一、JVM与Java体系结构

1、Java及JVM简介

2、Java发展重大事件

3、虚拟机与Java虚拟机

3.1 虚拟机概念

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（cpu、硬盘……，对硬件的模拟），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。（系统虚拟机上的应用软件）

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。

3.2 Java虚拟机

1. 概念说明

- 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。

  即JVM 并不限制字节码一定要由 Java 语言编写，JVM 所能处理的字节码是遵循 Java 虚拟机规范定义的一种通用格式，因此可以由其他编程语言生成的字节码也能在 JVM 上运行。

  1. Kotlin：一种现代的静态类型编程语言，它与 Java 无缝集成，并可以编译成 Java 字节码以在 JVM 上运行。
  2. Scala：一种多范式编程语言，它结合了面向对象编程和函数式编程的特性。Scala 代码也可以编译成 Java 字节码并在 JVM 上执行。
  3. Groovy：一种动态类型的编程语言，它允许在 Java 平台上进行简洁、优雅的编码。Groovy 代码也可以编译成 Java 字节码并在 JVM 上运行。
  4. Clojure：一种 Lisp 方言，它运行在 JVM 上，并将 Lisp 的强大功能与 Java 平台的广泛库集成起来。
  5. JRuby：Ruby 的一个实现，它可以编译成 Java 字节码并在 JVM 上运行。
  6. Jython：Python 的一个实现，同样可以编译成 Java 字节码并在 JVM 上执行。

  因此，JVM 并不局限于只能执行由 Java 编写的字节码文件，而是支持多种编程语言生成的字节码文件。
- 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

2. 作用

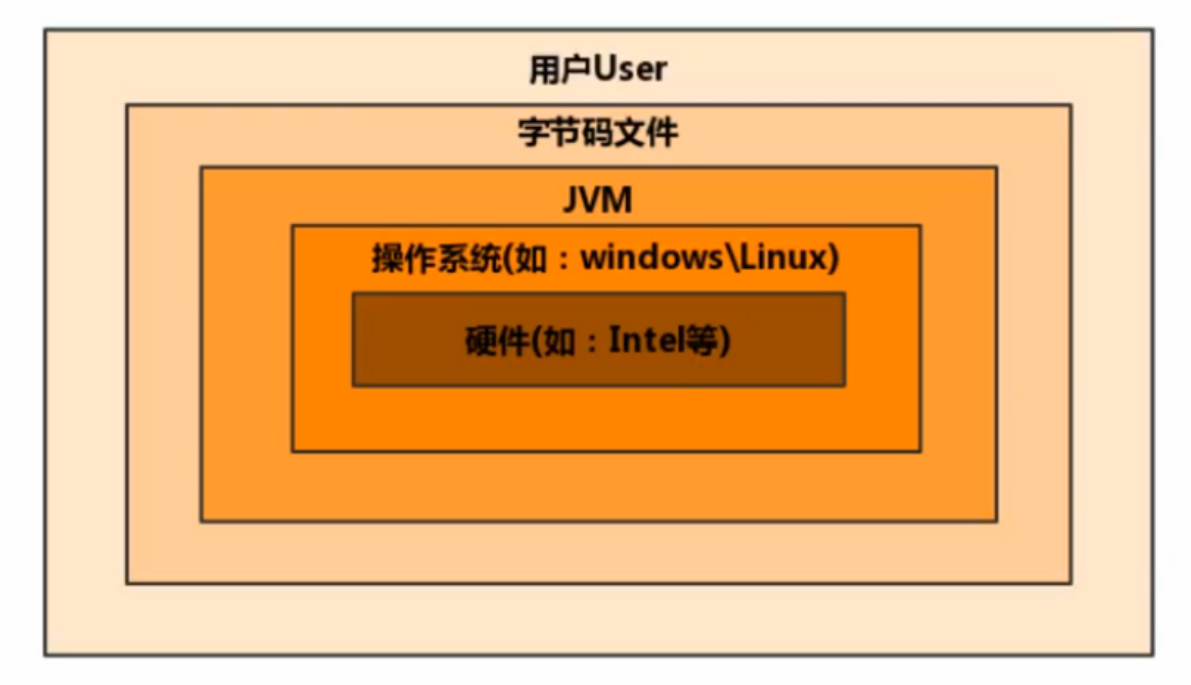
Java虚拟机就是二进制字节码的运行环境，负责装载字节码到其内部，解释编译为对应平台上的机器指令执行。每一条Java指令，Java虚拟机规范中都处理结果放在哪里。有详细定义，如怎么取操作数，怎么处理操作数，处理结果放在哪里。（但是，由于内存等管理这些都是JVM帮我们做好了，有很大的风险）

3. 特点

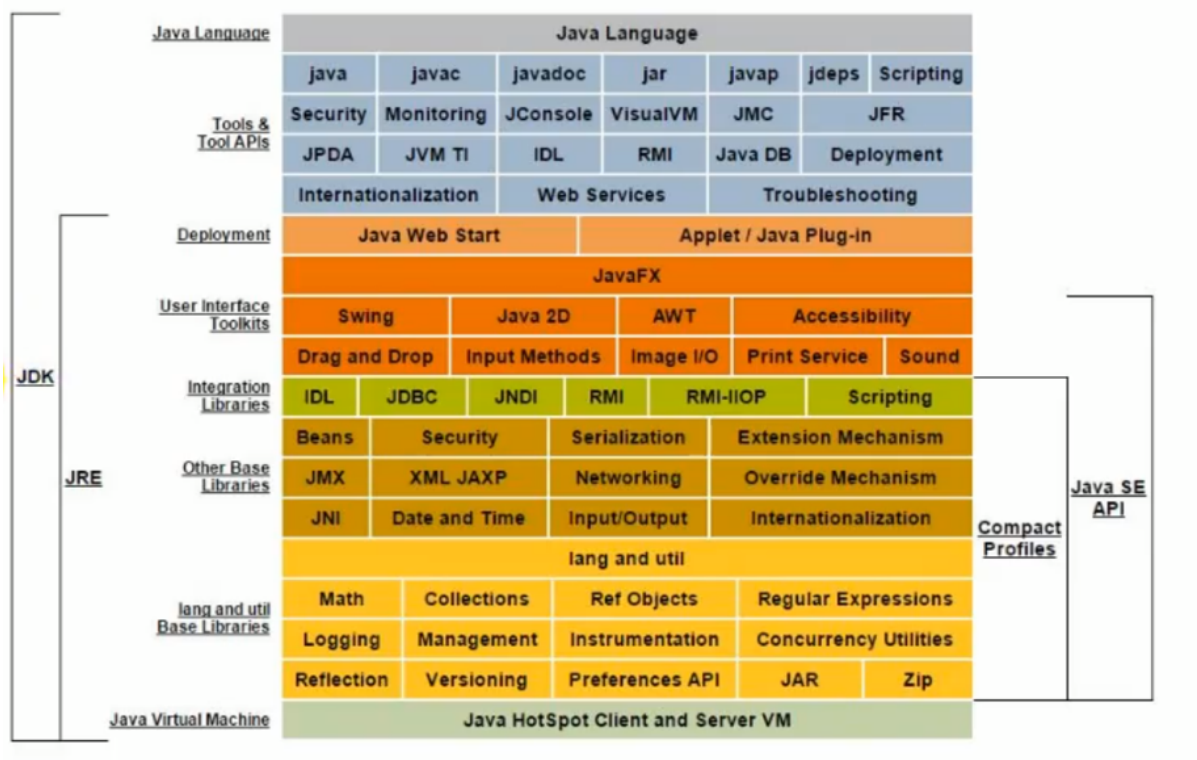
- 一次编译，到处运行
- 自动内存管理
- 自动垃圾回收功能

3.3 JVM位置

内两层为系统虚拟机（模拟整个计算机硬件，对物理计算机的仿真），JVM为程序虚拟机（专门为执行单个计算机程序而设计，JVM是运行在操作系统之上的，它与硬件没有直接的交互）。



JVM整体架构



4、JVM的整体结构

4.1 HotSpot VM

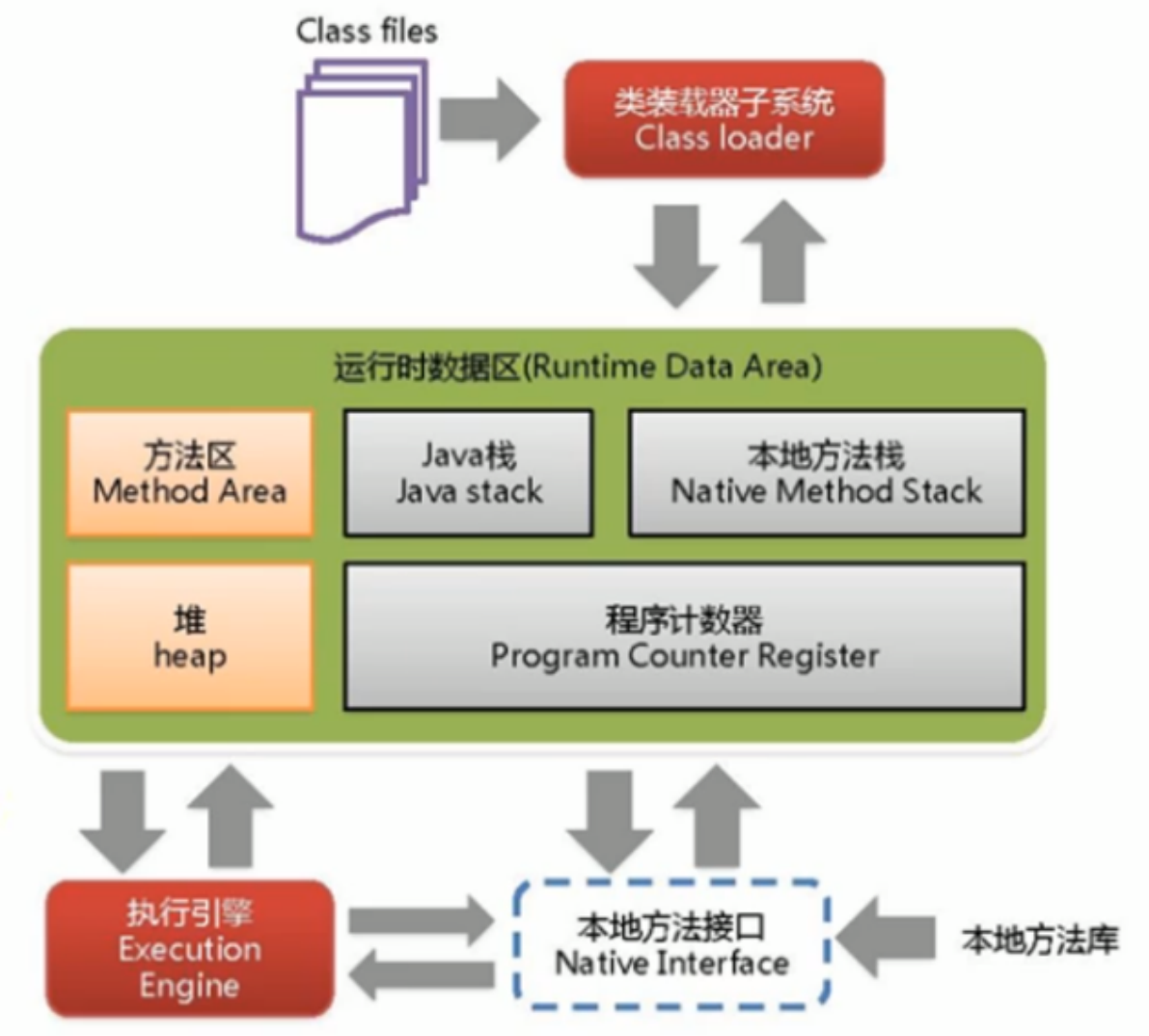
HotSpot VM是Java平台上的一种虚拟机，是oracle jdk 和 opean jdk共用的虚拟机，由Oracle开发和维护。它是Java虚拟机（JVM）规范的一种实现，用于执行Java程序。HotSpot VM是目前广泛使用的Java虚拟机之一，它提供了高性能的垃圾回收、即时编译（JIT）等特性，使得Java程序能够在各种环境下运行并获得良好的性能表现。

4.2 特点

- 它采用解释器与即时编译器并存的架构。
- HotSpot VM是目前市面上高性能虚拟机的代表作之一。
- 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。

4.3 HotSpot VM内存结构图说明

HotSpot VM内存结构图如下所示



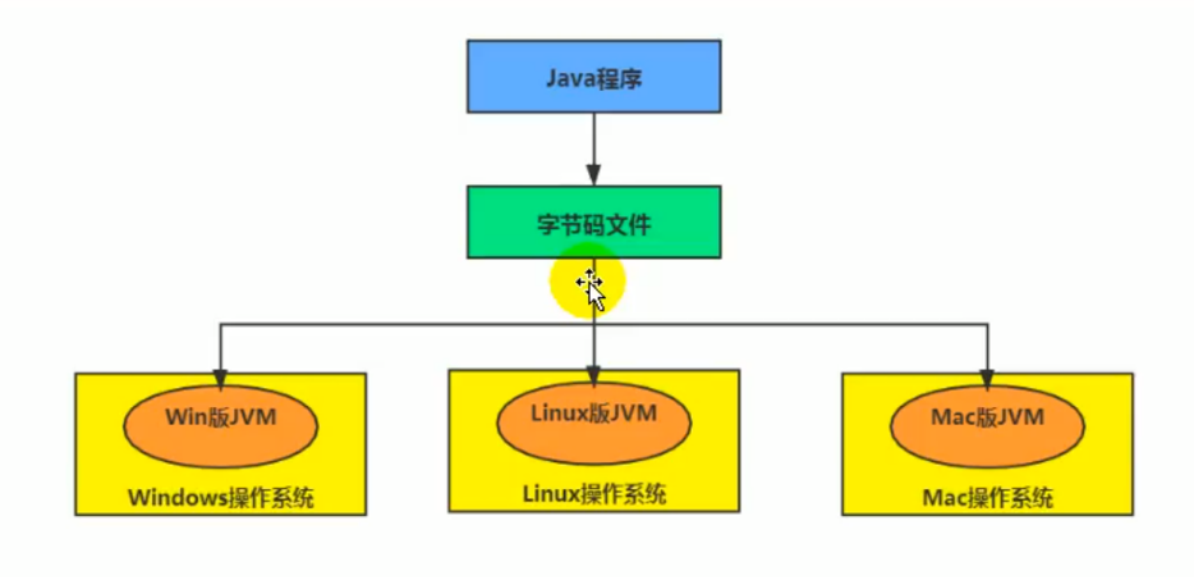
说明：

从上往下依，java虚拟机是用来解释运行字节码文件，因此入口/原材料就是Class files（字节码文件）。通过类装载器子系统（类装载器）将字节码文件加载到内存中，生成一个大的class对象实例（其中涉及加载、链接、初始化），在运行时数据区将大的class对象实例放到对应的内容，其中方法区、堆是所有线程共享的，而Java栈、本地方法栈和程序计数器是每个线程独有一份的。执行引擎分为解析器（将加载到内存中的内容进行解释运行）、及时编译器（对热点代码提前进行编译，注意java到字节码是编译期的前端，这里属于编译期的后端，用于将高级语言翻译成机器语言，即字节码到机器语言）和垃圾回收器。

5、Java代码执行流程

5.1 简单概述

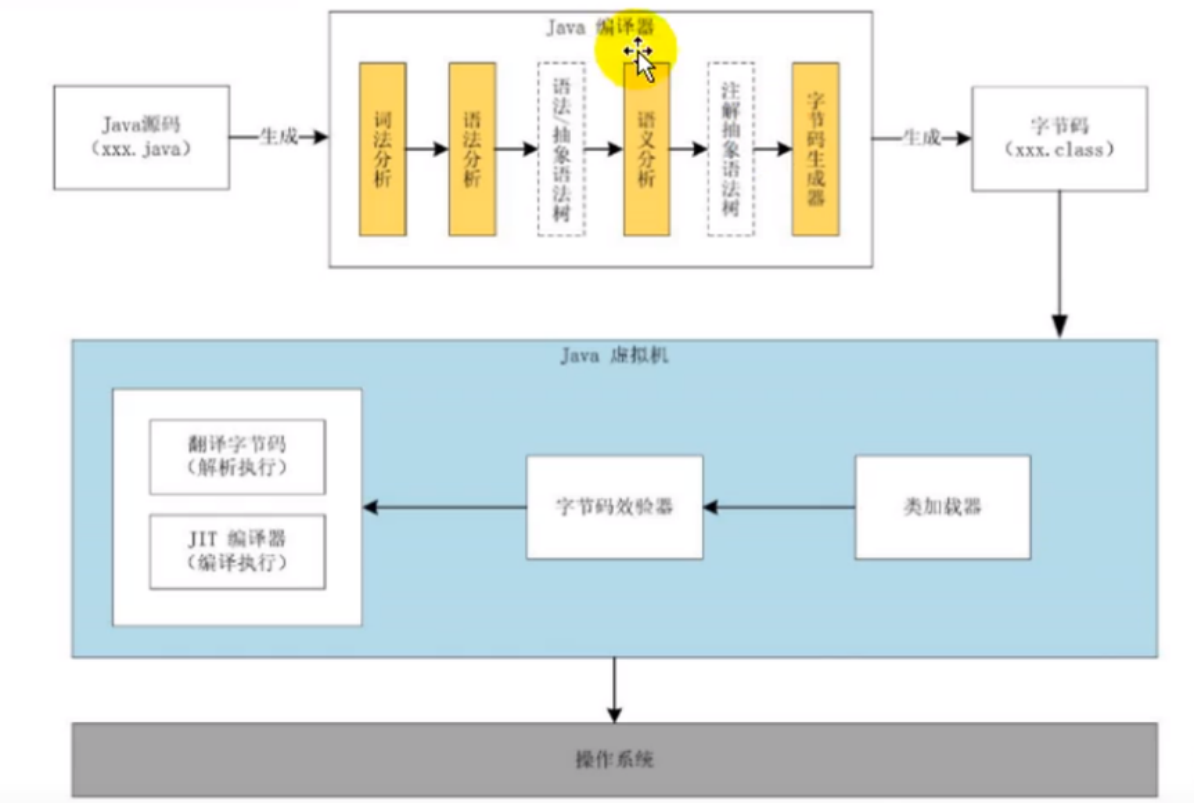
Java程序首先编译为字节码文件，然后字节码文件在对应的操作系统上的JVM中进行编译解析运行。



5.2 展开说明

扩展：只要字节码符合JVM的编译规范，JVM虚拟机就可以编译执行，并且我们可以自己去设计一个语言（包括语言源码和编译成字节码的方式）

整个java的执行流程如下图所示



注意：CPU/计算机只能识别机器指令，我们首先要把高级语言转换成汇编预约，再将汇编预约转换成机器指令，而这个过程就是JVM中执行引擎来完成的。（世面上主流的虚拟机都采用了解释执行、及时编译并行的方式）

- 解释器：解释字节码是实时将字节码文件转化为机器指令
- JIT编译器：会将热点代码转化为机器指令，并缓存起来，放在方法区中，下次可以直接调用。其主要用于提升jvm的性能。

其中涉及到两次编译，第一次是将java源码编译为字节码文件，第二次编译试讲字节码文件编译为机器指令。

6、JVM的架构模型

6.1 栈的指令集架构和寄存器的指令集架构

栈的指令集架构（Stack-based Instruction Set Architecture）和寄存器的指令集架构（Register-based Instruction Set Architecture）是两种不同的计算机指令集设计范式，它们在指令执行和操作数据的方式上有所不同。

1. 栈的指令集架构

说明

在栈的指令集架构中，操作数通常被压入栈中，然后通过栈顶的操作数进行操作。指令集中的操作通常被设计为将操作数从栈中弹出、执行操作，然后将结果压入栈中。这种架构通常不需要显式地指定操作数的位置，而是通过栈的先进先出（FIFO）机制来管理数据。

特点

- 设计和实现更简单，适用于资源受限的系统

资源受限的系统指的是具有限制性能或功能的计算机系统或设备。这些限制可能是由硬件、软件或环境因素引起的。

1. **计算资源**：包括处理器性能、内存容量、存储空间等。例如，嵌入式系统通常具有较低的处理器性能和内存容量，因此被认为是资源受限的系统。
2. **能源资源**：包括电池容量、功耗限制等。例如，移动设备如智能手机和平板电脑通常有限的电池容量，因此需要在使用资源方面进行有效管理。
3. **网络资源**：包括带宽、连接数限制等。例如，在云计算环境中，虚拟机实例可能受到网络带宽和连接数的限制。
4. **软件资源**：包括许可证限制、并发连接数限制等。例如，某些商业软件可能根据许可证限制同时连接的用户数或设备数。
5. **环境资源**：包括温度、湿度等环境条件的限制。例如，在工业控制系统中，由于环境条件不稳定，设备可能受到温度或湿度的限制。

- 避开了寄存器的分配难题:使用零地址指令方式分配。
    - 零地址指令方式分配：在零地址指令方式中，指令本身不包含操作数的地址或位置信息，这种方式下，指令被设计为直接作用于栈或寄存器中的操作数，因此不需要指定操作数的位置。典型的例子是栈式计算机中的指令，例如将栈顶两个元素相加并将结果压入栈顶的指令。
    - 一地址指令方式分配：在一地址指令方式中，指令包含一个操作数的地址或位置信息。指令操作的是指定地址上的数据，而不是直接操作寄存器或栈中的数据。这种方式下，指令通常具有固定的格式，其中一个字段用于指定操作数的地址或位置。典型的例子包括取址操作，例如从内存中加载数据到寄存器中的指令。
    - 二地址指令方式分配：在二地址指令方式中，指令包含两个操作数的地址或位置信息。指令操作的是指定地址上的两个数据，并且通常将结果存储在其中一个地址中。这种方式下，指令的格式通常具有固定的格式，其中两个字段分别指定两个操作数的地址。典型的例子包括加法操作，例如将两个寄存器中的数据相加并将结果存储在另一个寄存器中的指令。

- 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。

  栈的指令集一般是8位字节对为一个基本单位。
- 不需要硬件支持，可移植性更好，更好实现跨平台。

2. 寄存器的指令集架构

说明

在寄存器的指令集架构中，操作数通常存储在寄存器中，指令集中的操作则直接在寄存器上执行。每条指令通常需要指定操作数所在的寄存器，或者通过寄存器的隐含含义来操作数据。

寄存器是计算机中用于存储和处理数据的一种特殊类型的内存单元。与普通内存单元（如RAM）相比，寄存器通常是位于CPU内部的高速存储器，用于临时存储指令、数据和中间结果。寄存器具有非常快速的访问速度，通常以纳秒级的速度进行访问，这使得它们在处理器内部的数据传输和计算中起着重要作用。

特点

- 典型的应用是x86的二进制指令集，比如传统的PC以及Androia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

  指令集采用十六位双字节对齐方式
- 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

6.2 指令集架构具体示例对比

执行2+3这种逻辑操作，其指令分别如下：

栈的指令集架构（JVM）：

```
iconst_2 //常量2入栈
istore_1
iconst 3 //常量3入栈iconst 3
istore 2
iload_1
iload 2
iadd    //常量2、3出栈，执行相加
istore_8 //结果5入栈
```

寄存器的指令集架构：

```
mov eax,2 //将eax寄存器的值设为1
add eax,3 //使eax寄存器的值加3
```

6.3 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。

7、JVM的生命周期

7.1 JVM生命周期概述

1. 虚拟机的启动

Java虚拟机的启动时通过引导类加载器（bootstrap class loader）创建一个初始类（initial class）来完成的，这个类是由虚拟机的具体实现指定的。

2. 虚拟机的执行

- 一个运行中的Java虚拟机有着一个清洗的任务：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。（我们可以通过Java JDK中的一个命令行工具`jps`是Java Virtual Machine Process Status Tool的缩写，用于列出当前系统中正在运行的Java进程，获取正在执行的Java进程的名称和它们的进程ID（PID））

3. 虚拟机的退出

如下几种情况，会造成虚拟机的退出：

- 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Java虚拟机进程终止
- 某线程调用Runtime类（即JVM运行时数据区对应的单例类,Runtime类允许应用程序与其运行的环境进行交互，例如执行系统命令、查询系统信息等）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, 本地方法接口）规范描述了用JNI Invocation API来加载或卸载Java虚拟机时，Java虚拟机退出的情况。

8、JVM发展历程

8.1 Sun Classic VM

1. 说明

- 早在1996年Java1.0版本的时候，sun公司发布了一款名为SunclassicVM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，没有即时编译器，无法对热点代码进行缓存，需要逐行对字节码文件进行解析，效率慢、程序性能低，但是其响应的时间是非常快的。

- 如果使用JIT编译器，就需要进行外挂。但是一旦使用了JIT编译器，JIT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。

2. 注意

在实际过程中，如果只有JIT（Just-In-Time ，即时编译器）编译器也是不行的，因为其在程序运行前，需要等待时间，去对一些热点代码进行处理，并缓存，最好的方式应该是JIT编译器和解释器一起使用。

举个例子：从A点到B点，如果只有解释器，没有即是编译器，那么就想到与步行，从一开始就直接从A点出发，向B点前进。如果只有即是编译器，没有解释器，那么就好比等公交车，需要花费一点时间去进行等待，然后再上车，过程可能会超过之前步行的人，并且在行进过程中，可能涉及到换乘，需要频繁等待。如果只有解释器，也有即是编译器，那么我们可以在路近的节点选择步行，远的选择公交车，更加灵活，高效。

8.2 Exact VM

1. 说明

为了解决上一个虚拟机问题，jdk1.2时，sun公司提供了此虚拟机。

Exact VM（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（一般只知道在内存中的地址，而不知道是什么类型），具备现代高性能虚拟机的雏形
- 热点探测（即JIT编译器对热点代码进行编译、缓存）
- 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classicm
- 英雄气短，终被Hotspot虚拟机替换

8.3 HotSpot VM

1. 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
- JDK1.3时，Hotspot vM成为默认虚拟机

2. 说明

- 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JDK6，还是使用比例较多的JDK8中，默认的虚拟机都是HotSpot
  - Sun/oracle JDK和OpenJDK的默认虚拟机
  - 因此后续内容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（解释器）与最佳执行性能（编译器）中取得平衡

8.4 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IVM。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号)和硬件成本的减少(达50号)。
- 优势:全面的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JRockitRealime提供以毫秒或微秒级的JV响应时间，适合财务、军事指挥、电信网络的需要
    - MissionContro1服务套件，它是一组以极低的开销来监控、管理和分析生产环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。
- 高斯林:目前就职于谷歌，研究人工智能和水下机器人

8.5 IBM的I9

- 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、桌面应用、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服务器之一，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 Ecilpse openJ9

8.6 KVM和CDC/CLDC Hotspot

- Oracle在Java ME(Java平台的一个分支，专门设计用于嵌入式设备、移动设备和其他资源受限的环境中)产品线上的两款虚拟机为:CDC/CLDC HotSpot Implementation VM
- KVM(Kilobyte)是CLDC-HI早期产品
- 目前移动领域地位尴尬，智能手机被Android和ios二分天下
- KVM简单、轻量、高度可移植，而向更低端的设备上还维持自己的一片市场
    - 智能控制器、传感器
    - 老人手机、经济欠发达地区的功能手机
- 所有的虚拟机的原则:一次编译，到处运行。

8.7 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Azu1 VM和BEA Liquid V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。
- Azu1 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Azu1 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zingJVM，可以在通用x86平台上提供接近于vega系统的特性。

8.8 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 v即是现在的JRockit vE(virtual Editin),Liquid eVM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 项目也停止了。

8.9 Apache Harmony

- Apache也曾经推出过与JDK1.5和JDK1.6兼容的Java运行平台Apache Harmony.
- 它是IBM和Inte1联合开发的开源JM，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。

8.10 Microsoft IM

- 微软为了在IE3浏览器中支持Java pplets，开发了Microsoft JVM。
- 只能在window平台下运行。但确是当时windows下性能最好的Java V。
- 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windowsXP SP3中抹掉了其VM。现在windows上安装的jdk都是Hotspot。

8.11 taobaoJVM

- 由AliJVM 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 V 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ava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 JV 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IVM版本全部替换了。

8.12 Dalvik VM

- 谷歌开发的，应用于Android系统，并在Android2.2中提供了JIT，发展迅猛。，它没有遵循 Java
- Dalvik VM 只能称作虚拟机，而不能称作“Java 拟机”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(DalvikExecutable)文件。执行效率比较高。
    - 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ava pl。
- Android 5.0 使用文持提前编译(Ahead of Time Compilation，aoT)的ART VM替换Dalvik VM。

8.13 Graal VM

- 2018年4月，0racle abs公开了GraalVM，号称"Run Programs raster Anywhere"，勃勃野心。与1995年java的”write once,run anywhere"遥相呼应。
- Graal VM在HotSpot VM基础上增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工具集快速构建面问一种新谐言的解释器。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 V希望最大。但是Java的软件生态没有丝毫变化。