**中软国际考题-笔试**

1. 单选题

1、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区内，然后在堆区创建如下哪一个对象（D）

A.java.lang.Object B.42 C.java.lang.Thread D. java.lang.Class

2、什么是强引用（A）

A．只要引用存在，垃圾回收器永远不会回收。

B．非必须引用，内存溢出之前进行回收，可以通过以下代码实现

C．监控对象是否已经被标记为即将回收的垃圾，可以通过弱引用的isEnQueues方法

返回对象是否被垃圾回收器标记。

D垃圾回收时回收，无法通过引用取到对象值

3、下列哪一个可以实现Java程序的[跨平台](https://www.baidu.com/s?wd=%E8%B7%A8%E5%B9%B3%E5%8F%B0&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)运行。 （C）  
A. JDK  B. JRE  C. JVM D.JS

4、JVM是什么的缩写（A）

A.Java Virtual Machine B.Java View Model

C.Java View Machine D.Java Virtual Model

5、JVM的配置参数分类，说法不正确的是（D）

A. 跟踪参数 B 堆分配参数 C 栈分配参数 D 拥有者参数

6、下列对JVM中的复制算法的描述错误是（D）

将内存平均分成A、B两块，算法过程：

A.新生对象被分配到A块中未使用的内存当中。当A块的内存用完了， 把A块的存活对象对象复制到B块。

B.清理A块所有对象

C.新生对象被分配的B块中未使用的内存当中。当B块的内存用完了， 把B块的存活对象对象复制到A块

D.返回对A块的处理

7、下面关于JVM新生代堆空间大小参数配置说明错误的是 （A）

A.-XX: MaxPermSize 新生代大小调整。

B.-XX: NewRatio设置新生代与老年代在堆空间的大小

C.-XX: NewSize新生代的最小值

D.-XX: SurvivorRatio新生代中Eden所占区域的大小

8、以下不是JVM垃圾收集器的是（D）

A. 串行收集器 B. 并行收集器 C. CMS收集器 D.栈收集器

9、下列哪个是类加载器（D）

A. 启动类加载器 B. 扩展类加载器C. 系统类加载器 D.以上都是

10、 关于JVM的方法区，说法错误的是（A）

A、有时候也成为永久代，在该区内经常发生垃圾回收，但是并不代表不发生GC，在这里进行的GC主要是对方法区里的常量池和对类型的卸载

B、方法区主要用来存储已被虚拟机加载的类的信息、常量、静态变量和即时编译器编译后的代码等数据。

C、该区域是被线程共享的。

D、方法区里有一个运行时常量池，用于存放静态编译产生的字面量和符号引用。该常量池具有动态性，也就是说常量并不一定是编译时确定，运行时生成的常量也会存在这个常量池中。

11、编译 java application源程序文件将产生相应的字节码文件，这些字节码文件的扩展名为（B）

A、java B、.class c.html D.exe

12、**( C )可以实现Java程序的**[跨平台](https://www.baidu.com/s?wd=%E8%B7%A8%E5%B9%B3%E5%8F%B0&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)**运行。**   
 A.JDK  B.JRE  C.JVM

1. 多选题
2. 如下选项中哪些属于类的生命周期？（A、B、C、D、E）
3. 加载 B. 初始化 C. 连接

D. 使用 E. 卸载

1. 如下选项中哪些不属于JVM的内存结构？（A、B、C、D、E）
2. 堆（Heap） B. JVM栈（JVM Stacks） C. 方法区（Method Area）

D. 本地方法栈（Native Method Stacks） E. 程序计数器（Program Counter Register）

3、Jvm中内存分区有(C、D、E)

A.开发区 B.缓存区 C.堆区 D.栈区 E.方法区

4、下列关于JDK、JRE和JVM关系的描述中,不正确的是？（A,B,C）

A. JDK中包含了JRE,JVM中包含了JRE B. JRE中包含了JDK，JDK中包含了JVM

C. JRE中包含了JDK，JVM中包含了JRE D. JDK中包含了JRE,JRE中包含了JVM

5、下面哪些项说法是正确的？（A、BC）

A. JVM是一种用于计算设备的规范

B. 它是一个虚构出来的计算机

C. 是通过在实际的计算机上仿真模拟各种计算机功能来实现的

D. 可以运行所有编程语言

6、以下哪些是JVM监控查看工具？（A、B）

A.jstack B. jvisualvm C. jsoup D. jrockit

7、下面关于Java对象的创建过程描述正确的是（A,B,C）

A.类加载检查。 B.分配内存 C.执行init方法 D.引用计数

8、GC的两种判定方法：（A、B）

A.引用计数法 B.引用链法 C.标记清除法 D.标记整理法

E.复制算法

9、JVM内存分配的区域有哪些？（A、B、C）

A. 虚拟机栈 B. 堆 C.方法区

10、Java类加载器包括（ ABCD ）

A . 启动类加载器 B . 扩展类加载器

C . 系统类加载器 D . 自定义类加载器

11、JVM数据区域中，线程共享的部分包括（ ABC ）

A . 堆 B . 方法区 C . 直接内存 D . 本地方法栈

1. 填空题（每空1分，共10分）
2. 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)
3. Java类加载器包括：\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_\_。(启动Bootstrap类加载、扩展Extension类加载、系统System类加载)
4. java虚拟机所管理的内存中最大的一块内存区域：\_\_\_\_\_\_\_\_\_\_ 。
5. JVM的一个非常重要的特点就是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。(与平台无关性)
6. 堆内存设置的参数 -Xmx 设置堆的最大空间大小 -Xms 设置堆的最小空间大小
7. GC的两种判定方法：\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_。（引用计数、引用链）
8. 根据JVM规范，JVM内存共分为\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_、方法区、程序计数器、本地房发栈五个部分。(虚拟机栈、堆)
9. JVM在整个jdk中处于最底层,负责与操作系统的交互,用来屏蔽操作系统环境,提供一个完整的Java运行环境,因此也叫虚拟计算机.操作系统装入JVM是通过jdk中

来完成 。（java.exe）

1. JVM中堆空间可以分成三个大区，（ **新生代 ）、（ 老年代 ）、（ 永久代 ）**
2. 判断题
3. Java堆（Heap）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。（ T ）
4. 虚拟机栈动态扩展，当扩展无法申请到足够的内存空间时候，抛出StackOverFlowError异常（ F ）
5. 垃圾收集器中串行收集器是多线程，并行收集器是单线程（F）
6. Eden和Survivor的比例分配默认为8:1（T）
7. java虚拟机栈是线程私有，生命周期与线程相同。（ T ）
8. 虚拟机执行java程序的时候，每个方法都会创建一个栈帧，栈帧存放在java虚拟机堆中，通过压栈出栈的方式进行方法调用。（ F ）
9. java的8中基本类型的局部变量的值存放在虚拟机栈的局部变量表中，如果是引用型的变量，则只存储对象的引用地址。（ T ）
10. 本地方法栈为虚拟机使用到本地方法服务（native）。（ T ）
11. 本地方法栈不为线程私有。（ F ）
12. 程序[计数器](https://www.baidu.com/s?wd=%E8%AE%A1%E6%95%B0%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)，虚拟机栈，native栈是线程私有的，堆是线程共有的。（ T ）
13. 栈是最小的一块内存区域。（ F ）
14. 局部变量表: 存放了编译器可知的各种基本数据类型、对象引用。（ T ）
15. JVM是开发JAVA的平台。（ F ）
16. 调整JVM的配置参数是JVM调优的重要手段（ T ）
17. 类加载的过程分为加载、验证、准备、解析、初始化五个阶段（ T ）
18. 程序计数器能够实现循环、分支、异常处理、线程恢复等操作。（ T ）
19. JVM是一种用于计算设备的规范，它是一个虚构出来的计算机，是通过在实际的计算机上仿真模拟各种计算机功能来实现的。（ T ）
20. JVM只能在windows系统安装。（ F）
21. Java 虚拟机是一个可以执行 Java 字节码的虚拟机进程。Java 源文件被编译成能被 Java 虚拟机执行的字节码文件。( √ )
22. Java 被设计成允许应用程序可以运行在任意的平台，而不需要程序员为每一个平台单独重写或者是重新编译。( √ )
23. JVM在运行时数据区域分为线程私有的和线程共享的两部分（ √ ）

五、简答题

1、请简述在GC中，判断对象是否存活的方式是什么？

答案：

（1）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，计数为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。

（2）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，则证明此对象是不可用的，不可达对象。

2、请简述java中类的加载机制？

答案：

（1）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也将由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

（2）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路径中加载该类

（3）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，并将其转换成Class对象，存入缓存区。这就是为什么修改了Class后，必须重启JVM，程序的修改才会生效

3、类的实例化顺序，比如父类静态数据，构造函数，字段，子类静态数据，构造函数，字段，它们的执行顺序

答案：先静态、先父后子。

先静态：父静态 > 子静态

优先级：父类 > 子类 静态代码块 > 非静态代码块 > 构造函数

一个类的实例化过程：

1，父类中的static代码块，当前类的static

2，顺序执行父类的普通代码块

3，父类的构造函数

4，子类普通代码块

5，子类（当前类）的构造函数，按顺序执行。

6，子类方法的执行

4、JVM垃圾回收机制，何时触发MinorGC等操作

分代垃圾回收机制：不同的对象生命周期不同。把不同生命周期的对象放在不同代上，不同代上采用最合适它的垃圾回收方式进行回收。

JVM中共划分为三个代：年轻代、年老代和持久代，

年轻代：存放所有新生成的对象；

年老代：在年轻代中经历了N次垃圾回收仍然存活的对象，将被放到年老代中，故都是一些生命周期较长的对象；

持久代：用于存放静态文件，如Java类、方法等。

新生代的垃圾收集器命名为“minor gc”，老生代的GC命名为”Full Gc 或者Major GC”.其中用System.gc()强制执行的是Full Gc.

判断对象是否需要回收的方法有两种：

1.引用计数

当某对象的引用数为0时，便可以进行垃圾收集。

2.对象引用遍历

果某对象不能从这些根对象的一个（至少一个）到达，则将它作为垃圾收集。在对象遍历阶段，gc必须记住哪些对象可以到达，以便删除不可到达的对象，这称为标记（marking）对象。

触发GC（Garbage Collector）的条件：

1)GC在优先级最低的线程中运行，一般在应用程序空闲即没有应用线程在运行时被调用。

2)Java堆内存不足时，GC会被调用。

5、jdk、jre、jvm是什么关系？（10分）

答案：

（1）JRE(Java Runtime Environment)，也就是java平台。所有的java程序都要在JRE环境下才能运行。

（2）JDK(Java Development Kit)，是开发者用来编译、调试程序用的开发包。JDK也是JAVA程序需要在JRE上运行。

（3）JVM(Java Virtual Machine)，是JRE的一部分。它是一个虚构出来的计算机，是通过在实际的计算机上仿真模拟各种计算机功能来实现的。

JVM有自己完善的硬件架构，如处理器、堆栈、寄存器等，还具有相应的指令系统。

Java语言最重要的特点就是跨平台运行。使用JVM就是为了支持与操作系统无关，实现跨平台。

6、写出JVM的内存模型?（10分）

答案：程序计数器、java虚拟机栈、本地方法栈、堆、方法区。

7、**JVM的生命周期**（10分）

答案：

1. 启动。启动一个Java程序时，一个JVM实例就产生了，任何一个拥有public static void   
   main(String[] args)函数的class都可以作为JVM实例运行的起点   
   b) 运行。main()作为该程序初始线程的起点，任何其他线程均由该线程启动。JVM内部有两种线程：守护线程和非守护线程，main()属于非守护线程，守护线程通常由JVM自己使用，java程序也可以表明自己创建的线程是守护线程   
   c) 消亡。当程序中的所有非守护线程都终止时，JVM才退出；若安全管理器允许，程序也可以使用Runtime类或者System.exit()来退出

8、简述JVM与Java跨平台特性的关系（5分）

答案：JVM全名为Java Virtual Machine，java虚拟机。每个特定平台下都有特定版本的JVM，如Linux平台下有Linux版本的JVM。不同平台下的JVM负责将class文件解释成特定平台下的机器码，进而实现了跨平台。

9、请说明JVM的栈区有什么特点。

答案：栈分为java虚拟机栈和本地方法栈。重点是Java虚拟机栈，它是线程私有的，生命周期与线程相同。每个方法执行都会创建一个栈帧，用于存放局部变量表，操作栈，动态链接，方法出口等。每个方法从被调用，直到被执行完。对应着一个栈帧在虚拟机中从入栈到出栈的过程。

通常说的栈就是指局部变量表部分，存放编译期间可知的8种基本数据类型，及对象引用和指令地址。局部变量表是在编译期间完成分配，当进入一个方法时，这个栈中的局部变量分配内存大小是确定的。

会有两种异常StackOverFlowError和 OutOfMemoneyError。当线程请求栈深度大于虚拟机所允许的深度就会抛出StackOverFlowError错误；虚拟机栈动态扩展，当扩展无法申请到足够的内存空间时候，抛出OutOfMemoneyError。

10、请说明JVM的方法区有什么特点。

答案：被所有线程共享区域，用于存放已被虚拟机加载的类信息，常量，静态变量等数据。被Java虚拟机描述为堆的一个逻辑部分。习惯是也叫它永久代（permanment generation）

垃圾回收很少光顾这个区域，不过也是需要回收的，主要针对常量池回收，类型卸载。

常量池用于存放编译期生成的各种字节码和符号引用，常量池具有一定的动态性，里面可以存放编译期生成的常量；运行期间的常量也可以添加进入常量池中，比如string的intern()方法。

11、请说明JVM的堆区有什么特点。

答案：堆被所有线程共享区域，在虚拟机启动时创建，唯一目的存放对象实例。堆区是gc的主要区域，通常情况下分为两个区块年轻代和年老代。更细一点年轻代又分为Eden区最要放新创建对象，From survivor 和 To survivor 保存gc后幸存下的对象，默认情况下各自占比 8:1:1。

12、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。

13、JVM执行程序的过程 ?

答案 ：（1）、加载.class文件

（2）、管理并分配内存

（3）、执行垃圾收集

14、对于对象引用， GC会采用不同的方法进行回收，JVM对象的引用分为了几种种类型？

答案：（1）强引用：默认情况下，对象采用的均为强引用（这个对象的实例没有其他对象引用，GC时才会被回收）

（2）软引用：软引用是Java中提供的一种比较适合于缓存场景的应用（只有在内存不够用的情况下才会被GC）

（3）弱引用：在GC时一定会被GC回收

（4）虚引用：由于虚引用只是用来得知对象是否被GC

15、Java 虚拟机是如何判定两个 Java 类是相同的?

1、Java 虚拟机不仅要看类的全名是否相同，还要看加载此类的类加载器是否一样。

2、只有两者都相同的情况，才认为两个类是相同的。即便是同样的字节代码，被不同的类加载器加载之后所得到的类，也是不同的。

16、自定义类加载的意义：

1、加载特定路径的class文件

2、加载一个加密的网络class文件

3、热部署加载class文件

17、堆内存设置的参数是什么？

1、-Xmx 设置堆的最大空间大小

2、-Xms 设置堆的最小空间大小

18、Java内存堆和栈区别

1、栈内存用来存储基本类型的变量和对象的引用变量，堆内存用来存储Java中的对象，无论是成员变量，局部变量，还是类变量，它们指向的对象都存储在堆内存中

2、栈内存归属于单个线程，每个线程都会有一个栈内存，其存储的变量只能在其所属线程中可见，即栈内存可以理解成线程的私有内存，堆内存中的对象对所有线程可见。堆内存中的对象可以被所有线程访问

3、如果栈内存没有可用的空间存储方法调用和局部变量，JVM会抛出java.lang.StackOverFlowError，如果是堆内存没有可用的空间存储生成的对象，JVM会抛 出java.lang.OutOfMemoryError

4、栈的内存要远远小于堆内存，如果你使用递归的话，那么你的栈很快就会充满，-Xss选项设置栈内存的大小。-Xms选项可以设置堆的开始时的大小

19、jvm性能调优都做了什么

1、控制GC的行为.GC是一个后台处理,但是它也是会消耗系统性能的,因此经常会根据系统运行的程序的特性来更改GC行为

2、控制JVM堆栈大小.一般来说,JVM在内存分配上不需要你修改,(举例)但是当你的程序新生代对象在某个时间段产生的比较多的时候,就需要控制新生代的堆大小.同时,还要需要控制总的JVM大小避免内存溢出

3、控制JVM线程的内存分配.如果是多线程程序,产生线程和线程运行所消耗的内存也是可以控制的,需要通过一定时间的观测后,配置最优结果

20、介绍JVM中5个区域

1、程序计数器

2、java虚拟机栈

3、本地方法栈

4、Java堆

5、方法区

六、程序实现题

1、Integer 缓存源代码：（10分）

|  |
| --- |
| /\*\*  \*此方法将始终缓存-128到127（包括端点）范围内的值，并可以缓存此范围之外的其他值。  \*/  public static Integer valueOf(int i) {  if (i >= IntegerCache.low && i <= IntegerCache.high)  return IntegerCache.cache[i + (-IntegerCache.low)];  return new Integer(i);  } |

2、**String 对象的两种创建方式?**（10分）

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

String str1 = "abcd";

String str2 = **new** String("abcd");

}

}

3、编码产生StackOverflowError和OutOfMemoryError

（10分）

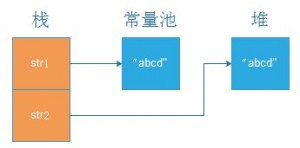
|  |
| --- |
| package pri.liyang.joke;  /\*\*  \* 让Java报StackOverFlow、OutOfMemory错误的简单代码实现  \* Author：李小白  \*/  public class ErrorGenerate {  /\*\*  \* 执行后会报StackOverflowError错误的方法  \*/  public void SOF(){  SOF();  }  /\*\*  \* 执行后会报OutOfMemoryError错误的方法  \*/  public void OOM(){  String str = "OOM";  for (int i = 0; i < Integer.MAX\_VALUE; i++) {  str = str + str;  }  }  /\*\*  \* 测试  \*/  public static void main(String[] args) {  //执行该方法，报错：StackOverflowError  new ErrorGenerate().SOF();  //执行该方法，报错：OutOfMemoryError  new ErrorGenerate().OOM();  }  } |

4、对于String对象有如下两种创建方法

1） String str1 = “abcd”;

2) String str2 = new String(“abcd”);

第一种方式是在常量池中拿对象，第二种方式直接在堆空间拿对象，如下所示：



请按照以上内存指向模型，完成以下代码的内存指向，并写出打印结果：

String str1 = “str”;

String str2 = “ing”;

String str3 = “str”+”ing”;

String str4 = str1 + str2;

String str5 = “string”;

System.out.println(str3 == str4);

System.out.println(str3 == str5);

System.out.println(str4 == str5);

|  |
| --- |
| 内存指向模型为：  IMG_256  输出结果为：false  true  false |

# 5、JVM锁机制：使用锁，维护计数器的串行访问与安全性

|  |
| --- |
| import java.util.ArrayList;  import java.util.List;    public class TestAddToList implements Runnable{  public static List<Integer> numberList = new ArrayList<Integer>();  int startNum = 0;  public TestAddToList(int startNum){  this.startNum = startNum;  }  @Override  public void run(){  int count = 0;  while(count < 1000000){  numberList.add(startNum);  startNum += 2;  count++;  }  }  public static void main(String[] args) throws Exception{  Thread t1 = new Thread(new TestAddToList(0));  Thread t2 = new Thread(new TestAddToList(1));  t1.start();  t2.start();  while(t1.isAlive()||t2.isAlive()){  Thread.sleep(1);  }  System.out.println(numberList.size());  }  } import java.util.ArrayList;  import java.util.List;    public class TestAddToList implements Runnable{  public static List<Integer> numberList = new ArrayList<Integer>();  int startNum = 0;  public TestAddToList(int startNum){  this.startNum = startNum;  }  @Override  public void run(){  int count = 0;  while(count < 1000000){  numberList.add(startNum);  startNum += 2;  count++;  }  }  public static void main(String[] args) throws Exception{  Thread t1 = new Thread(new TestAddToList(0));  Thread t2 = new Thread(new TestAddToList(1));  t1.start();  t2.start();  while(t1.isAlive()||t2.isAlive()){  Thread.sleep(1);  }  System.out.println(numberList.size());  }  } |