Java内存模型

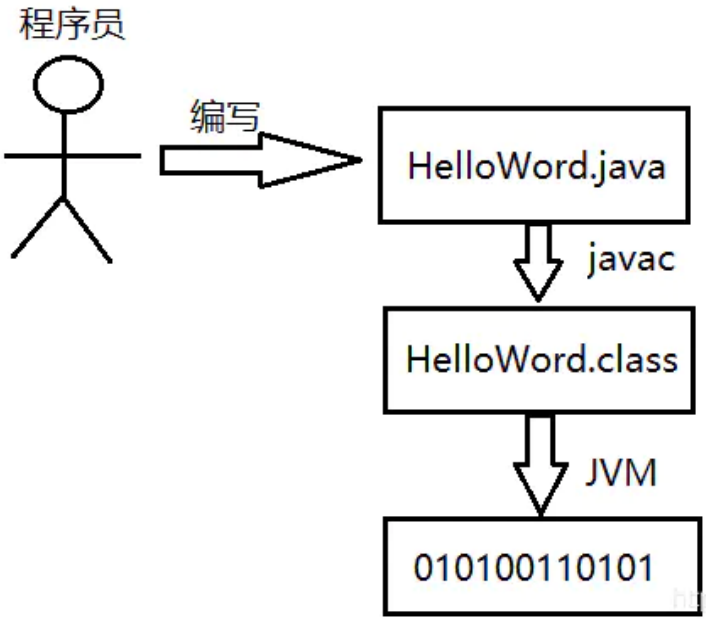
1. 我们开发人员编写的Java代码是怎么让电脑认识的

- 首先先了解电脑是二进制的系统，他只认识 01010101
- 比如我们经常要编写 HelloWord.java 电脑是怎么认识运行的
- HelloWord.java是我们程序员编写的，我们人可以认识，但是电脑不认识

Java文件编译的过程

1. 程序员编写的.java文件
2. 由javac编译成字节码文件.class：（为什么编译成class文件，因为JVM只认识.class文件）
3. 在由JVM编译成电脑认识的文件 （对于电脑系统来说 文件代表一切）

（这是一个大概的观念 抽象画的概念）



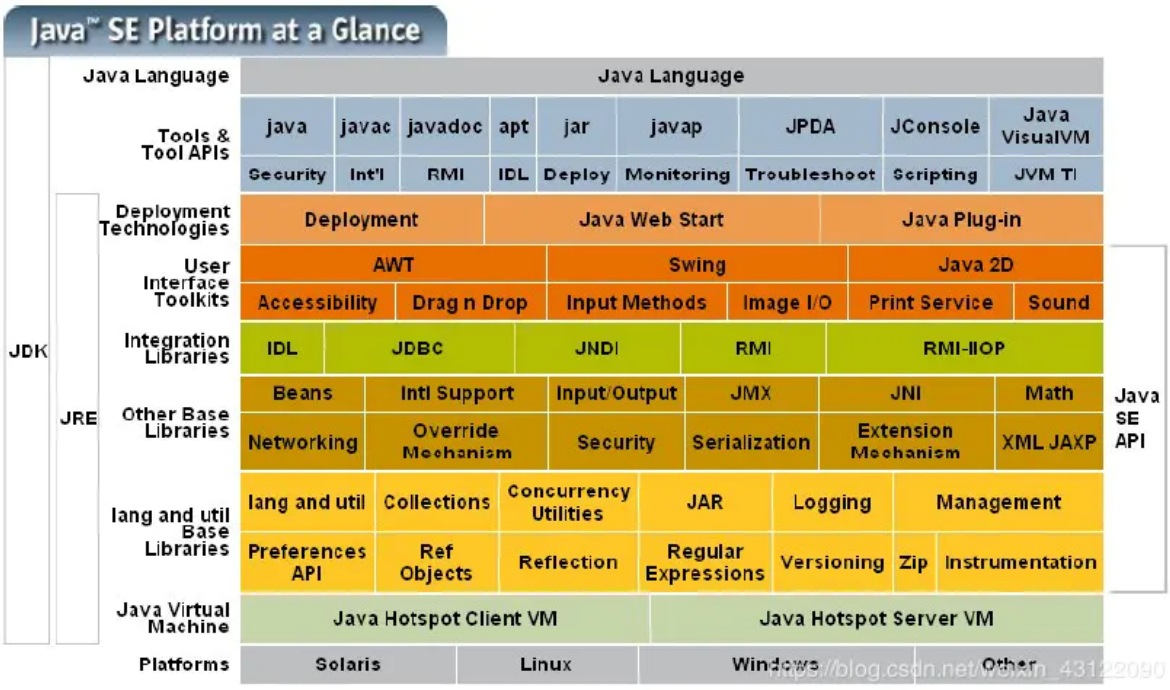
2. 为什么说java是跨平台语言

- 这个夸平台是中间语言（JVM）实现的夸平台
- Java有JVM从软件层面屏蔽了底层硬件、指令层面的细节让他兼容各种系统

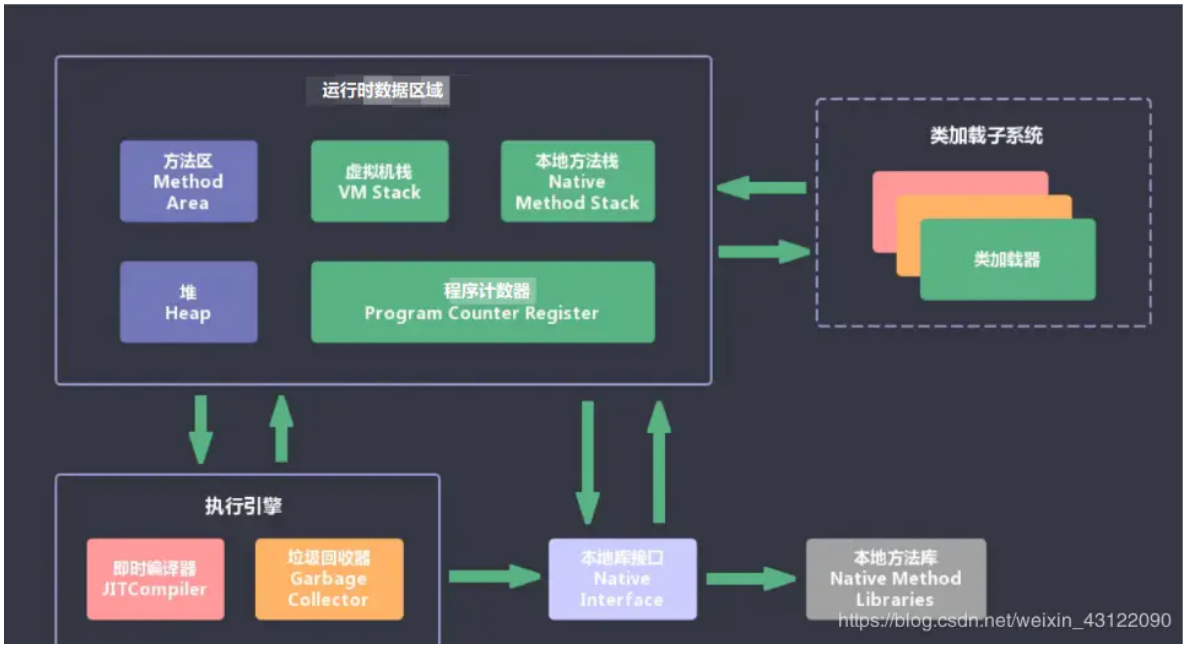
难道 C 和 C++ 不能夸平台吗 其实也可以 C和C++需要在编译器层面去兼容不同操作系统的不同层面，写过C和C++的就知道不同操作系统的有些代码是不一样

3. Jdk和Jre和JVM的区别

- Jdk包括了Jre和Jvm，Jre包括了Jvm
- Jdk是我们编写代码使用的开发工具包
- Jre 是Java的运行时环境，他大部分都是 C 和 C++ 语言编写的，他是我们在编译java时所需要的基础的类库
- Jvm俗称Java虚拟机，他是java运行环境的一部分，它虚构出来的一台计算机，在通过在实际的计算机上仿真模拟各种计算机功能来实现Java应用程序
- 看Java官方的图片，Jdk中包括了Jre，Jre中包括了JVM



4. 说一下 JVM由那些部分组成，运行流程是什么?

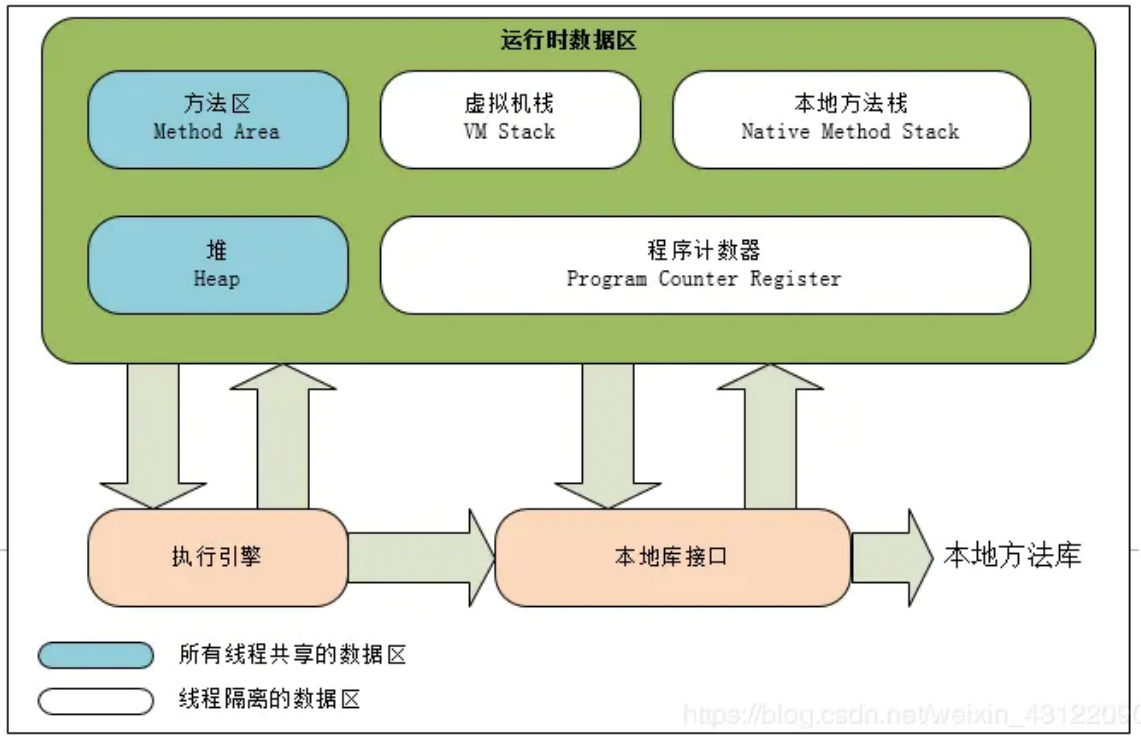


- JVM包含两个子系统和两个组件: 两个子系统为Class loader(类装载)、Execution engine(执行引擎)； 两个组件为Runtime data area(运行时数据区)、Native Interface(本地接口)。
  - Class loader(类装载)：根据给定的全限定名类名(如：java.lang.Object)来装载class文件到Runtime data area中的method area。
  - Execution engine（执行引擎）：执行classes中的指令。
  - Native Interface(本地接口)：与native libraries交互，是其它编程语言交互的接口。
  - Runtime data area(运行时数据区域)：这就是我们常说的JVM的内存。
- **流程**：首先通过编译器把 Java 代码转换成字节码，类加载器（ClassLoader）再把字节码加载到内存中，将其放在运行时数据区（Runtime data area）的方法区内，而字节码文件只是 JVM 的一套指令集规范，并不能直接交给底层操作系统去执行，因此需要特定的命令解析器执行引擎（Execution Engine），将字节码翻译成底层系统指令，再交由 CPU 去执行，而这个过程中需要调用其他语言的本地库接口（Native Interface）来实现整个程序的功能。

5. 说一下 JVM 运行时数据区

- Java 虚拟机在执行 Java 程序的过程中会把它所管理的内存区域划分为若干个不同的数据区域。这些区域都有各自的用途，以及创建和销毁的时间，有些区域随着虚拟机进程的启动而存在，有些区域则是依赖线程的启动和结束而建立和销毁。Java 虚拟机所管理的内存被划分为如下几个区域：

`简单的说就是我们java运行时的东西是放在那里的`



- 程序计数器（Program Counter Register）：当前线程所执行的字节码的行号指示器，字节码解析器的工作是通过改变这个计数器的值，来选取下一条需要执行的字节码指令，分支、循环、跳转、异常处理、线程恢复等基础功能，都需要依赖这个计数器来完成；

  为什么要线程计数器？因为线程是不具备记忆功能

- Java 虚拟机栈（Java Virtual Machine Stacks）：每个方法在执行的同时都会在Java 虚拟机栈中创建一个栈帧（Stack Frame）用于存储局部变量表、操作数栈、动态链接、方法出口等信息；

  栈帧就是Java虚拟机栈中的下一个单位

- 本地方法栈（Native Method Stack）：与虚拟机栈的作用是一样的，只不过虚拟机栈是服务 Java 方法的，而本地方法栈是为虚拟机调用 Native 方法服务的；

  Native 关键字修饰的方法是看不到的，Native 方法的源码大部分都是 C和C++ 的代码

- Java 堆（Java Heap）：Java 虚拟机中内存最大的一块，是被所有线程共享的，几乎所有的对象实例都在这里分配内存；
- 方法区（Methed Area）：用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译后的代码等数据。

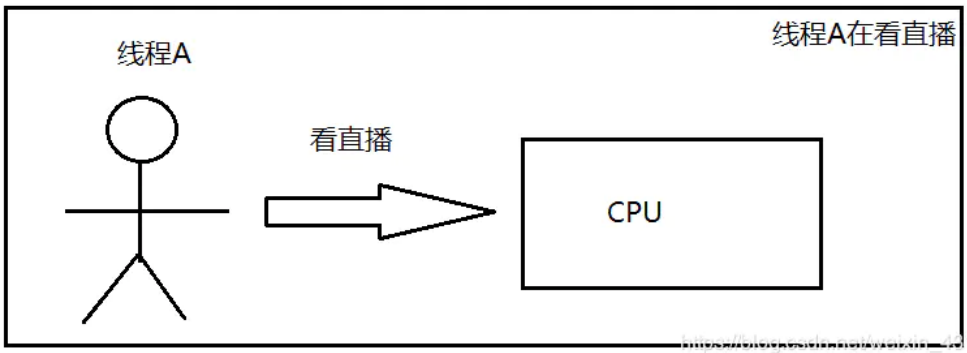
后面有详细的说明JVM 运行时数据区

6. 详细的介绍下程序计数器？（重点理解）

1. 程序计数器是一块较小的内存空间，它可以看作是：保存当前线程所正在执行的字节码指令的地址(行号)
2. 由于Java虚拟机的多线程是通过线程轮流切换并分配处理器执行时间的方式来实现的，一个处理器都只会执行一条线程中的指令。因此，为了线程切换后能恢复到正确的执行位置，每条线程都有一个独立的程序计数器，各个线程之间计数器互不影响，独立存储。称之为“线程私有”的内存。程序计数器内存区域是虚拟机中唯一没有规定OutOfMemoryError情况的区域。

总结：也可以把它叫做线程计数器

- **例子**：在java中最小的执行单位是线程，线程是要执行指令的，执行的指令最终操作的就是我们的电脑，就是 CPU。在CPU上面去运行，有个非常不稳定的因素，叫做调度策略，这个调度策略是时基于时间片的，也就是当前的这一纳秒是分配给那个指令的。
- **假如**：
  - 线程A在看直播



    - 突然，线程B来了一个视频电话，就会抢夺线程A的时间片，就会打断了线程A，线程A就会挂起

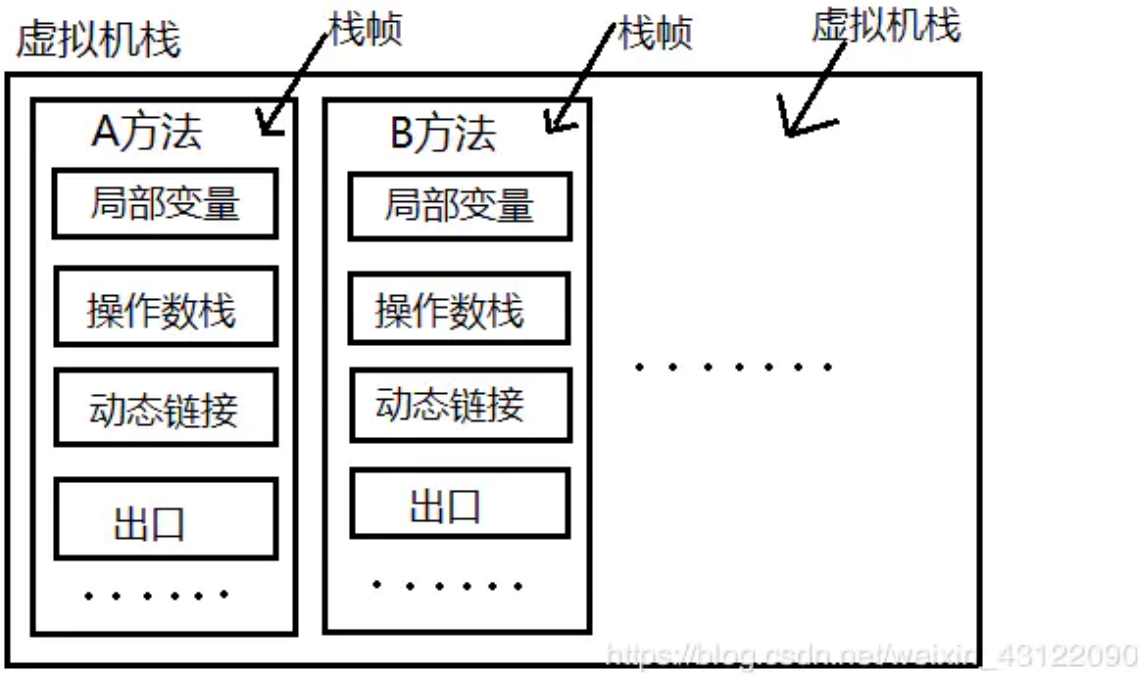
```
![在这里插入图片描述](https://user-gold-cdn.xitu.io/2020/4/13/171729fcc70da181?imageView2/0/w/1280/h/960/format/webp/ignore-error/1)

*    然后，视频电话结束，这时线程A究竟该干什么？ （线程是最小的执行单位，他不具备记忆功能，他只负责去干，那这个记忆就由：**程序计数器来记录**）
![在这里插入图片描述](https://user-gold-cdn.xitu.io/2020/4/13/171729fcc90c8a88?imageView2/0/w/1280/h/960/format/webp/ignore-error/1)
```

7. 详细介绍下Java虚拟机栈？（重点理解）

1. Java虚拟机是线程私有的，它的生命周期和线程相同。
2. 虚拟机栈描述的是Java方法执行的内存模型：`每个方法在执行的同时`都会创建一个栈帧（Stack Frame）用于存储局部变量表、操作数栈、动态链接、方法出口等信息。

- **解释**：虚拟机栈中是有单位的，单位就是**栈帧**，一个方法一个**栈帧**。一个**栈帧**中他又要存储，局部变量，操作数栈，动态链接，出口等。



解析栈帧：

1. 局部变量表：是用来存储我们临时8个基本数据类型、对象引用地址、returnAddress类型。（returnAddress中保存的是return后要执行的字节码的指令地址。）
2. 操作数栈：操作数栈就是用来操作的，例如代码中有个 i = 6*6，他在一开始的时候就会进行操作，读取我们的代码，进行计算后再放入局部变量表中去
3. 动态链接：假如我方法中，有个 service.add()方法，要链接到别的方法中去，这就是动态链接，存储链接的地方。
4. 出口：出口是什呢，出口正常的话就是return 不正常的话就是抛出异常落

一个方法调用另一个方法，会创建很多栈帧吗？

- 答：会创建。如果一个栈中有动态链接调用别的方法，就会去创建新的栈帧，栈中是由顺序的，一个栈帧调用另一个栈帧，另一个栈帧就会排在调用者下面

栈指向堆是什么意思？

- 栈指向堆是什么意思，就是栈中要使用成员变量怎么办，栈中不会存储成员变量，只会存储一个应用地址

递归的调用自己会创建很多栈帧吗？

- 答：递归的话也会创建多个栈帧，就是在栈中一直从上往下排下去

8. 你能给我详细的介绍Java堆吗？（重点理解）

- java堆（Java Heap）是java虚拟机所管理的内存中最大的一块，是被所有线程共享的一块内存区域，在虚拟机启动时创建。此内存区域的唯一目的就是存放对象实例。
- 在Java虚拟机规范中的描述是：所有的对象实例以及数组都要在堆上分配。
- java堆是垃圾收集器管理的主要区域，因此也被成为“GC堆”。
- 从内存回收角度来看java堆可分为：新生代和老生代。
- 从内存分配的角度看，线程共享的Java堆中可能划分出多个线程私有的分配缓冲区。
- 无论怎么划分，都与存放内容无关，无论哪个区域，存储的都是对象实例，进一步的划分都是为了更好的回收内存，或者更快的分配内存。

- 根据Java虚拟机规范的规定，java堆可以处于物理上不连续的内存空间中。当前主流的虚拟机都是可扩展的（通过 -Xmx 和 -Xms 控制）。如果堆中没有内存可以完成实例分配，并且堆也无法再扩展时，将会抛出OutOfMemoryError异常。

9. 能不能解释一下本地方法栈?

1. 本地方法栈很好理解，他很栈很像，只不过方法上带了 native 关键字的栈字
2. 它是虚拟机栈为虚拟机执行Java方法（也就是字节码）的服务方法
3. native关键字的方法是看不到的，必须要去oracle官网去下载才可以看的到，而且native关键字修饰的大部分源码都是C和C++的代码。
4. 同理可得，本地方法栈中就是C和C++的代码

10. 能不能解释一下方法区（重点理解）

1. 方法区是所有线程共享的内存区域，它用于存储已被Java虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。
2. 它有个别命叫Non-Heap（非堆）。当方法区无法满足内存分配需求时，抛出OutOfMemoryError异常。

11. 什么是JVM字节码执行引擎

- 虚拟机核心的组件就是执行引擎，它负责执行虚拟机的字节码，一般户先进行编译成机器码后执行。
- “虚拟机”是一个相对于“物理机”的概念，虚拟机的字节码是不能直接在物理机上运行的，需要JVM字节码执行引擎- 编译成机器码后才可在物理机上执行。

12. 你听过直接内存吗?

- 直接内存（Direct Memory）并不是虚拟机运行时数据区的一部分，也不是Java虚拟机中定义的内存区域。但是这部分内存也被频繁地使用，而且也可能导致 OutOfMemoryError 异常出现，所以我们放到这里一起讲解。
- 我的理解就是直接内存是基于物理内存和Java虚拟机内存的中间内存

13. 知道垃圾收集系统吗?

- 程序在运行过程中，会产生大量的内存垃圾（一些没有引用指向的内存对象都属于内存垃圾，因为这些对象已经无法访问，程序用不了它们了，对程序而言它们已经死亡），为了确保程序运行时的性能，java虚拟机在程序运行的过程中不断地进行自动的垃圾回收（GC）。
- 垃圾收集系统是Java的核心，也是不可少的，Java有一套自己进行垃圾清理的机制，开发人员无需手工清理
- 有一部分原因就是因为Java垃圾回收系统的强大导致Java领先市场

14. 堆栈的区别是什么?

对比	JVM堆	JVM栈
物理地址	堆的物理地址分配对对象是不连续的。因此性能慢些。在GC的时候也要考虑到不连续的分配，所以有各种算法。比如，标记-消除，复制，标记-压缩，分代（即新生代使用复制算法，老年代使用标记——压缩）	栈使用的是数据结构中的栈，先进后出的原则，物理地址分配是连续的。所以性能快。
内存分别	堆因为是不连续的，所以分配的内存是在运行期确认的，因此大小不固定。一般堆大小远远大于栈。	栈是连续的，所以分配的内存大小要在编译期就确认，大小是固定的。
存放的内容	堆存放的是对象的实例和数组。因此该区更关注的是数据的存储	栈存放：局部变量，操作数栈，返回结果。该区更关注的是程序方法的执行。
程序的可见度	堆对于整个应用程序都是共享、可见的。	栈只对于线程是可见的。所以也是线程私有。他的生命周期和线程相同。

- 注意：
  - 静态变量放在方法区
  - 静态的对象还是放在堆。

15. 深拷贝和浅拷贝

- 浅拷贝（shallowCopy）只是增加了一个指针指向已存在的内存地址，
- 深拷贝（deepCopy）是增加了一个指针并且申请了一个新的内存，使这个增加的指针指向这个新的内存，
- 浅复制：仅仅是指向被复制的内存地址，如果原地址发生改变，那么浅复制出来的对象也会相应的改变。
- 深复制：在计算机中开辟一块**新的内存地址**用于存放复制的对象。

16. Java会存在内存泄漏吗？请说明为什么？

- 内存泄漏是指不再被使用的对象或者变量一直被占据在内存中。理论上来说，Java是有GC垃圾回收机制的，也就是说，不再被使用的对象，会被GC自动回收掉，自动从内存中清除。
- 但是，即使这样，Java也还是存在着内存泄漏的情况，java导致内存泄露的原因很明确：长生命周期的对象持有短生命周期对象的引用就很可能发生内存泄露，`尽管短生命周期对象已经不再需要，但是因为长生命周期对象持有它的引用而导致不能被回收`，这就是java中内存泄露的发生场景。

垃圾回收机制及算法

17. 简述Java垃圾回收机制

- 在java中，程序员是不需要显示的去释放一个对象的内存的，而是由虚拟机自行执行。在JVM中，有一个垃圾回收线程，它是低优先级的，在正常情况下是不会执行的，只有在虚拟机空闲或者当前堆内存不足时，才会触发执行，扫面那些没有被任何引用的对象，并将它们添加到要回收的集合中，进行回收。

18. GC是什么？为什么要GC

- GC 是垃圾收集的意思（Gabage Collection）,内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java 提供的 GC 功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java 语言没有提供释放已分配内存的显示操作方法。

19. 垃圾回收的优点和缺点

- 优点：JVM的垃圾回收器都不需要我们手动处理无引用的对象了，这个就是最大的优点
- 缺点：程序员不能实时的对某个对象或所有对象调用垃圾回收器进行垃圾回收。

20. 垃圾回收器的原理是什么？有什么办法手动进行垃圾回收?

- 对于GC来说，当程序员创建对象时，GC就开始监控这个对象的地址、大小以及使用情况。
- 通常，GC采用有向图的方式记录和管理堆(heap)中的所有对象。通过这种方式确定哪些对象是"可达的"，哪些对象是"不可达的"。当GC确定一些对象为"不可达"时，GC就有责任回收这些内存空间。
- 可以。程序员可以手动执行System.gc()，通知GC运行，但是Java语言规范并不保证GC一定会执行。

21. JVM 中都有哪些引用类型?

- 强引用：发生 gc 的时候不会被回收。
- 软引用：有用但不是必须的对象，在发生内存溢出之前会被回收。
- 弱引用：有用但不是必须的对象，在下一次GC时会被回收。
- 虚引用（幽灵引用/幻影引用）：无法通过虚引用获得对象，用 PhantomReference 实现虚引用，虚引用的用途是在 gc 时返回一个通知。

22. 怎么判断对象是否可以被回收?

- 垃圾收集器在做垃圾回收的时候，首先需要判定的就是哪些内存是需要被回收的，哪些对象是存活的，是不可以被回收的；哪些对象已经死掉了，需要被回收。
- 一般有两种方法来判断：
    - 引用计数器法：为每个对象创建一个引用计数，有对象引用时计数器 +1，引用被释放时计数 -1，当计数器为 0 时就可以被回收。它有一个缺点不能解决循环引用的问题；（这个已经淘汰了）

- 可达性分析算法：从 GC Roots 开始向下搜索，搜索所走过的路径称为引用链。当一个对象到 GC Roots 没有任何引用链相连时，则证明此对象是可以被回收的。（市场上用的非常非常广泛）

23. Full GC是什么

- 清理整个堆空间—包括年轻代和老年代和永久代
- 因为Full GC是清理整个堆空间所以Full GC执行速度非常慢，在Java开发中最好保证少触发Full GC

24. 对象什么时候可以被垃圾器回收

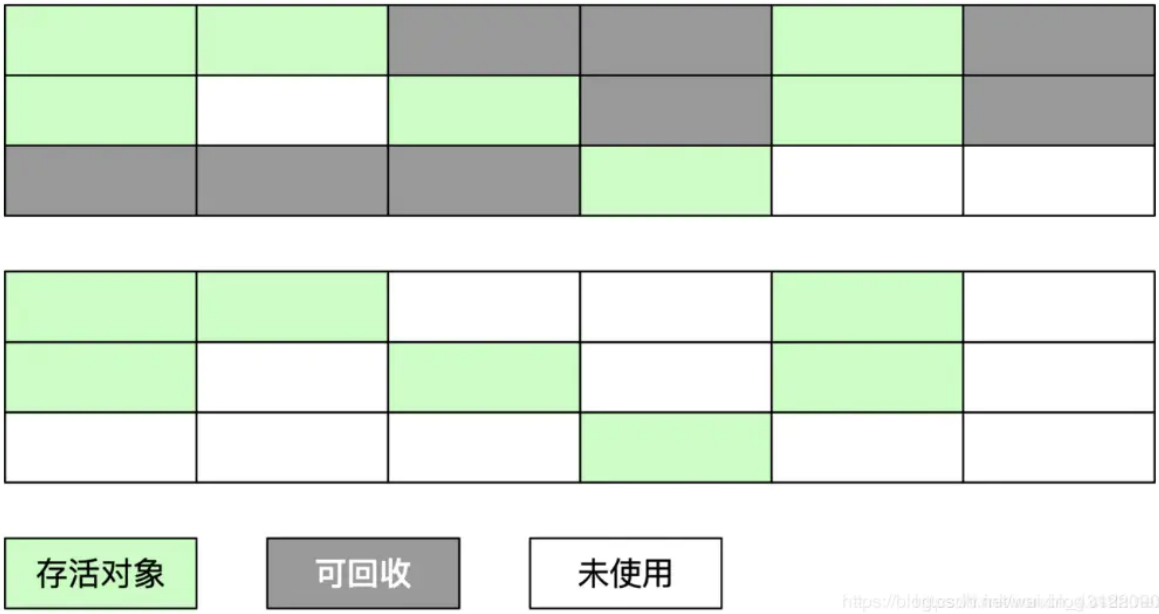
- 当对象对当前使用这个对象的应用程序变得不可触及的时候，这个对象就可以被回收了。
- 垃圾回收不会发生在永久代，如果永久代满了或者是超过了临界值，会触发完全垃圾回收(Full GC)。如果你仔细查看垃圾收集器的输出信息，就会发现永久代也是被回收的。这就是为什么正确的永久代大小对避免Full GC是非常重要的原因。

25. JVM 垃圾回收算法有哪些?

- 标记-清除算法：标记无用对象，然后进行清除回收。缺点：效率不高，无法清除垃圾碎片。
- 复制算法：按照容量划分二个大小相等的内存区域，当一块用完的时候将活着的对象复制到另一块上，然后再把已使用的内存空间一次清理掉。缺点：内存使用率不高，只有原来的一半。
- 标记-整理算法：标记无用对象，让所有存活的对象都向一端移动，然后直接清除掉端边界以外的内存。
- 分代算法：根据对象存活周期的不同将内存划分为几块，一般是新生代和老年代，新生代基本采用复制算法，老年代采用标记整理算法。

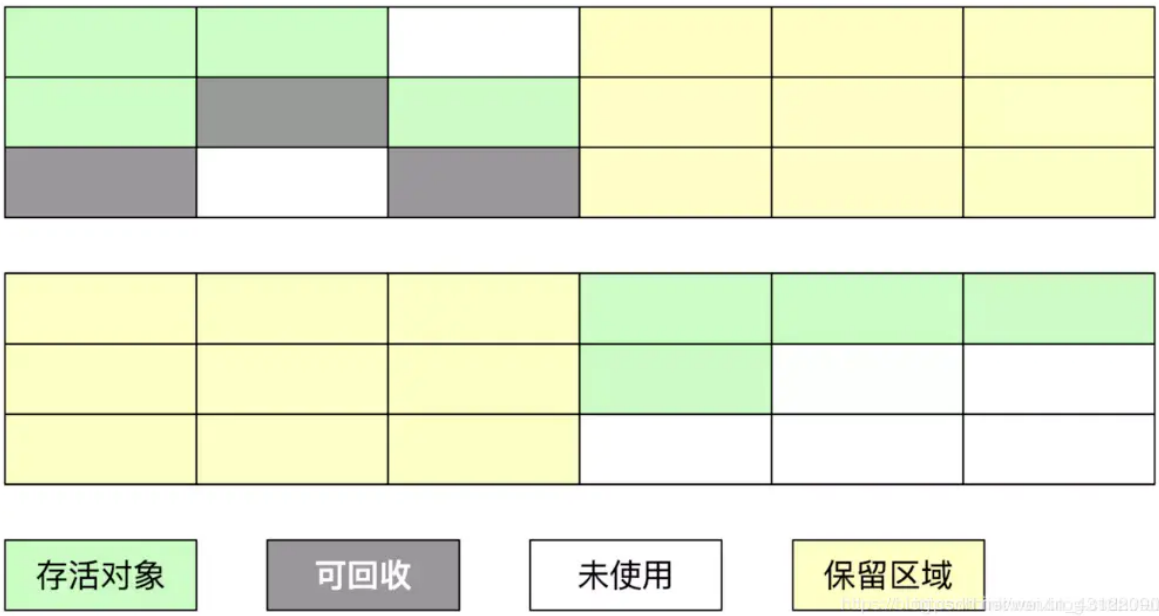
标记-清除算法

- 标记无用对象，然后进行清除回收。
- 标记-清除算法（Mark-Sweep）是一种常见的基础垃圾收集算法，它将垃圾收集分为两个阶段：
  - 标记阶段：标记出可以回收的对象。
  - 清除阶段：回收被标记的对象所占用的空间。
- 标记-清除算法之所以是基础的，是因为后面讲到的垃圾收集算法都是在此算法的基础上进行改进的。
- **优点**：实现简单，不需要对象进行移动。
- **缺点**：标记、清除过程效率低，产生大量不连续的内存碎片，提高了垃圾回收的频率。
- 标记-清除算法的执行的过程如下图所示



复制算法

- 为了解决标记-清除算法的效率不高的问题，产生了复制算法。它把内存空间划为两个相等的区域，每次只使用其中一个区域。垃圾收集时，遍历当前使用的区域，把存活对象复制到另外一个区域中，最后将当前使用的区域的可回收的对象进行回收。
- **优点**：按顺序分配内存即可，实现简单、运行高效，不用考虑内存碎片。
- **缺点**：可用的内存大小缩小为原来的一半，对象存活率高时会频繁进行复制。
- 复制算法的执行过程如下图所示

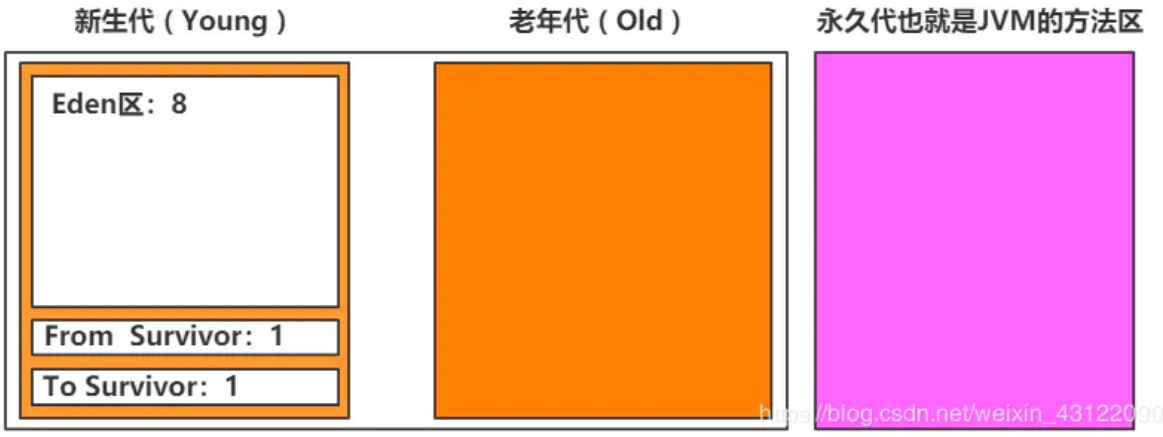


标记-整理算法

- 在新生代中可以使用复制算法，但是在老年代就不能选择复制算法了，因为老年代的对象存活率会较高，这样会有较多的复制操作，导致效率变低。标记-清除算法可以应用在老年代中，但是它效率不高，在内存回收后容易产生大量内存碎片。因此就出现了一种标记-整理算法（Mark-Compact）算法，与标记-整理算法不同的是，在标记可回收的对象后将所有存活的对象压缩到内存的一端，使他们紧凑的排列在一起，然后对端边界以外的内存进行回收。回收后，已用和未用的内存都各自一边。
- **优点**：解决了标记-清理算法存在的内存碎片问题。
- **缺点**：仍需要进行局部对象移动，一定程度上降低了效率。
- 标记-整理算法的执行过程如下图所示

分代收集算法

- 当前商业虚拟机都采用 `分代收集` 的垃圾收集算法。分代收集算法，顾名思义是根据对象的 `存活周期` 将内存划分为几块。一般包括 `年轻代`、`老年代` 和 `永久代`，如图所示：`（后面有重点讲解）`

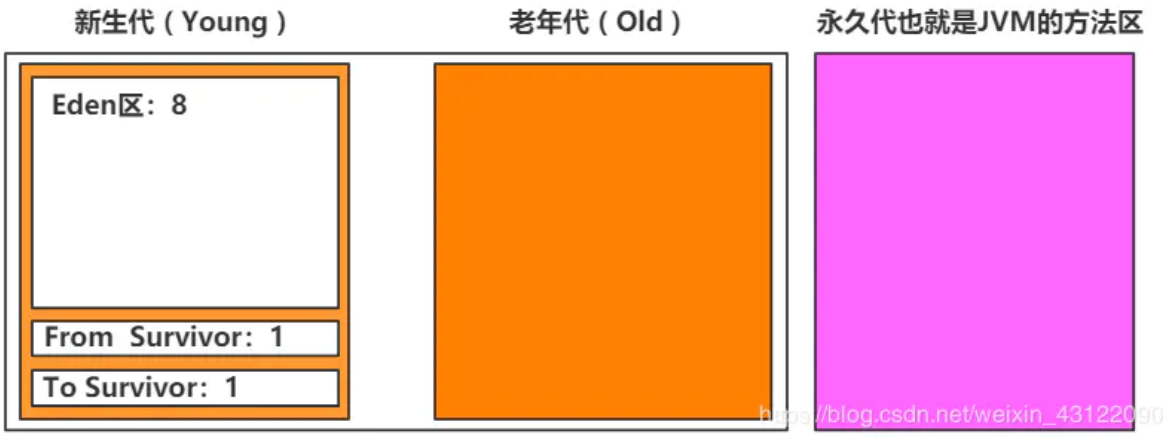


26. JVM中的永久代中会发生垃圾回收吗

- 垃圾回收不会发生在永久代，如果永久代满了或者是超过了临界值，会触发完全垃圾回收(Full GC)。如果你仔细查看垃圾收集器的输出信息，就会发现永久代也是被回收的。这就是为什么正确的永久代大小对避免Full GC是非常重要的原因。请参考下Java8：从永久代到元数据区
(注：Java8中已经移除了永久代，新加了一个叫做元数据区的native内存区)

垃圾收集器以及新生代、老年代、永久代

27. 讲一下新生代、老年代、永久代的区别



- 在 Java 中，堆被划分成两个不同的区域：新生代 (Young)、老年代 (Old)。而新生代 (Young) 又被划分为三个区域：Eden、From Survivor、To Survivor。这样划分的目的是为了使 JVM 能够更好的管理堆内存中的对象，包括内存的分配以及回收。
- 新生代中一般保存新出现的对象，所以每次垃圾收集时都发现大批对象死去，只有少量对象存活，便采用了`复制算法`，只需要付出少量存活对象的复制成本就可以完成收集。
- 老年代中一般保存存活了很久的对象，他们存活率高、没有额外空间对它进行分配担保，就必须采用`“标记-清理”或者“标记-整理”`算法。
- 永久代就是JVM的方法区。在这里都是放着一些被虚拟机加载的类信息，静态变量，常量等数据。这个区中的东西比老年代和新生代更不容易回收。

28. Minor GC、Major GC、Full GC是什么

1. Minor GC是新生代GC，指的是发生在新生代的垃圾收集动作。由于java对象大都是朝生夕死的，所以Minor GC非常频繁，一般回收速度也比较快。（一般采用复制算法回收垃圾）
2. Major GC是老年代GC，指的是发生在老年代的GC，通常执行Major GC会连着Minor GC一起执行。Major GC的速度要比Minor GC慢的多。（可采用标记清楚法和标记整理法）
3. Full GC是清理整个堆空间，包括年轻代和老年代

29. Minor GC、Major GC、Full GC区别及触发条件

- **Minor GC 触发条件一般为：**
    1. eden区满时，触发MinorGC。即申请一个对象时，发现eden区不够用，则触发一次MinorGC。
    2. 新创建的对象大小 > Eden所剩空间时触发Minor GC
- **Major GC和Full GC 触发条件一般为：** `Major GC通常是跟full GC是等价的`
    1. 每次晋升到老年代的对象平均大小>老年代剩余空间
    2. MinorGC后存活的对象超过了老年代剩余空间
    3. 永久代空间不足
    4. 执行System.gc()
    5. CMS GC异常
    6. 堆内存分配很大的对象

30. 为什么新生代要分Eden和两个 Survivor 区域?

- 如果没有Survivor，Eden区每进行一次Minor GC，存活的对象就会被送到老年代。老年代很快被填满，触发Major GC.老年代的内存空间远大于新生代，进行一次Full GC消耗的时间比Minor GC

长得多,所以需要分为Eden和Survivor。
- Survivor的存在意义，就是减少被送到老年代的对象，进而减少Full GC的发生，Survivor的预筛选保证，只有经历15次Minor GC还能在新生代中存活的对象，才会被送到老年代。
- 设置两个Survivor区最大的好处就是解决了碎片化，刚刚新建的对象在Eden中，经历一次Minor GC，Eden中的存活对象就会被移动到第一块survivor space S0，Eden被清空；等Eden区再满了，就再触发一次Minor GC，Eden和S0中的存活对象又会被复制送入第二块survivor space S1（这个过程非常重要，因为这种复制算法保证了S1中来自S0和Eden两部分的存活对象占用连续的内存空间，避免了碎片化的发生）

31. Java堆老年代(Old) 和新生代 (Young) 的默认比例?

- 默认的，新生代 (Young) 与老年代 (Old) 的比例的值为 1:2 (该值可以通过参数 –XX:NewRatio 来指定)，即：新生代 (Young) = 1/3 的堆空间大小。老年代 (Old) = 2/3 的堆空间大小。
- 其中，新生代 (Young) 被细分为 Eden 和 **两个 Survivor 区域**，Edem 和俩个Survivor 区域比例是 = 8 : 1 : 1 (可以通过参数 –XX:SurvivorRatio 来设定)，
- 但是JVM 每次只会使用 Eden 和其中的一块 Survivor 区域来为对象服务，所以无论什么时候，总是有一块 Survivor 区域是空闲着的。

32.为什么要这样分代：

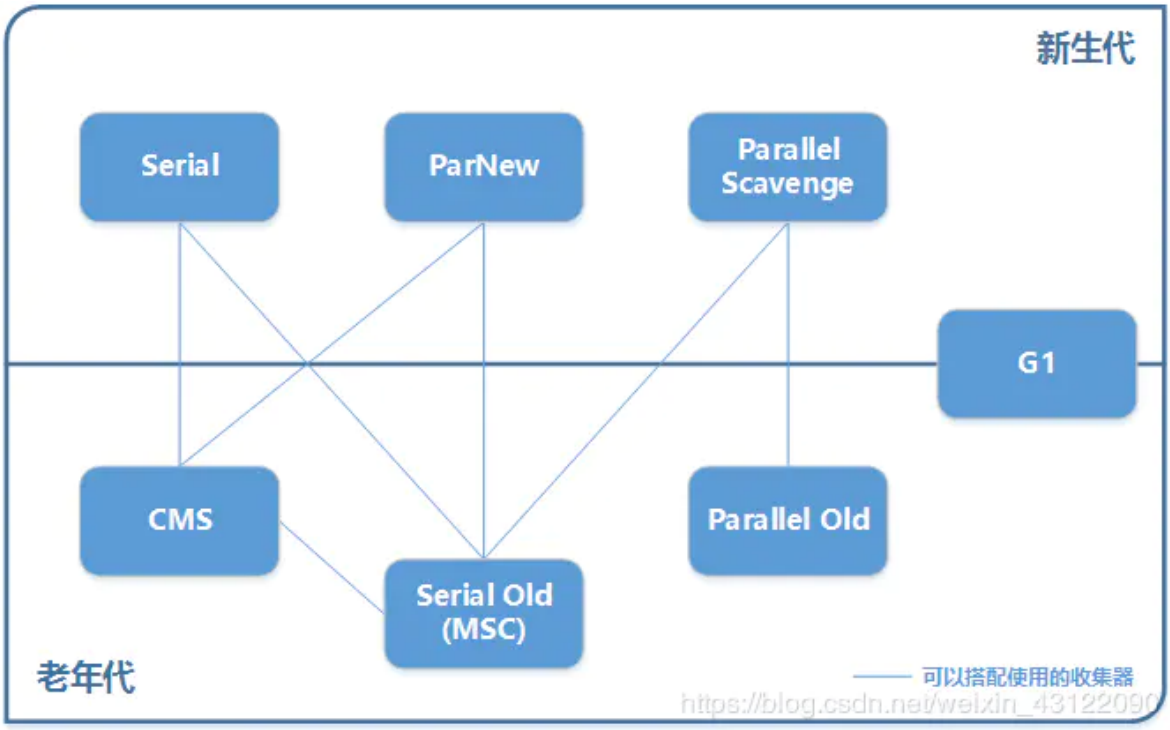
- 其实主要原因就是可以根据各个年代的特点进行对象分区存储，更便于回收，采用最适当的收集算法：
    - 新生代中，每次垃圾收集时都发现大批对象死去，只有少量对象存活，便采用了复制算法，只需要付出少量存活对象的复制成本就可以完成收集。
    - 而老年代中因为对象存活率高、没有额外空间对它进行分配担保，就必须采用“标记-清理”或者“标记-整理”算法。
- 新生代又分为Eden和Survivor （From与To，这里简称一个区）两个区。加上老年代就这三个区。数据会首先分配到Eden区当中（当然也有特殊情况，如果是大对象那么会直接放入到老年代（大对象是指需要大量连续内存空间的java对象）。当Eden没有足够空间的时候就会触发jvm发起一次Minor GC，。如果对象经过一次Minor-GC还存活，并且又能被Survivor空间接受，那么将被移动到Survivor空间当中。并将其年龄设为1，对象在Survivor每熬过一次Minor GC，年龄就加1，当年龄达到一定的程度（默认为15）时，就会被晋升到老年代中了，当然晋升老年代的年龄是可以设置的。

33. 什么是垃圾回收器他和垃圾算法有什么区别

- 垃圾收集器是垃圾回收算法（标记清楚法、标记整理法、复制算法、分代算法）的具体实现，不同垃圾收集器、不同版本的JVM所提供的垃圾收集器可能会有很在差别。

34. 说一下 JVM 有哪些垃圾回收器?

- 如果说垃圾收集算法是内存回收的方法论，那么垃圾收集器就是内存回收的具体实现。下图展示了7种作用于不同分代的收集器，其中用于回收新生代的收集器包括Serial、PraNew、Parallel Scavenge，回收老年代的收集器包括Serial Old、Parallel Old、CMS，还有用于回收整个Java堆的G1收集器。不同收集器之间的连线表示它们可以搭配使用。



垃圾回收器	工作区域	回收算法	工作线程	用户线程并行	描述
Serial	新生带	复制算法	单线程	否	Client模式下默认新生代收集器。简单高效
ParNew	新生带	复制算法	多线程	否	Serial的多线程版本，Server模式下首选， 可搭配CMS的新生代收集器
Parallel Scavenge	新生带	复制算法	多线程	否	目标是达到可控制的吞吐量
Serial Old	老年带	标记-整理	单线程	否	Serial老年代版本，给Client模式下的虚拟机使用
Parallel Old	老年带	标记-整理	多线程	否	Parallel Scavenge老年代版本，吞吐量优先
G1	新生带 + 老年带	标记-整理 + 复制算法	多线程	是	JDK1.9默认垃圾收集器

- Serial收集器（复制算法): 新生代单线程收集器，标记和清理都是单线程，优点是简单高效；
- ParNew收集器 (复制算法): 新生代收并行集器，实际上是Serial收集器的多线程版本，在多核CPU环境下有着比Serial更好的表现；
- Parallel Scavenge收集器 (复制算法): 新生代并行收集器，追求高吞吐量，高效利用 CPU。吞吐量 = 用户线程时间/(用户线程时间+GC线程时间)，高吞吐量可以高效率的利用CPU时间，尽快完成程

序的运算任务，适合后台应用等对交互相应要求不高的场景；
- Serial Old收集器 (标记-整理算法): 老年代单线程收集器，Serial收集器的老年代版本；
- Parallel Old收集器 (标记-整理算法)： 老年代并行收集器，吞吐量优先，Parallel Scavenge收集器的老年代版本；
- CMS(Concurrent Mark Sweep)收集器（标记-清除算法）： 老年代并行收集器，以获取最短回收停顿时间为目标的收集器，具有高并发、低停顿的特点，追求最短GC回收停顿时间。
- G1(Garbage First)收集器 (`标记整理 + 复制算法来回收垃圾`)： Java堆并行收集器，G1收集器是JDK1.7提供的一个新收集器，G1收集器基于“标记-整理”算法实现，也就是说不会产生内存碎片。此外，G1收集器不同于之前的收集器的一个重要特点是：G1回收的范围是整个Java堆(包括新生代，老年代)，而前六种收集器回收的范围仅限于新生代或老年代。

35. 收集器可以这么分配？（了解就好了）

```
Serial / Serial Old
Serial / CMS
ParNew / Serial Old
ParNew / CMS
Parallel Scavenge / Serial Old
Parallel Scavenge / Parallel Old
G1
```

36. 新生代垃圾回收器和老年代垃圾回收器都有哪些？有什么区别？

- 新生代回收器：Serial、ParNew、Parallel Scavenge
- 老年代回收器：Serial Old、Parallel Old、CMS
- 整堆回收器：G1

新生代垃圾回收器一般采用的是复制算法，复制算法的优点是效率高，缺点是内存利用率低；老年代回收器一般采用的是标记-整理的算法进行垃圾回收。

37. 简述分代垃圾回收器是怎么工作的？

- 分代回收器有两个分区：老生代和新生代，新生代默认的空间占比总空间的 1/3，老生代的默认占比是 2/3。
- 新生代使用的是复制算法，新生代里有 3 个分区：Eden、To Survivor、From Survivor，它们的默认占比是 8:1:1，它的执行流程如下：
  - 把 Eden + From Survivor 存活的对象放入 To Survivor 区；
  - 清空 Eden 和 From Survivor 分区；
  - From Survivor 和 To Survivor 分区交换，From Survivor 变 To Survivor，To Survivor 变 From Survivor。
- 每次在 From Survivor 到 To Survivor 移动时都存活的对象，年龄就 +1，当年龄到达 15（默认配置是 15）时，升级为老生代。大对象也会直接进入老生代。
- 老生代当空间占用到达某个值之后就会触发全局垃圾收回，一般使用标记整理的执行算法。以上这些循环往复就构成了整个分代垃圾回收的整体执行流程。

内存分配策略

38. 简述java内存分配与回收策率以及Minor GC和Major GC

- 所谓自动内存管理，最终要解决的也就是内存分配和内存回收两个问题。前面我们介绍了内存回收，这里我们再来聊聊内存分配。
- 对象的内存分配通常是在 Java 堆上分配（随着虚拟机优化技术的诞生，某些场景下也会在栈上分配，后面会详细介绍），对象主要分配在新生代的 Eden 区，如果启动了本地线程缓冲，将按照线程优先在 TLAB 上分配。少数情况下也会直接在老年代上分配。总的来说分配规则不是百分百固定的，其细节取决于哪一种垃圾收集器组合以及虚拟机相关参数有关，但是虚拟机对于内存的分配还是会遵循以下几种「普世」规则：

39. 对象优先在 Eden 区分配

- 多数情况，对象都在新生代 Eden 区分配。当 Eden 区分配没有足够的空间进行分配时，虚拟机将会发起一次 Minor GC。如果本次 GC 后还是没有足够的空间，则将启用分配担保机制在老年代中分配内存。
    - 这里我们提到 Minor GC，如果你仔细观察过 GC 日常，通常我们还能从日志中发现 Major GC/Full GC。
    - **Minor GC** 是指发生在新生代的 GC，因为 Java 对象大多都是朝生夕死，所有 Minor GC 非常频繁，一般回收速度也非常快；
    - **Major GC/Full GC** 是指发生在老年代的 GC，出现了 Major GC 通常会伴随至少一次 Minor GC。Major GC 的速度通常会比 Minor GC 慢 10 倍以上。

40. 为什么大对象直接进入老年代

- 所谓大对象是指需要大量连续内存空间的对象，频繁出现大对象是致命的，会导致在内存还有不少空间的情况下提前触发 GC 以获取足够的连续空间来安置新对象。
- 前面我们介绍过新生代使用的是标记-清除算法来处理垃圾回收的，如果大对象直接在新生代分配就会导致 Eden 区和两个 Survivor 区之间发生大量的内存复制。因此对于大对象都会直接在老年

代进行分配。

41. 长期存活对象将进入老年代

- 虚拟机采用分代收集的思想来管理内存，那么内存回收时就必须判断哪些对象应该放在新生代，哪些对象应该放在老年代。因此虚拟机给每个对象定义了一个对象年龄的计数器，如果对象在 Eden 区出生，并且能够被 Survivor 容纳，将被移动到 Survivor 空间中，这时设置对象年龄为 1。对象在 Survivor 区中每「熬过」一次 Minor GC 年龄就加 1，当年龄达到一定程度（默认 15） 就会被晋升到老年代。

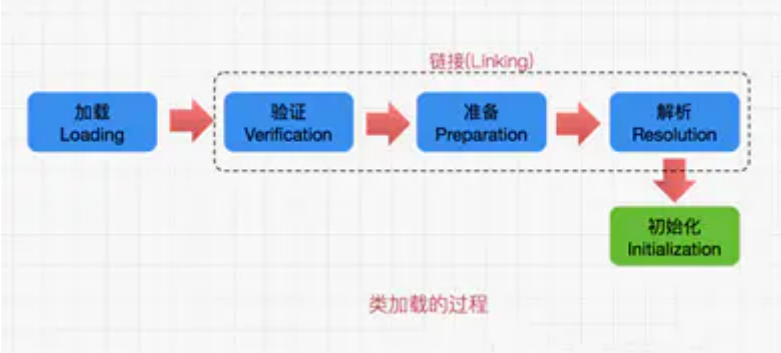
虚拟机类加载机制

42. 简述java类加载机制?

- 虚拟机把描述类的数据从Class文件加载到内存，并对数据进行校验，解析和初始化，最终形成可以被虚拟机直接使用的java类型。

43.类加载的机制及过程

- 程序主动使用某个类时，如果该类还未被加载到内存中，则JVM会通过加载、连接、初始化3个步骤来对该类进行初始化。如果没有意外，JVM将会连续完成3个步骤，所以有时也把这个3个步骤统称为类加载或类初始化。



1、加载

- 加载指的是将类的class文件读入到内存，并将这些静态数据转换成方法区中的运行时数据结构，并在堆中生成一个代表这个类的java.lang.Class对象，作为方法区类数据的访问入口，这个过程需要类加载器参与。
- Java类加载器由JVM提供，是所有程序运行的基础，JVM提供的这些类加载器通常被称为系统类加载器。除此之外，开发者可以通过继承ClassLoader基类来创建自己的类加载器。
- 类加载器，可以从不同来源加载类的二进制数据，比如：本地Class文件、Jar包Class文件、网络Class文件等等等。
- 类加载的最终产物就是位于堆中的Class对象（注意不是目标类对象），该对象封装了类在方法区中的数据结构，并且向用户提供了访问方法区数据结构的接口，即Java反射的接口

2、连接过程

- 当类被加载之后，系统为之生成一个对应的Class对象，接着将会进入连接阶段，连接阶段负责把类的二进制数据合并到JRE中（意思就是将java类的二进制代码合并到JVM的运行状态之中）。类连接又可分为如下3个阶段。

1. 验证：确保加载的类信息符合JVM规范，没有安全方面的问题。主要验证是否符合Class文件格式规范，并且是否能被当前的虚拟机加载处理。
2. 准备：正式为类变量（static变量）分配内存并设置类变量初始值的阶段，这些内存都将在方法区中进行分配
3. 解析：虚拟机常量池的符号引用替换为字节引用过程

3、初始化

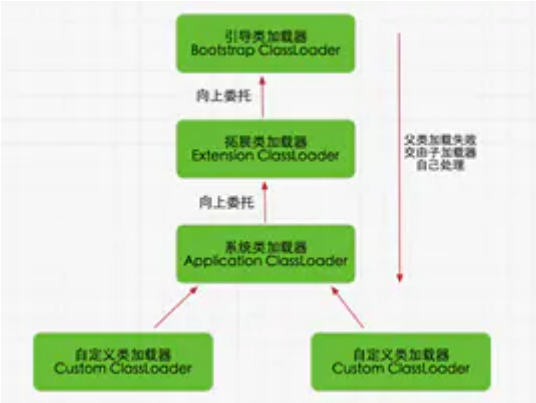
- 初始化阶段是执行类构造器`<clinit>`() 方法的过程。类构造器`<clinit>`()方法是由编译器自动收藏类中的`所有类变量的赋值动作和静态语句块(static块)中的语句合并产生，代码从上往下执行。`
- 当初始化一个类的时候，如果发现其父类还没有进行过初始化，则需要先触发其父类的初始化
- 虚拟机会保证一个类的`<clinit>`() 方法在多线程环境中被正确加锁和同步

`初始化的总结就是：初始化是为类的静态变量赋予正确的初始值`

44. 描述一下JVM加载Class文件的原理机制

- Java中的所有类，都需要由类加载器装载到JVM中才能运行。类加载器本身也是一个类，而它的工作就是把class文件从硬盘读取到内存中。在写程序的时候，我们几乎不需要关心类的加载，因为这些都是隐式装载的，除非我们有特殊的用法，像是反射，就需要显式的加载所需要的类。
- 类装载方式，有两种：
  - 1.隐式装载， 程序在运行过程中当碰到通过new 等方式生成对象时，隐式调用类装载器加载对应的类到jvm中，
  - 2.显式装载， 通过class.forname()等方法，显式加载需要的类
- Java类的加载是动态的，它并不会一次性将所有类全部加载后再运行，而是保证程序运行的基础类(像是基类)完全加载到jvm中，至于其他类，则在需要的时候才加载。这当然就是为了节省内存开销。

45. 什么是类加载器，类加载器有哪些?



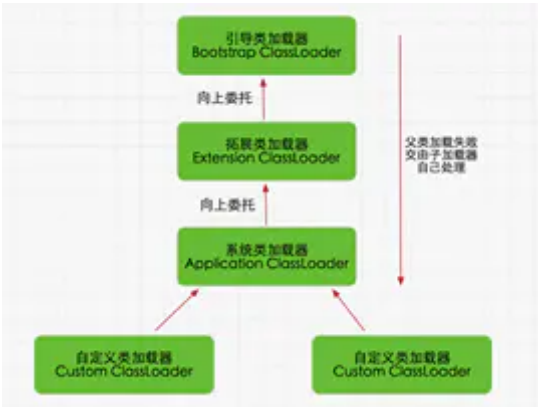
- 实现通过类的权限定名获取该类的二进制字节流的代码块叫做类加载器。
- 主要有一下四种类加载器:
  1. 启动类加载器(Bootstrap ClassLoader)用来加载java核心类库，无法被java程序直接引用。
  2. 扩展类加载器(extensions class loader):它用来加载 Java 的扩展库。Java 虚拟机的实现会提供一个扩展库目录。该类加载器在此目录里面查找并加载 Java 类。
  3. 系统类加载器（system class loader）：它根据 Java 应用的类路径（CLASSPATH）来加载 Java 类。一般来说，Java 应用的类都是由它来完成加载的。可以通过 ClassLoader.getSystemClassLoader()来获取它。
  4. 用户自定义类加载器，通过继承 java.lang.ClassLoader类的方式实现。

46. 说一下类装载的执行过程?

- 类装载分为以下 5 个步骤：
  - 加载：根据查找路径找到相应的 class 文件然后导入；
  - 验证：检查加载的 class 文件的正确性；
  - 准备：给类中的静态变量分配内存空间；
  - 解析：虚拟机将常量池中的符号引用替换成直接引用的过程。符号引用就理解为一个标示，而在直接引用直接指向内存中的地址；
  - 初始化：对静态变量和静态代码块执行初始化工作。

47. 什么是双亲委派模型?

- 在介绍双亲委派模型之前先说下类加载器。对于任意一个类，都需要由加载它的类加载器和这个类本身一同确立在 JVM 中的唯一性，每一个类加载器，都有一个独立的类名称空间。类加载器就是根据指定全限定名称将 class 文件加载到 JVM 内存，然后再转化为 class 对象。



- 类加载器分类：
  - 启动类加载器（Bootstrap ClassLoader），是虚拟机自身的一部分，用来加载 Java_HOME/lib/目录中的，或者被 -Xbootclasspath 参数所指定的路径中并且被虚拟机识别的类库；
  - 其他类加载器：
  - 扩展类加载器（Extension ClassLoader）：负责加载\lib\ext目录或Java. ext. dirs系统变量指定的路径中的所有类库；
  - 应用程序类加载器（Application ClassLoader）。负责加载用户类路径（classpath）上的指定类库，我们可以直接使用这个类加载器。一般情况，如果我们没有自定义类加载器默认就是用这个加载器。
- 双亲委派模型：如果一个类加载器收到了类加载的请求，它首先不会自己去加载这个类，而是把这个请求委派给父类加载器去完成，每一层的类加载器都是如此，这样所有的加载请求都会被传送到顶层的启动类加载器中，只有当父加载无法完成加载请求（它的搜索范围中没找到所需的类）时，子加载器才会尝试去加载类。
- 总结就是：`当一个类收到了类加载请求时，不会自己先去加载这个类，而是将其委派给父类，由父类去加载，如果此时父类不能加载，反馈给子类，由子类去完成类的加载。`

JVM调优

48. JVM 调优的参数可以在那设置参数值

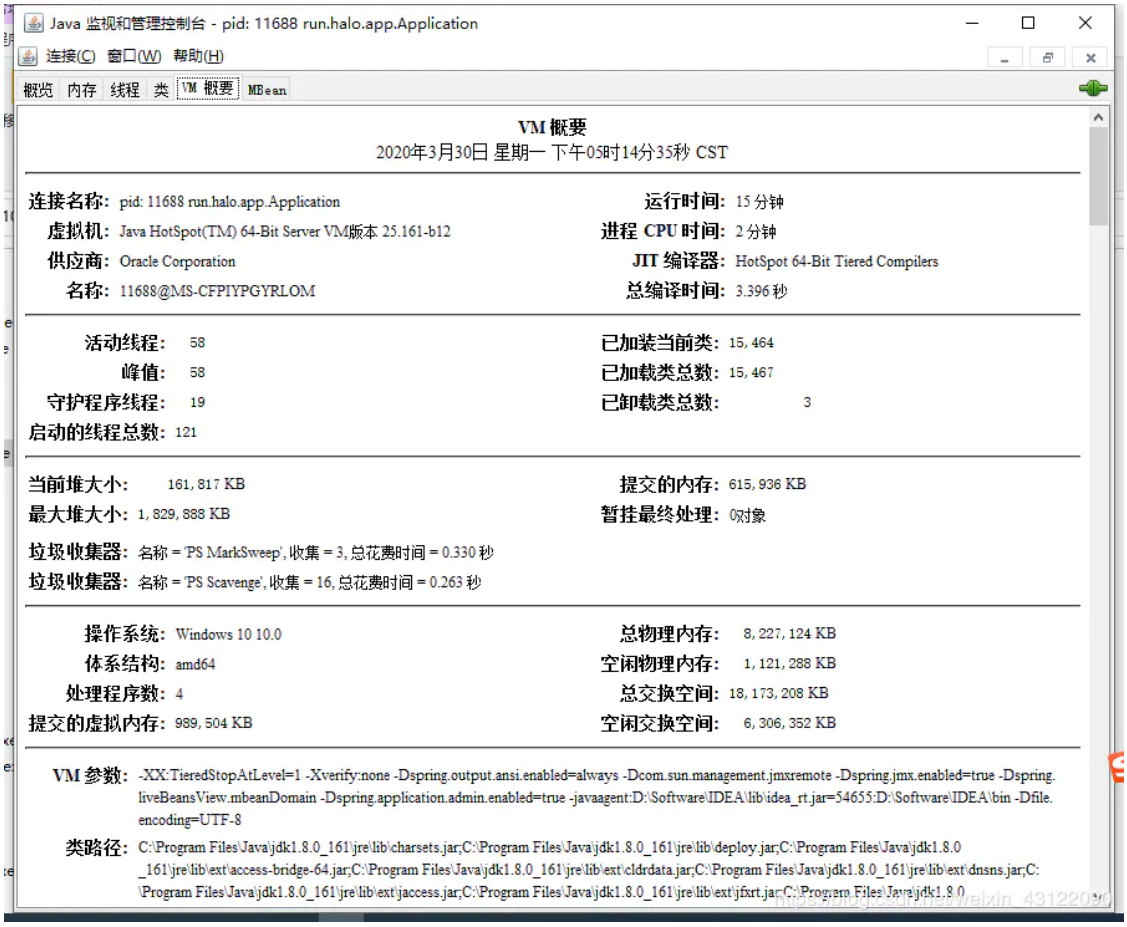
- 可以在IDEA，Eclipse，工具里设置

- 如果上线了是WAR包的话可以在Tomcat设置
- 如果是Jar包直接： java -jar 是直接插入JVM命令就好了

```
java -Xms1024m -Xmx1024m ...等等等 JVM参数 -jar springboot_app.jar &
```

49. 说一下 JVM 调优的工具?

- JDK 自带了很多监控工具，都位于 JDK 的 bin 目录下，其中最常用的是 jconsole 和 jvisualvm 这两款视图监控工具。
    - jconsole：用于对 JVM 中的内存、线程和类等进行监控；



    - jvisualvm：JDK 自带的全能分析工具，可以分析：内存快照、线程快照、程序死锁、监控内存的变化、gc 变化等。

```
![在这里插入图片描述](https://user-gold-cdn.xitu.io/2020/4/13/171729fd4d2a3018?imageView2/0/w/1280/h/960/format/webp/ignore-error/1)
```

50. 常用的 JVM 调优的参数都有哪些?

```
#常用的设置
-Xms：初始堆大小，JVM 启动的时候，给定堆空间大小。

-Xmx：最大堆大小，JVM 运行过程中，如果初始堆空间不足的时候，最大可以扩展到多少。

-Xmn：设置堆中年轻代大小。整个堆大小=年轻代大小+年老代大小+持久代大小。
```

-XX:NewSize=n 设置年轻代初始化大小大小

-XX:MaxNewSize=n 设置年轻代最大值

-XX:NewRatio=n 设置年轻代和年老代的比值。如: -XX:NewRatio=3，表示年轻代与年老代比值为 1：3，年轻代占整个年轻代+年老代和的 1/4

-XX:SurvivorRatio=n 年轻代中 Eden 区与两个 Survivor 区的比值。注意 Survivor 区有两个。8表示两个Survivor :eden=2:8 ,即一个Survivor占年轻代的1/10，默认就为8

-Xss：设置每个线程的堆栈大小。JDK5后每个线程 Java 栈大小为 1M，以前每个线程堆栈大小为 256K。

-XX:ThreadStackSize=n 线程堆栈大小

-XX:PermSize=n 设置持久代初始值

-XX:MaxPermSize=n 设置持久代大小

-XX:MaxTenuringThreshold=n 设置年轻带垃圾对象最大年龄。如果设置为 0 的话，则年轻代对象不经过 Survivor 区，直接进入年老代。

#下面是一些不常用的

-XX:LargePageSizeInBytes=n 设置堆内存的内存页大小

-XX:+UseFastAccessorMethods 优化原始类型的getter方法性能

-XX:+DisableExplicitGC 禁止在运行期显式地调用System.gc()，默认启用

-XX:+AggressiveOpts 是否启用JVM开发团队最新的调优成果。例如编译优化，偏向锁，并行年老代收集等，jdk6纸之后默认启动

-XX:+UseBiasedLocking 是否启用偏向锁，JDK6默认启用

-Xnoclassgc 是否禁用垃圾回收

-XX:+UseThreadPriorities 使用本地线程的优先级，默认启用

等等等.....

51. JVM的GC收集器设置

- -xx:+Use xxx GC
    - xxx 代表垃圾收集器名称

-XX:+UseSerialGC:设置串行收集器，年轻带收集器

-XX:+UseParNewGC:设置年轻代为并行收集。可与 CMS 收集同时使用。JDK5.0 以上，JVM 会根据系统配置自行设置，所以无需再设置此值。

-XX:+UseParallelGC:设置并行收集器，目标是目标是达到可控制的吞吐量

-XX:+UseParallelOldGC:设置并行年老代收集器，JDK6.0 支持对年老代并行收集。

-XX:+UseConcMarkSweepGC:设置年老代并发收集器

-XX:+UseG1GC:设置 G1 收集器，JDK1.9默认垃圾收集器