JVM相关的常见面试问题汇总：运筹策帷帐之中，决胜于千里之外

JVM相关的常见面试问题汇总：运筹策帷帐之中，决胜于千里之外

面试和笔试的要点其实差不多，基础知识和实战经验都是最重要的关注点（当然，面试时的态度和眼缘也很重要）。

实际面试时，因为时间有限，不可能所有问题都问一遍，一般是根据简历上涉及的内容，抽一部分话题来聊一聊。看看面试者的经验，态度，以及面对一层层的深入问题时的处理思路。借此了解面试者的技术水平，对深度、广度，以及思考和解决问题的能力。

常见的面试套路是什么呢?

- XXX是什么?
- 实现原理是什么?
- 为什么这样实现?
- 如果让你实现你会怎么做?
- 分析下你的实现有什么优缺点?

- 有哪些需要改进的地方?

下面总结一些比较常见的面试题，供大家参考。
针对这些问题，大家可以给自己打一个分。

- 0分：不清楚相关知识。
- 30分：有一点印象，知道一些名词。
- 60分：知道一些概念以及含义，了解功能和常见用途。
- 80分：能在参考答案的基础上进行补充。
- 100分：发现参考答案的问题。

下面我们来看看JVM相关面试问题。

1. 什么是JVM?

JVM全称是`Java Virtual Machine`，中文称为`Java虚拟机`。
JVM是Java程序运行的底层平台，与Java支持库一起构成了Java程序的执行环境。
分为JVM规范和JVM实现两个部分。简单来说，Java虚拟机就是指能执行标准Java字节码的虚拟计算机。

1.1 请问JDK与JVM有什么区别?

现在的JDK、JRE和JVM一般是整套出现的。
JDK = JRE + 开发调试诊断工具
JRE = JVM + Java标准库

1.2 你认识哪些JVM厂商?

常见的JDK厂商包括:

- Oracle公司，包括 Hotspot虚拟机、GraalVM；分为OpenJDK和OracleJDK两种版本。
- IBM 公司，J9虚拟机， 用在IBM的产品套件中
- Azul Systems公司，高性能的Zing和开源的Zulu
- 阿里巴巴，Dragonwell 是阿里开发的OpenJDK定制版
- 亚马逊，Corretto OpenJDK
- Red Hat公司的OpenJDK

- Adopt OpenJDK
- 此外，还有一些开源和试验性质的JVM实现，比如Go.JVM

1.3 OracleJDK与OpenJDK有什么区别?

各种版本的JDK一般来说都会符合Java虚拟机规范。
两者的区别一般来说包括：

- 两种JDK提供的工具套件略有差别，比如jmc等有版权的工具。
- 某些协议或配置不一样，比如美国限制出口的加密算法。
- 其他细微差别，比如JRE中某些私有的API不一样。

1.4 开发中使用哪个版本的JDK？生产环境呢？为什么这么选?

有一说一。选择哪个版本需要考虑研发团队的具体情况：比如机器的操作系统，团队成员的掌握情况，兼顾遗留项目等等。
当前Java最受欢迎的长期维护版本是Java8和Java11。

- Java8是经典LTS版本，性能优秀，系统稳定，良好支持各种CPU架构和操作系统平台。
- Java11是新的长期支持版，性能更强，支持更多新特性，而且经过几年的维护已经很稳定。

有的企业在开发环境使用OracleJDK，在生产环境使用OpenJDK。
也有的企业恰好相反，在开发环境使用OpenJDK，在生产环境使用OracleJDK。
也有的公司使用同样的打包版本。

开发和部署时只要进行过测试就没问题。
一般来说。 测试环境、预上线环境的JDK配置需要和生产环境一致。

2. 什么是Java字节码?

Java 中的字节码，是值 Java 源代码编译后的中间代码格式，一般称为字节码文件。

2.1 字节码文件中包含哪些内容？

字节码文件中，一般包含以下部分：

- 版本号信息

- 静态常量池（符号常量）
- 类相关的信息
- 字段相关的信息
- 方法相关的信息
- 调试相关的信息

可以说，大部分信息都是通过常量池中的符号常量来表述的。

2.2 什么是常量?

常量是指不变的量，字母 `'K'` 或者数字 `1024` 在UTF8编码中对应到对应的二进制格式都是不变的。同样地，字符串在Java中的二进制表示也是不变的, 比如 `"KK"` 。在Java中需要注意的是， `final` 关键字修饰的字段和变量，表示最终变量，只能赋值1次，不允许再次修改，由编译器和执行引擎共同保证。

2.3 你怎么理解常量池?

在Java中，常量池包括两层含义：

- 静态常量池，class文件中的一个部分，里面保存的是类相关的各种符号常量。
- 运行时常量池， 其内容主要由静态常量池解析得到，但也可以由程序添加。

3. JVM的运行时数据区有哪些?

根据 JVM规范，标准的JVM运行时数据区包括以下部分：

- 程序计数器
- Java虚拟机栈
- 堆内存
- 方法区
- 运行时常量池
- 本地方法栈

具体的JVM实现可根据实际情况进行优化或者合并，满足规范的要求即可。

3.1 什么是堆内存?

堆内存是指由程序代码自由分配的内存，与栈内存作区分。

在Java中，堆内存主要用于分配对象的存储空间，只要拿到对象引用，所有线程都可以访问堆内存。

3.2 堆内存包括哪些部分?

以Hotspot为例，堆内存（HEAP）主要由GC模块进行分配和管理， 可分为以下部分：

- 新生代
- 存活区
- 老年代

其中，新生代和存活区一般称为年轻代。

3.3 什么是非堆内存?

除堆内存之外，JVM的内存池还包括非堆（NON_HEAP），对应于JVM规范中的方法区，常量池等部分：

- MetaSpace
- CodeCache
- Compressed Class Space

4. 什么是内存溢出?

内存溢出（OOM）是指可用内存不足。
程序运行需要使用的内存超出最大可用值，如果不进行处理就会影响到其他进程，所以现在操作系统的处理办法是：只要超出立即报错，比如抛出 `内存溢出错误` 。

就像杯子装不下，满了要溢出来一样，比如一个杯子只有500ml的容量，却倒进去600ml，于是水就溢出造成破坏。

4.1 什么是内存泄漏?

内存泄漏（Memory Leak）是指本来无用的对象却继续占用内存，没有再恰当的时机释放占用的内存。
不使用的内存，却没有被释放，称为 `内存泄漏` 。 也就是该释放的没释放，该回收的没回收。
比较典型的场景是： 每一个请求进来，或者每一次操作处理，都分配了内存，却有一

部分不能回收（或未释放），那么随着处理的请求越来越多，内存泄漏也就越来越严重。

在Java中一般是指无用的对象却因为错误的引用关系，不能被GC回收清理。

4.2 两者有什么关系?

如果存在严重的内存泄漏问题，随着时间的推移，则必然会引起内存溢出。
内存泄漏一般是资源管理问题和程序BUG，内存溢出则是内存空间不足和内存泄漏的最终结果。

5. 给定一个具体的类，请分析对象的内存占用

```
public class MyOrder{
  private long orderId;
  private long userId;
  private byte state;
  private long createMillis;
}
```


一般来说，MyOrder 类的每个对象会占用40个字节。

5.1 怎么计算出来的?

计算方式为：

- 对象头占用12字节。
- 每个long类型的字段占用8字节，3个long字段占用24字节。
- byte 字段占用1个字节。
- 以上合计 37字节，加上以8字节对齐，则实际占用40个字节。

5.2 对象头中包含哪些部分?

对象头中一般包含两个部分：

- 标记字，占用一个机器字，也就是8字节。
- 类型指针，占用一个机器字，也就是8个字节。

- 如果堆内存小于32GB，JVM默认会开启指针压缩，则只占用4个字节。

所以前面的计算中，对象头占用12字节。
如果是数组，对象头中还会多出一个部分：

- 数组长度， int值，占用4字节。

6. 常用的JVM启动参数有哪些？

截止目前（2020年3月），JVM可配置参数已经达到1000多个，其中GC和内存配置相关的JVM参数就有600多个。
但在绝大部分业务场景下，常用的JVM配置参数也就10来个。
例如：

```
# JVM启动参数不换行
# 设置堆内存
-Xmx4g -Xms4g
# 指定GC算法
-XX:+UseG1GC -XX:MaxGCPauseMillis=50
# 指定GC并行线程数
-XX:ParallelGCThreads=4
# 打印GC日志
-XX:+PrintGCDetails -XX:+PrintGCDateStamps
# 指定GC日志文件
-Xloggc:gc.log
# 指定Meta区的最大值
-XX:MaxMetaspaceSize=2g
# 设置单个线程栈的大小
-Xss1m
# 指定堆内存溢出时自动进行Dump
-XX:+HeapDumpOnOutOfMemoryError
-XX:HeapDumpPath=/usr/local/
```


此外，还有一些常用的属性配置：

```
# 指定默认的连接超时时间
-Dsun.net.client.defaultConnectTimeout=2000
```

```
-Dsun.net.client.defaultReadTimeout=2000
# 指定时区
-Duser.timezone=GMT+08
# 设置默认的文件编码为UTF-8
-Dfile.encoding=UTF-8
# 指定随机数熵源(Entropy Source)
-Djava.security.egd=file:/dev/./urandom
```

6.1 设置堆内存XMX应该考虑哪些因素?

需要根据系统的配置来确定，要给操作系统和JVM本身留下一定的剩余空间。
推荐配置系统或容器里可用内存的 70-80% 最好。

6.2 假设物理内存是8G，设置多大堆内存比较合适?

比如说系统有 8G 物理内存，系统自己可能会用掉一点，大概还有 7.5G 可以用，那么建议配置 `-Xmx6g` 。

- 说明： `7.5G*0.8 = 6G` ，如果知道系统里有明确使用堆外内存的地方，还需要进一步降低这个值。

6.3 `-Xmx` 设置的值与JVM进程所占用的内存有什么关系?

JVM总内存=栈+堆+非堆+堆外+Native

6.4 怎样开启GC日志?

一般来说，JDK8及以下版本通过以下参数来开启GC日志：

```
-XX:+PrintGCDetails -XX:+PrintGCDateStamps -Xloggc:gc.log
```

如果是在JDK9及以上的版本，则格式略有不同：

```
-Xlog:gc*=info:file=gc.log:time:filecount=0
```

6.5 请指定使用G1垃圾收集器来启动 `Hello` 程序

```
java -XX:+UseG1GC
-Xms4g
-Xmx4g
-Xloggc:gc.log
-XX:+PrintGCDetails
-XX:+PrintGCDateStamps
Hello
```

7. Java8默认使用的垃圾收集器是什么?

Java8版本的Hotspot JVM，默认情况下使用的是并行垃圾收集器（Parallel GC）。其他厂商提供的JDK8基本上也默认使用并行垃圾收集器。

7.1 Java11的默认垃圾收集器是什么?

Java9之后，官方JDK默认使用的垃圾收集器是G1。

7.2 常见的垃圾收集器有哪些?

常见的垃圾收集器包括:

- 串行垃圾收集器: `-XX:+UseSerialGC`
- 并行垃圾收集器: `-XX:+UseParallelGC`
- CMS垃圾收集器: `-XX:+UseConcMarkSweepGC`
- G1垃圾收集器: `-XX:+UseG1GC`

7.3 什么是串行垃圾收集?

就是只有单个worker线程来执行GC工作。

7.4 什么是并行垃圾收集?

并行垃圾收集，是指使用多个GC worker 线程并行地执行垃圾收集，能充分利用多核CPU的能力，缩短垃圾收集的暂停时间。
除了单线程的GC，其他的垃圾收集器，比如 PS，CMS， G1等新的垃圾收集器都使

用了多个线程来并行执行GC工作。

7.5 什么是并发垃圾收集器?

并发垃圾收集器，是指在应用程序在正常执行时，有一部分GC任务，由GC线程在应用线程一起并发执行。
例如 CMS/G1的各种并发阶段。

7.6 什么是增量式垃圾收集?

首先， G1的堆内存不再单纯划分为年轻代和老年代，而是划分为多个（通常是 2048 个）可以存放对象的小块堆区域（smaller heap regions）。
每个小块，可能一会被定义成 Eden 区，一会被指定为 Survivor 区或者 Old 区。
这样划分之后，使得 G1 不必每次都去回收整个堆空间，而是以增量的方式来进行处理： 每次只处理一部分内存块，称为此次 GC 的回收集（collection set）。
下一次GC时在本次的基础上，再选定一定的区域来进行回收。增量式垃圾收集的好处是大大降低了单次GC暂停的时间。

7.7 什么是年轻代?

年轻代是分来垃圾收集算法中的一个概念，相对于老年代而言，年轻代一般包括：

- 新生代，Eden区。
- 存活区，执行年轻代GC时，用存活区来保存活下来的对象。 存活区也是年轻代的一部分，但一般有2个存活区，所以可以来回倒腾。

7.8 什么是GC停顿(GC pause)?

因为GC过程中，有一部分操作需要等所有应用线程都到达安全点，暂停之后才能执行，这时候就叫做GC停顿，或者叫做GC暂停。

7.9 GC停顿与STW停顿有什么区别?

这两者一般可以认为就是同一个意思。

8. 如果CPU使用率突然飙升，你会怎么排查?

缺乏经验的话，针对当前问题，往往需要使用不同的工具来收集信息，例如：

- 收集不同的指标（CPU，内存，磁盘IO，网络等等）
- 分析应用日志
- 分析GC日志
- 获取线程转储并分析
- 获取堆转储来进行分析

8.1 如果系统响应变慢，你会怎么排查?

一般根据APM监控来排查应用系统本身的问题。
有时候也可以使用Chrome浏览器等工具来排查外部原因，比如网络问题。

8.2 系统性能一般怎么衡量?

可量化的3个性能指标：

- 系统容量：比如硬件配置，设计容量；
- 吞吐量：最直观的指标是TPS；
- 响应时间：也就是系统延迟，包括服务端延时和网络延迟。

这些指标。可以具体拓展到单机并发，总体并发，数据量，用户数，预算成本等等。

9. 使用过哪些JVM相关的工具?

这个问题请根据实际情况回答，比如Linux命令，或者JDK提供的工具等。

9.1 查看JVM进程号的命令是什么?

可以使用 `ps -ef` 和 `jps -v` 等等。

9.2 怎么查看剩余内存?

比如： `free -m`，`free -h`，`top` 命令等等。

9.3 查看线程栈的工具是什么?

一般先使用 jps命令， 再使用 `jstack -l`

9.4 用什么工具来获取堆内存转储?

一般使用 jmap 工具来获取堆内存快照。

9.5 内存Dump时有哪些注意事项?

根据实际情况来看，获取内存快照可能会让系统暂停或阻塞一段时间，根据内存量决定。
使用jmap时，如果指定 `live` 参数，则会触发一次FullGC，需要注意。

9.6 使用JMAP转储堆内存大致的参数怎么处理?

示例:

```
jmap -dump:format=b,file=3826.hprof 3826
```

9.7 为什么转储文件以 `.hprof` 结尾?

JVM有一个内置的分析器叫做HPROF, 堆内存转储文件的格式，最早就是这款工具定义的。

9.8 内存Dump完成之后，用什么工具来分析?

一般使用 Eclipse MAT工具，或者 jhat 工具来处理。

9.9 如果忘记了使用什么参数你一般怎么处理?

上网搜索是比较笨的办法，但也是一种办法。
另外就是，各种JDK工具都支持 `-h` 选项来查看帮助信息，只要用得比较熟练，即使忘记了也很容易根据提示进行操作。

10. 开发性问题：你碰到过哪些JVM问题?

比如GC问题、内存泄漏问题、或者其他疑难杂症等等。
然后可能还有一些后续的问题。例如:

- 你遇到过的印象最深的JVM问题是什么?
- 这个问题是怎么分析和解决的?
- 这个过程中有哪些值得分享的经验?

此问题为开放性问题，请根据自身情况进行回答，可以把自己思考的答案发到本课程的微信群里，我们会逐个进行分析点评。