一. JVM基础概念

JVM的中文名称叫Java虚拟机，它是由软件技术模拟出计算机运行的一个虚拟的计算机。

JVM也充当着一个翻译官的角色，我们编写出的Java程序，是不能够被操作系统所直接识别的，这时候JVM的作用就体现出来了，它负责把我们的程序翻译给系统“听”，告诉它我们的程序需要做什么操作。

我们都知道Java的程序需要经过编译后，产生.class文件，JVM才能识别并运行它，JVM针对每个操作系统开发其对应的解释器，所以只要其操作系统有对应版本的JVM，那么这份Java编译后的代码就能够运行起来，这就是Java能一次编译，到处运行的原因。

二. JVM生命周期

JVM在Java程序开始执行的时候，它才运行，程序结束的时它就停止。

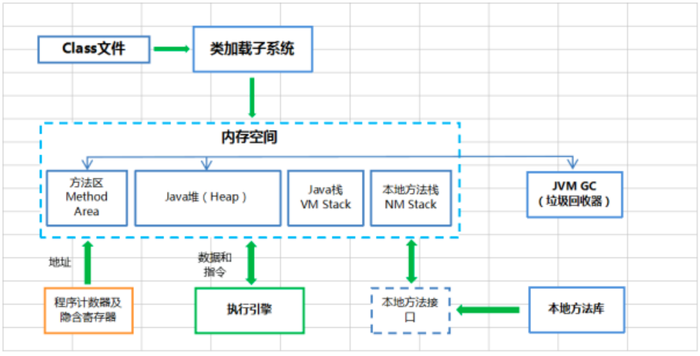
一个Java程序会开启一个JVM进程，如果一台机器上运行三个程序，那么就会有三个运行中的JVM进程。

JVM中的线程分为两种：守护线程和普通线程

- 守护线程是JVM自己使用的线程，比如垃圾回收（GC）就是一个守护线程。
- 普通线程一般是Java程序的线程，只要JVM中有普通线程在执行，那么JVM就不会停止。

权限足够的话，可以调用exit()方法终止程序。

三. JVM结构



四. JVM启动过程

1. **JVM的装入环境和配置**
   在学习这个之前，我们需要了解一件事情，就是JDK和JRE的区别。
   1). JDK是面向开发人员使用的SDK，它提供了Java的开发环境和运行环境，JDK中包含了JRE。
   2). JRE是Java的运行环境，是面向所有Java程序的使用者，包括开发者。

2. **装载JVM**
   通过第一步找到JVM的路径后，Java.exe通过LoadJavaVM来装入JVM文件。
   LoadLibrary装载JVM动态连接库，然后把JVM中的到处函数JNI_CreateJavaVM和JNI_GetDefaultJavaVMIntArgs 挂接到InvocationFunction 变量的CreateJavaVM和GetDafaultJavaVMInitArgs 函数指针变量上。JVM的装载工作完成。

3. **初始化JVM，获得本地调用接口**
   调用InvocationFunction -> CreateJavaVM也就是JVM中JNI_CreateJavaVM方法获得JNIEnv结构的实例。

4. **运行Java程序**
   JVM运行Java程序的方式有两种：jar包 与 Class
   1). **运行jar 的时候**，Java.exe调用GetMainClassName函数，该函数先获得JNIEnv实例然后调用JarFileJNIEnv类中getManifest()，从其返回的Manifest对象中取getAttrebutes("Main-Class")的值，即jar 包中文件：META-INF/MANIFEST.MF指定的Main-Class的主类名作为运行的主类。之后main函数会调用Java.c中LoadClass方法装载该主类（使用JNIEnv实例的FindClass）。
   2). **运行Class的时候**，main函数直接调用Java.c中的LoadClass方法装载该类。

五. Class文件

Class文件由Java编译器生成，我们创建的Java文件在经过编译器后，会变成.class的文件，这样才能被JVM所识别并运行。

六. 类加载系统

类加载子系统也可以称之为类加载器，JVM默认提供三个类加载器：

1. **BootStrap ClassLoader:** 称之为启动类加载器，是最顶层的类加载器，负责加载JDK中的核心类库，如 rt.jar、resources.jar、charsets.jar等。
2. **Extension ClassLoader:** 称之为扩展类加载器，负责加载Java的扩展类库，默认加载$JAVA_HOME中jre/lib/*.jar 或 -Djava.ext.dirs指定目录下的jar包。
3. **App ClassLoader**：称之为系统类加载器，负责加载应用程序classpath目录下所有jar和class文件。

BootStrap ClassLoader 不是一个普通的Java类，它底层由C++编写，已嵌入到了JVM的内核当中，当JVM启动后，BootStrap ClassLoader 也随之启动，负责加载完核心类库后，并构造Extension ClassLoader 和App ClassLoader 类加载器。

类加载器子系统不仅仅负责定位并加载类文件，它还严格按照以下步骤做了很多事情
1.加载：寻找并导入Class文件的二进制信息
2.连接：进行验证、准备和解析
   1）验证：确保导入类型的正确性
   2）准备：为类型分配内存并初始化为默认值
   3）解析：将字符引用解析为直接引用
3.初始化：调用Java代码，初始化类变量为指定初始值

七. JVM执行引擎

Java虚拟机相当于一台虚拟的“物理机”，这两种机器都有代码执行能力，其区别主要是物理机的执行引擎是直接建立在处理器、硬件、指令集和操作系统层面上的。而JVM的执行引擎是自己实现的，因此程序员可以自行制定指令集和执行引擎的结构体系，因此能够执行那些不被硬件直接支持的指令集格式。

在JVM规范中制定了虚拟机字节码执行引擎的概念模型，这个模型称之为JVM执行引擎的统一外观。JVM实现中，可能会有两种的执行方式：解释执行（通过解释器执行）和编译执行（通过即时编译器产生本地代码）。有些虚拟机只采用一种执行方式，有些则可能同时采用两种，甚至有可能包含几个不同级别的编译器执行引擎。

八. 本地方法接口（JNI）

JNI是Java Native Interface的缩写，它提供了若干的API实现了Java和其他语言的通信（主要是C和C++）。

1. JNI的适用场景

当我们有一些旧的库，已经使用C语言编写好了，如果要移植到Java上来，非常浪费时间，而JNI可以支持Java程序与C语言编写的库进行交互，这样就不必要进行移植了。或者是与硬件、操作系统进行交互、提高程序的性能等，都可以使用JNI。需要注意的一点是需要保证本地代码能工作在任何Java虚拟机环境。

2. **JNI的副作用**

一旦使用JNI，Java程序将丢失了Java平台的两个优点：
- 程序不再跨平台，要想跨平台，必须在不同的系统环境下程序编译配置本地语言部分。
- 程序不再是绝对安全的，本地代码的使用不当可能会导致整个程序崩溃。一个通用规则是，调用本地方法应该集中在少数的几个类当中，这样就降低了Java和其他语言之间的耦合。