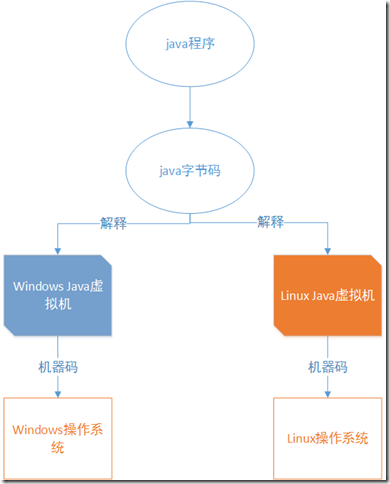
深入理解JVM（一）—基本原理

1. **前言**

JVM一直是java知识里面进阶阶段的重要部分，如果希望在java领域研究的更深入，则JVM则是如论如何也避开不了的话题，本系列试图通过简洁易读的方式，讲解JVM必要的知识点。

1. **运行流程**

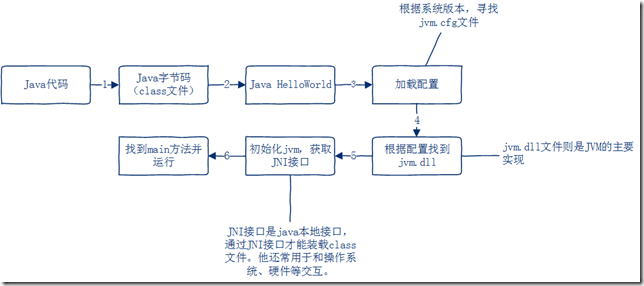
我们都知道java一直宣传的口号是：一次编译，到处运行。那么它如何实现的呢？我们看下图：



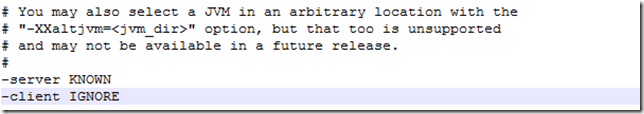
java程序经过一次编译之后，将java代码编译为字节码也就是class文件，然后在不同的操作系统上依靠不同的java虚拟机进行解释，最后再转换为不同平台的机器码，最终得到执行。这样我们是不是可以推演，如果要在mac系统上运行，是不是只需要安装mac java虚拟机就行了。那么了解了这个基本原理后，我们尝试去做更深的研究，一个普通的java程序它的执行流程到底是怎样的呢？例如我们写了一段这样的代码：

public class HelloWorld { public static void main(String[] args) { System.out.print("Hello world"); } }

这段程序从编译到运行，最终打印出“Hello world”中间经过了哪些步骤呢？我们直接上图：



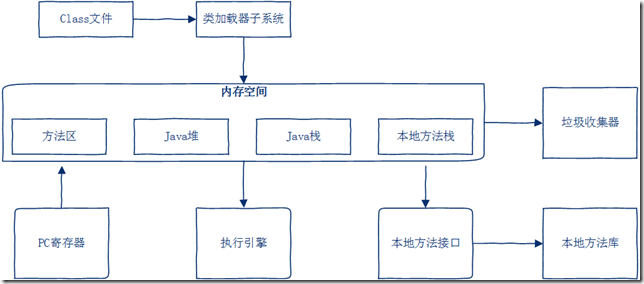
java代码通过编译之后生成字节码文件（class文件），通过：java HelloWorld执行，此时java根据系统版本找到jvm.cfg，各位可以搜索一下自己电脑上的jvm.cfg文件在哪，它会根据你的系统版本放在不同的位置，比如我的这个文件就在：C:\Program Files\Java\jdk1.8.0\_101\jre\lib\amd64\jvm.cfg，打开看一下：



这是我电脑上的文件，其中-server KNOWN就表示名称为server的jvm可用。如果这时你搜索一下你电脑上jvm.dll，你就会发现它一定在你的某个server目录下，比如我的：C:\Program Files\Java\jdk1.8.0\_101\jre\bin\server\jvm.dll。简而言之就是通过jvm.cfg文件找到对应的jvm.dll，jvm.dll则是java虚拟机的主要实现。接下来会初始化JVM,并且获取JNI接口，什么是JNI接口，就是java本地接口，你想啊java被编译成了class文件，JVM怎么从硬盘上找到这个文件并装载到JVM里呢，就是通过JNI接口（它还常用于java与操作系统、硬件交互），找到class文件后并装载进JVM，然后找到main方法，最后执行。

1. **JVM基本结构**

可能通过上面的描述，大家对JVM运行流程有了一个粗略的认识，那么JVM内部到底是怎么执行一个class文件的呢，也就是上图中最后一步第6步的内部细节是怎样的呢？要了解这个问题，我们首先得看一下JVM的内部结构：



从这个结构不难看出，class文件被jvm装载以后，经过jvm的内存空间调配，最终是由执行引擎完成class文件的执行。当然这个过程还有其他角色模块的协助，这些模块协同配合才能让一个java程序成功的运行，下面就详细介绍这些模板，它们也是后面学习jvm最重要的部分。

1. **内存空间**

JVM内存空间包含：方法区、java堆、java栈、本地方法栈。

方法区是各个线程共享的区域，存放类信息、常量、静态变量。

java堆也是线程共享的区域，我们的类的实例就放在这个区域，可以想象你的一个系统会产生很多实例，因此java堆的空间也是最大的。如果java堆空间不足了，程序会抛出OutOfMemoryError异常。

java栈是每个线程私有的区域，它的生命周期与线程相同，一个线程对应一个java栈，每执行一个方法就会往栈中压入一个元素，这个元素叫“栈帧”，而栈帧中包括了方法中的局部变量、用于存放中间状态值的操作栈，这里面有很多细节，我们以后再讲。如果java栈空间不足了，程序会抛出StackOverflowError异常，想一想什么情况下会容易产生这个错误，对，递归，递归如果深度很深，就会执行大量的方法，方法越多java栈的占用空间越大。

本地方法栈角色和java栈类似，只不过它是用来表示执行本地方法的，本地方法栈存放的方法调用本地方法接口，最终调用本地方法库，实现与操作系统、硬件交互的目的。

PC寄存器，说到这里我们的类已经加载了，实例对象、方法、静态变量都去了自己改去的地方，那么问题来了，程序该怎么执行，哪个方法先执行，哪个方法后执行，这些指令执行的顺序就是PC寄存器在管，它的作用就是控制程序指令的执行顺序。

执行引擎当然就是根据PC寄存器调配的指令顺序，依次执行程序指令。

1. **来源**

http://www.cnblogs.com/leefreeman/p/7344460.html