JVM 常见面试题指南

本文将重点介绍面试过程中常见的 JVM 题目。 将面试题分为三大类：基础题目，进阶题目，实战题目。

基础

1.1 JDK、 JRE、 JVM 的关系是什么?

什么是 JVM ?

英文名称 (Java Virtual Machine)，就是 JAVA 虚拟机， 它只识别 `.class` 类型文件，它能够将 class 文件中的字节码指令进行识别并调用操作系统向上的 API 完成动作。

什么是 JRE ?

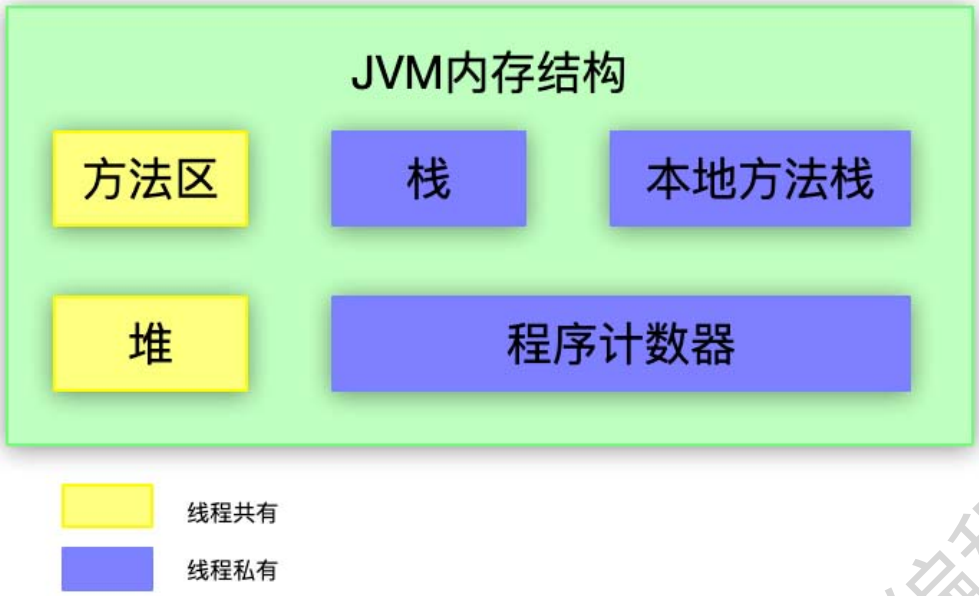
英文名称 （ Java Runtime Environment ） ， Java 运行时环境。它主要包含两个部分：JVM 的标准实现和 Java 的一些基本类库。相对于 JVM 来说，JRE多出来一部分 Java 类库。

什么是 JDK?
英文名称 （ Java Development Kit ） ， Java 开发工具包。JDK 是整个 Java 开发的核心，它集成了 JRE 和一些好用的小工具。例如：javac.exe、java.exe、jar.exe 等。

这三者的关系：一层层的嵌套关系。JDK > JRE > JVM。

1.2 JVM 的内存模型以及分区情况和作用

如下图所示:



黄色部分为线程共有，蓝色部分为线程私有。

方法区

用于存储虚拟机加载的类信息，常量，静态变量等数据。

堆

存放对象实例，所有的对象和数组都要在堆上分配。 是 JVM 所管理的内存中最大的一块区域。

栈

Java 方法执行的内存模型：存储局部变量表，操作数栈，动态链接，方法出口等信息。生命周期与线程相同。

本地方法栈

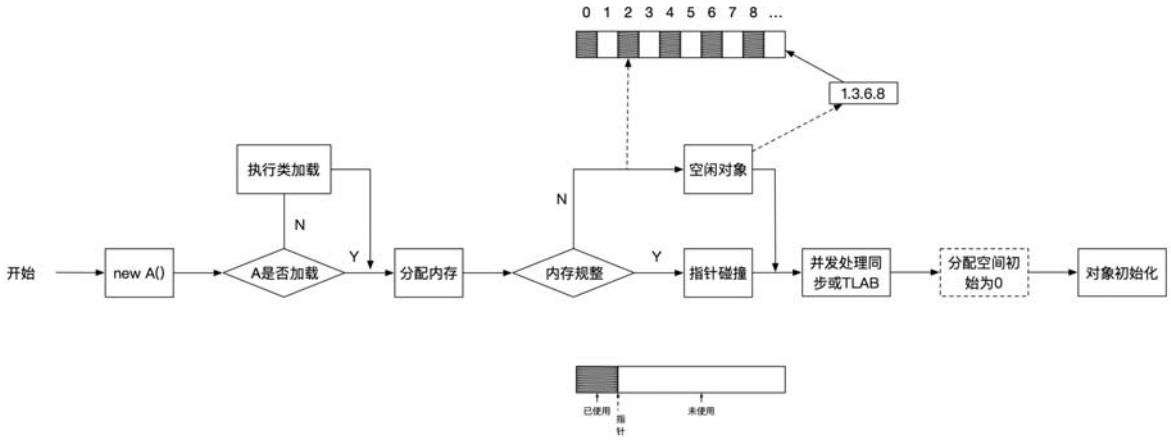
作用与虚拟机栈类似，不同点本地方法栈为 native 方法执行服务，虚拟机栈为虚拟机执行的 Java 方法服务。

程序计数器

当前线程所执行的行号指示器。是 JVM 内存区域最小的一块区域。执行字节码工作时就是利用程序计数器来选取下一条需要执行的字节码指令。

1.3 JVM 对象创建步骤流程是什么？

整体流程如下图所示：



第 1 步：虚拟机遇到一个 new 指令，首先将去检查这个指令的参数是否能在常量池中定位到这个类的符号引用，并且检查这个符号引用的类是否已经被加载&解析&初始化。

第 2 步：如果类已经被加载那么进行第 3 步；如果没有进行加载，那么就就需要先进行类的加载。

第 3 步：类加载检查通过之后，接下来进行新生对象的内存分配。

第 4 步：对象生成需要的内存大小在类加载完成后便可完全确定，为对象分配空间等同于把一块确定大小的内存从 Java 堆中划分出来

第 5 步：内存大小的划分分为两种情况：
第一种情况：JVM 的内存是规整的，所有的使用的内存都放到一边，空闲的内存在另外一边，中间放一个指针作为分界点的指示器。那么这时候分配内存就比较简单，只要讲指针向空闲空间那边挪动一段与对象大小相同的距离。这种就是“**指针碰撞**”。

第二种情况：JVM 的内存不是规整的，也就是说已使用的内存与未使用的内存相互交错。这时候就没办法利用指正碰撞了。这时候我们就需要维护一张表，用于记录那些内存可用，在分配的时候从列表中找到一块足够大的空间划分给对象实例，并更新到记录表上。

第 6 步：空间申请完成之后，JVM 需要将内存的空间都初始化为 0 值。如果使用 TLAB，就可以在 TLAB 分配的时候就可以进行该工作。

第 7 步：JVM 对对象进行必要的设置。例如，这个对象是哪个类的实例、对象的哈希码、GC 年代等信息。

第 8 步：完成了上面的步骤之后 从 JVM 来看一个对象基本上完成了，但从 Java 程序代码绝对来看，对象创建才刚刚开始，需要执行 < init > 方法，按照程序中设定的初始化操作初始化，这时候一个真正的程序对象生成了。

1.4 垃圾回收算法有几种类型？他们对应的优缺点又是什么？

常见的垃圾回收算法有：

> 标记-清除算法、复制算法、标记-整理算法、分代收集算法

标记-清除算法

标记—清除算法包括两个阶段：“标记”和“清除”。
标记阶段：确定所有要回收的对象，并做标记。
清除阶段：将标记阶段确定不可用的对象清除。

缺点：
1. 标记和清除的效率都不高。
2. 会产生大量的碎片，而导致频繁的回收。

复制算法

内存分成大小相等的两块，每次使用其中一块，当垃圾回收的时候，
把存活的对象复制到另一块上，然后把这块内存整个清理掉。

缺点：
1. 需要浪费额外的内存作为复制区。
2. 当存活率较高时，复制算法效率会下降。

标记-整理算法

标记—整理算法不是把存活对象复制到另一块内存，而是把存活对象往内存的一端移动，然后直接回收边界以外的内存。

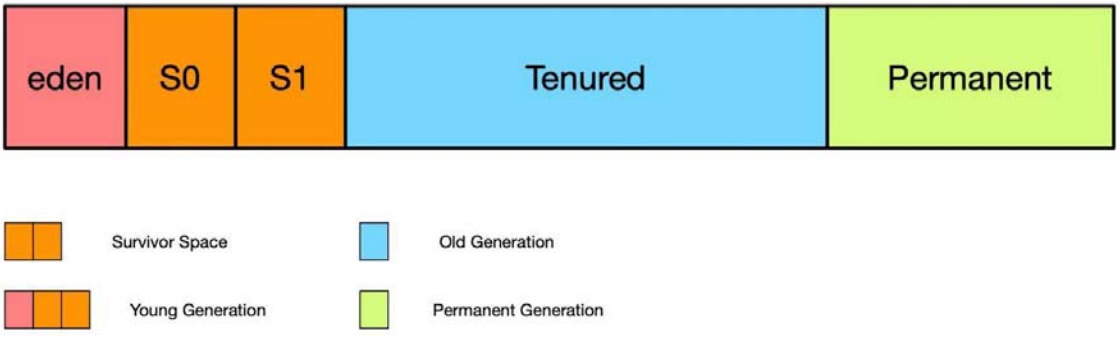
缺点：
算法复杂度大，执行步骤较多

分代收集算法

目前大部分 JVM 的垃圾收集器采用的算法。根据对象存活的生命周期将内存划分为若干个不同的区域。一般情况下将堆区划分为新生代（Young Generation 和老年代（Tenured Generation），永久代（Permanet Generation）。

老年代的特点是每次垃圾收集时只有少量对象需要被回收，而新生代的特点是每次垃圾回收时都有大量的对象需要被回收，那么就可以根据不同代的特点采取最适合的收集算法。

如下图所示：



Young：存放新创建的对象，对象生命周期非常短，几乎用完可以立即回收，也叫 Eden 区。

Tenured： young 区多次回收后存活下来的对象将被移到 tenured 区，也叫 old 区。

Perm：永久带，主要存加载的类信息，生命周期长，几乎不会被回收。

缺点：
算法复杂度大，执行步骤较多。

1.5 简单介绍一下什么是类加载机制?

Class 文件由类装载器装载后，在 JVM 中将形成一份描述 Class 结构的元信息对象，通过该元信息对象可以获知 Class 的结构信息：如构造函数，属性和方法等。

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，转换解析和初始化，最终形成可以被虚拟机直接使用的 Java 类型，这就是虚拟机的类加载机制。

1.6 类的加载过程是什么？简单描述一下每个步骤

类加载的过程包括了：

> 加载、验证、准备、解析、初始化五个阶段

第一步：加载

查找并加载类的二进制数据。

加载是类加载过程的第一个阶段，虚拟机在这一阶段需要完成以下三件事情：

- 通过类的全限定名来获取其定义的二进制字节流
- 将字节流所代表的静态存储结构转化为方法区的运行时数据结构
- 在 Java 堆中生成一个代表这个类的 java.lang.Class 对象，作为对方法区中这些数据的访问入口

第二步：验证

确保被加载的类的正确性。

这一阶段是确保 Class 文件的字节流中包含的信息符合当前虚拟机的规范，并且不会损害虚拟机自身的安全。包含了四个验证动作：文件格式验证，元数据验证，字节码验证，符号引用验证。

第三步：准备

为类的静态变量分配内存，并将其初始化为默认值。

准备阶段是正式为类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分配。

第四步：解析

把类中的符号引用转换为直接引用。

解析阶段是虚拟机将常量池内的符号引用替换为直接引用的过程，解析动作主要针对类或接口、字段、类方法、接口方法、方法类型、方法句柄和调用点限定符 7 类符号引用进行。

第五步：初始化

类变量进行初始化

为类的静态变量赋予正确的初始值，JVM 负责对类进行初始化，主要对类变量进行初始化。

1.7 JVM 预定义的类加载器有哪几种？分别什么作用？

> 启动（Bootstrap）类加载器、标准扩展（Extension）类加载器、应用程序类加载器（Application）

启动（Bootstrap）类加载器

引导类装入器是用本地代码实现的类装入器，它负责将 < Java_Runtime_Home >/lib 下面的类库加载到内存中。由于引导类加载器涉及到虚拟机本地实现细节，开发者无法直接获取到启动类加载器的引用。

标准扩展（Extension）类加载器

扩展类加载器负责将 < Java_Runtime_Home >/lib/ext 或者由系统变量 java.ext.dir 指定位置中的类库加载到内存中。开发者可以直接使用标准扩展类加载器。

应用程序类加载器（Application）

应用程序类加载器（Application ClassLoader）：负责加载用户路径（classpath）上的类库。

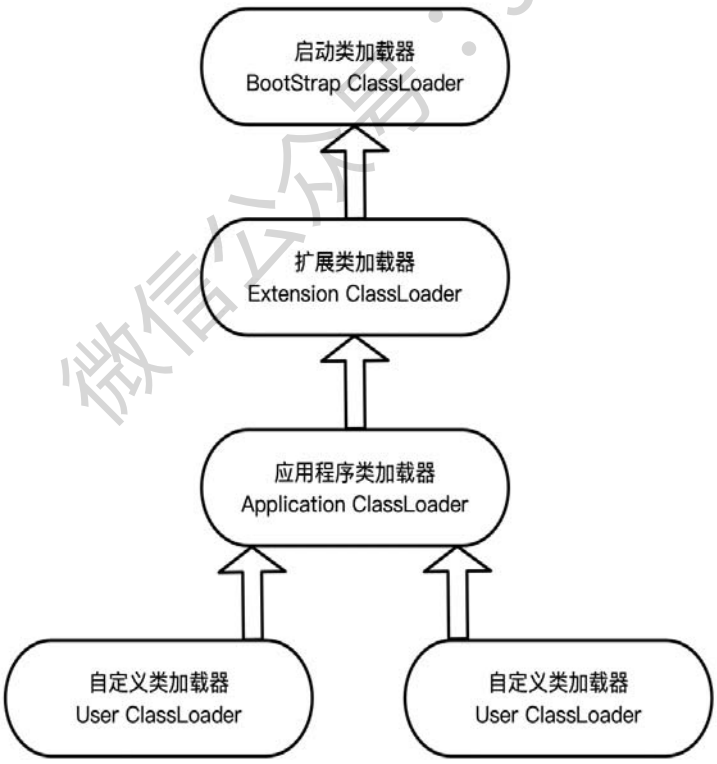
1.8 什么是双亲委派模式？有什么作用？

基本定义：
双亲委派模型的工作流程是：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，而是把请求委托给父加载器去完成，依次向上，因此，所有的类加载请求最终都应该被传递到顶层的启动类加载器中，只有当父加载器没有找到所需的类时，子加载器才会尝试去加载该类。

双亲委派机制：

1. 当 AppClassLoader 加载一个 class 时，它首先不会自己去尝试加载这个类，而是把类加载请求委派给父类加载器 ExtClassLoader 去完成。
2. 当 ExtClassLoader 加载一个 class 时，它首先也不会自己去尝试加载这个类，而是把类加载请求委派给 BootStrapClassLoader 去完成。
3. 如果 BootStrapClassLoader 加载失败，会使用 ExtClassLoader 来尝试加载；
4. 若 ExtClassLoader 也加载失败，则会使用 AppClassLoader 来加载，如果 AppClassLoader 也加载失败，则会报出异常 ClassNotFoundException。

如下图所示：



双亲委派作用：

- 通过带有优先级的层级关可以避免类的重复加载；
- 保证 Java 程序安全稳定运行，Java 核心 API 定义类型不会被随意替换。

1.9 介绍一下 JVM 中垃圾收集器有哪些？他们特点分别是什么？

新生代垃圾收集器

Serial 收集器

特点：
Serial 收集器只能使用一条线程进行垃圾收集工作，并且在进行垃圾收集的时候，所有的工作线程都需要停止工作，等待垃圾收集线程完成以后，其他线程才可以继续工作。

使用算法：复制算法

ParNew 收集器

特点：
ParNew 垃圾收集器是Serial收集器的多线程版本。为了利用 CPU 多核多线程的优势，ParNew 收集器可以运行多个收集线程来进行垃圾收集工作。这样可以提高垃圾收集过程的效率。

使用算法：复制算法

Parallel Scavenge 收集器

特点：
Parallel Scavenge 收集器是一款多线程的垃圾收集器，但是它又和 ParNew 有很大的不同点。

Parallel Scavenge 收集器和其他收集器的关注点不同。其他收集器，比如 ParNew 和 CMS 这些收集器，它们主要关注的是如何缩短垃圾收集的时间。而 Parallel Scavenge 收集器关注的是如何控制系统运行的吞吐量。这里说的吞吐量，指的是 CPU 用于运行应用程序的时间和 CPU 总时间的占比，吞吐量 = 代码运行时间 /（代码运行时间 + 垃圾收集时间）。如果虚拟机运行的总的 CPU 时间是 100 分钟，而用于执行垃圾收集的时间为 1 分钟，那么吞吐量就是 99%。

使用算法：复制算法

老年代垃圾收集器

Serial Old 收集器

特点：
Serial Old 收集器是 Serial 收集器的老年代版本。这款收集器主要用于客户端应用程序中作为老年代的垃圾收集器，也可以作为服务端应用程序的垃圾收集器。

使用算法：标记-整理

Parallel Old 收集器

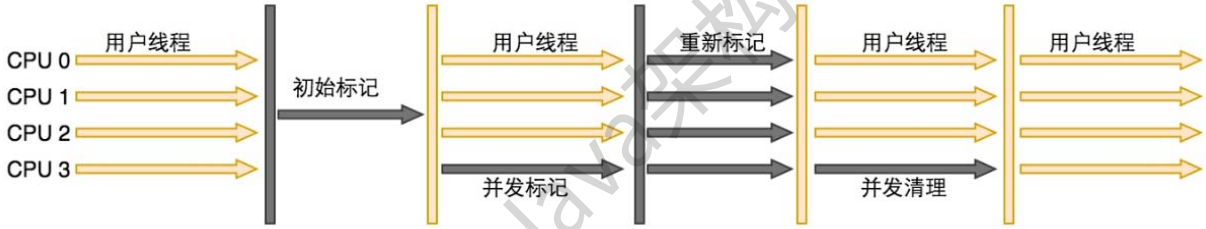
特点：
Parallel Old 收集器是 Parallel Scavenge 收集器的老年代版本这个收集器是在 JDK1.6 版本中出现的，所以在 JDK1.6 之前，新生代的 Parallel Scavenge 只能和 Serial Old 这款单线程的老年代收集器配合使用。Parallel Old 垃圾收集器和 Parallel Scavenge 收集器一样，也是一款关注吞吐量的垃圾收集器，和 Parallel Scavenge 收集器一起配合，可以实现对 Java 堆内存的吞吐量优先的垃圾收集策略。

使用算法：标记-整理

CMS 收集器

特点：
CMS 收集器是目前老年代收集器中比较优秀的垃圾收集器。CMS 是 Concurrent Mark Sweep，从名字可以看出，这是一款使用”标记-清除”算法的并发收集器。

CMS 垃圾收集器是一款以获取最短停顿时间为目标的收集器。如下图所示：



从图中可以看出，CMS 收集器的工作过程可以分为 4 个阶段：

- 初始标记（CMS initial mark）阶段
- 并发标记（CMS concurrent mark）阶段
- 重新标记（CMS remark）阶段
- 并发清除(（CMS concurrent sweep）阶段

使用算法：复制+标记清除

其他

G1 垃圾收集器

特点：
主要步骤： 初始标记，并发标记，重新标记，复制清除。

使用算法：复制 + 标记整理

1.10 什么是 Class 文件？ Class 文件主要的信息结构有哪些？

Class 文件是一组以 8 位字节为基础单位的二进制流。各个数据项严格按顺序排列。

Class 文件格式采用一种类似于 C 语言结构体的伪结构来存储数据。这样的伪结构仅仅有两种数据类型：无符号数和表。

无符号数：是基本数据类型。以 u1、u2、u4、u8 分别代表 1 个字节、2 个字节、4 个字节、8 个字节的无符号数，能够用来描写叙述数字、索引引用、数量值或者依照 UTF-8 编码构成的字符串值。

表：由多个无符号数或者其它表作为数据项构成的复合数据类型。全部表都习惯性地以 `_info` 结尾。

1.11 对象“对象已死” 是什么概念?

对象不可能再被任何途径使用，称为对象已死。
判断对象已死的方法有：引用计数法与可达性分析算法。

进阶

2.1 Java 语言怎么实现跨平台的?

我们编写的 Java 源码，编译后会生成一种 .class 文件，称为字节码文件。字节码不能直接运行，必须通过 JVM 翻译成机器码才能运行。

JVM 是一个”桥梁“，是一个”中间件“，是实现跨平台的关键。Java 代码首先被编译成字节码文件，再由 JVM 将字节码文件翻译成机器语言，从而达到运行 Java 程序的目的。

2.2 JVM 数据运行区，哪些会造成 OOM 的情况?

除了数据运行区，其他区域均有可能造成 OOM 的情况。

```
堆溢出：java.lang.OutOfMemoryError: Java heap space
栈溢出：java.lang.StackOverflowError
永久代溢出：java.lang.OutOfMemoryError: PermGen space
```

2.3 详细介绍一下对象在分带内存区域的分配过程?

1. JVM 会试图为相关 Java 对象在 Eden 中初始化一块内存区域。
2. 当 Eden 空间足够时，内存申请结束；否则到下一步。
3. JVM 试图释放在 Eden 中所有不活跃的对象（这属于 1 或更高级的垃圾回收）。释放后若 Eden 空间仍然不足以放入新对象，则试图将部分 Eden 中活跃对象放入 Survivor 区。

4. Survivor 区被用来作为 Eden 及 Old 的中间交换区域，当 Old 区空间足够时，Survivor 区的对象会被移到 Old 区，否则会被保留在 Survivor 区。
5. 当 Old 区空间不够时，JVM 会在 Old 区进行完全的垃圾收集。
6. 完全垃圾收集后，若 Survivor 及 Old 区仍然无法存放从 Eden 复制过来的部分对象，导致 JVM 无法在 Eden 区为新对象创建内存区域，则出现 “ out of memory ” 错误。

1.4 G1 与 CMS 两个垃圾收集器的对比

细节方面不同

1. G1 在压缩空间方面有优势。
2. G1 通过将内存空间分成区域（Region）的方式避免内存碎片问题。
3. Eden, Survivor, Old 区不再固定、在内存使用效率上来说更灵活。
4. G1 可以通过设置预期停顿时间（Pause Time）来控制垃圾收集时间避免应用雪崩现象。
5. G1 在回收内存后会马上同时做合并空闲内存的工作、而 CMS 默认是在 STW（stop the world）的时候做。
6. G1 会在 Young GC 中使用、而 CMS 只能在 O 区使用。

整体内容不同

```
吞吐量优先：G1
响应优先：CMS
```

CMS 的缺点是对 cpu 的要求比较高。G1 是将内存化成了多块，所有对内段的大小有很大的要求。

CMS 是清除，所以会存在很多的内存碎片。G1 是整理，所以碎片空间较小。

2.5 线上常用的 JVM 参数有哪些?

数据区设置

- Xms：初始堆大小
- Xmx：最大堆大小
- Xss:Java 每个线程的Stack大小
- XX:NewSize=n：设置年轻代大小
- XX:NewRatio=n：设置年轻代和年老代的比值。如：为 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。如：3，表示 Eden：Survivor=3：2，一个 Survivor 区占整个年轻代的 1/5。
- XX：MaxPermSize=n：设置持久代大小。

收集器设置

- XX:+UseSerialGC：设置串行收集器
- XX:+UseParallelGC:：设置并行收集器
- XX:+UseParalledlOldGC：设置并行年老代收集器
- XX:+UseConcMarkSweepGC：设置并发收集器

GC日志打印设置

- XX:+PrintGC：打印 GC 的简要信息
- XX:+PrintGCDetails：打印 GC 详细信息
- XX:+PrintGCTimeStamps：输出 GC 的时间戳

2.6 对象什么时候进入老年代?

对象优先在 Eden 区分配内存

当对象首次创建时, 会放在新生代的 eden 区, 若没有 GC 的介入，会一直在 eden 区，GC 后，是可能进入 survivor 区或者年老代

大对象直接进入老年代

所谓的大对象是指需要大量连续内存空间的 Java 对象，最典型的大对象就是那种很长的字符串以及数组，大对象对虚拟机的内存分配就是坏消息，尤其是一些朝生夕灭的短命大对象，写程序时应避免。

长期存活的对象进入老年代

虚拟机给每个对象定义了一个对象年龄（Age）计数器，对象在 Survivor 区中每熬过一次 Minor GC，年龄就增加 1，当他的年龄增加到一定程度（默认是 15 岁），就将会被晋升到老年代中。

2.7 什么是内存溢出，内存泄露？他们的区别是什么?

内存溢出 out of memory，是指程序在申请内存时，没有足够的内存空间供其使用，出现 out of memory；

内存泄露 memory leak，是指程序在申请内存后，无法释放已申请的内存空间，一次内存泄露危害可以忽略，但内存泄露堆积后果很严重，无论多少内存，迟早会被占光。

内存溢出就是你要求分配的内存超出了系统能给你的，系统不能满足需求，于是产生溢出。

内存泄漏是指你向系统申请分配内存进行使用（new），可是使用完了以后却不归还（delete），结果你申请到的那块内存你自己也不能再访问（也许你把它的地址给弄丢了），而系统也不能再次将它分配给需要的程序。

2.8 引起类加载操作的行为有哪些?

1. 遇到 new、getstatic、putstatic 或 invokestatic 这四条字节码指令。
2. 反射调用的时候，如果类没有进行过初始化，则需要先触发其初始化。
3. 子类初始化的时候，如果其父类还没初始化，则需先触发其父类的初始化。
4. 虚拟机执行主类的时候（有 main（ string[] args））。
5. JDK1.7 动态语言支持。

2.9 介绍一下 JVM 提供的常用工具

1. **jps：**用来显示本地的 Java 进程，可以查看本地运行着几个 Java 程序，并显示他们的进程号。
   命令格式：jps

2. **jinfo：**运行环境参数：Java System 属性和 JVM 命令行参数，Java class path 等信息。
   命令格式：jinfo 进程 pid

3. **jstat：**监视虚拟机各种运行状态信息的命令行工具。
   命令格式：jstat -gc 123 250 20

4. **jstack：**可以观察到 JVM 中当前所有线程的运行情况和线程当前状态。
   命令格式：jstack 进程 pid

5. **jmap：**观察运行中的 JVM 物理内存的占用情况（如：产生哪些对象，及其数量）。
   命令格式：jmap [option] pid

2.10 Full GC 、 Major GC、 Minor GC 之间区别?

Minor GC：
从新生代空间（包括 Eden 和 Survivor 区域）回收内存被称为 Minor GC。

Major GC：
清理 Tenured 区，用于回收老年代，出现 Major GC 通常会出现至少一次 Minor GC。

Full GC：
Full GC 是针对整个新生代、老年代、元空间（metaspace，java8 以上版本取代 perm gen）的全局范围的 GC。

2.11 什么时候触发 Full GC ?

1. 调用 System.gc 时，系统建议执行 Full GC，但是不必然执行。
2. 老年代空间不足。

3. 方法区空间不足。
4. 通过 Minor GC 后进入老年代的平均大小大于老年代的可用内存。
5. 由 Eden 区、survivor space1（From Space）区向 survivor space2（To Space）区复制时，对象大小大于 To Space 可用内存，则把该对象转存到老年代，且老年代的可用内存小于该对象大小。

2.12 什么情况下会出现栈溢出

1. 方法创建了一个很大的对象，如 List，Array。
2. 是否产生了循环调用、死循环。
3. 是否引用了较大的全局变量。

2.13 说一下强引用、软引用、弱引用、虚引用以及他们之间和 gc 的关系

1. 强引用：new 出的对象之类的引用，只要强引用还在，永远不会回收。
2. 软引用：引用但非必须的对象，内存溢出异常之前，回收。
3. 弱引用：非必须的对象，对象能生存到下一次垃圾收集发生之前。
4. 虚引用：对生存时间无影响，在垃圾回收时得到通知。

2.14 Eden 和 Survivor 的比例分配是什么情况？为什么？

默认比例 8:1。
大部分对象都是朝生夕死。
复制算法的基本思想就是将内存分为两块，每次只用其中一块，当这一块内存用完，就将还活着的对象复制到另外一块上面。复制算法不会产生内存碎片。

实战

3.1 CPU 资源占用过高

1. top 查看当前 CPU 情况，找到占用 CPU 过高的进程 PID=123。
2. top -H -p123 找出两个 CPU 占用较高的线程，记录下来 PID=2345, 3456 转换为十六进制。
3. jstack -l 123 > temp.txt 打印出当前进程的线程栈。
4. 查找到对应于第二步的两个线程运行栈，分析代码。

3.2 OOM 异常排查

1. 使用 top 指令查询服务器系统状态。
2. ps -aux|grep java 找出当前 Java 进程的 PID。

3. jstat -gcutil pid interval 查看当前 GC 的状态。
4. jmap -histo:live pid 可用统计存活对象的分布情况，从高到低查看占据内存最多的对象。
5. jmap -dump:format=b,file= 文件名 [pid] 利用 Jmap dump。
6. 使用性能分析工具对上一步 dump 出来的文件进行分析，工具有 MAT 等。

总结

上面介绍了 JVM 常见的面试题目，希望对大家接下里的面试或者对于 JVM 的深入学习有所帮助。