



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

## 第三版：JVM 18 道

### 详细介绍一下 JVM 内存模型

根据 JVM 规范，JVM 内存共分为虚拟机栈、堆、方法区、程序计数器、本地方法栈五个部分。

具体可能会聊聊 jdk1.7 以前的 PermGen（永久代），替换成 Metaspace（元空间）。

- 1、原本永久代存储的数据：符号引用(Symbols)转移到了 native heap；字面量(interned strings)转移到了 java heap；类的静态变量(class statics)转移到了 java heap
- 2、Metaspace（元空间）存储的是类的元数据信息（metadata）
- 3、元空间的本质和永久代类似，都是对 JVM 规范中方法区的实现。不过元空间与永久代之间最大的区别在于：元空间并不在虚拟机中，而是使用本地内存。
- 4、替换的好处：一、字符串存在永久代中，容易出现性能问题和内存溢出。二、永久代会为 GC 带来不必要的复杂度，并且回收效率偏低。

### 讲讲什么情况下会出现内存溢出，内存泄漏？

内存泄漏的原因很简单：

- 1、对象是可达的(一直被引用)

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

## 2、但是对象不会被使用

常见的内存泄漏例子：

```
public static void main(String[] args) {
    Set<Object> set = new HashSet<>();

    for (int i = 0; i < 10; i++) {
        Object object = new Object();
        set.add(object);

        // 设置为空，该对象不再使用
        object = null;
    }

    // 但是 set 集合中还维护 object 的引用，gc 不会回收 object 对象
    System.out.println(set);
    System.out.println(set.size());
}
}
```

输出结果

```
[java.lang.Object@74a14482,
java.lang.Object@677327b6,
java.lang.Object@6d6f6e28,
java.lang.Object@4554617c,
java.lang.Object@45ee12a7,
java.lang.Object@1b6d3586,
java.lang.Object@7f31245a,
java.lang.Object@135fbaa4,
```

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

```
java.lang.Object@1540e19d,
```

```
java.lang.Object@14ae5a5]
```

```
10
```

```
Process finished with exit code 0
```

解决这个内存泄漏问题也很简单，将 set 设置为 null，那就可以避免上述内存泄漏问题了。其他内存泄漏得一步一步分析了。

#### 内存溢出的原因：

- 1、 内存泄露导致堆栈内存不断增大，从而引发内存溢出。
- 2、 大量的 jar， class 文件加载，装载类的空间不够，溢出
- 3、 操作大量的对象导致堆内存空间已经用满了，溢出
- 4、 nio 直接操作内存，内存过大导致溢出

#### 解决：

- 1、 查看程序是否存在内存泄漏的问题
- 2、 设置参数加大空间
- 3、 代码中是否存在死循环或循环产生过多重复的对象实体、
- 4、 查看是否使用了 nio 直接操作内存。

### 说说线程栈

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

这里的线程栈应该指的是虚拟机栈吧...

- 1、JVM 规范让每个 Java 线程拥有自己的独立的 JVM 栈，也就是 Java 方法的调用栈。
- 2、当方法调用的时候，会生成一个栈帧。栈帧是保存在虚拟机栈中的，栈帧存储了方法的局部变量表、操作数栈、动态连接和方法返回地址等信息
- 3、线程运行过程中，只有一个栈帧是处于活跃状态，称为“当前活跃栈帧”，当前活动栈帧始终是虚拟机栈的栈顶元素。
- 4、通过 jstack 工具查看线程状态

## 常用 JVM 基本配置参数

- 1、-Xmx：最大分配内存，默认为物理内存的 1/4
- 2、-Xms：初始分配内存，默认为物理内存的 1/64
- 3、-Xss：等价于-XX:ThreadStackSize，单个线程栈空间大小，默认一般为 512k-1024k，通过 jinfo 查看为 0 时，表示使用默认值
- 4、-Xmn：设置年轻代大小
- 5、-XX:MetaspaceSize：设置元空间大小（默认 21M 左右，可以配置大一些），元空间的本质可永久代类似，都是对 JVM 规范中方法区的实现，不过元空间与永久代的最大区别在于：元空间不在虚拟机中，而是使用本地内存，因此，默认情况下，元空间大小仅受本地内存大小限制

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

6、 典型设置案例：-Xms128m -Xmx4096m -Xss1024k

-XX:MetaspaceSize=512m -XX:+PrintCommandLineFlags

-XX:+PrintGCDetails -XX:+UseSerialGC

7、 -XX:+PrintGCDetails：打印垃圾回收细节，打印 GC： 打印 Full GC：

8、 -XX:SurvivorRatio：调整 Eden 中 survivor 区比例，默认 -XX:SurvivorRatio=8 (8:1:1)，调整为 -XX:SurvivorRatio=4 (4:1:1)，一般使用默认值

9、 -XX:NewRatio：调整新生代与老年代的比例，默认为 2 (新生代 1，老年代 2，年轻代占整个堆的 1/3)，调整为 -XX:NewRatio=4 表示 (新生代 1，老年代 4，年轻代占堆的 1/5)，一般使用默认值

10、 -XX:MaxTenuringThreshold：设置垃圾的最大年龄 (经历多少次垃圾回收进入老年代)，默认 15 (15 次垃圾回收后依旧存活的对象进入老年代)，JDK1.8 设置必须  $0 < -XX:MaxTenuringThreshold < 15$

## 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，他们的执行顺序

答：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

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

- 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，

## JVM 年轻代到老年代的晋升过程的判断条件是什么呢？

- 1、 部分对象会在 From 和 To 区域中复制来复制去,如此交换 15 次(由 JVM 参数 MaxTenuringThreshold 决定,这个参数默认是 15),最终如果还是存活,就存入到老年代。
- 2、 如果对象的大小大于 Eden 的二分之一会直接分配在 old, 如果 old 也分配不下, 会做一次 majorGC, 如果小于 eden 的一半但是没有足够的空间, 就进行 minorgc 也就是新生代 GC。
- 3、 minor gc 后, survivor 仍然放不下, 则放到老年代
- 4、 动态年龄判断, 大于等于某个年龄的对象超过了 survivor 空间一半, 大于等于某个年龄的对象直接进入老年代

## JVM 出现 fullGC 很频繁, 怎么去线上排查问题

这题就依据 full GC 的触发条件来做:

关注公众号: 磊哥聊编程, 回复: 面试题, 获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

- 1、如果有 perm gen 的话(jdk1.8 就没了)，要给 perm gen 分配空间，但没有足够的空间时，会触发 full gc。
- 2、 所以看看是不是 perm gen 区的值设置得太小了。
- 3、 System.gc()方法的调用
- 4、 这个一般没人去调用吧~~~
- 5、 当统计得到的 Minor GC 晋升到旧生代的平均大小大于老年代的剩余空间，则会触发 full gc(这就可以从多个角度上看了)。
- 6、 是不是频繁创建了大对象(也有可能 eden 区设置过小)(大对象直接分配在老年代中，导致老年代空间不足--->从而频繁 gc)
- 7、 是不是老年代的空间设置过小了(Minor GC 几个对象就大于老年代的剩余空间了)

**请解释 StackOverflowError 和 OutOfMemeryError 的区**

**别?**

通过之前的分析可以发现，实际上每一块内存中都会存在有一部分的可变伸缩区，其基本流程为：如果空间内存不足，在可变范围之内扩大内存空间，当一段时间之后发现内存充足，会缩小内存空间。

永久代 (JDK 1.8 后消失了)

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

虽然 java 的版本是 JDK1.8，但是 java EE 的版本还是 jdk1.7，永久代存在于堆内存之中

## 元空间

元空间在 Jdk1.8 之后才有的，其功能实际上和永久代没区别，唯一的区别在于永久代使用的是 JVM 的堆内存空间，元空间使用的是物理内存，所以元空间的大小受本地内存影响，一般默认在 2M 左右。

范例：设置一些参数，让元空间出错

```
Java -XX:MetaspaceSize=1m
```

## 类加载为什么要使用双亲委派模式，有没有什么场景是打破了这个模式？

双亲委托模型的重要用途是为了解决类载入过程中的安全性问题。

- 1、 假设有一个开发者自己编写了一个名为 `java.lang.Object` 的类，想借此欺骗 JVM。现在他要使用自定义 `ClassLoader` 来加载自己编写的 `java.lang.Object` 类。
- 2、 然而幸运的是，双亲委托模型不会让他成功。因为 JVM 会优先在 `Bootstrap ClassLoader` 的路径下找到 `java.lang.Object` 类，并载入它

Java 的类加载是否一定遵循双亲委托模型？

- 1、 在实际开发中，我们可以通过自定义 `ClassLoader`，并重写父类的 `loadClass` 方法，来打破这一机制。

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题





微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

2、 SPI 就是打破了双亲委托机制的(SPI: 服务提供发现)。

## 类的实例化顺序

- 1、 父类静态成员和静态初始化块 ， 按在代码中出现的顺序依次执行
- 2、 子类静态成员和静态初始化块 ， 按在代码中出现的顺序依次执行
- 3、 父类实例成员和实例初始化块 ， 按在代码中出现的顺序依次执行
- 4、 父类构造方法
- 5、 子类实例成员和实例初始化块 ， 按在代码中出现的顺序依次执行
- 6、 子类构造方法

检验一下是不是真懂了：

```
public class Base {  
    private String name = "博客：Soinice";  
  
    public Base() {  
        tellName();  
        printName();  
    }  
  
    public void tellName() {  
        System.out.println("Base tell name: " + name);  
    }  
}
```

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

```
public void printName() {  
    System.out.println("Base print name: " + name);  
}  
}
```

```
public class Dervied extends Base {  
    private String name = "Java3y";  
  
    public Dervied() {  
        tellName();  
        printName();  
    }  
  
    @Override  
    public void tellName() {  
        System.out.println("Dervied tell name: " + name);  
    }  
  
    @Override  
    public void printName() {  
        System.out.println("Dervied print name: " + name);  
    }  
  
    public static void main(String[] args) {  
        new Dervied();  
    }  
}
```

输出数据：

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

```

Dervied tell name: null
Dervied print name: null
Dervied tell name: Java3y
Dervied print name: Java3y

```

```

Process finished with exit code 0

```

第一次做错的同学点个赞，加个关注不过分吧(hahaha。

## JVM 垃圾回收机制，何时触发 MinorGC 等操作

当 young gen 中的 eden 区分配满的时候触发 MinorGC(新生代的空间不够放的时候)。

## JVM 中一次完整的 GC 流程 (从 ygc 到 fgc) 是怎样的

这题不是很明白意思(水平有限...如果知道这题的意思可在评论区留言呀~~)

因为按我的理解：执行 fgc 是不会执行 ygc 的呀~~

### YGC 和 FGC 是什么

1、YGC：对新生代堆进行 gc。频率比较高，因为大部分对象的存活寿命较短，在新生代里被回收。性能耗费较小。

2、FGC：全堆范围的 gc。默认堆空间使用到达 80%(可调整)的时候会触发 fgc。以我们生产环境为例，一般比较少会触发 fgc，有时 10 天或一周左右会有一次。

### 什么时候执行 YGC 和 FGC

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

1、 a.eden 空间不足,执行 young gc

2、 b.old 空间不足, perm 空间不足, 调用方法 System.gc() , ygc 时的悲观策略, dump live 的内存信息时(jmap -dump:live), 都会执行 full gc

## 各种回收算法

GC 最基础的算法有三种:

1、 标记 -清除算法

2、 复制算法

3、 标记-压缩算法

我们常用的垃圾回收器一般都采用分代收集算法(其实就是组合上面的算法, 不同的区域使用不同的算法)。

具体:

1、 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。

2、 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, 每次只使用其中的一块。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, 然后再把已使用过的内存空间一次清理掉。

关注公众号: 磊哥聊编程, 回复: 面试题, 获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

3、 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行清理，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

4、 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代，这样就可以根据各个年代的特点采用最适当的收集算法。

## 各种回收器，各自优缺点，重点 CMS、G1

图来源于《深入理解 Java 虚拟机：JVM 高级特效与最佳实现》，图中两个收集器之间有连线，说明它们可以配合使用。

- 1、 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，但可能会产生较长的停顿，只使用一个线程去回收。
- 2、 ParNew 收集器，ParNew 收集器其实就是 Serial 收集器的多线程版本。
- 3、 Parallel 收集器，Parallel Scavenge 收集器类似 ParNew 收集器，Parallel 收集器更关注系统的吞吐量。
- 4、 Parallel Old 收集器，Parallel Old 是 Parallel Scavenge 收集器的老年代版本，使用多线程“标记 - 整理”算法
- 5、 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的收集器。它需要消耗额外的 CPU 和内存资源，在 CPU 和内存资源紧张，CPU 较少时，会加重系统负担。CMS 无法处理浮动垃圾。CMS 的“标记-清除”算法，会导致大量空间碎片的产生。

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

6、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容量内存的机器、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。

## stackoverflow 错误, permgen space 错误

stackoverflow 错误主要出现:

在虚拟机栈中(线程请求的栈深度大于虚拟机栈锁允许的最大深度)

permgen space 错误(针对jdk 之前 1.7 版本):

1、大量加载 class 文件

2、常量池内存溢出

## JVM 的引用类型有哪些?

引用内型:

强引用:

当内存不足的时候, JVM 宁可出现 OutOfMemoryError 错误停止, 也需要进行保存, 并且不会将此空间回收。在引用期间和栈有联系就无法被回收

软引用:

当内存不足的时候, 进行对象的回收处理, 往往用于高速缓存中; mybatis 就是其中

关注公众号: 磊哥聊编程, 回复: 面试题, 获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

弱引用：

不管内存是否紧张，只要有垃圾了就立即回收

幽灵引用：

和没有引用是一样的

## 谈谈 JVM 中，对类加载器的认识

类加载器是 JVM 的组成部分之一。将字节码文件加载进 JVM。

类加载分为四部分：

- 1、BootstrapClassLoader, 即跟类加载器, 加载 java 运行时所需的类, 如 String, Integer 等存在  $\{\text{java\_home}\}/\text{jre}/\text{lib}/\text{rt.jar}$  包类的所有类。
- 2、ExtensionClassLoader, 扩展类加载器, 加载一些扩展类, 即  $\{\text{java\_home}\}/\text{jre}/\text{lib}/\text{ext}/\text{*}.jar$  包
- 3、AppClassLoader, 系统加载类, 加载自定义的类, 级 classpath 下的所有类
- 4、ClassLoader 抽象类加载器: 用户自定义的类加载器, 用户定义的类加载器都要继承次 ClassLoader
- 5、Jvm 默认采用的是双亲委派类加载机制, 即先加载父类在加载子类, 对上面四个类加载器采用自顶向下加载

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

## GC 的回收流程是怎样的？

GC 回收流程如下：

- 1、对于整个的 GC 流程里面，那么最需要处理的就是新生代和老年代的内存清理操作，而元空间（永久代）都不在 GC 范围内
- 2、当现在有一个新的对象产生，那么对象一定需要内存空间，平均每个栈内存存 4k，每个堆内存存 8k，那么对象一定需要进行堆空间的申请
- 3、首先会判断 Eden 区是否有内存空间，如果此时有内存空间，则直接将新对象保存在伊甸园区。
- 4、但是如果此时在伊甸园区内存不足，那么会自动执行一个 Minor GC 操作，将伊甸园区的无用内存空间进行清理，Minor GC 的清理范围只在 Eden 园区，清理之后会继续判断 Eden 园区的内存空间是否充足？如果内存空间充足，则将新对象直接在 Eden 园区进行空间分配。
- 5、如果执行 Minor GC 之后发现伊甸园区的内存空间依然不足，那么这个时候会执行存活区的判断，如果存活区有剩余空间，则将 Eden 园区部分活跃对象保存在存活区，那么随后继续判断 Eden 园区的内存空间是否充足，如果充足则则将新对象直接在 Eden 园区进行空间分配。
- 6、此时如果存活区没有内存空间，则继续判断老年区。则将部分存活对象保存在老年代，而后存活区将有空余空间。
- 7、如果这个时候老年代也满了，那么这个时候将产生 Major GC (Full GC) ,那么这个时候将进行老年代的清理

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题





微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

8、 如果老年代执行 Full GC 之后，无法进行对象的保存，则会产生 OOM 异常,OutOfMemoryError 异常

磊哥聊编程  
号，磊哥聊编程：得最新版，面试题  
关注公众号，磊哥聊编程：得最新版，面试题  
关注公众号，磊哥聊编程：得最新版，面试题  
关注公众号，磊哥聊编程：得最新版，面试题  
关注公众号，磊哥聊编程：得最新版，面试题  
公众号，磊哥聊编程：得最新版，面试题  
磊哥聊编程

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题