|  |  |
| --- | --- |
| **在HotSpot源代码和文档中可以找到术语表。**  **(https://openjdk.java.net/groups/hotspot/docs/RuntimeOverview.html)** | |
| A work in progress, especially as the HotSpot VM evolves. But a place to put definitions of things so we only have to define them once. There are empty entries (marked *TBD* for "to be defined") because we think of things that we need to define faster than we think of good definitions.  **adaptive spinning**  An optimization technique whereby a thread spins waiting for a change-of-state to occur (typically a flag that represents some event has occurred - such as the release of a lock) rather than just blocking until notified that the change has occurred. The "adaptive" part comes from the policy decisions that control how long the thread will spin until eventually deciding to block.  **biased locking**  An optimization in the VM that leaves an object as logically locked by a given thread even after the thread has released the lock. The premise is that if the thread subsequently reacquires the lock (as often happens), then reacquisition can be achieved at very low cost. If a different thread tries to acquire a biased lock then the bias must be revoked from the current bias owner.  **block start table**  A table that shows, for a region of the heap, where the object starts that comes on to this region from lower addresees. Used, for example, with the [card table](https://openjdk.java.net/groups/hotspot/docs/HotSpotGlossary.html#cardTable) variant of the [remembered set](https://openjdk.java.net/groups/hotspot/docs/HotSpotGlossary.html#rememberedSet).  **bootstrap classloader**  The logical classloader that has responsibility for loading the classes (and resources) that are found in the boot-classpath - typically the core Java platform classes. Typically implemented as part of the VM, by historical convention the bootstrap classloader is represented by NULL at the Java API level.  **bytecode verification**  A step in the linking process of a class where the methods bytecodes are analyzed to ensure type-safety.  **C1 compiler**  Fast, lightly optimizing bytecode compiler. Performs some value numbering, inlining, and class analysis. Uses a simple CFG-oriented SSA "high" IR, a machine-oriented "low" IR, a linear scan register allocation, and a template-style code generator.  **C2 compiler**  Highly optimizing bytecode compiler, also known as 'opto'. Uses a "sea of nodes" SSA "ideal" IR, which lowers to a machine-specific IR of the same kind. Has a graph-coloring register allocator; colors all machine state, including local, global, and argument registers and stack. Optimizations include global value numbering, conditional constant type propagation, constant folding, global code motion, algebraic identities, method inlining (aggressive, optimistic, and/or multi-morphic), intrinsic replacement, loop transformations (unswitching, unrolling), array range check elimination.  **card table**  A kind of [remembered set](https://openjdk.java.net/groups/hotspot/docs/HotSpotGlossary.html#rememberedSet) that records where oops have changed in a generation.  **class data sharing**  A startup optimization that records the in-memory form of some classes, so that that form can be mapped into memory by a subsequent run of the virtual machine, rather than loading those classes from their class files.  **class hierachy analysis**  Also known as 'CHA'. Analysis of the class tree used by a compiler to determine if the receiver at a virtual call site has a single implementor. If so, the callee can be inlined or the compiler can employ some other static call mechanism.  **code cache**  A special heap that holds compiled code. These objects are not relocated by the GC, but may contain oops, which serve as GC roots.  **compaction**  A garbage collection technique that results in live objects occupying a dense portion of the virtual address space, and available space in another portion of the address space. Cf. [free list](https://openjdk.java.net/groups/hotspot/docs/HotSpotGlossary.html#freeList).  **concurrency**  Concurrency, or more specifically concurrent programming, is the logical simultaneous execution of multiple instruction streams. If multiple processors are available then the logical simultaneity can be physical simultaneity - this is known as 'parallelism'  **concurrent garbage collection**  A garbage collection algorithm that does most (if not all) of its work while the Java application threads are still running.  **copying garbage collection**  A garbage collection algorithm that moves objects during the collection.  **deoptimization**  The process of converting an compiled (or more optimized) stack frame into an interpreted (or less optimized) stack frame. Also describes the discarding of an nmethod whose dependencies (or other assumptions) have been broken. Deoptimized nmethods are typically recompiled to adapt to changing application behavior. Example: A compiler initially assumes a reference value is never null, and tests for it using a trapping memory access. Later on, the application uses null values, and the method is deoptimized and recompiled to use an explicit test-and-branch idiom to detect such nulls.  **dependency**  An optimistic assumption associated with an nmethod, which allowed the compiler to emit improved code into the nmethod. Example: A given class has no subclasses, which simplifies method dispatch and type testing. The loading of new classes (or replacement of old classes) can cause dependencies to become false, which requires dependent nmethods to be discarded and activations of those nmethods to be deoptimized.  **eden**  A part of the Java object heap where object can be created efficiently.  **free list**  A storage management technique in which unused parts of the Java object heap are chained one to the next, rather than having all of the unused part of the heap in a single block.  **garbage collection**  The automatic management of storage.  **garbage collection root**  A pointer into the Java object heap from outside the heap. These come up, e.g., from static fields of classes, local references in activation frames, etc.  **GC map**  A description emitted by the JIT (C1 or C2) of the locations of oops in registers or on stack in a compiled stack frame. Each code location which might execute a safepoint has an associated GC map. The GC knows how to parse a frame from a stack, to request a GC map from a frame's nmethod, and to unpack the GC map and manage the indicated oops within the stack frame.  **generational garbage collection**  A storage management technique that separates objects expected to be referenced for different lengths of time into different regions of the heap, so that different algorithms can be applied to the collection of those regions.  **handle**  A memory word containing an oop. The word is known to the GC, as a root reference. C/C++ code generally refers to oops indirectly via handles, to enable the GC to find and manage its root set more easily. Whenever C/C++ code blocks in a safepoint, the GC may change any oop stored in a handle. Handles are either 'local' (thread-specific, subject to a stack discipline though not necessarily on the thread stack) or global (long-lived and explicitly deallocated). There are a number of handle implementations throughout the VM, and the GC knows about them all.  **hot lock**  A lock that is highly contended.  **interpreter**  A VM module which implements method calls by individually executing bytecodes. The interpreter has a limited set of highly stylized stack frame layouts and register usage patterns, which it uses for all method activations. The Hotspot VM generates its own interpreter at start-up time.  **JIT compilers**  An on-line compiler which generates code for an application (or class library) during execution of the application itself. ("JIT" stands for "just in time".) A JIT compiler may create machine code shortly before the first invocation of a Java method. Hotspot compilers usually allow the interpreter ample time to "warm up" Java methods, by executing them thousands of times. This warm-up period allows a compiler to make better optimization decisions, because it can observe (after initial class loading) a more complete class hierarchy. The compiler can also inspect branch and type profile information gathered by the interpreter.  **JNI**  The Java Native Interface - a specification and API for how Java code can call out to native C code, and how native C code can call into the Java VM  **JVM TI**  The Java Virtual Machine Tools Interface - a standard specification and API that is used by development and monitoring tools. See [JVM TI](https://openjdk.java.net/groups/hotspot/docs/Serviceability.html#tjvmti) for more information.  **klass pointer**  The second word of every object header. Points to another object (a metaobject) which describes the layout and behavior of the original object. For Java objects, the "klass" contains a C++ style "vtable".  **mark word**  The first word of every object header. Usually a set of bitfields including synchronization state and identity hash code. May also be a pointer (with characteristic low bit encoding) to synchronization related information. During GC, may contain GC state bits.  **nmethod**  A block of executable code which implements some Java bytecodes. It may be a complete Java method, or an 'OSR' method. It routinely includes object code for additional methods inlined by the compiler.  **object header**  Common structure at the beginning of every GC-managed heap object. (Every oop points to an object header.) Includes fundamental information about the heap object's layout, type, GC state, synchronization state, and identity hash code. Consists of two words. In arrays it is immediately followed by a length field. Note that both Java objects and VM-internal objects have a common object header format.  **object promotion**  The act of copying an object from one generation to another.  **old generation**  A region of the Java object heap that holds object that have remained referenced for a while.  **on-stack replacement**  Also known as 'OSR'. The process of converting an interpreted (or less optimized) stack frame into a compiled (or more optimized) stack frame. This happens when the interpreter discovers that a method is looping, requests the compiler to generate a special nmethod with an entry point somewhere in the loop (specifically, at a backward branch), and transfers control to that nmethod. A rough inverse to deoptimization.  **oop**  An object pointer. Specifically, a pointer into the GC-managed heap. (The term is traditional. One 'o' may stand for 'ordinary'.) Implemented as a native machine address, not a handle. Oops may be directly manipulated by compiled or interpreted Java code, because the GC knows about the liveness and location of oops within such code. (See GC map.) Oops can also be directly manipulated by short spans of C/C++ code, but must be kept by such code within handles across every safepoint.  **parallel classloading**  The ability to have multiple classes/type be in the process of being loaded by the same classloader at the same time.  **parallel garbage collection**  A garbage collection algorithm that uses multiple threads of control to perform more efficiently on multi-processor boxes.  **permanent generation**  A region of the address space that holds object allocated by the virtual machine itself, but which is managed by the garbage collector. The permanent generation is mis-named, in that almost all of the objects in it *can* be collected, though they tend to be referenced for a long time, so they rarely become garbage.  **remembered set**  A data structure that records pointers between generations.  **safepoint**  A point during program execution at which all GC roots are known and all heap object contents are consistent. From a global point of view, all threads must block at a safepoint before the GC can run. (As a special case, threads running JNI code can continue to run, because they use only handles. During a safepoint they must block instead of loading the contents of the handle.) From a local point of view, a safepoint is a distinguished point in a block of code where the executing thread may block for the GC. Most call sites qualify as safepoints. There are strong invariants which hold true at every safepoint, which may be disregarded at non-safepoints. Both compiled Java code and C/C++ code be optimized between safepoints, but less so across safepoints. The JIT compiler emits a GC map at each safepoint. C/C++ code in the VM uses stylized macro-based conventions (e.g., TRAPS) to mark potential safepoints.  **sea-of-nodes**  The high-level intermediate representation in C2. It is an SSA form where both data and control flow are represented with explicit edges between nodes. It differs from forms used in more traditional compilers in that nodes are not bound to a block in a control flow graph. The IR allows nodes to float within the sea (subject to edge constraints) until they are scheduled late in the compilation process.  **Serviceability Agent (SA)**  The Serviceablity Agent is collection of Sun internal code that aids in debugging HotSpot problems. It is also used by several JDK tools - jstack, jmap, jinfo, and jdb. See [SA](https://openjdk.java.net/groups/hotspot/docs/Serviceability.html#tsa) for more information.  **stackmap**  Refers to the StackMapTable attribut e or a particular StackMapFrame in the table.  **StackMapTable**  An attribute of the Code attribute in a classfile which contains type information used by the new verifier during verification. It consists of an array of StackMapFrames. It is generated automatically by javac as of JDK6.  **survivor space**  A region of the Java object heap used to hold objects. There are usually a pair of survivor spaces, and collection of one is achieved by copying the referenced objects in one survivor space to the other survivor space.  **synchronization**  In general terms this is the coordination of concurrent activities to ensure the safety and liveness properties of those activities. For example, protecting access to shared data by using a lock to guard all code paths to that data.  **TLAB**  Thread-local allocation buffer. Used to allocate heap space quickly without synchronization. Compiled code has a "fast path" of a few instructions which tries to bump a high-water mark in the current thread's TLAB, successfully allocating an object if the bumped mark falls before a TLAB-specific limit address.  **uncommon trap**  When code generated by C2 reverts back to the interpreter for further execution. C2 typically compiles for the common case, allowing it to focus on optimization of frequently executed paths. For example, C2 inserts an uncommon trap in generated code when a class that is uninitialized at compile time requires run time initialization.  **verifier**  The software code in the VM which performs bytecode verification.  **VM Operations**  Operations in the VM that can be requested by Java threads, but which must be executed, in serial fashion by a specific thread known as the VM thread. These operations are often synchronous, in that the requester will block until the VM thread has completed the operation. Many of these operations also require that the VM be brought to a safepoint before the operation can be performed - a garbage collection request is a simple example.  **write barrier**  Code that is executed on every oop store. For example, to maintain a remembered set.  **young generation**  A region of the Java object heap that holds recently-allocated objects. | 正在进行的工作，特别是在HotSpot VM发展的过程中。而是一个可以定义事物的地方，所以我们只需要定义一次。有一些空条目(标记为TBD表示“待定义”)，因为我们考虑需要定义的东西比考虑好的定义要快。  **自适应旋转**  一种优化技术，线程旋转以等待状态的改变(通常是表示某个事件已经发生的标志——比如释放锁)，而不是仅仅阻塞直到被通知发生了改变。“自适应”部分来自控制线程在最终决定阻塞前旋转多长时间的策略决策。  **偏向锁**  虚拟机中的一种优化，即使在线程释放了锁之后，对象仍然被给定的线程逻辑锁定。前提是，如果线程随后重新获得锁(经常发生)，那么可以以非常低的成本实现重新获取。如果另一个线程试图获取偏置锁，则必须从当前偏置所有者撤销该偏置。  **块开始表**  一个表，它显示了堆的一个区域，从这里开始的对象从较低的地址进入这个区域。例如，与记忆集合的卡片表变体一起使用。  **引导类加载器**  负责加载在引导类路径中找到的类(和资源)的逻辑类装入器——通常是核心Java平台类。通常是作为VM的一部分实现的，根据历史惯例，引导类装入器在Java API级别上由NULL表示。  **字节码验证**  类链接过程中的一个步骤，在此步骤中分析方法字节码以确保类型安全。  **C1编译器**  快速，轻微优化的字节码编译器。执行一些值编号、内联和类分析。使用一个简单的面向cfg的SSA“高”IR，一个面向机器的“低”IR，一个线性扫描寄存器分配和一个模板样式的代码生成器。  **C2编译器**  高度优化的字节码编译器，也称为“opto”。使用“节点的海洋”SSA“理想”IR，它降低到相同类型的特定于机器的IR。具有图形着色寄存器分配器;给所有机器状态着色，包括局部、全局和参数寄存器和堆栈。优化包括全局值编号、条件常量类型传播、常量折叠、全局代码运动、代数恒等式、方法内联(积极的、乐观的和/或多态的)、内在替换、循环转换(不切换、展开)、数组范围检查消除。  **卡表**  一种记忆集记录了在一代人中发生的变化。  **类数据共享**  一种启动优化，记录一些类在内存中的形式，这样在虚拟机的后续运行中，这种形式可以映射到内存中，而不是从类文件加载这些类。  **类层次分析**  又称“CHA”。编译器用来确定虚调用站点的接收者是否只有一个实现者的类树分析。如果是这样，被调用方可以内联，或者编译器可以使用一些其他的静态调用机制。  代码缓存  保存已编译代码的特殊堆。这些对象不会被GC重新定位，但可能包含作为GC根的oops。  压实  一种垃圾收集技术，导致活动对象占用虚拟地址空间的密集部分，并占用地址空间的另一部分的可用空间。参见空闲列表。  并发性  并发，或者更具体地说并发编程，是多个指令流的逻辑同步执行。如果有多个处理器可用，那么逻辑上的同时性也可以是物理上的同时性——这被称为“并行性”。  并发垃圾收集  一种垃圾收集算法，在Java应用程序线程仍在运行时完成其大部分(如果不是全部)工作。  复制垃圾收集  垃圾收集算法，在收集期间移动对象。  逆优化  将已编译(或优化程度更高)的堆栈帧转换为解释(或优化程度较低)的堆栈帧的过程。还描述了抛弃依赖项(或其他假设)被破坏的nmethod。未优化的n方法通常会重新编译以适应不断变化的应用程序行为。示例:编译器最初假设一个参考值永远不会为空，然后使用捕获内存访问来测试它。稍后，应用程序将使用空值，该方法将被取消优化并重新编译，以使用显式的test-and-branch习惯用法来检测此类空值。  依赖  一个与n方法相关的乐观假设，它允许编译器将改进的代码发送到n方法中。示例:给定的类没有子类，这简化了方法分派和类型测试。加载新类(或替换旧类)可能会导致依赖变为false，这需要丢弃依赖的n方法，并取消对这些n方法的激活。  伊甸园  Java对象堆的一部分，可以在其中高效地创建对象。  空闲列表  一种存储管理技术，其中将Java对象堆中未使用的部分一个链接到另一个，而不是将堆中所有未使用的部分都放在一个块中。  垃圾收集  存储的自动化管理。  垃圾收集根  从堆外指向Java对象堆的指针。例如，它们来自类的静态字段、激活框架中的局部引用等。  GC地图  JIT (C1或C2)对已编译堆栈帧中oops在寄存器或堆栈上的位置的描述。每个可能执行安全点的代码位置都有一个关联的GC映射。GC知道如何从堆栈中解析帧，如何从帧的n方法中请求GC映射，以及如何解包GC映射并在堆栈帧中管理指示的oops。  分代垃圾收集  一种存储管理技术，它将需要在不同时间长度内引用的对象分离到堆的不同区域，以便不同的算法可以应用于这些区域的集合。  处理  一个包含oop的存储字。GC知道这个词，作为根引用。C/ c++代码通常通过句柄间接地引用oops，以便GC更容易地找到和管理其根集。每当C/ c++代码块在安全点时，GC可以更改存储在句柄中的任何oop。句柄要么是“局部的”(线程特定的，服从堆栈规则，但不一定在线程堆栈上)，要么是全局的(长期存在并显式释放)。VM中有许多句柄实现，并且GC知道所有这些句柄实现。  热的锁  一种高度竞争的锁。  翻译  一个VM模块，它通过单独执行字节码来实现方法调用。解释器有一组高度程式化的堆栈框架布局和寄存器使用模式，用于所有方法的激活。Hotspot VM在启动时生成自己的解释器。  JIT编译器  在线编译器，它在应用程序本身执行期间为应用程序(或类库)生成代码。(“JIT”代表“及时”。)JIT编译器可能会在第一次调用Java方法之前不久创建机器码。Hotspot编译器通常允许解释器有充足的时间来“预热”Java方法，通过执行它们数千次。这个预热阶段允许编译器做出更好的优化决策，因为它可以观察(在初始类加载之后)一个更完整的类层次结构。编译器还可以检查由解释器收集的分支和类型概要信息。  JNI  Java本地接口——Java代码如何调用本机C代码，以及本机C代码如何调用Java VM的规范和API  JVM TI  Java虚拟机工具接口——开发和监控工具使用的标准规范和API。参见JVM TI了解更多信息。  klass指针  每个对象头部的第二个单词。指向另一个对象(元对象)，它描述了原始对象的布局和行为。对于Java对象，“klass”包含一个c++风格的“vtable”。  标志词  每个对象头部的第一个单词。通常是一组位域，包括同步状态和标识哈希码。也可以是一个指针(具有低位编码特征)来同步相关信息。在GC期间，可能包含GC状态位。  nmethod  实现一些Java字节码的可执行代码块。它可能是一个完整的Java方法，也可能是一个“OSR”方法。它通常包括编译器内联的其他方法的对象代码。  对象头  在每个gc管理的堆对象的开头使用通用结构。(每个oop都指向一个对象头。)包括堆对象的布局、类型、GC状态、同步状态和标识哈希码的基本信息。由两个字组成。在数组中，紧随其后的是一个长度字段。注意，Java对象和vm内部对象都有共同的对象头格式。  对象推广  将一个对象从一代复制到另一代的操作。  老的代  Java对象堆中的一个区域，其中保存着被引用一段时间的对象。  堆栈上替换  又称“OSR”。将解释的(或优化程度较低的)堆栈帧转换为编译的(或优化程度较高的)堆栈帧的过程。当解释器发现一个方法正在循环时，请求编译器生成一个特殊的n方法，该方法在循环的某个地方有一个入口点(特别是在向后分支)，并将控制权转移给该n方法。一个粗略的反优化。  oop  一个对象的指针。具体来说，是指向gc管理的堆的指针。(这个术语是传统的。一个“o”可以代表“ordinary”。)实现为本机机器地址，而不是句柄。Oops可以由编译或解释的Java代码直接操作，因为GC知道这些代码中的Oops的活跃度和位置。GC(见地图)。Oops也可以由短时间的C/ c++代码直接操作，但是必须由这些代码在每个安全点的句柄中保存。  并行类加载  让多个类/类型在同一时间被同一类装入器加载的能力。  并行垃圾收集  一种垃圾收集算法，使用多个控制线程在多处理器上更有效地执行。  永久的一代  地址空间中的一个区域，保存由虚拟机本身分配的对象，但由垃圾回收器管理。永久生成的名称是错误的，因为它中的几乎所有对象都可以被收集，尽管它们往往会被引用很长时间，所以它们很少会变成垃圾。  记得设置  一种记录代与代之间指针的数据结构。  safepoint  在程序执行期间，已知所有GC根且所有堆对象内容一致的点。从全局的角度来看，在GC可以运行之前，所有线程都必须阻塞在一个安全点。(作为一种特殊情况，运行JNI代码的线程可以继续运行，因为它们只使用句柄。在一个安全的地方，它们必须阻塞而不是装入把手中的内容。)从本地的角度来看，安全点是代码块中一个独特的点，在这里执行的线程可能会阻塞GC。大多数呼叫站点都是安全地点。强不变量在每个安全点都成立，在非安全点可以忽略。编译后的Java代码和C/ c++代码都在安全点之间进行了优化，但在安全点之间的优化就不那么明显了。JIT编译器在每个安全点发出一个GC映射。VM中的C/ c++代码使用程式化的基于宏的约定(例如，TRAPS)来标记潜在的安全点。  sea-of-nodes  C2中的高级中间表示。它是一种SSA形式，其中数据和控制流都用节点之间的显式边表示。它与更传统的编译器中使用的表单不同，节点没有绑定到控制流图中的块。IR允许节点在海洋中浮动(受边缘约束)，直到它们被安排在编译过程的后期。  可服务性代理(SA)  可服务性代理是Sun内部代码的集合，用于帮助调试热点问题。它也被一些JDK工具使用——jstack、jmap、jinfo和jdb。请参阅SA了解更多信息。  stackmap  指的是StackMapTable属性或表中的一个特定的StackMapFrame。  StackMapTable  类文件中Code属性的一个属性，它包含新验证者在验证期间使用的类型信息。它由一个StackMapFrames数组组成。它是由JDK6的javac自动生成的。  幸存者空间  Java对象堆中用于保存对象的区域。通常有一对幸存者空间，其中一个的收集是通过将一个幸存者空间中的引用对象复制到另一个幸存者空间来实现的。  同步  一般来说，这是对并行活动的协调，以确保这些活动的安全性和活性属性。例如，通过使用锁来保护到该数据的所有代码路径来保护对共享数据的访问。  TLAB  线程本地分配缓冲区。用于快速分配堆空间而不需要同步。编译后的代码有一些指令的“快速路径”，这些指令试图在当前线程的TLAB中碰撞一个高水位标记，如果碰撞标记落在特定的TLAB限制地址之前，则成功分配一个对象。  不常见的陷阱  当C2生成的代码返回到解释器以便进一步执行时。C2通常针对常见情况进行编译，允许它专注于优化频繁执行的路径。例如，当编译时未初始化的类需要运行时初始化时，C2会在生成的代码中插入一个不常见的陷阱。  验证器  虚拟机中执行字节码校验的软件代码。  虚拟机的操作  Java线程可以请求的VM中的操作，但这些操作必须由称为VM线程的特定线程以串行方式执行。这些操作通常是同步的，因为请求者将阻塞，直到VM线程完成操作。其中许多操作还要求在执行操作之前将VM带到一个安全点—垃圾收集请求就是一个简单的例子。  写障碍  在每个oop存储上执行的代码。例如，维护记忆集。  年轻的一代  Java对象堆中的一个区域，用于保存最近分配的对象。 |