**第8章 虚拟机字节码执行引擎**

字节码执行引擎是Java虚拟机最核心的组成部分之一。在Java虚拟机规范中制定了虚拟机字节码执行引擎的**概念模型**，这个概念模型成为**各种虚拟机执行引擎的统一外观（Facade）**。

在不同虚拟机实现里面，执行引擎在执行Java代码的时候可能会有解释执行（通过解释器执行）和编译执行（通过即时编译器产生本地代码执行）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。但从外观上看起来，所有的Java虚拟机的执行引擎都是一致的：输入的是字节码文件，处理过程是字节码解析的等效过程，输出的是执行结果。本章主要从概念模型的角度来讲解虚拟机的**方法调用**和**字节码执行**。

运行时栈帧结构

**栈帧（Stack Frame）**是用于支持虚拟机进行**方法调用**和**方法执行**的数据结构，它是虚拟机运行时数据区中的虚拟机栈（VM Stack）的栈元素。每一个栈帧存储了每一个方法的**局部变量表**、**操作数栈**、**动态连接**和**方法返回地址**等信息。

每一个**方法从调用开始至执行完成**的过程，都对应着一个**栈帧**在虚拟机栈里从**入栈到出栈**的过程。

在编译程序代码时，每一个栈帧（方法）中需要多大的局部变量表、多深的操作数栈都已经完全确定，并写入到对应方法表的Code属性中。因此一个栈帧需要分配多少内存，不会受到程序运行期变量数据的影响，而仅仅取决于具体的虚拟机实现。

一个线程中的方法调用链可能会很长，所以会有很多方法都同时处于执行状态。而对于执行引擎来说，在活动线程中，只有位于栈顶的栈帧才是有效的，称为**当前栈帧**（Current Stack Frame），与这个栈帧相关联的方法称为**当前方法**（Current Method）。执行引擎运行的所有字节码指令都只针对当前栈帧进行操作。概念模型的栈帧结构如下所示：

**活动线程（当前线程）** 线程n 线程n-1

**当前栈帧**

局部变量表

Local Variable Table

**……**

操作数栈

Operand Stack

动态连接

Dynamic Linking

返回地址

Return Address

**……**

**……**

**栈帧2**

**栈帧1**

栈帧的概念结构

局部变量表

局部变量表（Local Variable Table）是一组变量值存储空间。用于存放**方法参数**和方法内部定义的**局部变量**。在Java程序编译Class文件时，就在方法的Code属性的max\_locals数据项中确定了该方法所需要分配的局部变量表的最大容量。

局部变量表的容量以变量槽（Variable Slot）为最小单位。一个Slot可以存放一个32位以内的数据类型，Java中占用32位以内的数据类型有boolean、byte、char、short、int、float、reference（表示对一个对象实例的引用，在64位虚拟机中则实际长度为64位。可从此引用中直接或间接地查找到对象在Java堆中的数据存放的起始地址索引，也可从此引用中直接或间接地查找到对象所属数据类型在方法区中的存储的类型信息）和returnAddress（为字节码指令jsr、jsr\_w、ret服务，指向了一条字节码指令的地址）。

对于64位的数据类型，虚拟机会以高位对齐的方式为其分配两个连续的Slot空间。64位的数据类型只有long和double（以及64位虚拟机中的reference）。

虚拟机通过索引定位的方式使用局部变量表，索引值的范围从0开始至局部变量表最大的Slot数量。如果访问的是32位数据类型的变量，索引n就代表使用第n个Slot，如果是64位数据类型的变量，则说明会同时使用n和n+1两个Slot。JVM不允许用户采用任何方式单独访问存放64位数据的两个Slot中的某一个，否则在类加载的校验阶段会抛出异常。

在方法执行时，虚拟机是使用局部变量表完成参数值到参数变量列表的传递过程的。如果执行的是**实例方法**（非static），则**局部变量表中第0位索引**的Slot默认是用于传递**方法所属对象实例的引用**，在方法中可以通过关键字**“this”**来访问到这个隐含的参数。**其余参数则按照参数表顺序排列**，占用从1开始的局部变量Slot，参数值分配完毕后，再根据方法体内部定义的**局部变量**顺序和作用域分配其余的Slot。

为节省栈帧空间，局部变量表中的Slot是可以重用的。方法体中定义的变量，其作用域并不一定会覆盖到整个方法体，如果当前字节码PC计数器的值已经超出了某个变量的作用域，那这个变量对应的Slot就可以交给其他变量使用。

操作数栈

操作数栈（Operand Stack）也称为操作栈。操作数栈的最大深度也在编译时写入到Code属性的max\_stacks数据项中。操作数栈的每一个元素可以是任意的Java数据类型，包括long和double。32位数据类型所占的栈容量为1,64位数据类型所占的栈容量为2。在方法执行的任何时候，操作数栈的深度都不会超过在max\_stacks数据项中设定的最大值。

当一个方法刚开始执行的时候，该方法的操作数栈是空的，在方法的执行过程中，会有各种字节码指令往操作数栈中写入和提取内容，即入栈/出栈操作。

另外，在概念模型中，两个栈帧作为虚拟机栈的元素，是完全相互独立的。但在大多虚拟机实现里都会做一些优化处理，令两个栈帧出现一部分重叠。让下面栈帧的部分操作数栈与上面栈帧的部分局部变量表重叠在一起，这样在进行方法调用时就可以共用一部分数据，无须进行额外的参数复制传递。

Java虚拟机的解释执行引擎称为“基于栈的执行引擎”，其中所指的“栈”就是操作数栈。

操作数栈

----------------------------------其他栈帧信息

----------------------------------局部变量表

**局部变量表共享区域**

**操作数栈共享区域**

操作数栈

----------------------------------其他栈帧信息

----------------------------------局部变量表

两个栈帧之间的数据共享

动态连接

每个栈帧都包含一个指向运行时常量池中该栈帧所属方法的引用，持有这个引用是为了支持方法调用过程中的动态连接（Dynamic Linking）。Class文件的常量池中存有大量的符号引用，字节码中的方法调用指令就以常量池中指向方法的符号引用作为参数。

**这些符号引用一部分会在类加载阶段或者第一次使用的时候就转化为直接引用，这种转化称为静态解析。另外一部分将在每一次运行期间转化为直接引用，这部分称为动态连接**。

方法返回地址