http://www.jianshu.com/p/4e0f5942790b

执行引擎在执行Java代码时候可能会有**解释执行**和**编译执行**两种选择，也可能两者兼备，甚至还可能会包含几个不同级别的编译器执行引擎。

# 栈帧

- 局部变量表
- 操作数
- 动态连接
- 方法返回地址
- 附加信息

# 方法调用

## 解析

方法在程序真正运行之前就有一个可确定的调用版本，并且这个方法的调用版本在运行期是不可变得。

- invokestatic
- invokespecial
- invlkevirtual
- invokeinterface
- invokedynamic

## 分派

- 1.静态分派

**静态类型（Static Type）**

**实际类型（Actual Type）**

静态类型在编译期就可知，实际类型只有在运行期才能确定。

所有依赖j**静态类型**来定位方法执行版本的分派动作称为**静态分派**，典型应用时**方法重载**。

- 2.动态分派

重写（Override）

运行期根据**实际类型**确定方法执行版本的分派过程称为**动态分派**

- 3.单分派与多分派

方法的接收者与方法的参数统称为方法的**宗量**。根据分派基于多少种宗量，可以将分派划分为**单分派**和**多分派**两种。单分派是根据一个宗量对目标方法进行选择，多分派则是根据多于一个宗量对目标方法进行选择。

- 虚拟机动态分派的实现

虚方法表

接口方法表

## 动态类型语言支持

- 1.动态类型语言

什么是动态类型语言？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在编译期。如JavaScript，Lua，静态类型语言如C++、Java

- 2.JDK1.7与动态类型
- 3.java.lang.invoke包

MethodHandle

与反射的区别

- 1）.从本质上讲，Reflection和MethodHandle机制都是在模仿方法的调用，但Reflection是在模拟**Java代码**层次的方法调用,而MethodHandle是在模拟**字节码**层次的方法调用。
- 2）.Reflection中的java.lang.reflect.Method对象远比MethodHandle机制中的java.lang.invoke.MethodHandle对象所包含的信息多。前者是方法在Java一端的全面映像，包含了方法的签名、描述符以及方法属性表中各个属性的Java端表示方式，还包含执行权限等运行期信息。而后者仅仅包含于执行该方法相关的信息。用通俗的话来将，Reflection是重量级的，而MethodHandle是轻量级。

- 3）.由于MethodHandle是对字节码的方法指令调用模拟，所以理论上虚拟机在这方面做的各种优化，在MethodHandle上也应当可以采用类似思路去支持。而通过反射去调用方法则不行。
- 4.invokedynamic指令
- 5.掌控方法分派规则