概述

" "

具体是如何执行的呢?

方法 栈

栈模型

运行时栈帧结构

“栈帧” 用以支持虚拟机进行方法调用和方法执行的数据结构它也是虚拟机运行时数据区中的虚拟机栈的栈元素。

" "

仅仅取决于序源码和具体的虚拟机实现的栈内存布局形式。

线程私有 虚拟机栈

运行状态

概念模型

局部变量表

变量值的存储空间 局部变量 最大容量
Code属性 变量槽(Variable Slot)

或者更小的物理内存来存储
不同的变化

表示对一个对象实例的引用

操作数栈

栈 入栈 " "

动态连接

" "

" "

接引用 静态解析 动态连接

编译期可知，运行期不可变

" "

方法返回地址

返回地址

" !"

" !"

" !"

!!

操作数栈

方法调用

方法调用 “如何确定要调用的方法？”

解析

方法在程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间是不可改变的。进行编译的那一刻就一定确定了

静态方法

私有方法

" "

invokestatic
invokespecial
invokevirtual
invokeinterface
invokedynamic

支持解析的方法

类加载
非-虚方法

非-虚方法 **虚方法**

非-虚方法 **虚方法** **可以被覆写的法都可以称作虚方法**
虚方法

虚方法 **非-虚方法**

invokeinterface指令存在的必要性

invokedynamic指令存在的必要性

"invoke "
如何查找目标方法的决定权从虚拟机转嫁到具体用户代码中

分派

解析调用

静态分派

静态分派
静态分派 方法的版本

"　　!"

"　　!"

"　　!"

!
!
!

静态类型（Static Type）　实际类型（Actual Type）

静态类型在编译期间便是可知的而实际类型的变化结果在运行期间才可确定，编译器在编译程序时并不知道一个对象的实际类型是什。

"

"

动态分派

" "

" "

Java虚拟机是如何判应该调用哪个方法的呢?

实际类型不同 Java虚拟机是如根据实际类型来确定分派方法执行版本呢?

```
#                                   $
    #                                   $      "        "
#                                   $
    #                                   $      "        "

    #                                   $
    #                                   $
#                                   $
    #                                   $      "        "

    #                                   $
```

准备动作

都相符的方法 权限校验

据方法接受者的

际类型来选择方法

遮蔽

单分派与多分派

基于宗量数量的不同，可以将分派分成单分派和多分派

" "

一个宗量

宗量

"　　"

"　　"

"　　"

"　　"

方法选择的过程

因而Java语言的静态分派属于多分派类型

"　"

唯一可以影响虚拟机选择的因素只有该方法的接受者是Father是Son

静态多分派、动态单分派的语言

方法执行

节码指令流 基于栈的指令集架构 寄存器的指令集

两种指令集的区别

不带参数的

优点 可移植 而基于栈的指令集则与硬件无关，因而具有灵活的可移植性。

缺点 基于栈的指令集会稍慢一些 更多。

基于栈的解释器执行过程

上边执行过程实际上仅仅是一个概念模型，虚拟机最终会对行过程做出一系列优化来提高性能，实际运行过程并不会完全按照概念模型的描述来。

总结

解析　分派

参考