栈帧(Stack frame)

- 栈帧是一种帮助虚拟机方法调用与方法执行的数据结构。
- 栈帧本身是一种数据结构, 封装了方法的局部变量表, 动态链接信息, 方法的返回地址以及操作数帧等信息。
- 可以通过slot 存储局部变量，slot 可复用的。
- 符号引用, 直接引用
    - 有些符号引用在类加载阶段或是第一次使用就会转换为直接引用，这种转换叫做静态解析；另外一些符号引用则是在每次运行期都会转换为直接引用。 这种转换叫做动态链接。这体现为Java的多态性。

方法重载与"invokevirtual"字节码指令

a. invokeinterface：调用接口中的方法，实际上是运行期决定的，决定到底调用实现接口的哪一个对象的特定方法。

a. invokestatic: 调用静态方法。

a. invokespecial: 调用自己的私有方法，构造方法（）以及父类的方法

a. invokevirtual： 调用虚方法，运行期动态查找的过程

a. invokedynamic: 动态调用方法

- 静态解析的4种情况
    1. 静态方法
    2. 父类方法
    3. 构造方法
    4. 私有方法（无法被重写）
- 以上4种方法称作为非虚方法，他们是在类加载阶段就可以将符号引用转化为直接引用。
- 方法的静态分派

```
Grandpa g1 = new Father();
```

以上代码g1的静态类型是 Grandpa，而g1的实际类型(真正执行的类型)是 Father

我们可以得出这样一个结论：变量的静态类型是不会发生变化的，而变量的实际类型则 是可以发生变化的(多态的一种体现)，实际类型是在运行期可以确定的。

- 方法重载是一种静态的行为，编译器就可以完全确定
- 方法的动态分派
    - 方法动态分派涉及到重要的一个概念：方法的接受者 invokevirtual 字节码指定的多态查找流程
    - 比较方法重载(overload) 与方法重写 (overwrite), 我们可以得到这样的一个结论：
        - 方法重载是静态的编译期行为
        - 方法重写是动态的运行期行为
    - 操作数栈顶第一个元素找到特定方法，找不到就按照继承关系从子类到父类去查找。Runtime的时候去确定。

*方法的接受者来确定是静态分配还是动态分配

    - 针对于方法调用动态分派的过程，虚拟机会在类的方法区建立一个虚方法表的数据结构(virtual method table, vtable)
    - 针对于 invokeinterface 指令来说，虚拟机会建立一个叫做接口方法表的数据结构(interface method table, itable)

指令集与寄存器的指令

现代JVM在执行Java代码的时候，通过都会讲解释执行与编译执行二者结合起来运行。

所谓解释执行，通过解释器来读取字节码，遇到相应的指令就去执行该指令。

所谓编译执行，就是通过即时编译器 (Just In time, JIT) 将字节码转换为本地机器码来执行；现代JVM会根据代码热点来生成响应的本地机器码.

- 基于栈的指令集与基于寄存器的指令集之间的关系;

  a. JVM 执行指令时所采取的方式是基于栈的指令集.

  b. 基于栈的指令集主要是操作有入栈与出栈两种.

  c. 基于栈的指令集的优势在于它可以在不同平台之间转义，而基于寄存器的指令集就是与硬件架构紧密关联，无法做到可移植。

  d. 基于栈的指令集的缺点在于完成相同的操作，指令数量通常要比基于寄存器的指令集数量要多，基于栈的指令集是在内存中完成的操作的。 而基于寄存器的指令集是直接由CPU来执行的, 它是在高速缓冲区执行的， 速度要快很多，虽然虚拟机可以采用一些优化手段，但是整体来说基于栈的指令集 的执行速度要慢一些。

2-1

1. iconst_1 //将减数1压入栈顶.
2. iconst_2 //将减数2压入栈顶.
3. isub //将栈顶以及下面的弹出对于响应的数字执行减法2-1,然后将结果压入栈顶.
4. istore_0 //将1放入局部变量表0的位置上.