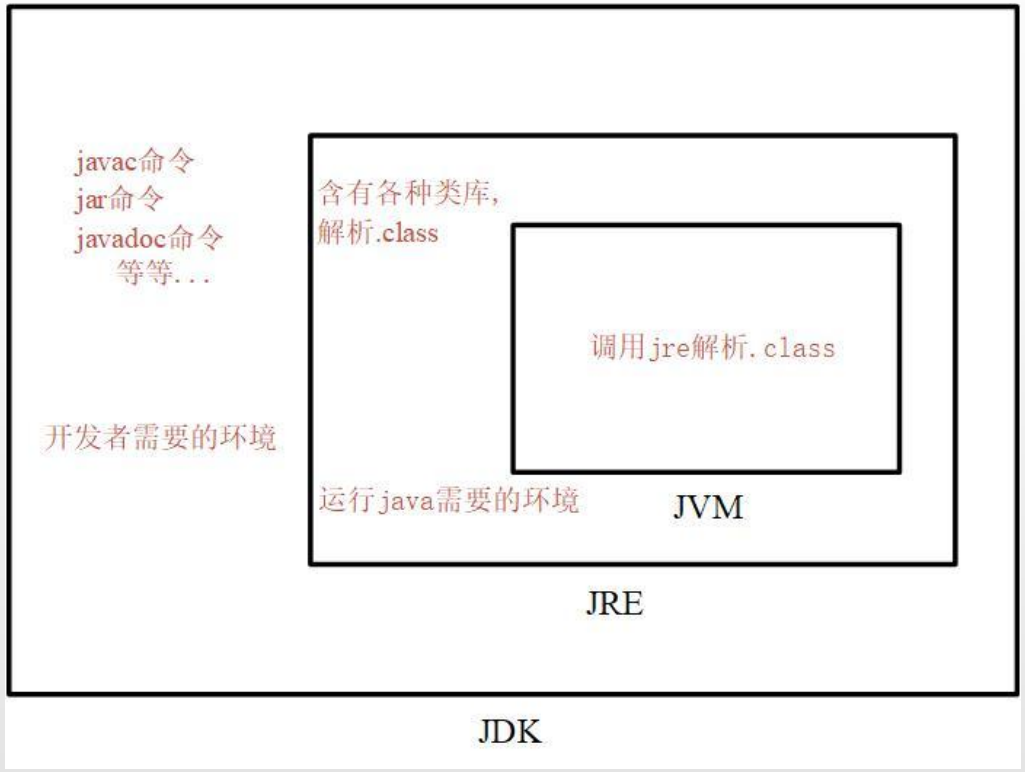
JVM：从入门开始（一）

PCLT Lab

zhangxiang@nj.iscas.ac.cn

2021.6.30

认识JDK、JVM、JRE



JDK：java development kit, java开发工具包，JDK是整个JAVA的核心，包括了Java的运行环境（Java Runtime Environment）、一堆Java工具（javac、java、jdb等）和Java基础的类库（即Java API包括rt.jar）。

JRE：java runtime environment, java运行时环境，针对java用户，也就是拥有可运行的.class文件包（jar或者war）的用户。

JVM：就是我们常说的java虚拟机，它是整个java实现跨平台的最核心的部分。Java Virtual Machine（Java 虚拟机）JVM是JRE的一部分，它是一个虚拟出来的计算机，是通过在实际的计算机上仿真模拟各种计算机功能来实现的。JVM有自己完善的硬件架构，如处理器、堆栈、寄存器等，还具有相应的指令系统。

总结

- 有JVM就可以解释执行字节码文件(.class).
- JVM解释执行这些字节码文件的时候需要调用类库，如果没有这些类库JVM就不能正确的执行字节码文件，JVM+类库=JRE,
- 有了JRE就可以正确的执行java程序了，但是光有JRE不能开发Java程序，所以JRE+开发工具=JDK,有了JDK,就可同时开发，执行JRE.

JAVA的跨平台特点

- C和C++之类的语言，会在编译期就直接编译成平台相关的机器指令，对于不同平台，可执行文件类型也不一样，如Linux为ELF，Windows为PE，而MacOS为Mach-O。而写Java的应该都清楚，java之所以跨平台性比较强，是因为Java在编译期没有被直接编译成机器指令，而是被编译成一种中间语言：字节码。

- 所有的java程序会首先被编译为.class的类文件，这种类文件可以在虚拟机上执行。也就是说class并不直接与机器的操作系统相对应，而是经过虚拟机间接与操作系统交互，由虚拟机将程序解释给本地系统执行。

为什么java不编译成机器码？

- 1. 机器码是与平台相关的，也就是操作系统相关，不同操作系统能识别的机器码不同，如果编译成机器码那岂不是和 C、C++差不多了，不能跨平台，Java 就没有那响亮的口号 “一次编译，到处运行”；

- 2.之所以不一次性全部编译，是因为有一些代码只运行一次，没必要编译，直接解释运行就可以。而那些“热点”代码，反复解释执行肯定很慢，JVM在运行程序的过程中不断优化，用JIT编译器编译那些热点代码，让他们不用每次都逐句解释执行；

- **那么，什么又是热点代码呢？**

字节码、机器码

- 字节码是指平常所了解的 .class 文件，Java 代码通过 javac 命令编译成字节码

- 机器码和本地代码都是指机器可以直接识别运行的代码，也就是机器指令

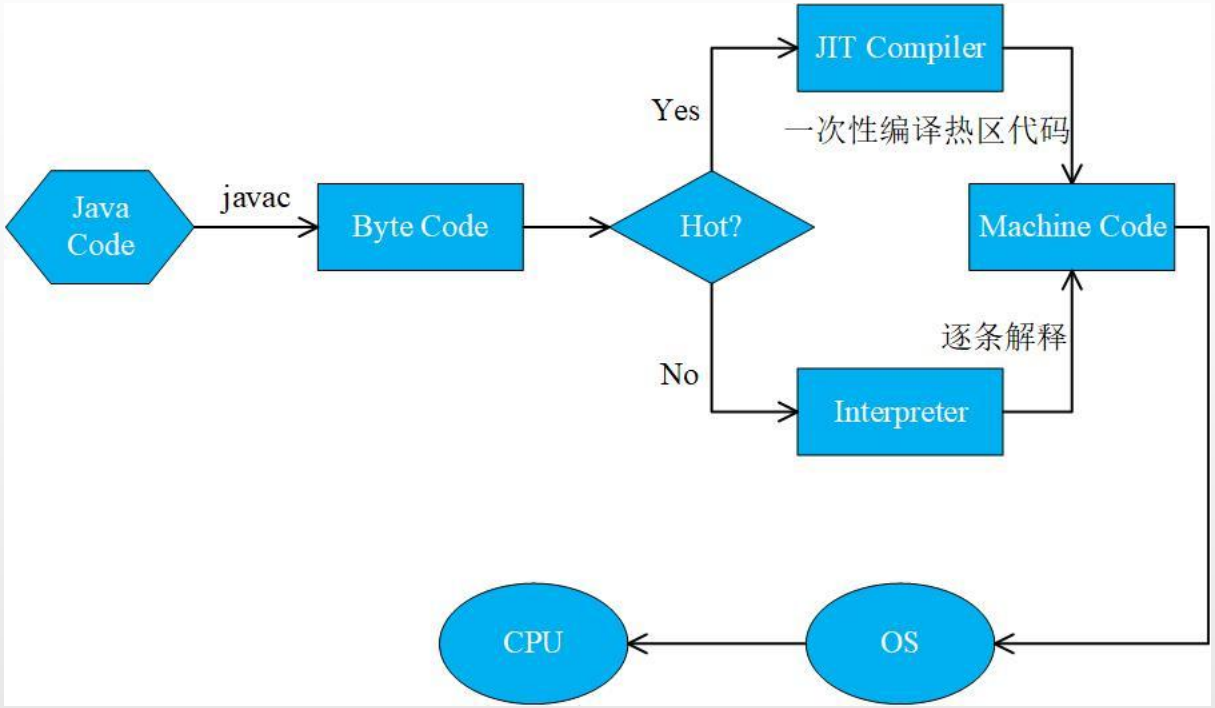
- 字节码是不能直接运行的，需要经过 JVM 解释或编译成机器码才能运行

编译器与解释器

- 编译器：把源程序的每一条语句都编译成机器语言,并保存成二进制文件,这样运行时计算机可以直接以机器语言来运行此程序,当程序需要迅速启动和执行的时候，解释器可以首先发挥作用，省去编译的时间，立即执行;
- 抽象来看：输入的代码 -> [编译器 编译] -> 编译后的代码 -> [执行] -> 执行结果

- 解释器：只在执行程序时,才一条一条的解释成机器语言给计算机来执行,所以运行速度是不如编译后的程序运行的快的，但是在程序运行后，随着时间的推移，编译器逐渐发挥作用，把越来越多的代码编译成本地代码之后，可以获取更高的执行效率;
- 抽象来看：输入的代码 -> [解释器 解释执行] -> 执行结果

java代码的编译过程



JIT（即时编译技术）

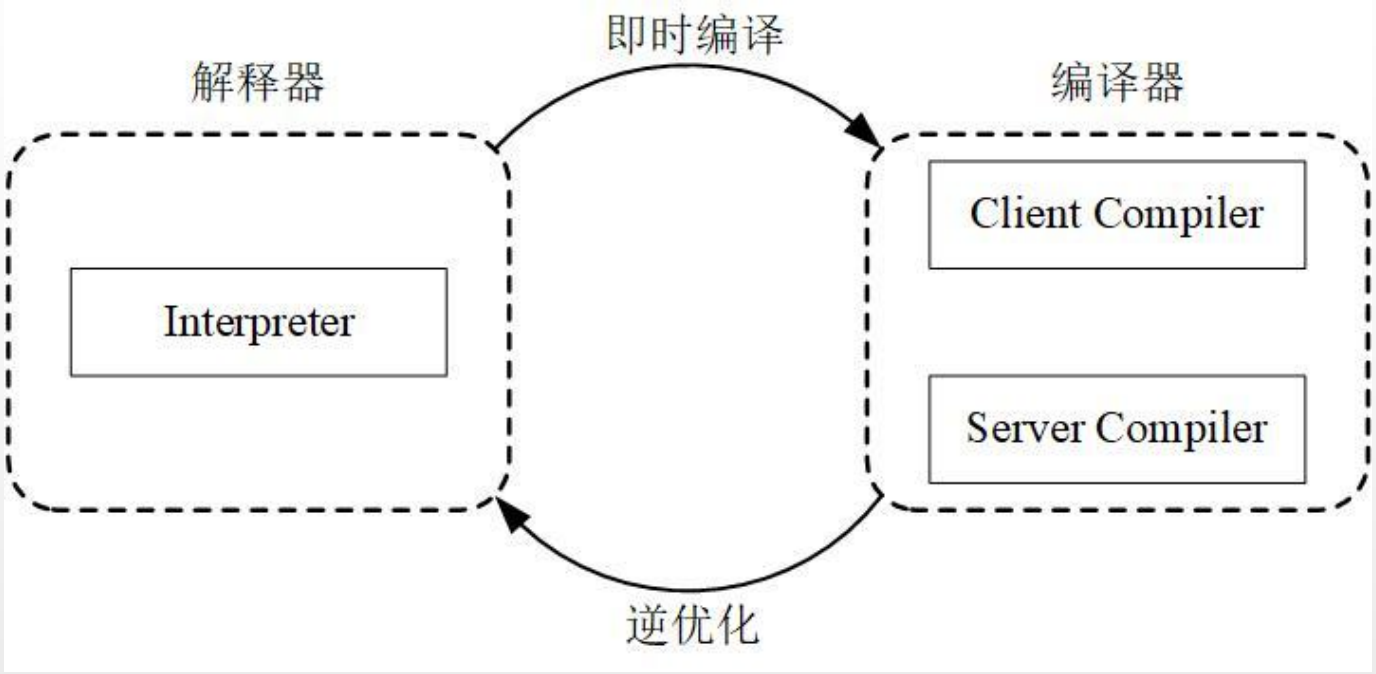
- Java 需要将字节码逐条翻译成对应的机器指令并且执行，这就是传统的 JVM 的解释器的功能，正是由于解释器逐条翻译并执行这个过程的效率低，引入了 JIT 即时编译技术。

- 具体来说，虚拟机发现某个方法或者代码块的运行频率高，就认定这些代码是“热点代码”（hot spot code）.为了提高热点代码的执行效率，虚拟机会把这些代码编译成与本地代码相关的机器码，并进行各种层次的优化，完成这个过程的就叫做JIT compiler（即时编译器）。

- 在HotSpot VM中内嵌有两个JIT编译器,分别为Client Compiler 和 Server Compiler,但大多数情况下我们简称为C1编译器和C2编译器:
- -client: 指定Java虚拟机运行在Client模式下,并使用C1编译器。
- C1编译器会对字节码进行简单和可靠的优化,耗时短。以达到更快的编译速度。

- -server: 指定Java虚拟机运行在Server模式下,并使用C2编译器。
- C2进行耗时较长的优化,以及激进优化。但优化的代码执行效率更高

解释器与编译器的“互动”

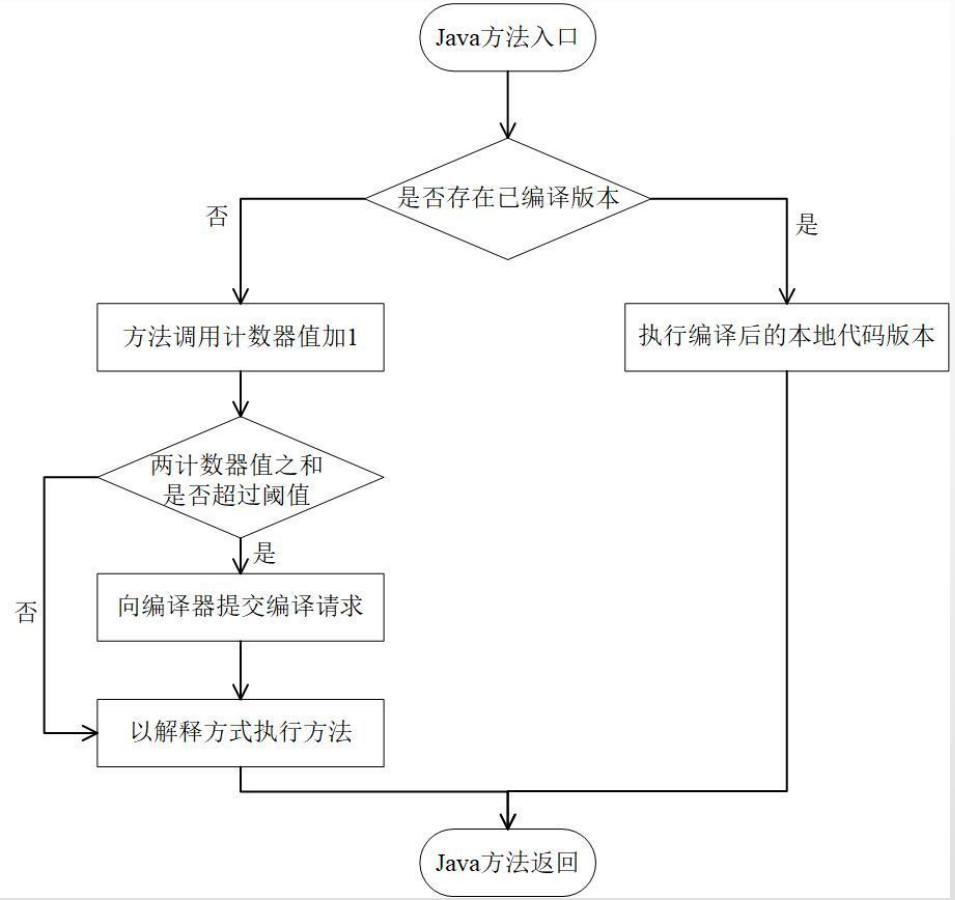


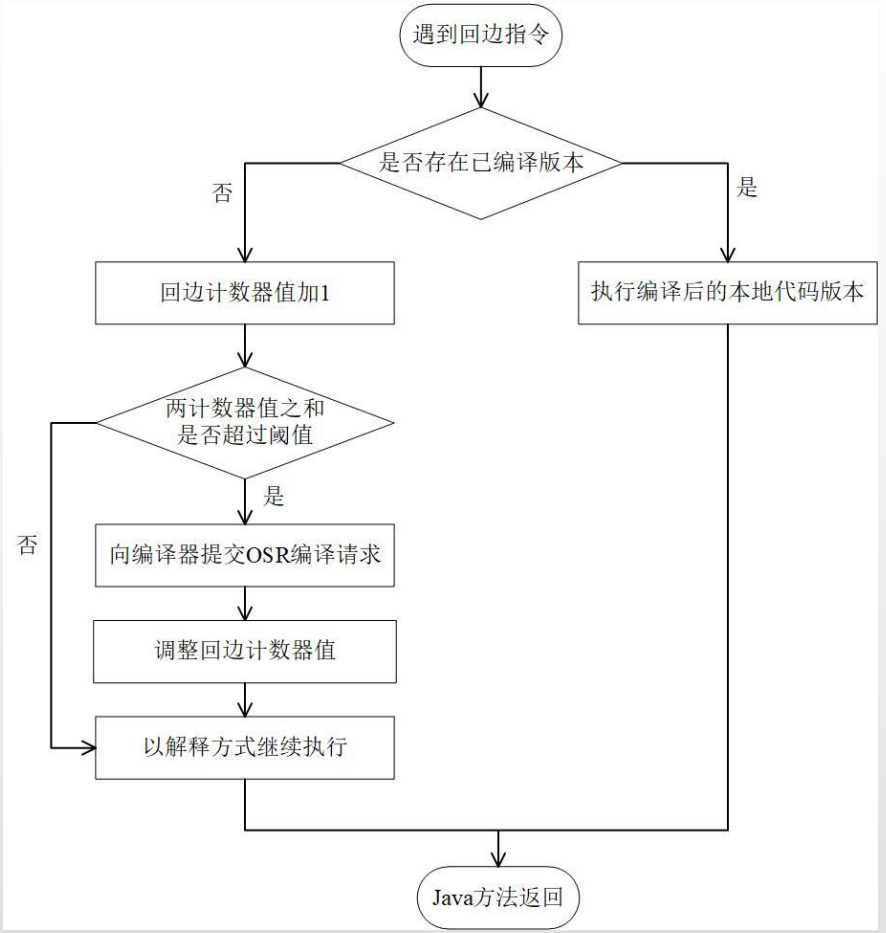
在程序运行环境中内存资源限制较大时，可以使用解释执行节约内存，反之可以使用编译执行来提升效率。当通过编译器优化时，发现并没有起到优化作用，，可以通过逆优化退回到解释状态继续执行。

热点代码

- 什么是热点代码：
- 被多次调用的方法
- 被多次执行的循环体
- **多次**到底怎么定义的？----涉及到了我们所说的“热点”，也就是说一旦触及到了多次的这个临界点，那么就触发及时编译
- 这就引入两个新的技术--
- 基于采样的热点探测：虚拟机周期性检查各个线程的调用栈顶，如果发现某个方法经常出现在栈顶，那该方法就是热点方法。实现简单高效，容易获取方法调用关系，缺点是很难精确的确认一个方法的热度，容易因为受到线程阻塞或别的因素影响扰乱热点探测。
- 基于计数器的热点探测：为每个方法建立计数器，统计方法的执行次数，如果执行次数超过一定的阈值就认为是热点方法。统计结果相对严谨，但是为每个方法维护计数器，实现比较麻烦，而且不能直接获取到方法的调用关系。

方法调用计数器　回边计数器





编译技术分析

- 提前编译：
- 一条分支是做与传统C、C++编译器类似的，在程序运行之前把程序代码编译成机器码的静态翻译工作。这点刚好对应了即时编译器的一个缺陷：即时编译器要占用程序运行时间和运算资源。
- 另外一条分支是把原本即时编译器在运行时要做的编译工作提前做好并保存下来，下次运行到这些代码（譬如公共库代码在被同一台机器其他Java进程使用）时直接把它加载进来使用。相对于即时编译器来说，本质上是给即时编译器做缓存加速，去改善Java程序的启动时间，以及需要一段时间预热后才能达到最高性能的问题，我们也可以称之为“动态提前编译（Dynamic AOT）”或者称为即时编译缓存。

即时编译相对于提前编译器的天然优势

- 性能分析制导优化
- 激进预测性优化
- 链接时优化

优化技术

- 方法内联：一是去除方法调用的成本，二是为其他优化建立良好的基础。方法内联膨胀之后可以便于在更大范围上进行后续的优化，可以取得更好的优化效果，各编译器一般会把内联优化放在优化序列的最前面。

```
static class B{
  int value;
  final int get(){
    return value;
  }
  public void foo(){
    y=b.get();
    //...do stuff...
    z=b.get();
    sum=y+z;
  }
}
```

```
public void foo(){
  y=b.value;
  //...do stuff...
  z=b.value;
  sum=y+z;
}
```

逃逸分析

- 逃逸分析的原理：分析对象动态作用域，当一个对象在方法里面被定义之后，它可能被外部方法所引用，例如作为调用参数传递到其它方法中，这种称为方法逃逸；甚至还有可能被外部线程访问到，譬如赋值给可以在其他线程中访问的实例变量，这种称为线程逃逸。
- 栈上分配--使用栈上分配，那大量的对象就会随着方法的结束而自动销毁了，垃圾收集子系统的压力将会下降很多。栈上分配可以支持方法逃逸，不能支持线程逃逸
- 标量替换--如果把一个java对象拆散，根据程序访问的情况，将其用到的成员变量恢复为初始类型来访问，那这个过程就是变量替换。
- 同步消除：线程同步本身是一个相对耗时的过程，如果逃逸分析能够确定一个变量不会逃逸出线程，无法被其他线程访问，那么这个变量的读写肯定不会有竞争，对这个变量实施的同步措施也就可以安全的消除掉。

实例分析

```
public int foo(int x){//初始代码，一个Point类
int xx=x+2;
Point p=new Point(xx,42);
return p.getX();
}
```

原始代码

```
public int foo(int x){//初始代码，一个Point类
int xx=x+2;
Point
p=point_memory_alloc();
p.x=xx;
p.y=42;
return p.x;
}
```

内联优化

```
public int foo(int x){
int xx=x+2;
p.x=xx;
p.y=42;
return p.x;
}
```

逃逸分析--标量替换

```
public int foo(int x){
return x+2;
}
```

无效代码消除

局限性

- 目前来说逃逸分析的应用还不够成熟，主要是它的计算成本很高，甚至不能保证逃逸分析带来的性能收益会高于它的消耗。如果逃逸分析完毕后发现几乎找不到几个不逃逸的对象，那这些运行期耗用的时间就白白浪费了，所以目前的虚拟机只能采用不那么准确，但时间压力相对较小的算法来完成分析。

公共子表达式消除

- 如果一个表达式E之前已经被计算过了，并且从先前的计算到现在E中所有变量的值都没有发生变化，那么E 的这次出现就称为公共子表达式。
- **实例解析**

```
int d = (c * b) * 12 + a + (a + b * c);
```
(1)

```
int d = E * 12 + a + (a + E);
```
(2)

```
int d = E * 13 + a + a;
```
(3)

- 参考链接：
- java虚拟机的编译技术学习（一）
- java虚拟机的编译技术学习（二）