# Java Stack Frame

当新建一个线程时，JVM就会为这个线程创建一个Java栈（Java Stack）[1]。JVM在Java栈上只有两个操作：pop和push。需要说明的是，每个新建的线程在JVM中都是一个执行引擎(Execution Engine).

Current method: 线程正在执行的方法

Current frame: 当前方法的栈帧

Current class: 当前方法所在的类

Current constant pool: 当前类的常量池

当线程触发（调用）一个Java方法时，JVM就会创建一个新的帧并将其push到当前线程的Java栈上。这个帧成为Current frame（当前帧），执行对应方法时，JVM用这个帧保存参数，本地变量，中间计算数值和其他数据。

当前方法结束时（正常结束或者发生异常），JVM会pop出当前方法的栈帧并丢弃，上一级（或者多级-异常时）帧成为当前栈帧。

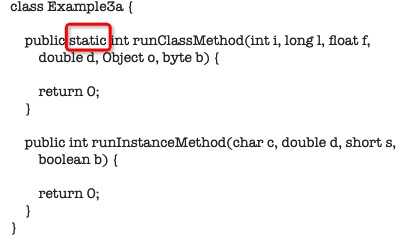
Java栈上的数据是当前线程私有的。

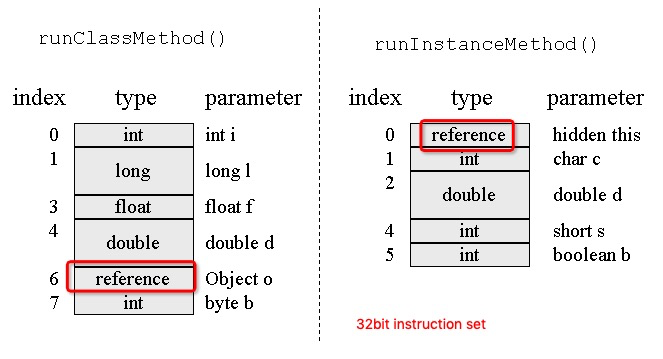
## 栈帧

栈帧包含三部分：本地变量，操作数栈和帧数据。本地变量和操作数栈的大小在编译时就已经确定了。帧数据由实现决定。当JVM执行方法时，它通过class数据确定本地变量和操作数需要的空间大小并创建适当大小的栈帧然后将其push到Java栈上。

### 本地变量

本地变量在栈帧中表现为数组形式，指令通过索引（index）引用其中的值。Int，float，reference和返回地址占据一个entry，long和double占用两个entry。小于int变量空间大小的值转成int后保存。本地变量以字大小对齐（[1]中的字大小是4，32位机，所以reference也说成了4字节）。





操作数栈中，各数据的存储对齐和本地变量一样处理。当数据写回到堆中或方法区（**什么情况？**）时，才会转成对应的类型。

本地变量数据，方法参数排列在前面。参数后面是方法内使用的变量，这部分变量如何在数组中存储没有严格约定，同一个index可能被不同变量占据（类比下C中的寄存器）。

### 操作数栈

和本地变量一样，操作数栈也组织为字数组。但操作数栈不以索引方式访问，而是通过pop和push操作。前一个指令push到操作数栈中的值，后一个指令可以pop出这个值来使用。

JVM像保存本地变量一样支持在操作数栈中保存各种不同类型的变量，returnType（返回值）也在操作数栈中。数值类型也一样处理。

JVM是基于栈实现的（而不是寄存器）。Java指令从操作数栈获取操作数，也可以是紧跟操作指令的操作数，还可以是常量池。JVM指令集主要关注的是操作数栈。

操作数栈是JVM的工作空间。

iload\_0 // push the int in local variable 0 (**第一个参数不是reference，所以是static**)

iload\_1 // push the int in local variable 1

iadd // pop two ints, add them, push result

istore\_2 // pop int, store into local variable 2

我们不妨猜猜这几个指令做了什么？可能是这样的

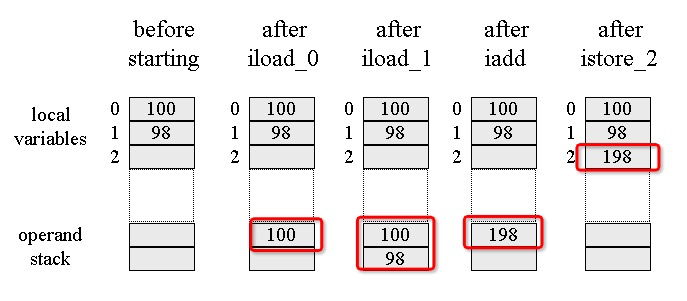
static int sum(int a, int b){

int c = a + b;

return c;

}

当然也可能不是。



### 帧数据

除本地变量和操作数栈外，Java栈帧还包含支持常量池解析、正常方法返回以及异常分发的数据。这部分数据保存在Java栈帧中的**帧数据**部分。

许多JVM指令集中的指令引用常量池（通常提供类似于符号表的功能，当然不仅仅如此）。其中一些指令仅仅是将常量池中的整数、字符串等常量push到操作数栈中。另外一些引用常量池继而引用类或数组进行初始化、访问字段或者触发方法调用。还有的指令则是通过常量池中的数据判断对象的class或者接口类型。

遇到指令访问常量池数据时，JVM使用帧数据中指向常量池的指针访问数据。常量池中的引用数据初始时是符号，当对常量池中引用类、接口、字段或者方法的数据时，这些引用仍旧是符号。此时，JVM必须解析这些引用。

帧数据还要帮助JVM处理方法正常和异常调用。正产返回时，JVM要恢复触发方法的栈帧。JVM要将PC寄存器设置为刚刚被调用函数完成执行后的下一条指令。如果刚刚完成的方法有返回值，JVM必须将返回值push到触发方法的栈帧中。

帧数据必须包含被执行方法的异常表的引用，以便被调用方法发生异常时进行处理。异常表定义了被调用方法catch语句保护的范围。异常表中每项数据标识catch语句保护的起止位置，一个指向常量池的索引（标识捕获到的异常类），和catch语句代码的开始位置。

当方法执行时抛出异常，JVM使用帧数据中的异常表确定如何处理异常。如果可以找到匹配的catch语句，那么就跳转到catch语句的开始处。如果找不到，方法即是异常完成。JVM使用帧数据中的信息回复触发方法的帧（调用者的帧），并在调用者中重新抛出异常。

帧数据还可能包含支持调试的数据。

### 动态链接

指通过帧数据中的方法符号解析符号，触发加载必要的类并进而实现方法调用。

1. <https://www.artima.com/insidejvm/ed2/jvm8.html>
2. <https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-2.html#jvms-2.6>
3. <https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-2.html#jvms-2.5.2>

Code:

       0: iload\_0

       1: iload\_1

       2: iadd

       3: istore\_2

       4: iload\_2

       5: ireturn

Code:

       0: iload\_1

       1: iload\_2

       2: iadd

       3: istore\_3

       4: iload\_3

       5: ireturn