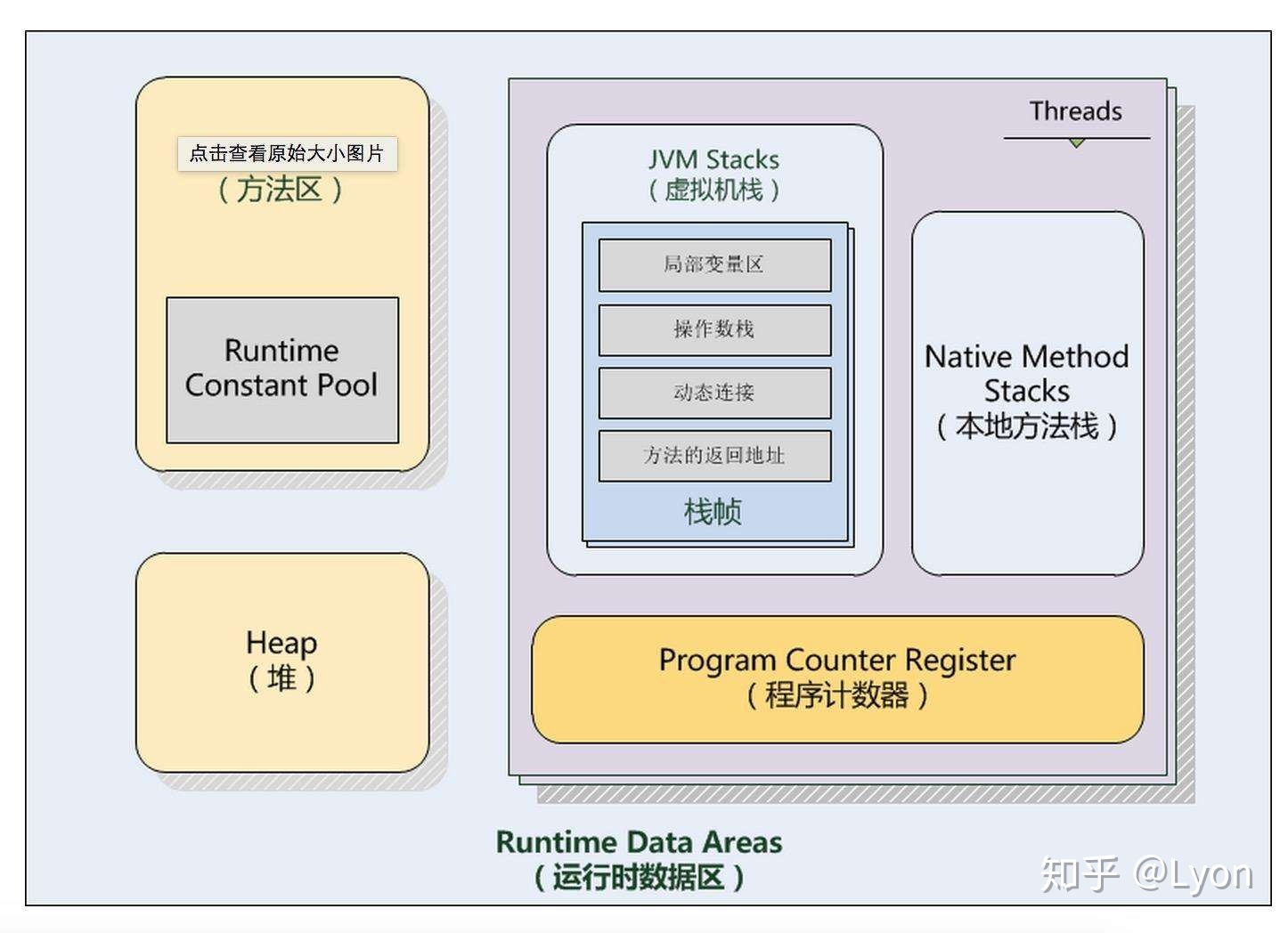
## Java虚拟机栈

在之前的文章：Java虚拟机—堆、栈、运行时数据区中，我们整体介绍了JVM在运行时的一些数据区域如堆、方法区、程序计数器、虚拟机栈、本地方法栈。本篇文章，我们围绕其中的一个区域展开——虚拟机栈中的栈元素栈帧Java虚拟机是基于「栈」架构的，如图所示：



为什么要深入研究虚拟机栈呢？因为它hin重要。除了一些native方法是基于本地方法栈实现的，所有的Java方法几乎都是通Java虚拟机栈来实现方法的调用和执行过程(当然，需要程序计数器、堆、方法区的配合)，所以Java虚拟机栈是虚拟机执行引擎的核心之一。而Java虚拟机栈中出栈入栈的元素就称为「栈帧」。

## 0.栈帧的概念

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。栈帧存储了方法的局部变量表、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。

一个线程中方法的调用链可能会很长，很多方法都同时处于执行状态。对于JVM执行引擎来说，在在活动线程中，只有位于JVM虚拟机栈栈顶的元素才是有效的，即称为当前栈帧，与这个栈帧相关连的方法称为当前方法，定义这个方法的类叫做当前类。

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。如果当前方法调用了其他方法，或者当前方法执行结束，那这个方法的栈帧就不再是当前栈帧了。

调用新的方法时，新的栈帧也会随之创建。并且随着程序控制权转移到新方法，新的栈帧成为了当前栈帧。方法返回之际，原栈帧会返回方法的执行结果给之前的栈帧(返回给方法调用者)，随后虚拟机将会丢弃此栈帧。

栈帧是线程本地的私有数据，不可能在一个栈帧中引用另外一个线程的栈帧。

在概念模型上，典型的栈帧结构如下：



**关于「栈帧」，《Java虚拟机规范》中的描述如下：**

栈帧是用来存储数据和部分过程结果的数据结构，同时也用来处理动态连接、方法返回值和异常分派。

栈帧随着方法调用而创建，随着方法结束而销毁——无论方法正常完成还是异常完成都算作方法结束。

栈帧的存储空间由创建它的线程分配在Java虚拟机栈之中，每一个栈帧都有自己的本地变量表(局部变量表)、操作数栈和指向当前方法所属的类的运行时常量池的引用。

接下来，详细讲解一下栈帧中的局部变量表、操作数栈、动态连接、方法返回地址等各个部分的数据结构和作用。

## 1.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定义的局部变量。局部变量表的容量以变量槽(Variable Slot)为最小单位，Java虚拟机规范并没有定义一个槽所应该占用内存空间的大小，但是规定了一个槽应该可以存放一个32位以内的数据类型。

在Java程序编译为Class文件时,就在方法的Code属性中的max\_locals数据项中确定了该方法所需分配的局部变量表的最大容量。(最大Slot数量)

一个局部变量可以保存一个类型为boolean、byte、char、short、int、float、reference和returnAddress类型的数据。reference类型表示对一个对象实例的引用。returnAddress类型是为jsr、jsr\_w和ret指令服务的，目前已经很少使用了。

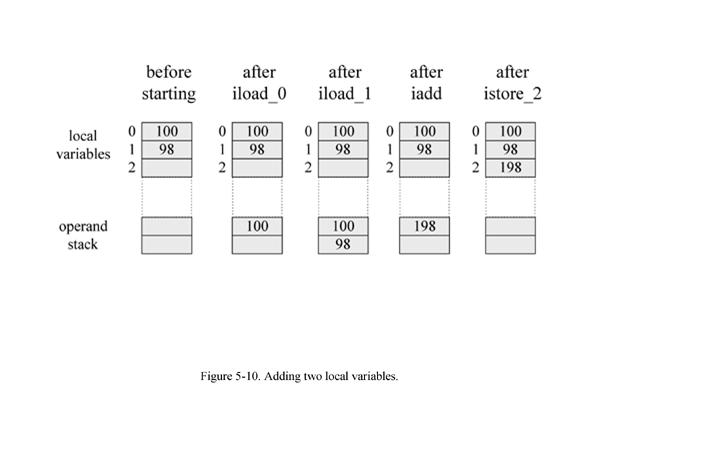
虚拟机通过索引定位的方法查找相应的局部变量，索引的范围是从0~局部变量表最大容量。如果Slot是32位的，则遇到一个64位数据类型的变量(如long或double型)，则会连续使用两个连续的Slot来存储。

## 2.操作数栈

操作数栈(Operand Stack)也常称为操作栈，它是一个后入先出栈(LIFO)。同局部变量表一样，操作数栈的最大深度也在编译的时候写入到方法的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Java数据类型，32位的数据类型占一个栈容量，64位的数据类型占2个栈容量,且在方法执行的任意时刻，操作数栈的深度都不会超过max\_stacks中设置的最大值。

当一个方法刚刚开始执行时，其操作数栈是空的，随着方法执行和字节码指令的执行，会从局部变量表或对象实例的字段中复制常量或变量写入到操作数栈，再随着计算的进行将栈中元素出栈到局部变量表或者返回给方法调用者，也就是出栈/入栈操作。一个完整的方法执行期间往往包含多个这样出栈/入栈的过程。



## 3.动态连接

在一个class文件中，一个方法要调用其他方法，需要将这些方法的符号引用转化为其在内存地址中的直接引用，而符号引用存在于方法区中的运行时常量池。

Java虚拟机栈中，每个栈帧都包含一个指向运行时常量池中该栈所属方法的符号引用，持有这个引用的目的是为了支持方法调用过程中的动态连接(Dynamic Linking)。

这些符号引用一部分会在类加载阶段或者第一次使用时就直接转化为直接引用，这类转化称为静态解析。另一部分将在每次运行期间转化为直接引用，这类转化称为动态连接。

## 4.方法返回

当一个方法开始执行时，可能有两种方式退出该方法：

**1、正常完成出口**

正常完成出口是指方法正常完成并退出，没有抛出任何异常(包括Java虚拟机异常以及执行时通过throw语句显示抛出的异常)。如果当前方法正常完成，则根据当前方法返回的字节码指令，这时有可能会有返回值传递给方法调用者(调用它的方法)，或者无返回值。具体是否有返回值以及返回值的数据类型将根据该方法返回的字节码指令确定。

**2、异常完成出口**

异常完成出口是指方法执行过程中遇到异常，并且这个异常在方法体内部没有得到处理，导致方法退出。

无论是Java虚拟机抛出的异常还是代码中使用athrow指令产生的异常，只要在本方法的异常表中没有搜索到相应的异常处理器，就会导致方法退出。

方法退出过程实际上就等同于把当前栈帧出栈，因此退出可以执行的操作有：恢复上层方法的局部变量表和操作数栈，把返回值(如果有的话)压如调用者的操作数栈中，调整PC计数器的值以指向方法调用指令后的下一条指令。

一般来说，方法正常退出时，调用者的PC计数值可以作为返回地址，栈帧中可能保存此计数值。而方法异常退出时，返回地址是通过异常处理器表确定的，栈帧中一般不会保存此部分信息。

## 5.附加信息

虚拟机规范允许具体的虚拟机实现增加一些规范中没有描述的信息到栈帧之中，例如和调试相关的信息，这部分信息完全取决于不同的虚拟机实现。在实际开发中，一般会把动态连接，方法返回地址与其他附加信息一起归为一类，称为栈帧信息。