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。

"

"

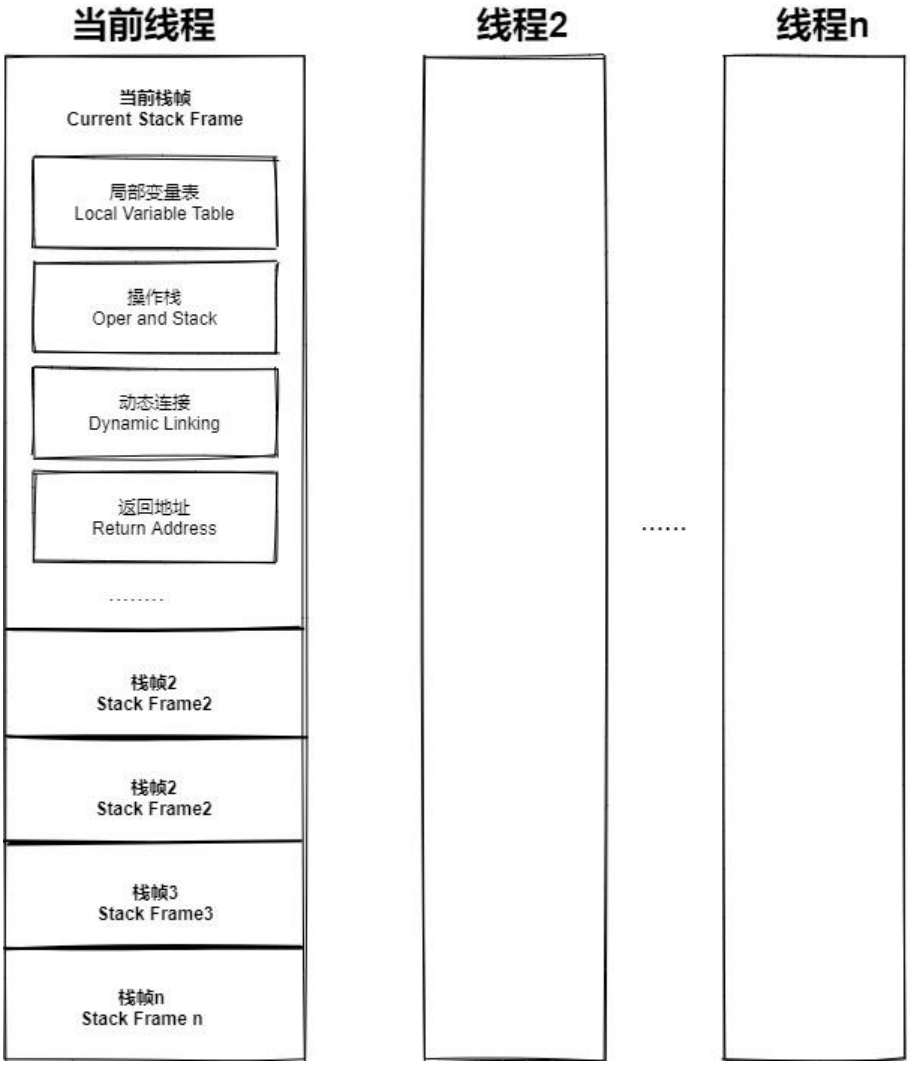
仅仅取决于

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

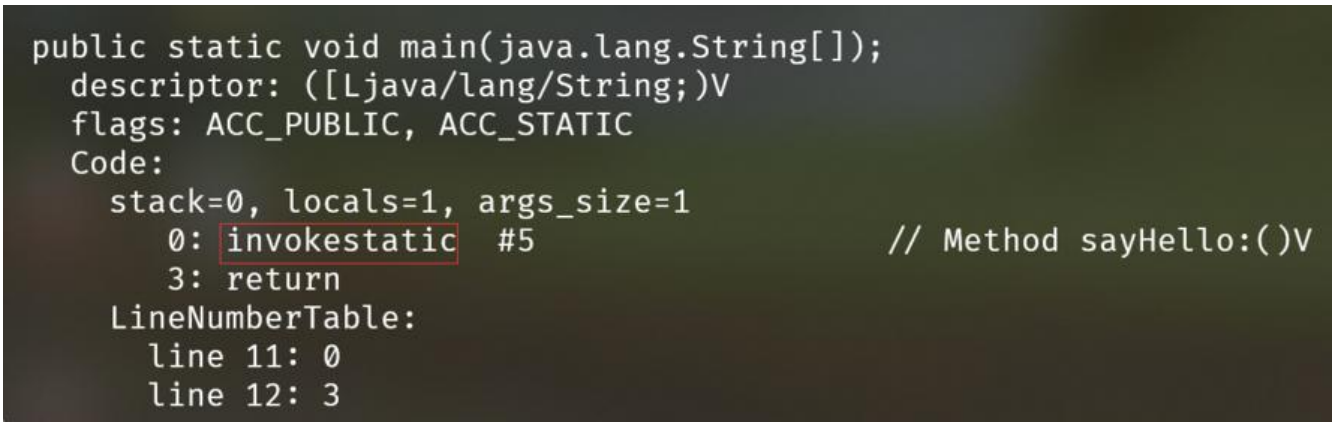
方法调用 “如何确定要调用的方法？”

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么。

"
"

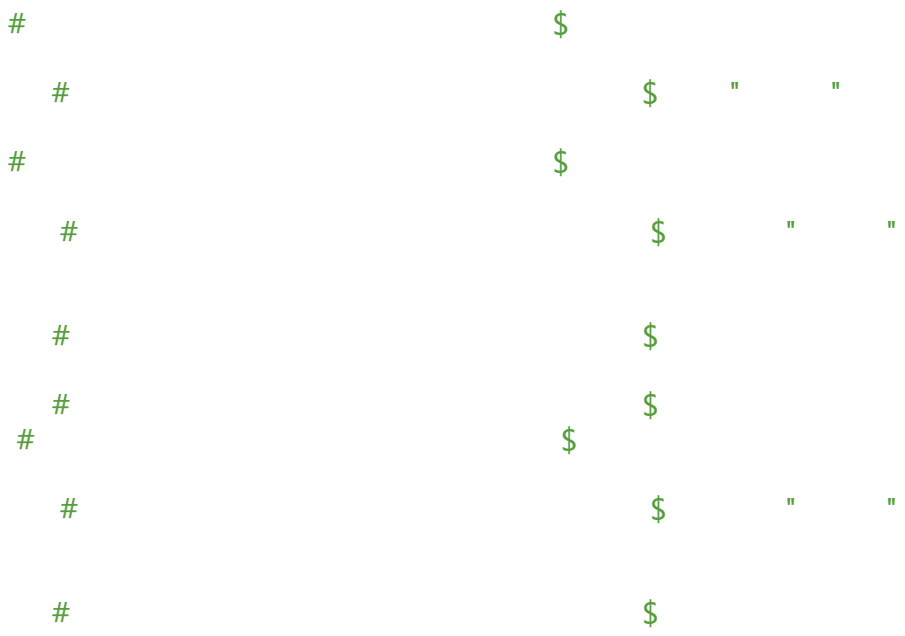
动态分派

" "

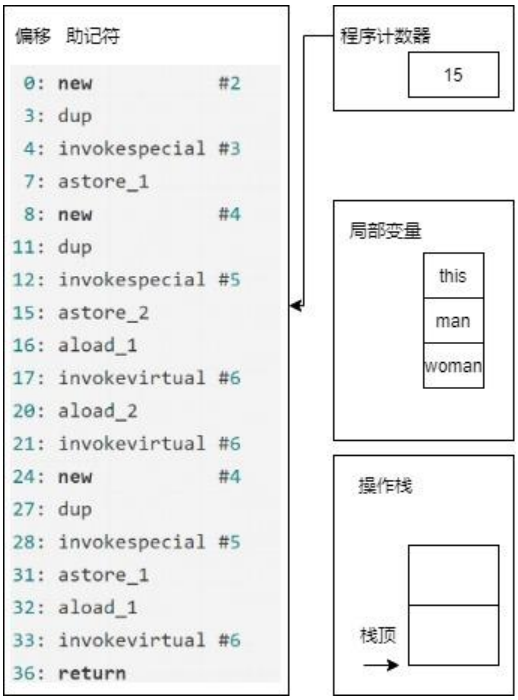
" "

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?



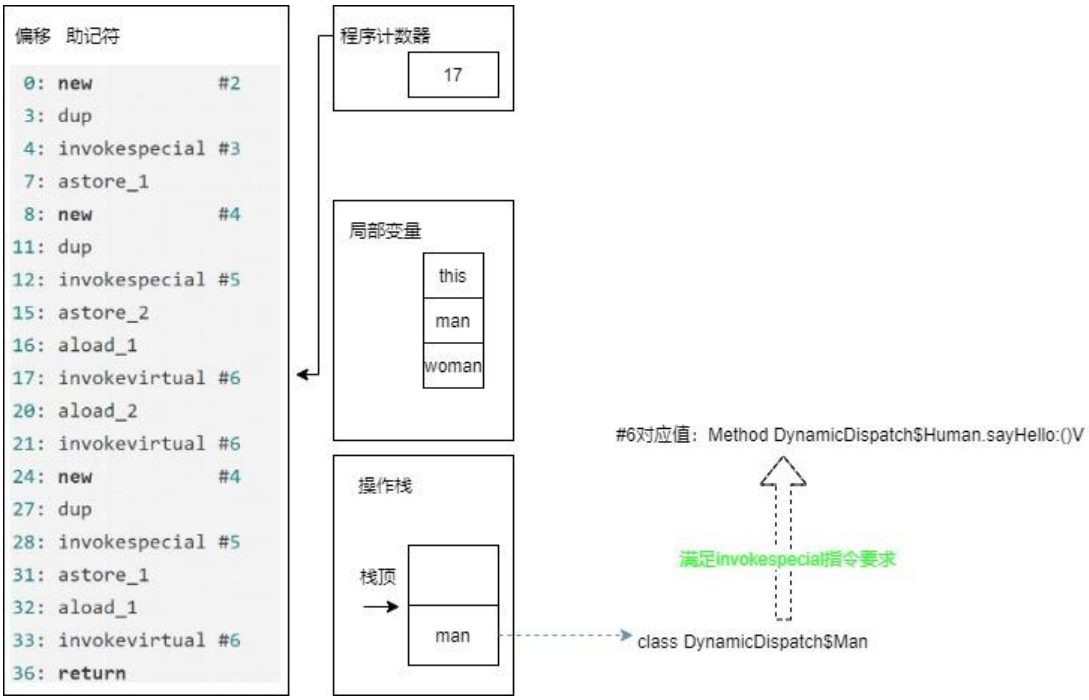
准备动作

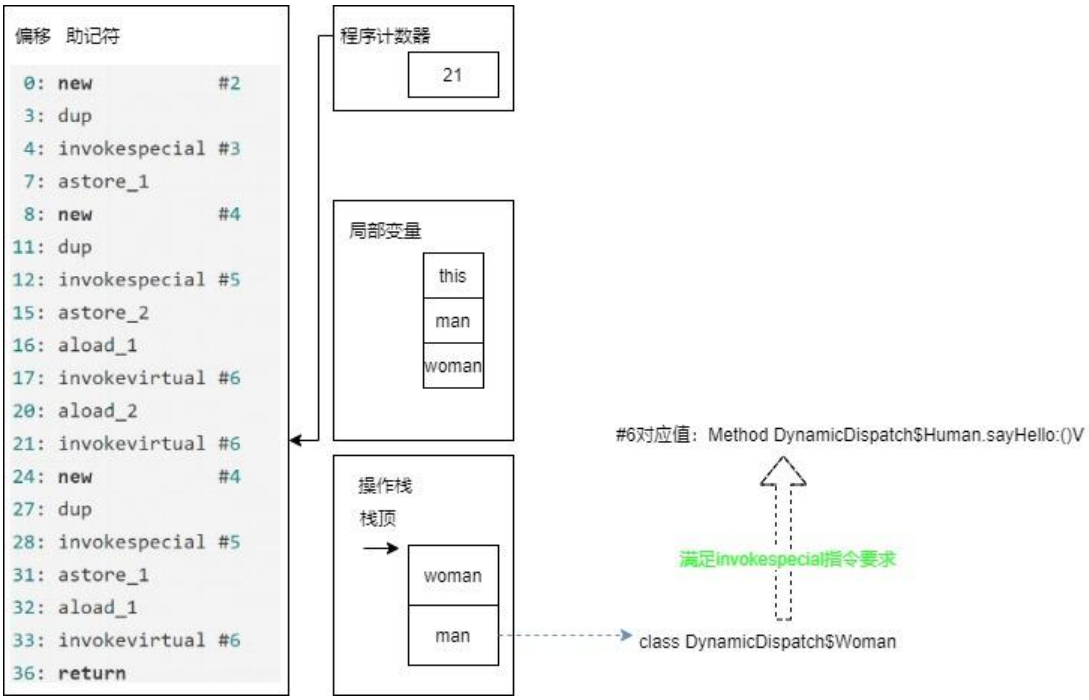


都相符的方法　　权限校验

据方法接受者的

际类型来选择方法


偏移 助记符
0: new #2
3: dup
4: invokespecial #3
7: astore_1
8: new #4
11: dup
12: invokespecial #5
15: astore_2
16: aload_1
17: invokevirtual #6
20: aload_2
21: invokevirtual #6
24: new #4
27: dup
28: invokespecial #5
31: astore_1
32: aload_1
33: invokevirtual #6
36: return
程序计数器
17
局部变量
this
man
woman
操作栈
栈顶
man
#6对应值：Method DynamicDispatch$Human.sayHello:()V
满足invokespecial指令要求
class DynamicDispatch$Man


偏移 助记符
0: new #2
3: dup
4: invokespecial #3
7: astore_1
8: new #4
11: dup
12: invokespecial #5
15: astore_2
16: aload_1
17: invokevirtual #6
20: aload_2
21: invokevirtual #6
24: new #4
27: dup
28: invokespecial #5
31: astore_1
32: aload_1
33: invokevirtual #6
36: return
程序计数器
21
局部变量
this
man
woman
操作栈
栈顶
woman
man
#6对应值：Method DynamicDispatch$Human.sayHello:()V
满足invokespecial指令要求
class DynamicDispatch$Woman

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

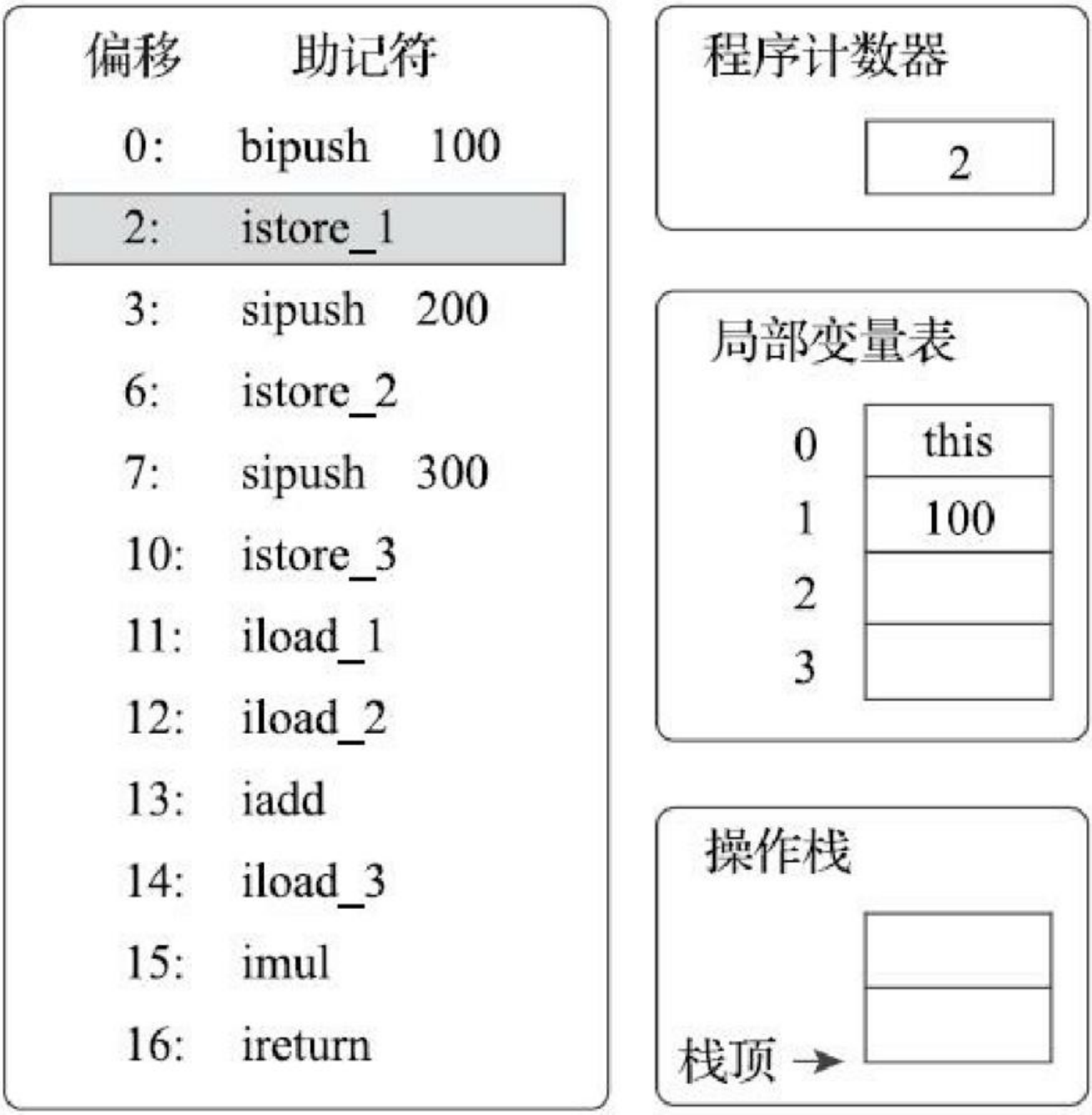
不带参数的

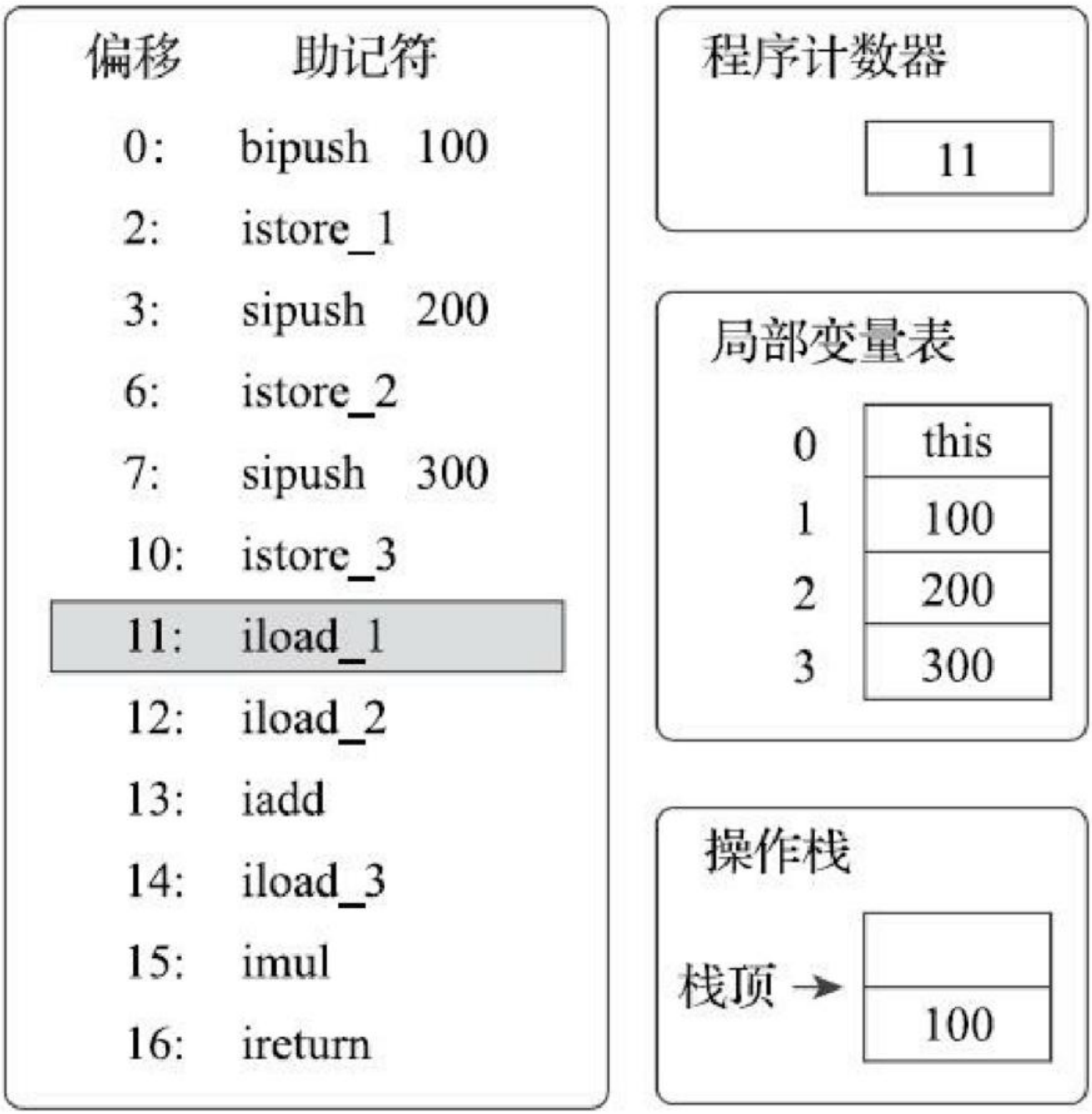
优点　　可移植
而基于栈的指令集则与硬件无关，因而具有灵活的可移植性。

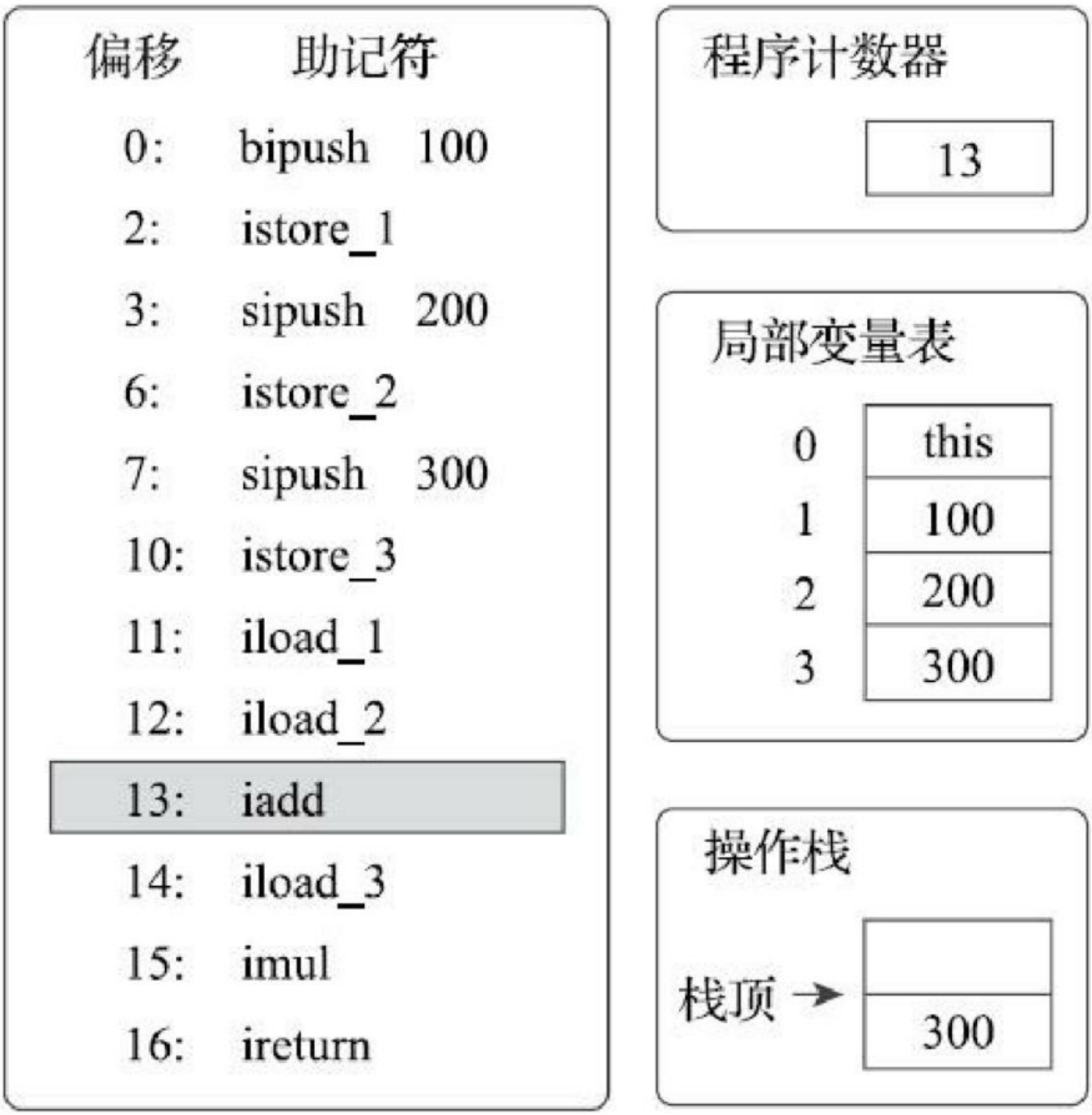
缺点　　基于栈的指令集会稍慢一些
更多。

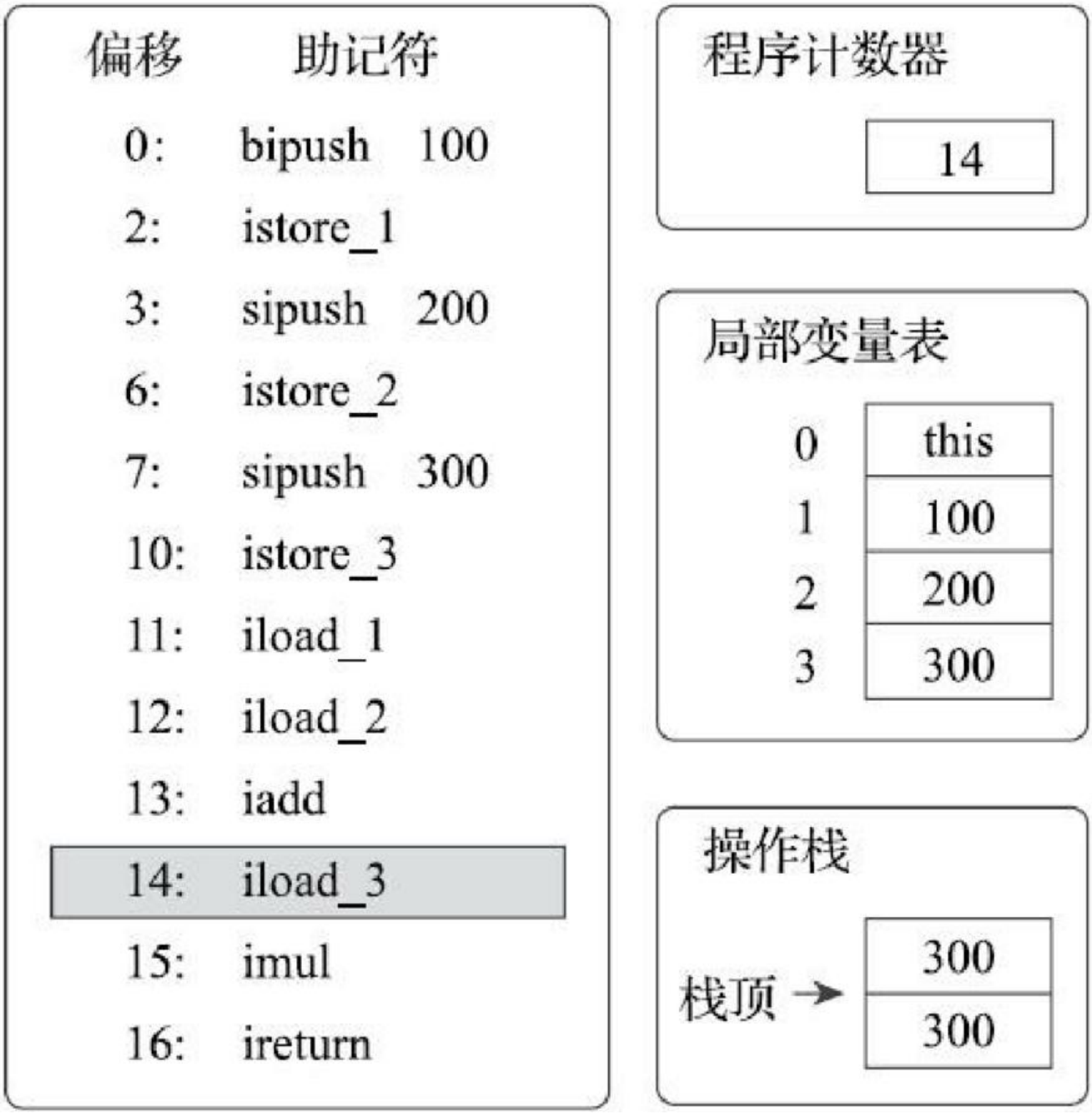
基于栈的解释器执行过程

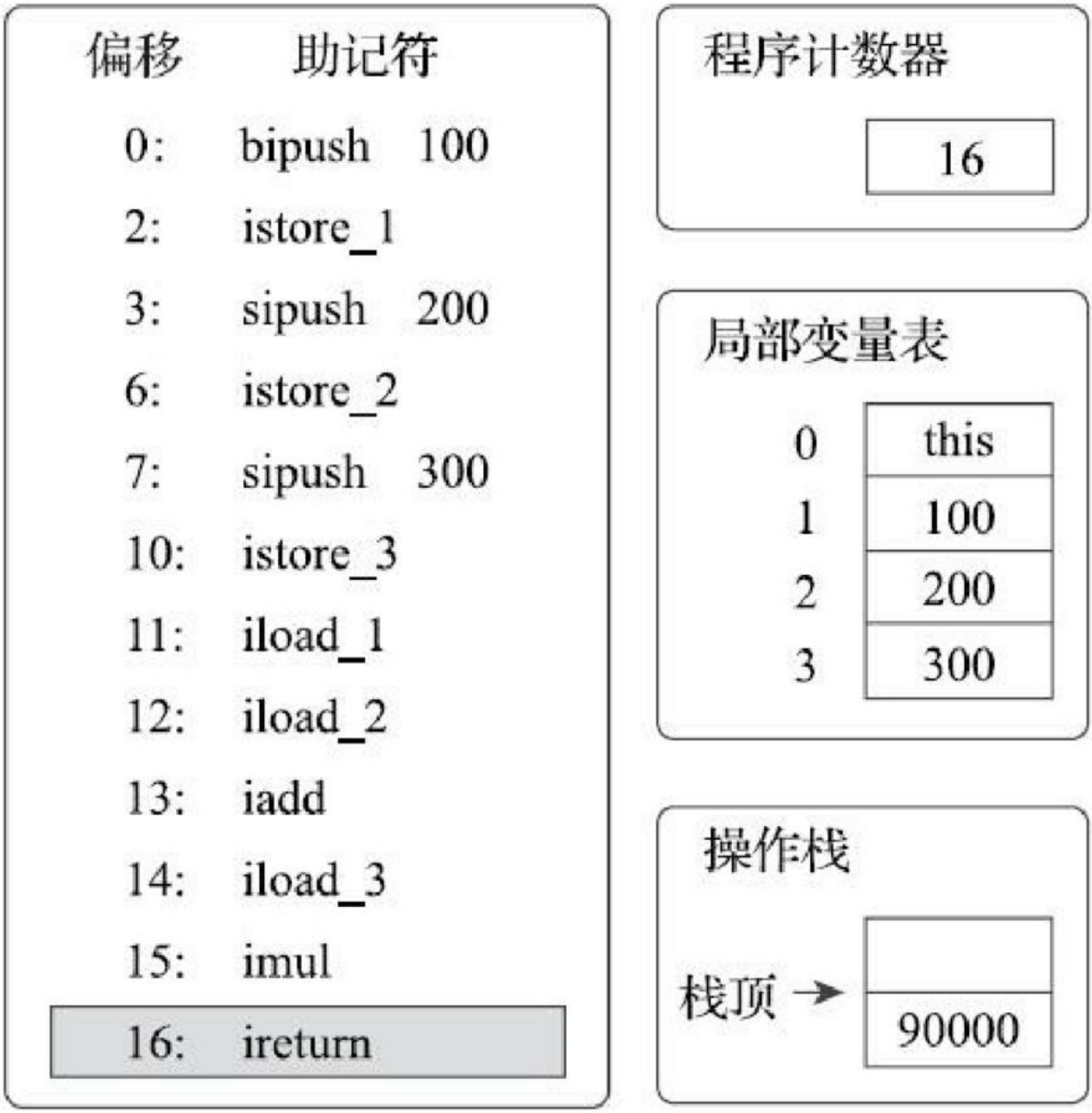

偏移 助记符
0: bipush 100
2: istore_1
3: sipush 200
6: istore_2
7: sipush 300
10: istore_3
11: iload_1
12: iload_2
13: iadd
14: iload_3
15: imul
16: ireturn
程序计数器
0
局部变量表
0 this
1
2
3
操作栈
栈顶
100


偏移 助记符
0: bipush 100
2: istore_1
3: sipush 200
6: istore_2
7: sipush 300
10: istore_3
11: iload_1
12: iload_2
13: iadd
14: iload_3
15: imul
16: ireturn
程序计数器
2
局部变量表
0 this
1 100
2
3
操作栈
栈顶


偏移 助记符
0: bipush 100
2: istore_1
3: sipush 200
6: istore_2
7: sipush 300
10: istore_3
11: iload_1
12: iload_2
13: iadd
14: iload_3
15: imul
16: ireturn
程序计数器
11
局部变量表
0 this
1 100
2 200
3 300
操作栈
栈顶
100


偏移 助记符
0: bipush 100
2: istore_1
3: sipush 200
6: istore_2
7: sipush 300
10: istore_3
11: iload_1
12: iload_2
13: iadd
14: iload_3
15: imul
16: ireturn
程序计数器
13
局部变量表
0 this
1 100
2 200
3 300
操作栈
栈顶
300


偏移 助记符
0: bipush 100
2: istore_1
3: sipush 200
6: istore_2
7: sipush 300
10: istore_3
11: iload_1
12: iload_2
13: iadd
14: iload_3
15: imul
16: ireturn
程序计数器
14
局部变量表
0 this
1 100
2 200
3 300
操作栈
栈顶
300
300



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