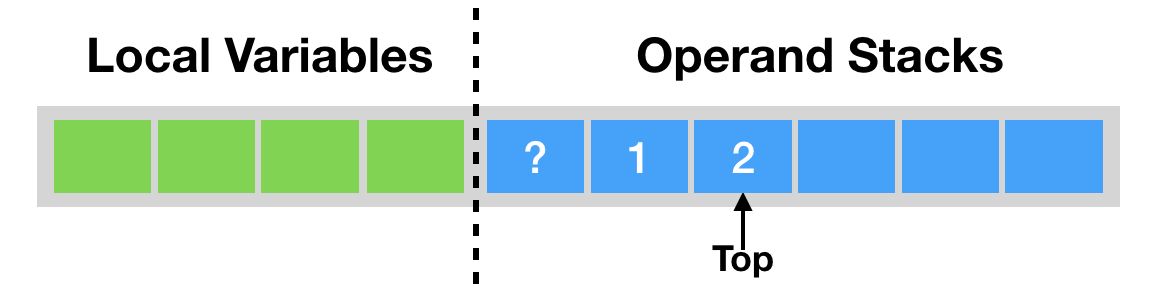
Java字节码

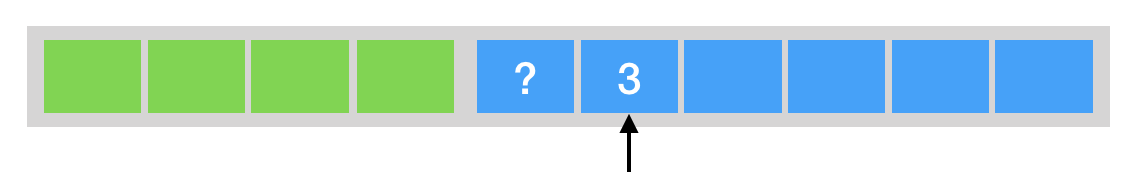
字节码是虚机使用的指令集合。

## 操作数栈

执行每一条指令之前，Java虚机要求该指令的操作数已经被压入操作数栈中。在执行指令的时候，虚机会将该指令所需的操作数弹出，并将指令的结果重新压入栈中。



以加法iadd为例，假设在执行该指令的时候，栈顶两个元素分别是int值1和int值2，那么iadd指令将弹出这两个int，并将求得的int值压入栈中。



由于iadd指令只消耗栈顶的两个元素，因此，对于离栈顶距离为2的元素，即图中的？号，iadd指令并不关心它是否存在，更加不会对其进行修改。

Java字节码好几个指令是直接作用在操作数栈上的。最为常用的就是

dup：复制栈顶元素，

pop：舍弃栈顶元素。

dup长用于复制new指令所生成的未初始化的引用。例如：

**public void** foo() {  
 **Object** o = **new** Object();  
}

// 对应字节码如下：

public void foo();

Code:

0: new #2 // class java/lang/Object

3: dup

4: invokespecial #1 // Method java/lang/Object."<init>":()V

7: astore\_1

8: return

当执行new指令时候，java虚机将指向一块已经分配的、未初始化的内存的引用压入操作数栈中。然后以这个引用作为调度者，调用其构造器，也就是invokespecial指令。该指令将消耗操作数栈上的元素，作为它的调用者及参数（不过Object不需要参数）

因此，我们需要利用dup指令复制一份new指令的结果，并用来调用构造器。当调用结束后，操作数栈上仍然有原本由new指令生成的引用，可用于接下来的操作（即偏移量为7的字节码）

pop指令常用于舍弃调用指令的返回结果。比如下面的bar方法，调用静态的bar方法，但是却不用其返回值。

由于对应的invokestatic指令仍然会将返回值压入foo方法的操作数栈中，因此java虚机需要额外执行pop指令，将返回值舍弃。

**public static boolean** bar() {  
 **return false**;  
}  
  
**public void** foo2() {  
 bar();  
}

foo2对应的字节码如下：

public void foo2();

Code:

0: invokestatic #3 // Method bar:()Z

3: pop

4: return

注意：dup指令和pop只能处理非long或者double类型的值。因为double类型或者long类型的值，需要占据两个单元。当遇到这些个值的时候，需要复制栈顶两个单元的值dup2和弹出栈顶两个单元的值pop2指令。

除此之外，直接用于操作数栈的指令还有swap指令，将交换栈顶两个元素的值。

Java虚机既可以通过iconst指令加载-1到5之间的值，也可以通过bipush、sipush加载一个字节、两个字节所能代表的int值。

Idc加载常量池中的值。例如：Idc #18将加载常量池中的第18项。

这些常量包括：int、long、float、double、String以及Class类型的常量。



常数加载指令表。

当抛出异常的时候，虚机会清除操作数栈上的所有内容，而后将异常实例压入操作数栈上。

## 局部变量区

Java栈帧的另外一个重要组成部分就是局部变量区，字节码程序可以将计算结果缓存在局部变量区之中。

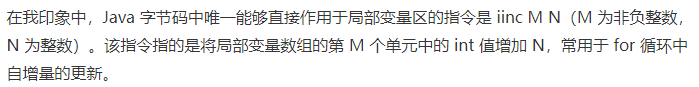
被认为是一个数组，以此存放this指针（仅非静态方法），所传入的参数，以及字节码的局部变量。

加载、存储指令是区分类型的



局部变量区访问指令表

局部变量数组的加载、存储指令都要指明所加载单元的下标。aload 0指的是加载第0个单元所存储的引用。



**public void** foo3() {  
 **for** (**int** i = 1\_0\_0; i > 0; i -- ) {  
 }  
}

// 编译后字节码如下

public void foo3();

Code:

0: bipush 100

2: istore\_1

3: iload\_1

4: ifle 13

7: iinc 1, -1

10: goto 3

13: return

综合事例：

**public static int** bar(**int** i) {  
 **return** ((i + 1) - 2) \* 3 / 4;  
}

public static int bar(int);

Code:

0: iload\_0

1: iconst\_1

2: iadd

3: iconst\_2

4: isub

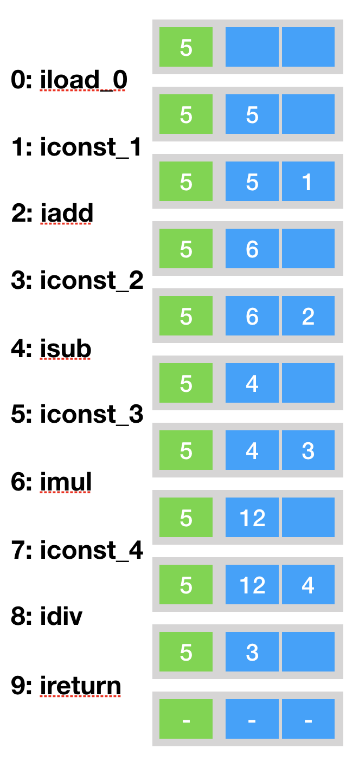
5: iconst\_3

6: imul

7: iconst\_4

8: idiv

9: ireturn



## Java字节码简介

上面介绍了——加载常量指令、操作数专用指令、局部变量访问区指令。

java相关指令：

1. 包括具有高层语义的字节码
   1. new（后跟目标类，生成该类的未初始化的对象）
   2. Instanceof（后跟目标类，判断栈顶元素是否为目标类/接口实例。是则压入1，否则压入0）
   3. checkcast（后跟目标类，判断栈顶元素是否为目标类/接口实例。不是抛异常）
   4. athrow（将栈顶异常抛出）
   5. monitorenter（为栈顶对象加锁）
   6. monitorexit（为栈顶对象解锁）
2. 字段访问指令
   1. 静态字段访问指令
      1. getstatic
      2. putstatic
   2. 实例字段访问指令
      1. getfield
      2. putfield
3. 方法调用指令(除了invokedynamic指令，其他方法调用指令所消耗的操作数栈元素是根据调用类型以及目标方法描述符来确定的。在进行方法调用之前，程序需要依次压入调用者（invokestatic不需要），以及各个参数。
   1. Invokestatic
   2. Invokespecial
   3. Invokevirtual
   4. Invokeinterface
   5. Invokedynamic
4. 数组相关指令
   1. newarray（新建基本类型数组）
   2. anewarray（生成多维数组multianewarray）
   3. arraylength（求数组长度）
   4. 
5. 控制流指令，除了返回指令外，其他的控制流指令均附带一个或者多个字节码偏移量，代表需要跳转到的位置
   1. goto（无条件跳转）
   2. tableswitch和lookupswitch（条件跳转指令，前者针对密集的cases，后者针对稀疏的cases）
   3. 返回指令
   4. jsr,ret（已经废弃的指令）

精解：

Q：为什么局部变量要初始化？

A：若虚机初始化局部变量，那么它需要掌握每个变量的生命周期，以便初始化共享同一下标的局部变量实现起来费事。从代码规范来看，使用未经初始化的局部便量意义不明。

Q：C/C++的汇编指令，会用到大量寄存器操作，java的指令会用到吗？

A：java字节码不会，但是底下的实现（比如解释执行器，即时编译器）会用到。