会。
* 同年， JDK 11 发布，LTS 版本的 JDK ，**发布革命性的 ZGC，调整 JDK 授权许可**
* 2019 年，JDK 12 发布，加入 Redhat 领导开发的 **Shenandoah GC**

## 5 - 虚拟机与 Java 虚拟机

### 虚拟机

所谓虚拟机（ Virtual Machine ），就是一台虚拟的计算机。它是一款软件，用来执行一系列虚拟计算机指令。大体上，虚拟机可以分为**系统虚拟机**和**程序虚拟机**

* 大名鼎鼎的 Visual Box ， Vmware 就属于系统虚拟机，它们完全是**对物理计算机的仿真**，提供了一个可运行完整操作系统的软件平台。
* 程序虚拟机的典型代表就是 Java 虚拟机，它**专门为执行单个计算机程序而设计**，在 Java 虚拟机中执行的指令我们称为 Java 字节码指令

无论是系统虚拟机还是程序虚拟机，在上面运行的软件都被限制于虚拟机提供的资源中。

### Java 虚拟机

* Java 虚拟机是一台执行 Java 字节码的虚拟计算机，它拥有独立的运行机制，其运行的 Java 字节码也未必由 Java 语言编译而成。
* JVM 平台的各种语言可以共享 Java 虚拟机带来的跨平台性、优秀的垃圾回器，以及可靠的即时编译器。
* **Java 技术的核心就是 Java 虚拟机**（ JVM ， Java Virtual Machine ），因为所有的 Java 程序都运行在 Java 虚拟机内部。

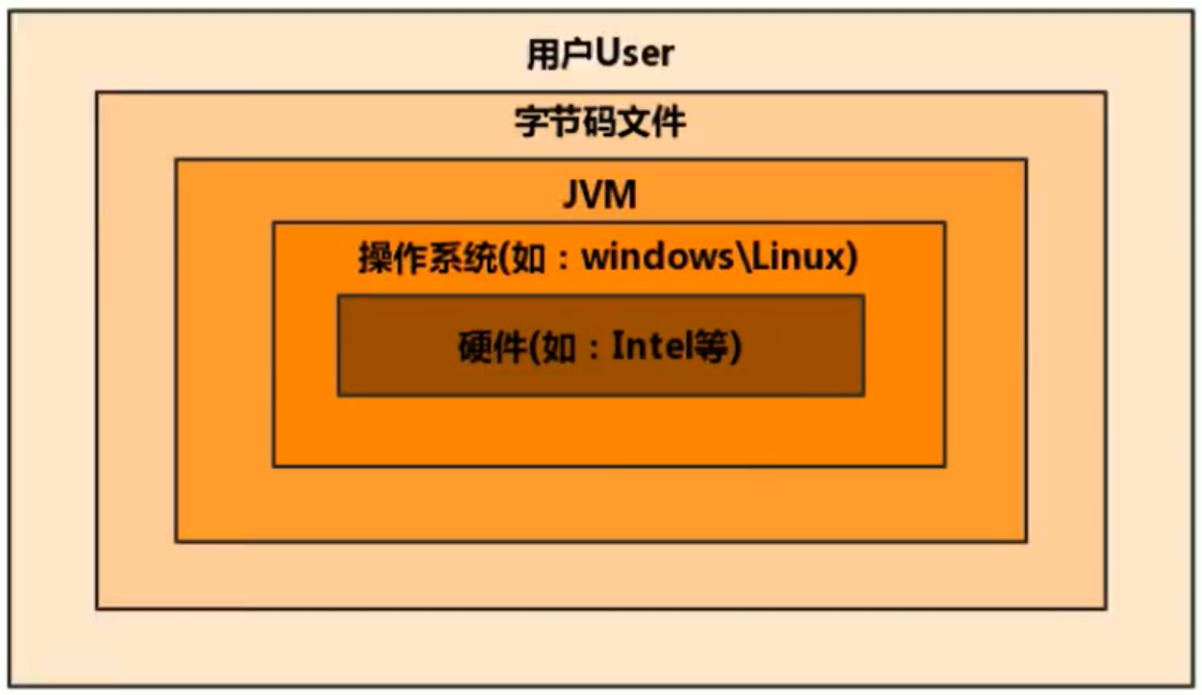
##### 作用

**Java 虚拟机就是二进制字节码的运行环境**，负责装载字节码到其内部，解释/编译为对应平台上的机器指令执行。每一条 Java 指令，Java 虚拟机规范中都有详细定义，如怎么取操作数，怎么处理操作数，处理结果放在哪里。

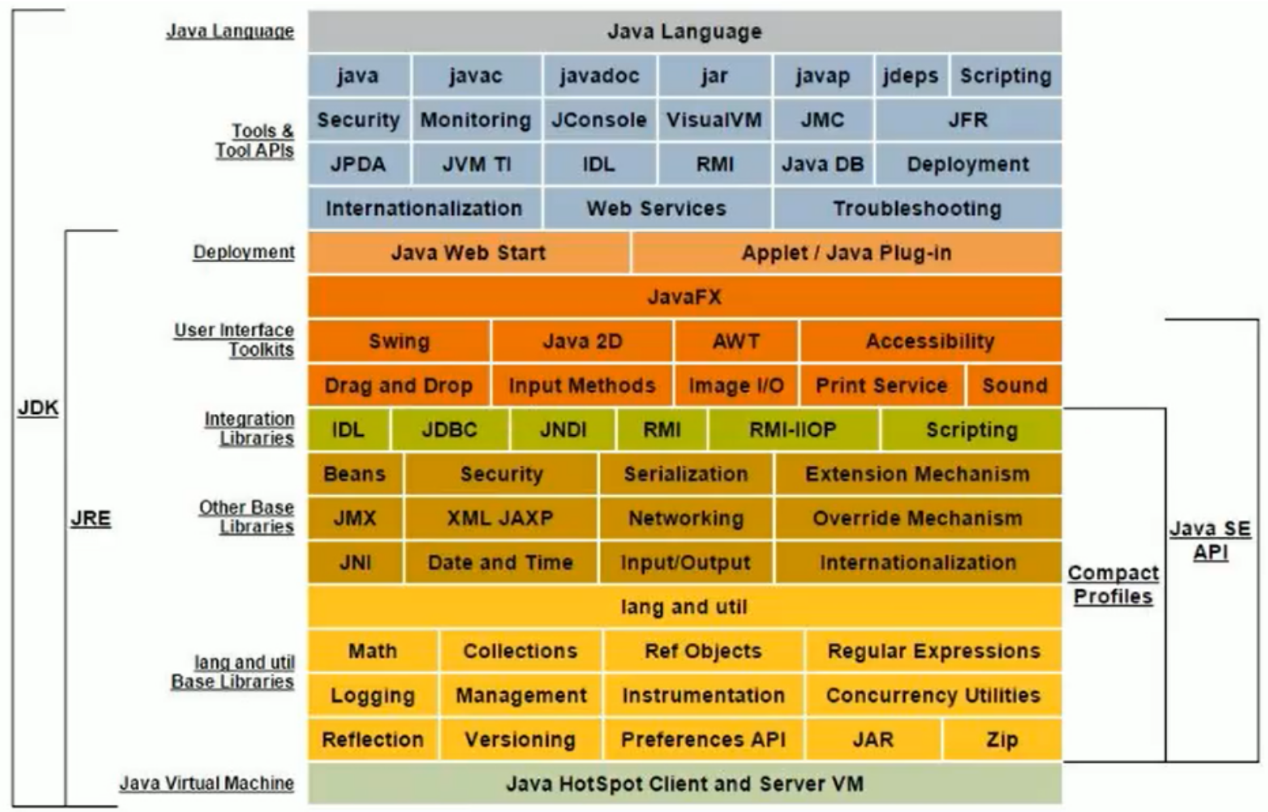
##### 特点

* 一次编译，到处运行
* 自动内存管理
* 自动垃圾回收功能

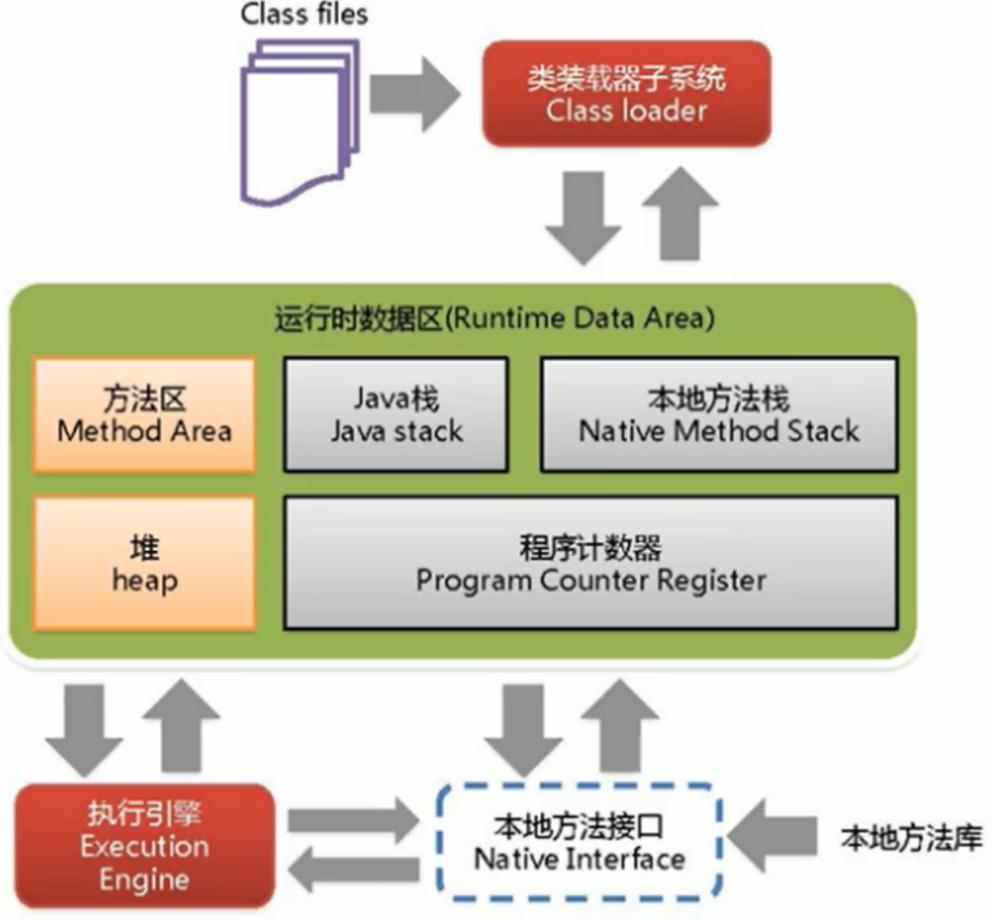
#### JVM 的位置



JVM 是运行在操作系统之上的，它与硬件没有直接的交互。

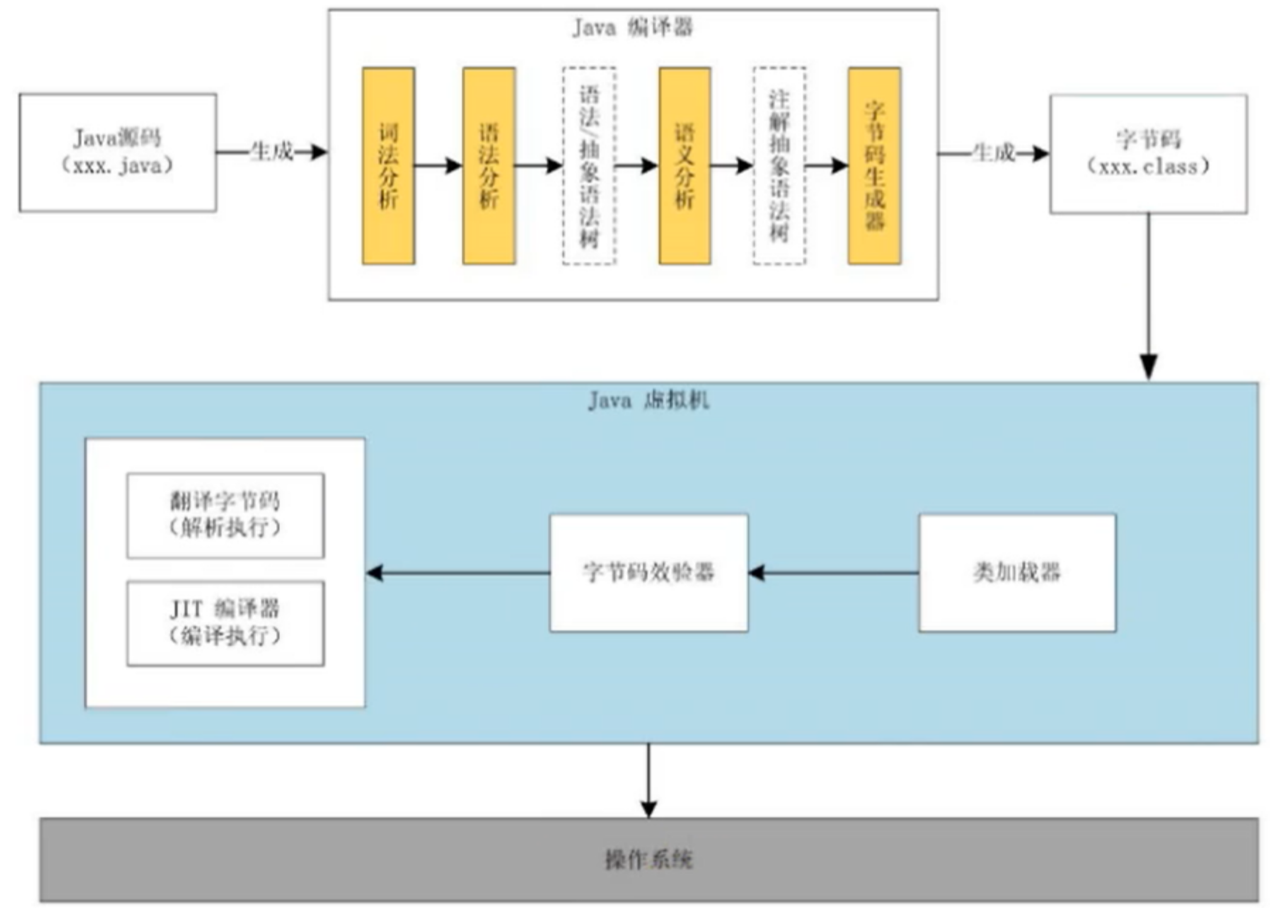


## 6 - JVM 的整体结构



* HotSpot VM 是目前市面上高性能虚拟机的代表作之一。
* 它采用解释器与即时编译器并存的架构。
* 在今天，Java程序的运行性能早已脱胎换骨，已经达到了可以和 C/C++ 程序一较高下的地步。

## 7 - Java 代码执行流程



## 8 - JVM 的架构模型

**Java 编译器输入的指令流基木上是一种基于栈的指令集架构**，另外一种指令集架构则是**基于寄存器的指令集架构**。

具体来说，这两种架构之间的区别：

**基于栈式架构的特点**

* 设计和实现更简单，适用于资源受限的系统；
* 避开了寄存器的分配难题：使用零地址指令方式分配。
* 指令流中的指令大部分是零地址指令，其执行过程依赖于操作栈。指令集更小，编译器容易实现。
* 不需要硬件支持，可移植性更好，更好实现跨平台

**基于寄存器架构的特点**

* 典型的应用是 x86 的二进制指令集：比如传统的 PC 以及 Android 的 Davlik 虚拟机。
* **指令集架构则完全依赖硬件，可移植性差**
* **性能优秀和执行更高效**
* 花费更少的指令去完成一项操作。
* 在大部分情况下，基于寄存器架构的指令集往往都以一地址指令、二地址指令和三地址指令为主，而基于栈式架构的指令集却是以零地址指令为主。

**总结**：

**由于跨平台性的设计， Java 的指令都是根据栈来设计的。**不同平台CPU架构不同，所以不能设计为基于寄存器的。优点是跨平台，指令集小，编译器容易实现，缺点是性能下降，实现同样的功能需要更多的指令。

时至今日，尽管嵌入式平台已经不是 Java 程序的主流运行平台了（准确来说应该是 HotSpot VM 的宿主环境己经不局限于嵌入式平台了），那么为什么不将架构更换为基于寄存器的架构呢？

**原因：**设计和实现更简单；在非资源受限的环境中，基于栈的设计仍然可用。

栈：跨平台性、指令集小、指令多；执行性能比寄存器差。

## 9 - JVM 的生命周期

1. **虚拟机的启动**

Java 虚拟机的启动是通过引导类加载器（ bootstrap class loader ）创建初始类（ initial class ）来完成的，这个类是由虚拟机的具体实现指定的。

1. **虚拟机的执行**

* 一个运行中的 Java 虚拟机有着一个清晰的任务：执行 Java 程序。
* 程序开始执行时他才运行，程序结束时他就停止。
* 执行一个所谓的 Java 程序的时候，真真正正在执行的是一个叫做 Java 虚拟机的进程。

1. **虚拟机的退出（有如下几种情况）**

* 程序正常执行结束
* 程序在执行过程中遇到了异常或错误而异常终止
* 由于操作系统出现错误而导致中 Java 虚拟机进程终止
* 某线程调用 Runtime 类或 System 类的 exit 方法，或 Runtime 类的 halt 方法，并且 Java 安全管理器也允许这次 exit 或 halt 操作。
* 除此之外， JNI （ Java Native Interface ）规范描述了用 JNI Invocation API 来加载或卸载 Java 虚拟机时，Java 虚拟机的退出情况

## 10 - JVM 发展历程

#### Sun Classic VM

早在 1996 年 Java 1.0 版本的时候， Sun 公司发布了一款名为 Sun Classic VM的Java 虚拟机，它同时也是世界上第一款商用 Java 虚拟机， JDK1.4 时完全被淘汰。

这款虚拟机内部只提供解释器。如果使用 JIT 编译器，就需要进行外挂。但是一旦使用了 J 编译器， J エ T 就会接管虚拟机的执行系统。解释器就不再工作。解释器和编译器不能配合工作。

现在 hotspot 内置了此虚拟机。

#### Exact VM

为了解決上一个虚拟机问题，jdk1.2 时， sun 提供了此虚拟机。

Exact Memory Management ：准确式内存管理。

* 也可以叫 Non-Conservative / Accurate Memory Management
* 虚拟机可以知道内存中某个位置的数据具体是什么类型。

具备现代高性能虚拟机的雏形

* 热点探测
* 编译器与解释器混合工作模式

只在 Solaris 平台短暂使用，其他平台上还是 classic vm

英雄气短，终被 Hotspot 虚拟机替换

#### SUN 公司的 Hotspot VM

Hotspot 历史

* 最初由一家名为 “ Longview Technologies " 的小公司设计
* 1997 年，此公司被 Sun 收购； 2009 年， Sun 公司被甲骨文收购。
* JDK 1. 3 时， Hotspot VM 成为默认虚拟机

目前 **Hotspot 占有绝对的市场地位，称霸武林。**

* 不管是现在仍在广泛使用的 JDK 6，还是使用比例较多的 JDK 8中，默认的虛拟机都是 Hotspot
* Sun / Oracle JDK 和 OpenJDK 的默认虚拟机
* 因此本课程中默认介绍的虚拟机都是 Hotspot，相关机制也主要是指 Hotspot 的 GC 机制。（比如其他两个商用虚拟机都没有方法区的概念）

从服务器、桌面到移动端、嵌入式都有应用。

名称中的 Hotspot 指的就是它的热点代码探测技术。

* 通过计数器找到最具编译价值代码，触发即时编译或栈上替换
* 通过编译器与解释器协同工作，在最优化的程序响应时间与最佳执行性能中取得平衡

#### BEA 的 JRockit

**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。

大量的行业基准测试显示， **JRockit JVM 是世界上最快的 JVM** 。

* 使用 JRockit 产品，客户已经体验到了显著的性能提高（一些超过了 70%）和硬件成本的减少（达 50%）。

优势：全而的 Java 运行时解决方案组合

* JRockit 面向延迟敏感型应用的解决方案 JRockit Real Time 提供以毫秒或微秒级的 JVM 响应时间，适合财务、军事指挥、电信网络的需要
* MissionControl 服务套件，它是一组以极低的开销来监控、管理和分析生产坏境中的应用程序的工具。

2008 年， BEA 被 Oracle 收购

Oracle 表达了整合两大优秀虚拟机的工作，大致在 JDK8 中完成。整合的方式是在 Hotspot 的基础上，移植 JRockit的优秀特性。

#### IBM 的 J9

全称：IBM Technology for Java Virtual Machine ，简称 IT4J ，内部代号： J9

市场定位与 Hotspot 接近，服务器端、桌而应用、嵌入式等多用途 VM

广泛用于 IBM 的各种 Java 产品。

目前，**有影响力的三大商用虚拟机之一**，也号称是世界上最快的Java虚拟机。

2017 年左右，IBM 发布了开源 J9 VM ，命名为 OpenJ9，交给 Eclipse 基金会管理，也称为 Eclipse OpenJ9

#### Azul VM

前面三大 “高性能 Java 虚拟机” 使用在通用硬件平台上

这里 Azul VM 和 BEA Liquid VM 是**与特定硬件平台绑定、软硬件配合的专有虚拟机**

* 高性能 Java 虚拟机中的战斗机。

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、专有硬件优化的线程调度等优秀特性**

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

高性能 Java 虚拟机中的战斗机。

BEA 公司开发的，直接运行在自家 Hypervisor 系统上

Liquid VM 即是现在的 JRockit VE（ Virtual Edition ）， 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，如线程调度、文件系统、网络支持等。

随着 JRockit 虚拟机终止开发， Liquid VM项目也停止了。

#### Apache Harmony

Apache 也曾经推出过与 JDK 1.5 和 JDK 1.6 兼容的 Java 运行平台 Apache Harmony

它是 IBM 和 Intel 联合开发的开源 JVM ，受到同样开源的 OpenJDK 的压制，Sun 坚决不让 Harmony 获得 JCP 认证，最终于 2011 年退役， IBM 转而参与 OpenJDK

虽然目前并没有 Apache Harmony 被大规模商用的案例，但是它的 Java 类库代码吸纳进了 Android SDK。

#### Microsoft JVM

微软为了在工 E3 浏览器中支持 Java Applets ，开发了 Microsoft JVM 。

只能在 Windows 平台下运行。但确是当时 Windows 下性能最好的 Java VM 。

1997 年， Sun 以侵犯商标、不正当竞争罪名指控微软成功，赔了 Sun 很多钱。微软在 Windows SP3 中抹掉了其 VM 。现在 Windows 上安装的 JDK 都是 Hotspot

TaobaoJVM

由 Ali JVM团队发布。阿里，国内使用 Java 最强大的公司，覆盖云计算、金融、物流电商等众多领域，需要解决高并发、高可用、分布式的复合问题。有大量的开源产品

**基于 OpenJDK 开发了自己的定制版本 AlibabaJDK** ，简称 AJDK。是整个阿里 Java体系的基石。

基于 OpenJDK Hotspot VM 发布的国内第一个优化、**深度定制且开源的高性能服务器版 Java虚拟机**。

* 创新的 GCIH（ GC invisible heap ）技术实现了 off-heap ，**即将生命周期较长的 Java对象从 heap 中移到heap 之外，并且 GC 不能管理 GCIH 内部的 Java对象，以此达到降低 GC 的回收频率和提升 GC 的回收效率的目的**
* GCIH 中的**对象还能够在多个 Java虚拟机进程中实现共享**
* 使用 crc32 指令实现 JVM intrinsic 降低 JNI 的调用开销 \
* PMU hardware 的 Java profiling tool 和诊断协助功能
* 针对大数据场景的 ZenGC

taobao vm 应用在阿里产品上性能高，**硬件严重依赖 intel 的 cpu** ，损失了兼容性，但提高了性能

* 目前已经在淘宝、天猫上线，把 Oracle 官方 JVM版木全部替换了。

#### Dalvik VM

谷歌开发的，应用于 Android 系统，并在 Android 2.2 中提供了 JIT ，发展迅猛。

**Dalvik vm 只能称作虚拟机，而不能称作“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

基于寄存器架构，不是 jvm 的栈架构

执行的是编译以后的 dex（ Dalvik Executable ）文件。执行效率比较高

* 它执行的 dek（ 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。

Android 5. 0 使用支持提前编译（ Ahead of Time Compilation ， AOT ）的 ART VM 替换 Dalvik VM 。

#### Graal VM

2018 年 4 月， Oracle Labs 公开了 Graal VM，号称 " **Run Programs Faster Anywhere** "，勃勃野心。

与 1995 年 Java 的 " write once, run anywhere " 遥相呼应。

Graal VM 在 Hotspot VM 基础上増强而成的**跨语言全栈虚拟机，可以作为 “任何语言” 的运行平台使用**。语言包括： Java 、Scala 、 Groovy 、 Kotlin ； C 、 C ++、 JavaScript、 Ruby、 Python 、 R 等

支持不同语言中混用对方的接口和对象，支持这些语言使用已经编写好的本地库文件

工作原理是将这些语言的源代码或源代码编译后的中间格式，通过解释器转换为能被 Graal VM 接受的中间表示。Graal VM 提供 Truffle 工具集快速构建面向一种新语言的解释器。在运行时还能进行即时编译优化，获得比原生编译器更优秀的执行效率。

**如果说 HotSpot 有一天真的被取代，Graal VM 希望最大**。但是 Java 的软件生态没有丝毫变化

#### 总结

具体 JVM 的内存结构，其实取決于其实现，不同厂商的 JVM ，或者同一厂商发布的不同版本，都有可能存在一定差异。**本套课程主要以 Oracle Hotspot VM 为默认虚拟机。**