链滴

JVM 知识点总览 - 高级 Java 工程师面试必备

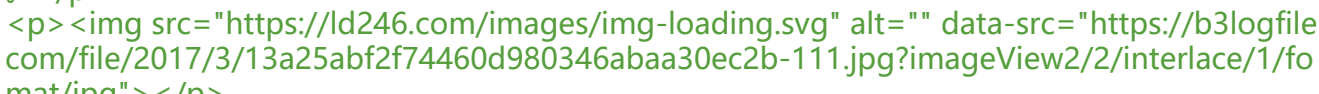
作者：whxiaobu

原文链接：https://ld246.com/article/1488526771990

来源网站：链滴

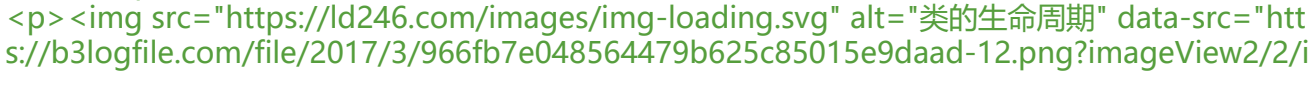
<blockquote>
来源：jvm 知识点总览-高级 Java 工程师面试备
作者：纯洁的微笑
版权归作者所有，转载请注明出处
</blockquote>
在江湖中要练就绝世武功必须内外兼备，精妙的招式和深厚的内功，武功的基础是内功。对于武低（就像江南七怪）的人，招式更重要，因为他们不能靠内功直接去伤人，只能靠招式，利刃上优势取胜了，但是练到高手之后，内功就更主要了。一个内功低的人招式在奇妙也打不过一个内功高的人比如，你剑法再厉害，一剑刺过来，别人一掌打断你的剑，你还怎么使剑法，你一掌打到一个武功高人身上，那人没什么事，却把你震伤了，你还怎么打。同样两者也是相辅相成的，内功深厚之后，原普通的一招一式威力也会倍增。
对于搞开发的我们其实也是一样，现在流行的框架越来越多，封装的也越来越完善，各种框架可搞定一切，几乎不用关注底层的实现，初级程序员只要熟悉基本的使用方法，便可以快速的开发上线但对于高级程序员来讲，内功的修炼却越发的重要，比如算法、设计模式、底层原理等，只有把这些础熟练之后，才能在开发过程中知其然知其所以然，出现问题时能快速定位到问题的本质。
对于 Java 程序员来讲，spring 全家桶几乎可以搞定一切，spring 全家桶便是精妙的招式，jvm 是内功心法很重要的一块，线上出现性能问题，jvm 调优更是不可回避的问题。因此 JVM 基础知识于高级程序员的重要性不必言语，我司在面试高级开发的时候，jvm 相关知识也必定是考核的标准之。本篇文章会根据之前写的 jvm 系列文章梳理出 jvm 需要关注的所有考察点。
<h2 id="jvm-总体梳理">jvm 总体梳理</h2>
jvm 体系总体分四大块：

类的加载机制
jvm 内存结构
GC 算法 垃圾回收
GC 分析 命令调优

当然这些知识点在之前的文章中都有详细的介绍，这里只做主干的梳理
这里画了一个思维导图，将所有的知识点进行了陈列，因为图比较大可以点击右键下载了放大查。


<h2 id="类的加载机制">类的加载机制</h2>
主要关注点：

什么是类的加载
类的生命周期
类加载器
双亲委派模型

什么是类的加载
类的加载指的是将类的.class 文件中的二进制数据读入到内存中，将其放在运行时数据区的方法内，然后在堆区创建一个 java.lang.Class 对象，用来封装类在方法区内的数据结构。类的加载的最产品是位于堆区中的 Class 对象，Class 对象封装了类在方法区内的数据结构，并且向 Java 程序员提了访问方法区内的数据结构的接口。
类的生命周期
类的生命周期包括这几个部分，加载、连接、初始化、使用和卸载，其中前三部是类的加载的过程如下图:


terlace/1/format/jpg">

加载，查找并加载类的二进制数据，在 Java 堆中也创建一个 java.lang.Class 类的对象
连接，连接又包含三块内容：验证、准备、初始化。1）验证，文件格式、元数据、字节码、符引用验证；2）准备，为类的静态变量分配内存，并将其初始化为默认值；3）解析，把类中的符号引转换为直接引用
初始化，为类的静态变量赋予正确的初始值
使用，new 出对象程序中使用
卸载，执行垃圾回收

<blockquote>
几个小问题？
1、JVM 初始化步骤？ 2、类初始化时机？ 3、哪几种情况下，Java 虚拟机将结束生命周期?/em>
答案参考这篇文章 jvm 系列一):java 类的加载机制
</blockquote>
类加载器
<img src="https://ld246.com/images/img-loading.svg" alt="类加载器" data-src="https://3logfile.com/file/2017/3/2d3457a4c4104cec814c10a7bbe90627-13.png?imageView2/2/interlce/1/format/jpg">

启动类加载器：Bootstrap ClassLoader，负责加载存放在 JDK\jre\lib(JDK 代表 JDK 的安装目，下同)下，或被-Xbootclasspath 参数指定的路径中的，并且能被虚拟机识别的类库
扩展类加载器：Extension ClassLoader，该加载器由 sun.misc.Launcher$ExtClassLoader 实，它负责加载 DK\jre\lib\ext 目录中，或者由 java.ext.dirs 系统变量指定的路径中的所有类库（如 javax.*开头的类），开发者可以直接使用扩展类加载器。
应用程序类加载器：Application ClassLoader，该类加载器由 sun.misc.Launcher$AppClassLoder 来实现，它负责加载用户类路径（ClassPath）所指定的类，开发者可以直接使用该类加载器

类加载机制

全盘负责，当一个类加载器负责加载某个 Class 时，该 Class 所依赖的和引用的其他 Class 也将该类加载器负责载入，除非显示使用另外一个类加载器来载入
父类委托，先让父类加载器试图加载该类，只有在父类加载器无法加载该类时才尝试从自己的类径中加载该类
缓存机制，缓存机制将会保证所有加载过的 Class 都会被缓存，当程序中需要使用某个 Class 时类加载器先从缓存区寻找该 Class，只有缓存区不存在，系统才会读取该类对应的二进制数据，并将转换成 Class 对象，存入缓存区。这就是为什么修改了 Class 后，必须重启 JVM，程序的修改才会生

<h2 id="jvm内存结构">jvm 内存结构</h2>
主要关注点：

jvm 内存结构都是什么
对象分配规则

jvm 内存结构
<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfilecom/file/2017/3/fa1968da09fb4b52b90f61d8429114b4-14.png?imageView2/2/interlace/1/fomat/jpg">
<blockquote>
方法区和对是所有线程共享的内存区域；而 java 栈、本地方法栈和程序员计数器是运行是线程

有的内存区域。
</blockquote>

Java 堆（Heap）,是 Java 虚拟机所管理的内存中最大的一块。Java 堆是被所有线程共享的一块存区域，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，几乎所有的对象实例都在里分配内存。
方法区（Method Area）,方法区（Method Area）与 Java 堆一样，是各个线程共享的内存区，它用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。
程序计数器（Program Counter Register）,程序计数器（Program Counter Register）是一块小的内存空间，它的作用可以看做是当前线程所执行的字节码的行号指示器。
JVM 栈（JVM Stacks）,与程序计数器一样，Java 虚拟机栈（Java Virtual Machine Stacks）是线程私有的，它的生命周期与线程相同。虚拟机栈描述的是 Java 方法执行的内存模型：每个方法执行的时候都会同时创建一个栈帧（Stack Frame）用于存储局部变量表、操作栈、动态链接、方法口等信息。每一个方法被调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中从入栈到出栈的程。
本地方法栈（Native Method Stacks）,本地方法栈（Native Method Stacks）与虚拟机栈所发的作用是非常相似的，其区别不过是虚拟机栈为虚拟机执行 Java 方法（也就是字节码）服务，而本方法栈则是为虚拟机使用到的 Native 方法服务。

对象分配规则

对象优先分配在 Eden 区，如果 Eden 区没有足够的空间时，虚拟机执行一次 Minor GC。
大对象直接进入老年代（大对象是指需要大量连续内存空间的对象）。这样做的目的是避免在 Edn 区和两个 Survivor 区之间发生大量的内存拷贝（新生代采用复制算法收集内存）。
长期存活的对象进入老年代。虚拟机为每个对象定义了一个年龄计数器，如果对象经过了 1 次 Mior GC 那么对象会进入 Survivor 区，之后每经过一次 Minor GC 那么对象的年龄加 1，知道达到阀对象进入老年区。
动态判断对象的年龄。如果 Survivor 区中相同年龄的所有对象大小的总和大于 Survivor 空间的半，年龄大于或等于该年龄的对象可以直接进入老年代。
空间分配担保。每次进行 Minor GC 时，JVM 会计算 Survivor 区移至老年区的对象的平均大小如果这个值大于老年区的剩余值大小则进行一次 Full GC，如果小于检查 HandlePromotionFailure置，如果 true 则只进行 Monitor GC,如果 false 则进行 Full GC。

<blockquote>
如何通过参数来控制个各个内存区域
参考此文章：jvm 系列(二):JM 内存结构
</blockquote>
<h2 id="GC算法-垃圾回收">GC 算法 垃圾回收</h2>
主要关注点：

对象存活判断
GC 算法
垃圾回收器

对象存活判断
判断对象是否存活一般有两种方式：

引用计数：每个对象有一个引用计数属性，新增一个引用时计数加 1，引用释放时计数减 1，计为 0 时可以回收。此方法简单，无法解决对象相互循环引用的问题。
可达性分析（Reachability Analysis）：从 GC Roots 开始向下搜索，搜索所走过的路径称为引链。当一个对象到 GC Roots 没有任何引用链相连时，则证明此对象是不可用的，不可达对象。

GC 算法
GC 最基础的算法有三种：标记 -清除算法、复制算法、标记-压缩算法，我们常用的垃圾回收器般都采用分代收集算法。

标记 -清除算法，“标记-清除”（Mark-Sweep）算法，如它的名字一样，算法分为“标记”和清除”两个阶段：首先标记出所有需要回收的对象，在标记完成后统一回收掉所有被标记的对象。</l>
复制算法，“复制”（Copying）的收集算法，它将可用内存按容量划分为大小相等的两块，每只使用其中的一块。当这一块的内存用完了，就将还存活着的对象复制到另外一块上面，然后再把已用过的内存空间一次清理掉。
标记-压缩算法，标记过程仍然与“标记-清除”算法一样，但后续步骤不是直接对可回收对象进清理，而是让所有存活的对象都向一端移动，然后直接清理掉端边界以外的内存
分代收集算法，“分代收集”（Generational Collection）算法，把 Java 堆分为新生代和老年，这样就可以根据各个年代的特点采用最适当的收集算法。

垃圾回收器

Serial 收集器，串行收集器是最古老，最稳定以及效率高的收集器，可能会产生较长的停顿，只用一个线程去回收。
ParNew 收集器，ParNew 收集器其实就是 Serial 收集器的多线程版本。
Parallel 收集器，Parallel Scavenge 收集器类似 ParNew 收集器，Parallel 收集器更关注系统吞吐量。
Parallel Old 收集器，Parallel Old 是 Parallel Scavenge 收集器的老年代版本，使用多线程和标记 - 整理”算法
CMS 收集器，CMS（Concurrent Mark Sweep）收集器是一种以获取最短回收停顿时间为目标收集器。
G1 收集器，G1 (Garbage-First)是一款面向服务器的垃圾收集器,主要针对配备多颗处理器及大量内存的机器. 以极高概率满足 GC 停顿时间要求的同时,还具备高吞吐量性能特征

<blockquote>
GC 算法和垃圾回收器算法图解以及更详细内容参考 jvm 系列(三):GC 算法 垃圾收集器
</blockquote>
<h2 id="GC分析-命令调优">GC 分析 命令调优</h2>
主要关注点：

GC 日志分析
调优命令
调优工具

GC 日志分析
摘录 GC 日志一部分（前部分为年轻代 gc 回收；后部分为 full gc 回收）：
<pre><code class="highlight-chroma">2016-07-05T10:43:18.093+0800: 25.395: [GC [PSYoungGen: 274931K->10738K(274944)] 371093K->147186K(450048K), 0.0668480 secs] [Times: user=0.17 sys=0.08, real=0.07 sec]
2016-07-05T10:43:18.160+0800: 25.462: [Full GC [PSYoungGen: 10738K->0K(274944K)] [ParOldGen: 136447K->140379K(302592K)] 147186K->140379K(577536K) [PSPermGen: 85411K->85376K(11008K)], 0.6763541 secs] [Times: user=1.75 sys=0.02, real=0.68 secs]
</code></pre>
通过上面日志分析得出，PSYoungGen、ParOldGen、PSPermGen 属于 Parallel 收集器。其中 PSYoungGen 表示 gc 回收前后年轻代的内存变化；ParOldGen 表示 gc 回收前后老年代的内存变

；PSPermGen 表示 gc 回收前后永久区的内存变化。young gc 主要是针对年轻代进行内存回收比频繁，耗时短；full gc 会对整个堆内存进行回城，耗时长，因此一般尽量减少 full gc 的次数

调优命令
Sun JDK 监控和故障处理命令有 jps jstat jmap jhat jstack jinfo

jps，JVM Process Status Tool,显示指定系统内所有的 HotSpot 虚拟机进程。
jstat，JVM statistics Monitoring 是用于监视虚拟机运行时状态信息的命令，它可以显示出虚机进程中的类装载、内存、垃圾收集、JIT 编译等运行数据。
jmap，JVM Memory Map 命令用于生成 heap dump 文件
jhat，JVM Heap Analysis Tool 命令是与 jmap 搭配使用，用来分析 jmap 生成的 dump，jhat 内置了一个微型的 HTTP/HTML 服务器，生成 dump 的分析结果后，可以在浏览器中查看
jstack，用于生成 java 虚拟机当前时刻的线程快照。
jinfo，JVM Configuration info 这个命令作用是实时查看和调整虚拟机运行参数。

<blockquote>
详细的命令使用参考这里 jvm 系列(四):jvm 调优-命令篇
</blockquote>
调优工具
常用调优工具分为两类,jdk 自带监控工具：jconsole 和 jvisualvm，第三方有：MAT(Memory Analyzer Tool)、GChisto。

jconsole，Java Monitoring and Management Console 是从 java5 开始，在 JDK 中自带的 java 监控和管理控制台，用于对 JVM 中内存，线程和类等的监控
jvisualvm，jdk 自带全能工具，可以分析内存快照、线程快照；监控内存变化、GC 变化等。

MAT，Memory Analyzer Tool，一个基于 Eclipse 的内存分析工具，是一个快速、功能丰富的 Java heap 分析工具，它可以帮助我们查找内存泄漏和减少内存消耗
GChisto，一款专业分析 gc 日志的工具

<blockquote>
工具使用参考 jvm 系列(七):jvm 调优-工具篇
</blockquote>