1. 动态类型语言支持
2. 概述

动态类型语言的关键特征是它的类型检查的主体过程是在运行期而不是编译期进行的。Java语言在JDK7之前通过编译将方法的符号引用作为方法调用指令的参数存储到Class文件中，这个符号引用包含了该方法定义在哪个具体类型中，方法的名字以及参数顺序，参数类型和方法返回值等信息，通过这个符号引用，Java虚拟机可以翻译出该方法的直接引用。

对于动态类型语言而言，在编译期是无法确定方法所在的具体类型（即方法接受者不固定），变量无类型而变量值有类型，这个特点是动态类型语言的核心特征。

1. 动态与静态语言的优缺点

静态类型语言能够在编译器确定变量类型，编译器可以进行全面的类型检查，利于稳定性；

动态类型语言在运行期才确定类型，可以为开发者提供极大的灵活性，提升开发效率；

1. 原有方法调用指令

JDK7以前的方法调用指令中，如invokevirtual, invokespecial, invokestatic, invokeinterface的第一个参数都是被调用的方法的符号引用，要实现Java的动态类型语言，就要找到具体的类型，可以在编译时留个占位符类型，运行时动态生成字节码实现具体类型到占位符类型的适配。但是，这样的内存消耗是很大的。

1. 方法句柄
2. 概述

通过方法句柄可以获取到调用的目标方法，lookup方法在指定类中查找符合给定的方法名称，方法类型并且符合调用权限的方法句柄，返回这个方法句柄，即最终调用方法的一个引用。

1. 方法句柄与反射的区别
2. 方法句柄和反射都是在模拟方法调用，但是反射是在模拟Java代码层次的方法调用，而方法句柄是在模拟字节码层次的方法调用；
3. 反射中所包含的信息比方法句柄中要多多了；
4. 方法句柄是对字节码的方法指令调用的模拟，那么虚拟机在这方面做的各种优化在方法句柄上也可以使用，反射就没有这些优化；
5. 反射仅仅是为Java语言服务的，而方法句柄可服务与所有Java虚拟机之上的语言；
6. invokedynamic指令
7. 概述

该指令的作用是为了解决原有的指令方法分派规则完全固化在虚拟机之中的问题，将如何查找目标方法的决定权从虚拟机转移到具体用户代码中，让用户有更大的自由度。

1. 动态调用点

每一处含有invokedynamic指令的位置都称为动态调用点，这条指令的第一个参数不再是代表方法符号引用的常量，而是JDK7新加入的CONSTANT\_InvokeDynamic\_info常量，该常量中有三种信息：引导方法（该方法存放在新增的BootStrapMethods属性中），方法类型和名称。引导方法有固定的参数，返回值是java.lang.invoke.CallSite对象，这个对象代表了真正要执行的目标方法调用，通过它来执行真正的方法。

1. 字节码解析

从动态调用代码的编译后的字节码看出，原有的方法调用指令已经被替换为了invokedynamic了，它的第一个参数为常量池中的常量，表示方法名称和描述符，第二个参数0是占位符，目的是给常量池缓存留出空间。

1. 字节码指令的执行
2. 解释执行

Java语言在一开始时可以理解为解释型语言，但是随着主流的虚拟机都包含了即时编译器后，Class文件中的代码到底会被解释执行还是编译执行，就只有虚拟机能够决定了。经典的编译思路是执行前先对程序源码进行词法分析和语法分析，将源码转换为抽象语法树，再遍历语法树生成线性的字节码指令流，最后经过解释器的解释执行。

1. 基于栈的指令集

Javac编译器输出的字节码指令流，基本上是一种基于栈的指令集架构，字节码指令流里面的指令大部分都是零地址指令，依赖操作数栈工作，不需要参数，直接使用操作数栈中的数据作为指令的运算输入，指令的运算结果也存储在操作数栈中。如下所示：

iconst\_1

iconst\_1

iadd

istore\_0

优点是可移植，代码相对更加紧凑，字节码中每个字节对应一条指令，而多地址指令集中还需要存放参数，编译器实现更加简单，不需要考虑空间分配，直接在栈上操作。

缺点是在解释执行前提下执行速度更慢，因为指令数量更多，出栈入栈操作产生大量指令，且栈实现在内存中，会产生频繁的内存访问。可以通过采取栈顶缓存的方式，将最常用的操作映射到寄存器中避免内存直接访问。

1. 基于寄存器的指令集

典型的应用时x86的二地址指令集，即主流PC机中物理硬件直接支持的指令集架构，依赖寄存器工作，这种指令都包含两个单独的输入参数，依赖寄存器访问和存储数据。如下所示：

mov eax, 1

add eax, 1