## 深入理解JVM虚拟机

### 二、虚拟机类加载

#### 1、类加载机制

参数：

xverify. ：none关掉大部分类验证措施，缩短类加载时间

##### 1）加载

通过类全限定名获取比类的二进制字节流

生成class对象

##### 2）验证

文件格式，元数据，字节码，符号引用等验证

##### 3）准备

类变量分配内存并赋初始值

##### 4）解析

常量池的符号引用替换为直接引用

什么是符号引用，直接引用？

符号引用，任意形式的字面量

直接引用，直接目标指针相对偏移量，目标句柄

四种解析

类或接口解析

字段解析

方法解析

接口方法解析

##### 5）初始化

执行类构造器方法的过程

#### 2、类与类加载器

任意一个类，必须由加载他的类加载器和这个类本身确定其在JAVA虚拟机中的唯一性

类加载器

##### A）启动类加载器

javahome. lib

##### B）扩展类加载器

javalhome. lib. ext

##### C）应用程序类加载器

#### 3、双亲委派模型

类加载器的加载请求委派给父类加载器，父类完不成，再由子类登场

双亲委派模型组织类加载器之间的关系，保证Java类型的完整性，Java程序的稳定运作

双亲委派模型的实现

classloader. loadclass()

检查请求加载的类型是否已经加载过，没有，调用父类加载器的loadclass方法，若父类加载器为空，则默认使用启动类加载器作为父类加载器

### 三、程序编译与代码优化

#### 1、前端编译与优化

前端编译器：把\*.java文件转变成\*.class文件的过程

JDK的Javac

后端编译器：运行期把字节码转变成本地机器码的过程

HotSpot的C1、C2编译器，Graal编译器

提前编译器：直接把程序编译成与目标机器指令集相关的二进制代码的过程

JDK的Jaotc

##### Javac编译器

从Javac代码的总体结构来看，编译过程大致可以分为1个准备过程和3个处理过程

###### A）准备过程：初始化插入式注解处理器。

###### B）解析与填充符号表过程

·词法、语法分析。将源代码的字符流转变为标记集合，构造出抽象语法树。

·填充符号表。产生符号地址和符号信息。

###### C）插入式注解处理器的注解处理过程

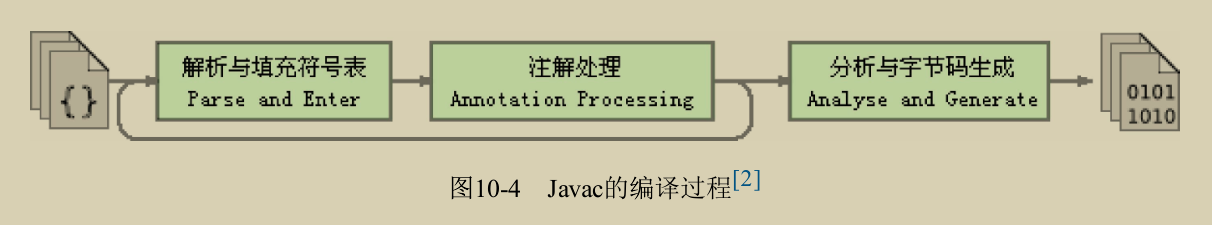
###### D）分析与字节码生成过程

·标注检查。对语法的静态信息进行检查。

·数据流及控制流分析。对程序动态运行过程进行检查。

·解语法糖。将简化代码编写的语法糖还原为原有的形式。

·字节码生成。将前面各个步骤所生成的信息转化成字节码。



##### 2）Java语法糖

###### A）泛型

泛型的本质是**参数化类型**（Parameterized Type）或者**参数化多态**（Parametric Polymorphism）的应用，即可以将操作的数据类型指定为方法签名中的一种特殊参数，这种参数类型能够用在类、接口和方法的创建中，分别构成泛型类、泛型接口和泛型方法

Java选择的泛型是**类型擦除式泛型**，并且在相应的地方加入了强制类型转换

**泛型的类型擦除：**

编译时期将泛型类型字节码擦除，不过元数据保留了泛型信息

Java泛型的问题：

反编译后泛型不见了，泛型类型变为裸类型，只是访问的时候强制转换

泛型变慢的原因

遇到需要自动拆装箱的数据，有这部分的额外开销，影响泛型速度

1. 首先，使用擦除法实现泛型直接导致了对原始类型（Primitive Types）数据的支持又成了新的麻烦

2）第二，运行期无法取到泛型类型信息，会让一些代码变得相当啰嗦

此外，运行期无法取得泛型的类型信息

最后泛型遇到重载，泛型擦除导致签名一模一样而无法重载

自动拆装箱缺陷

包装类在 == 时候不会自动拆箱

##### 3）插入式注解处理器

#### 2、后端编译与优化

后端编译：

编译器无论在何时、在何种状态下把Class文件转换成与本地基础设施（硬件指令集、操作系统）相关的二进制机器码，它都可以视为整个编译过程的后端

即时编译器：这里指hotspot虚拟机内部的内置即时编译器

##### 1）即时编译器：

频繁被虚拟机运行的代码称为热点代码，为提升热点代码的执行效率，虚拟机把热点代码翻译为本地机器码（这就是即时编译器干的事情），并以各种手段尽可能地进行代码优化

##### 2）解决问题：

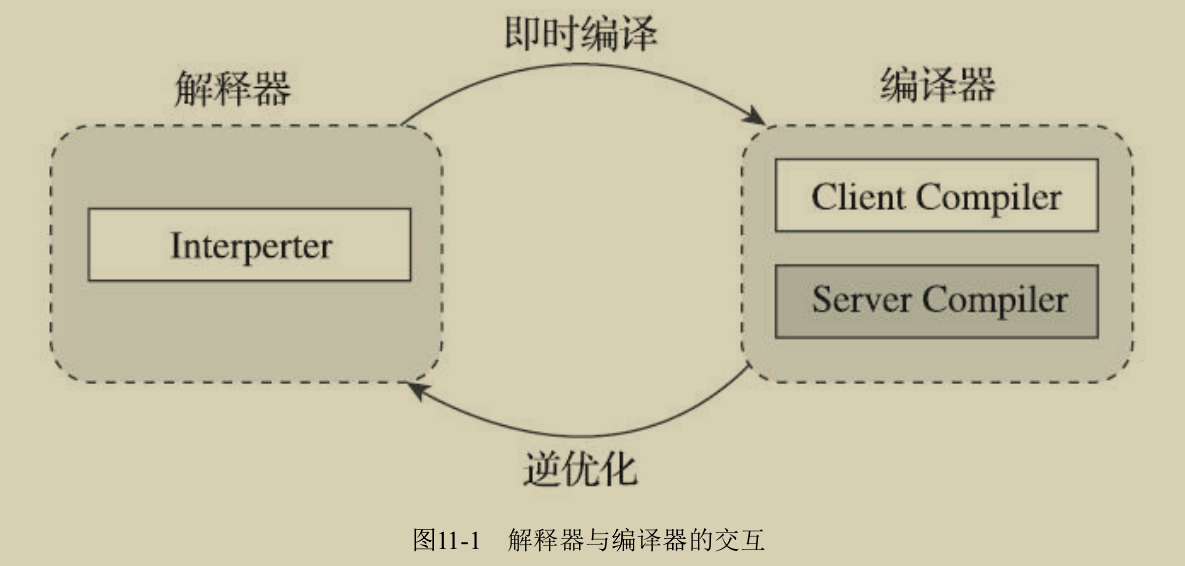
·为何HotSpot虚拟机要使用解释器与即时编译器并存的架构？

·为何HotSpot虚拟机要实现两个（或三个）不同的即时编译器？

·程序何时使用解释器执行？何时使用编译器执行？

·哪些程序代码会被编译为本地代码？如何编译本地代码？

·如何从外部观察到即时编译器的编译过程和编译结果？



HotSpot虚拟机中内置了两个（或三个）即时编译器，其中有两个编译器存在已久，分别被称为“客户端编译器”（Client Compiler）和“服务端编译器”（Server Compiler），或者简称为C1编译器和C2编译器（部分资料和JDK源码中C2也叫Opto编译器），第三个是在JDK 10时才出现的、长期目标是代替C2的Graal编译器

hotspot内置两到三个即时编译器

C1：客户端编译器

C2：服务端编译器

Graal

解释器与编译器搭配使用的方式在虚拟机中被称为“混合模式”（Mixed Mode）

“-Xint”强制虚拟机运行于“解释模式”

“-Xcomp”强制虚拟机运行于“编译模式”

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虚拟机在编译子系统中加入了分层编译的功能

##### 3）hopspot分层编译：

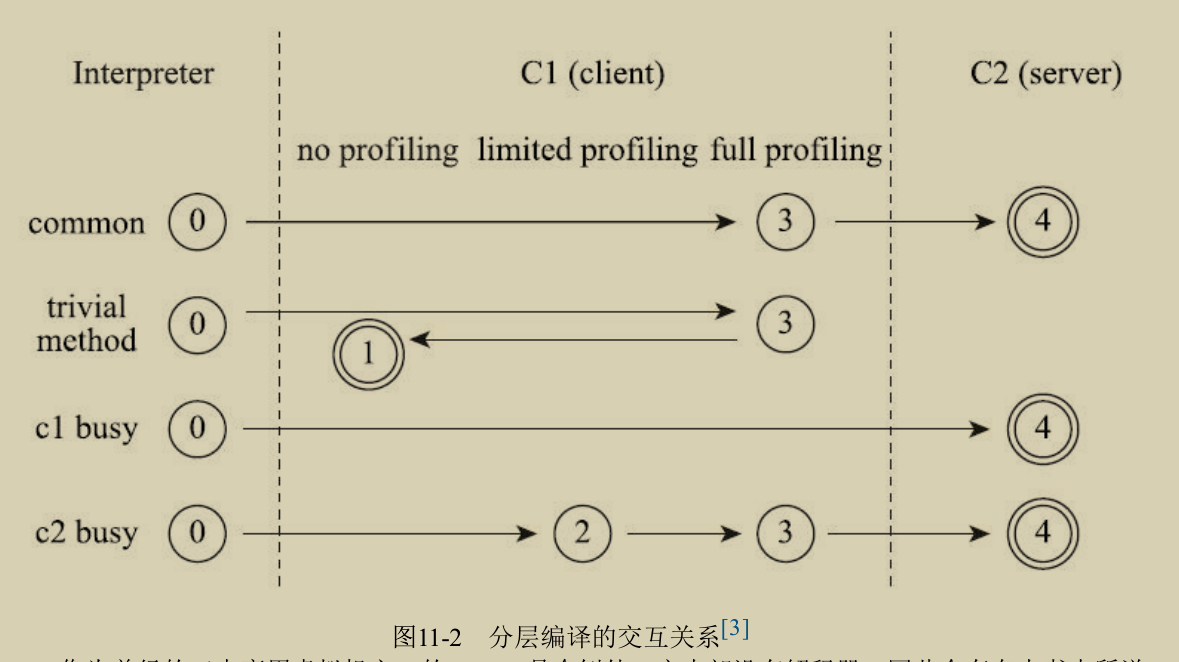
**·第0层。程序纯解释执行**，并且解释器不开启性能监控功能（Profiling）。

**·第1层。使用客户端编译器将字节码编译为本地代码来运行**，进行简单可靠的稳定优化，不开启性能监控功能。

**·第2层。仍然使用客户端编译器执行**，仅**开启方法及回边次数统计**等有限的性能监控功能。

**·第3层。仍然使用客户端编译器执行，开启全部性能监控**，除了第2层的统计信息外，还会收集如分支跳转、虚方法调用版本等全部的统计信息。

**·第4层。使用服务端编译器将字节码编译为本地代码**，相比起客户端编译器，服务端编译器会启用更多编译耗时更长的优化，还会根据性能监控信息进行一些不可靠的激进优化。



编译对象触发条件

**热点代码：**

·被多次调用的方法。

·被多次执行的循环体。

**热点探测**

1）基于采样的热点探测，J9

实现简单高效，但容易因为受到线程阻塞或别的外界因素的影响而扰乱热点探测

2）基于计数器的热点探测，hotspot

它的统计结果相对来说更加精确严谨

Hotspot：方法调用计数器，回边计数器（提供阈值）

**方法调用计数器**，服务端默认10000次，-XX: CompileThreshold可设置

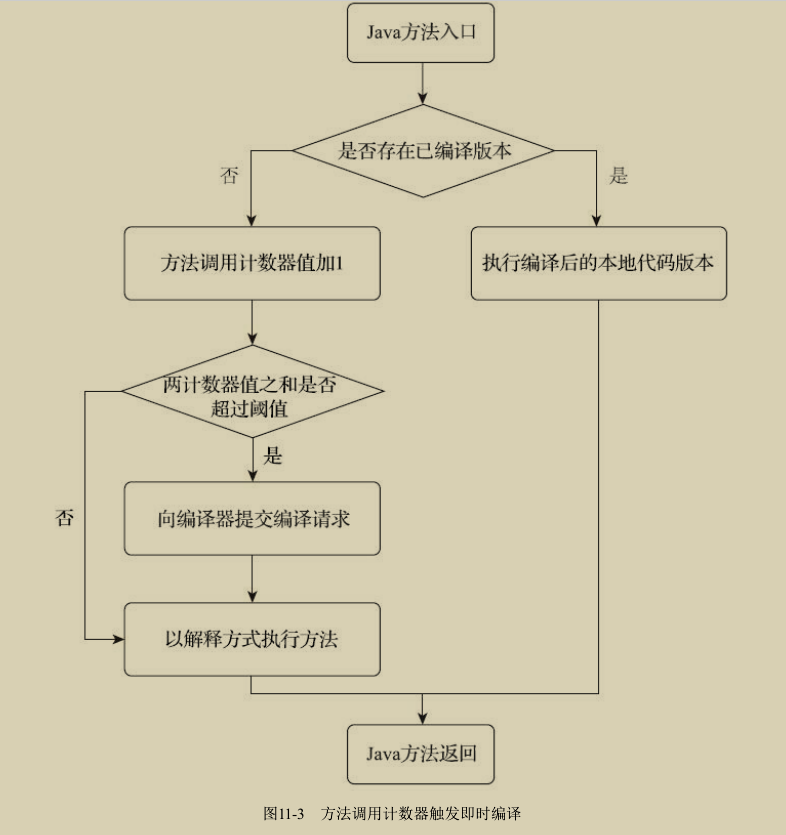
**热度衰减**

一定时间内（半衰周期），方法调用次数达不到即时编译器限定次数，此时调用计数器会衰减一半

可以使用

-XX: UseCounterDecay 关闭热度衰减

-XX: CounterHalfLifeTime 参数设置半衰周期的时间，单位是秒。



回边计数器：作用是统计一个方法中循环体代码执行的次数

阀值，该参数控制阀值

-XX：OnStackReplacePercentage来间接调整回边计数器的阈值

·虚拟机运行在**客户端**模式下，回边计数器阈值计算公式为：方法调用计数器阈值（-XX：CompileThreshold）乘以OSR比率（-XX：OnStackReplacePercentage）除以100。

·虚拟机运行在**服务端**模式下，回边计数器阈值的计算公式为：方法调用计数器阈值（-XX：

CompileThreshold）乘以（OSR比率（-XX：OnStackReplacePercentage）减去解释器监控比率（-XX：InterpreterProfilePercentage）的差值）除以100。

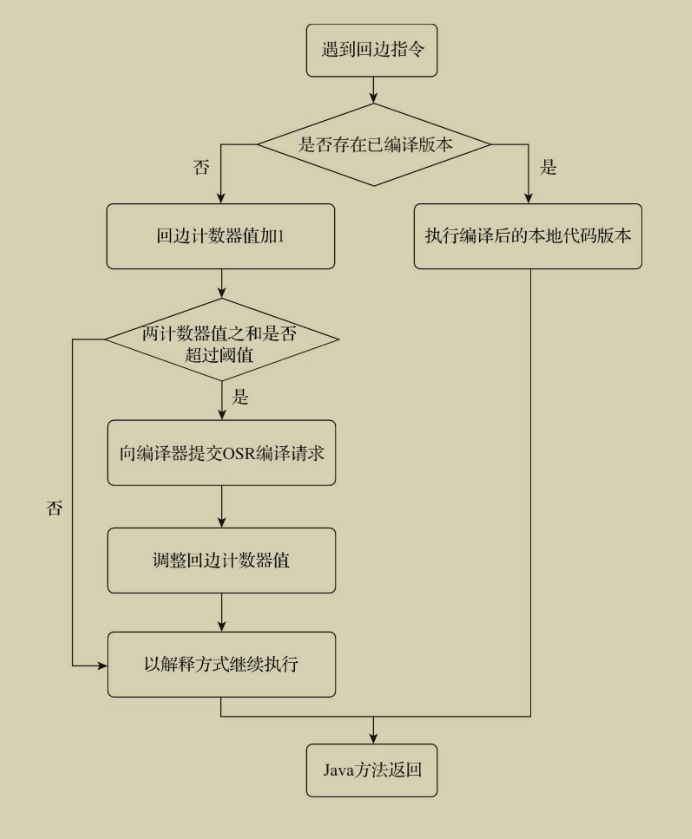


图11-4　回边计数器触发即时编译

编译过程：

