链滴

《深入理解 Java 虚拟机》读书笔记：晚期（运行期）优化

作者：jingqueyimu

原文链接：https://ld246.com/article/1585667447323

来源网站：链滴

正文

在部分商用虚拟机（Sun HotSpot、IBM J9）中，Java 程序最初是通过解释器进行解释执行的，当拟机发现某个方法或代码块的运行特别频繁时，就会把这些代码认定为“热点代码”。为了提高热点码的执行效率，在运行时，虚拟机会把这些代码编译成本地机器码，并进行各种层次的优化。完成这任务的编译器称为即时编译器（Just In Time Compiler，简称 JIT 编译器）。

Java 虚拟机规范并没有规定必须要有即时编译器存在，更没有限定或指导即时编译器如何去实现。所即时编译器的功能完全与虚拟机的具体实现相关。

一、HotSpot 虚拟机内的即时编译器

1、解释器与编译器

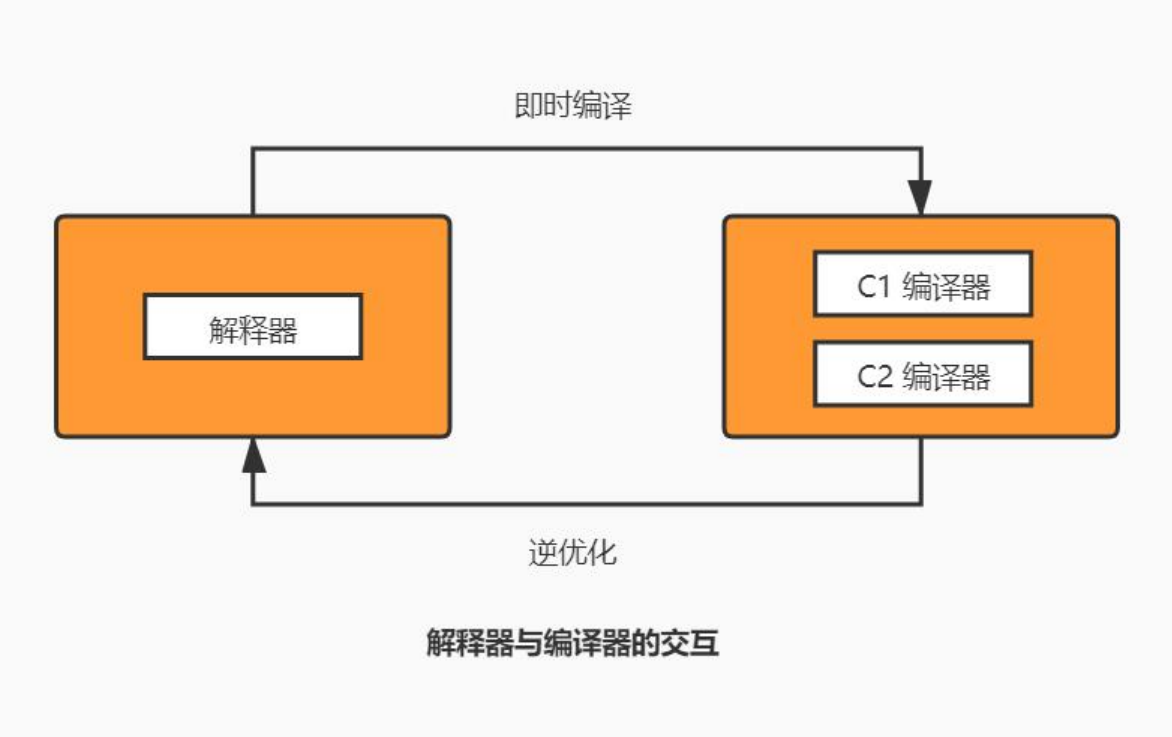
许多主流的商用虚拟机（如 HotSpot、J9），都采用解释器与编译器并存的架构。

（1）解释器、编译器

当程序需要迅速启动和执行时，解释器可以首先发挥作用，省去编译的时间，立即执行。在程序运行，随着时间的推移，编译器把越来越多的代码编译成本地代码后，可以获取更高的执行效率。

当程序运行环境中内存资源限制较大（如部分嵌入式系统），可以使用解释执行节约内存，反之可以用编译执行提升效率。

解释器可以作为编译器激进优化时的一个“逃生门”，让编译器根据概率选择一些大多数时候都能提运行速度的激进优化手段，当激进优化不成立时，可以通过逆优化退回到解释状态继续执行。



解释器与编译器的交互

（2）C1、C2 编译器

HopSpot 虚拟机内置了两个即时编译器，分别称为 Client Compiler（C1 编译器）和 Server Compiler（C2 编译器）。默认采用解释器与其中一个编译器配合的方式工作，程序使用哪个编译器，取决虚拟机是以 Client 模式还是 Server 模式运行。虚拟机会根据自身版本与宿主机器的硬件性能自动选运行模式，用户也可以使用“-client”或“-server”参数强制指定虚拟机的运行模式。

（3）混合模式、解释模式与编译模式

- 混合模式：解释器与编译器搭配使用的方式。
- 解释模式：全部代码都使用解释方式执行，编译器完全不介入工作。可使用“-Xint”参数强制虚拟运行于解释模式。
- 编译模式：优先采用编译方式执行，但是解释器仍会在编译无法进行时介入执行过程。可使用“-Xcmp”强制虚拟机运行于编译模式。

（4）分层编译

为了在程序启动响应速度与运行效率之间达到最佳平衡，HotSpot 虚拟机会逐渐启用分层编译的策略。

分层编译根据编译器编译、优化的规模与耗时，划分出不同的编译层次，其中包括：

- 第 0 层：程序解释执行，解释器不开启性能监控功能，可触发第 1 层编译。
- 第 1 层：也称 C1 编译，将字节码编译为本地代码，进行简单、可靠的优化，必要时加入性能监控逻辑。
- 第 2 层（或 2 层以上）：也称 C2 编译，也是将字节码编译为本地代码，但会启用一些编译耗时较的优化，甚至会根据性能监控信息进行一些不可靠的激进优化。

实施分层编译后，C1、C2 编译器将会同时工作，用 C1 编译器获取更高的编译速度，用 C2 编译器取更好的编译质量，解释执行时也无须再承担收集性能监控信息的任务。

2、编译对象与触发条件

（1）热点代码及编译对象

热点代码：

- 被多次调用的方法。
- 被多次执行的循环体。

两种热点代码的编译对象都是整个方法。第一种热点代码的编译，由于是由方法调用触发的，理所当会以整个方法作为编译对象。第二种热点代码的编译，尽管是由循环体触发的，但编译器仍会以整个法（而不是单独的循环体）作为编译对象。

栈上替换：

被多次执行的循环体成为热点代码时，所触发的编译。因为编译发生在方法执行过程中，因此称之为上替换（也称 OSR 编译），即方法栈帧还在栈上，方法就被替换了。

（2）热点探测

判断一段代码是不是热点代码，是不是需要触发即时编译，这样的行为称为热点探测。

热点探测方式：

● 基于采样的热点探测：虚拟机周期性地检查各个线程的栈顶，如果发现某个方法经常出现在栈顶，这个方法就是“热点代码”。实现简单高效，但很难精确地确认方法的热度。

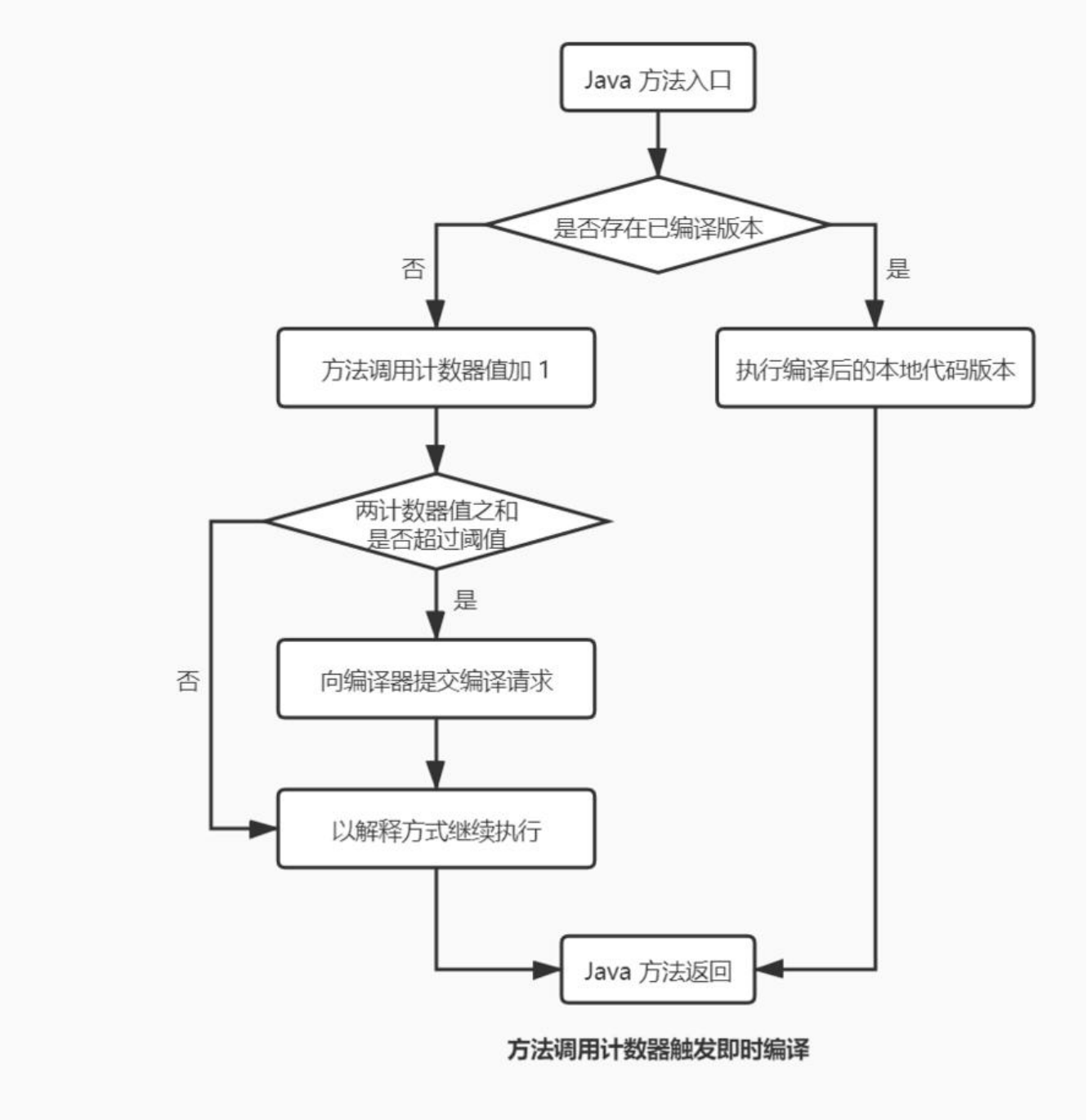
● 基于计数器的热点探测：虚拟机为每个方法（甚至是代码块）建立计数器，统计方法的执行次数，果执行次数超过一定阈值就认为它是“热点代码”。实现麻烦，但统计结果更加精确严谨。

HotSpot 虚拟机使用的是基于计数器的热点探测方法，它为每个方法准备了两类计数器：

● 方法调用计数器：统计方法被调用的次数。

● 回边计数器：统计一个方法中循环体代码执行的次数。在字节码中遇到控制流向后跳转的指令称为回边”。

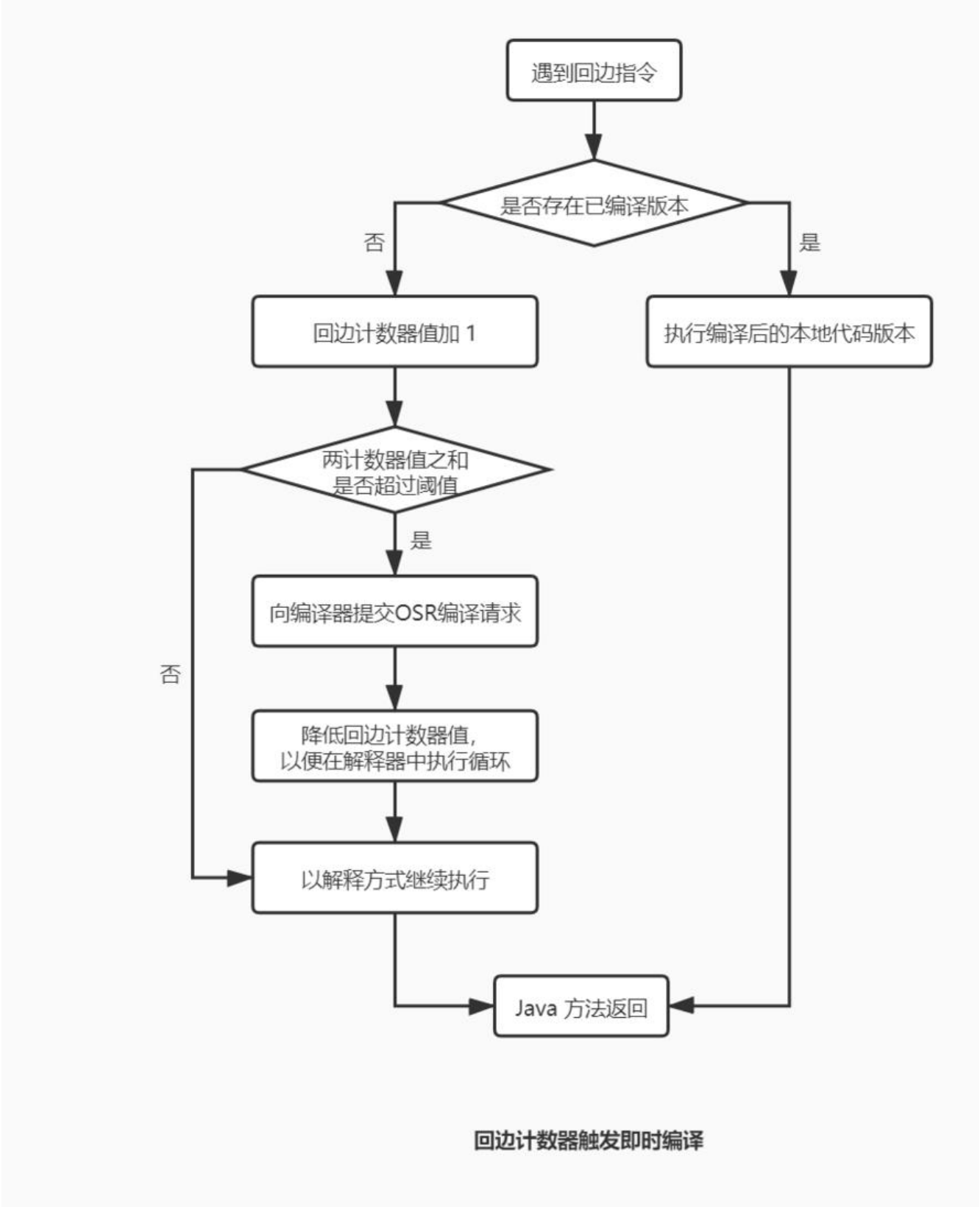
方法调用计数器触发即时编译：



方法调用计数器触发即时编译

原文链接：《深入理解 Java 虚拟机》读书笔记：晚期（运行期）优化

回边计数器触发即时编译：



回边计数器触发即时编译

3、编译过程

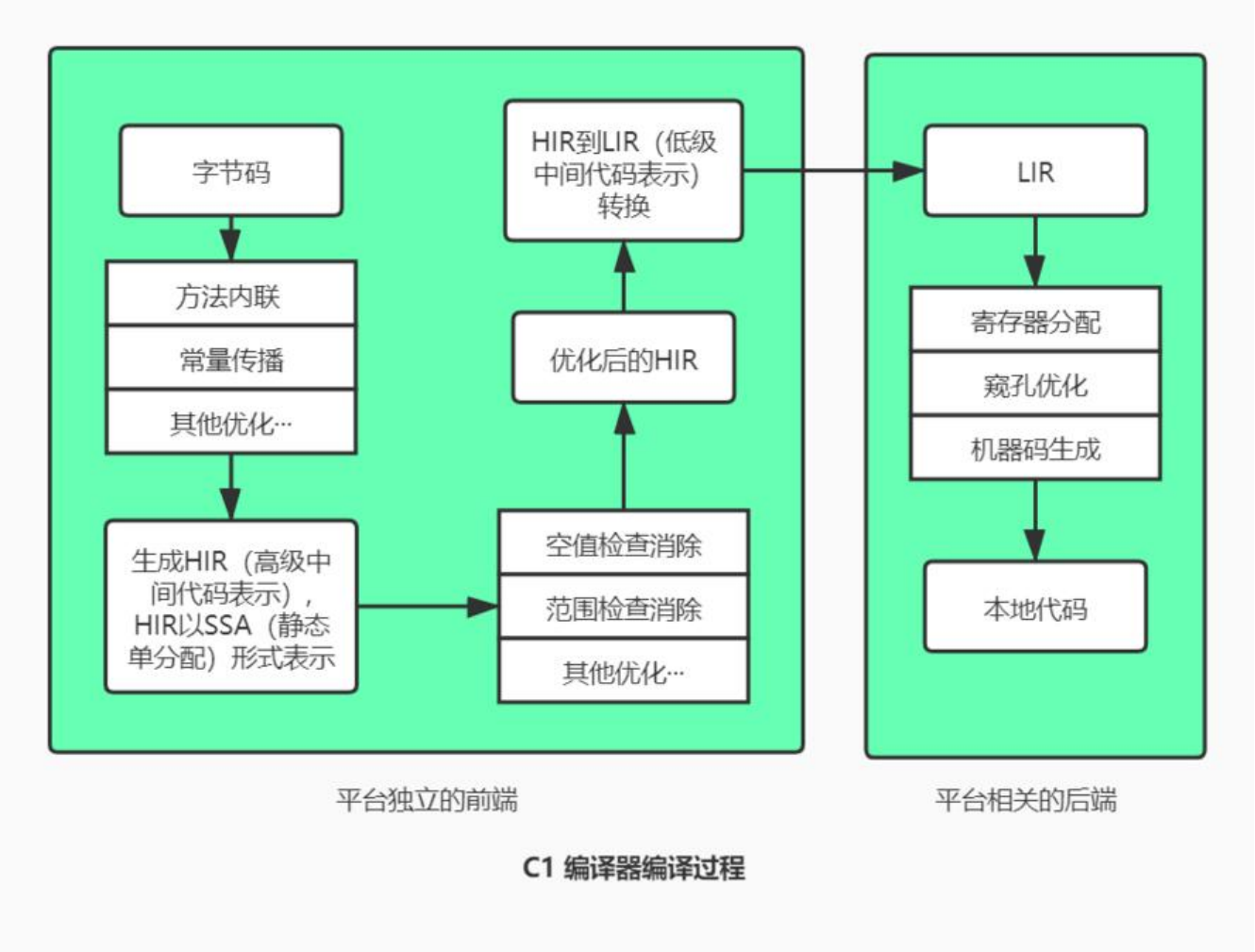
默认设置下，无论是方法调用产生的即时编译请求，还是 OSR 编译请求，虚拟机在代码编译还未完之前，仍会按照解释方式继续执行，而编译动作则在后台的编译线程中进行。

C1 编译器编译过程：

● 第一阶段：一个平台独立的前端将字节码构造成一种高级中间代码表示（HIR），HIR 使用静态单配（SSA）的形式来代表代码值。在此之前，编译器会在字节码上进行方法内联、常量传播等优化。

● 第二阶段：一个平台相关的后端从 HIR 中产生低级中间代码表示（LIR）。在此之前，编译器会在 HR 上进行空值检查消除、范围检查消除等优化。

● 第三阶段：平台相关的后端使用线性扫描算法，在 LIR 上分配寄存器、做窥孔优化，然后产生机器码。



C1 编译器编译过程

二、编译优化技术

1、公共子表达式消除

如果一个表达式 E 已经计算过了，并且从先前计算到现在 E 中所有变量的值都没有变化，那么 E 的次出现就成了公共子表达式。对于这种表达式，没有必要再次进行计算，直接用前面计算过的表达式果代替 E 即可。

2、数组边界检查消除

Java 语言访问数组元素时，虚拟机系统会自动进行上下界的范围检查，一旦访问超出范围，将抛出一运行时异常：java.lang.ArrayIndexOutOfBoundsException。

数组边界检查使得程序员即便没有专门编写防御代码，也可以避免大部分的溢出攻击。但对于虚拟机执行子系统来说，每次数组元素的读写都带有一次隐含的条件判定操作，如果程序中拥有大量数组访代码，无疑大大增加了性能负担。

编译器可以通过数据流分析判定数组下标是否会越界，如果分析后确定不会越界，那么可以把数组的下界检查消除。

3、方法内联

把目标方法的代码“复制”到发起调用的方法之中，避免发生真实的方法调用。

（1）类型继承关系分析

对于一个虚方法，编译期做内联的时候根本无法确定应该使用哪个方法版本，为了解决这个问题，引了类型继承关系分析（Class Hierarchy Analysis，CHA）技术。CHA 用于确定在目前已加载的类中某个接口是否有多于一种的实现，某个类是否存在子类、子类是否为抽象类等信息。

（2）方法内联过程

- 如果是非虚方法，直接进行内联。
- 如果是虚方法，则向 CHA 查询是否有多个目标版本。
  - 如果只有一个版本，则进行守护内联。
  - 如果有多个版本，则使用内联缓存完成方法内联。

守护内联：

当虚方法只有一个目标版本时，也可以进行内联，但这种内联属于激进优化，需要预留一个“逃生门，这种内联称为守护内联。进行守护内联时，如果后续执行过程中，加载了导致继承关系发生变化的类，则需要抛弃已经编译的代码，退回到解释状态执行，或者重新进行编译。

内联缓存：

内联缓存是一个建立在目标方法正常入口之前的缓存。它的工作原理是：在未发生方法调用前，内联存状态为空，第一次调用发生后，缓存记录下方法接收者的版本信息，并且每次进行方法调用时都会较接收者版本。如果接收者版本一致，那么这个内联还可以用下去，如果不一致，说明程序使用了虚法的多态特性，此时会取消内联，查找虚方法表进行方法分派。

4、逃逸分析

逃逸分析的基本行为就是分析对象动态作用域：当一个对象在方法中定义后，如果它被外部方法所引或被外部线程访问到，那么就说这个对象发生了逃逸。

- 方法逃逸：对象被外部方法所引用，比如作为调用参数传递到其他方法中。
- 线程逃逸：对象被外部线程访问到。

如果能证明一个对象不会逃逸到方法或线程之外，也就是别的方法或线程无法通过任何途径访问到这对象，则可能为这个变量进行一些高效的优化。

（1）栈上分配

如果确定一个对象不会逃逸出方法之外，那么可以让这个对象在栈上分配内存。这样对象所占用的内空间就可以随栈帧出栈而销毁，从而减少了垃圾收集系统的压力。

（2）同步消除

如果确定一个变量不会逃逸出线程，那么这个变量的读写肯定不会有竞争，因此可以消除掉这个变量线程同步措施。

（3）标量替换

如果确定一个对象不会被外部访问，并且这个对象可以被拆散的话，那么程序真正执行时可能不创建个对象，而改为创建它的若干个被这个方法使用到的成员变量来代替，这个过程称为标量替换。

将对象拆分后，除了可以让对象的成员变量在栈上分配和读写之外，还可以为后续进一步的优化手段建条件。

标量与聚合量：

- 标量：如果一个数据无法再分解成更小的数据来表示，则称为标量。比如 int、long 等原始数据类。
- 聚合量：如果一个数据可以继续分解，则称为聚合量。比如 Java 对象。