面试真题

【美团】讲一下你熟悉垃圾回收器？

1. Serial GC 单线程GC，是JVM Client模式默认垃圾回收器

**缺点：**全程StopTheWorld

1. Parallel GC 多线程GC，是JDK8默认垃圾回收器

**缺点：**停顿时间较长。

1. ConcurrentMarkSweep GC 最短停顿时间

初始标记、并发标记、重新标记、并发清除

缺点：内存碎片化、CPU敏感

1. Garbage 1st GC 停顿时间可控、压缩避免碎片，JDK9+用G1替代CMS

划分堆为2048个1~32MB区域，采用Mixed GC模式，通过-XX:MaxGCPauseMillis停顿时间可控

缺点：内存占用高

如何优化JVM的垃圾回收？

1. 合适的垃圾回收器
2. 合适的堆内存大小，避免频繁Full GC  
   Full GC：JVM回收包括年轻代和老年代在内的整个堆内存，导致长停顿。
3. 合适的新生代和老年代的比例

【美团】新生代和老年代的占比有了解吗？

新生代**占**堆内存的 1/3**：Eden区、Survivor区 (S0和S1)**

老年代**占**堆内存的 2/3

新生代和老年代占比的调整

-XX:NewRatio=2 表示老年代占总堆内存的 **2/3**，新生代占 **1/3**

如何根据应用场景调整新生代与老年代的比例？

对于短生命周期对象较多的，如Web应用，**增**加新**生代的内存**。

对于长生命周期对象较多的，如大数据处理，**增**加老**年代的内存**。

知识框架

JVM基础

JVM内存模型包含哪些核心区域？各自作用是什么？

JVM内存分为：

1. 堆：线程共享，存放对象实例，由GC管理，可能发生OOM。
2. 方法区（元空间）：存储类元信息、常量池（JDK8后由元空间实现）。
3. 虚拟机栈：线程私有，存储方法调用的栈帧（局部变量、操作数栈）。
4. 本地方法栈：服务于Native方法。
5. 程序计数器：记录当前线程执行指令的地址，无OOM。

JDK8方法区为什么用元空间替代永久代？

哪些场景会导致栈溢出（StackOverflowError）？

双亲委派模型的工作机制是什么？有什么优缺点？

**机制**：类加载器收到请求后，先委派给父类加载器（Bootstrap→Extension→Application），父类无法完成时自己加载。  
**优点**：避免重复加载核心类，保证安全性（如自定义String类不会被加载）。  
**缺点**：无法解决基础类调用用户代码的场景（如JDBC需用线程上下文类加载器打破委派）。

如何自定义类加载器并打破双亲委派？

Tomcat为什么要破坏双亲委派？

堆和栈的区别是什么？

堆：存对象实例、对象存活到GC回收、线程共享、堆异常OOM

栈：存方法调用和局部变量、随方法结束销毁、线程私有、栈异常StackOverflowError

局部变量在栈上，对象一定在堆上吗？（逃逸分析后的栈上分配）

如何通过JVM参数调整堆和栈大小？

常用的JVM启动参数有哪些？如何设置堆内存？

* **-Xms**：初始堆内存（建议与-Xmx一致避免扩容抖动）；
* **-Xmx**：最大堆内存（如-Xmx4G）；
* **-Xmn**：年轻代大小（如-Xmn1G）；
* **-XX:MetaspaceSize**：元空间初始大小（替代PermSize）。

为什么推荐-Xms和-Xmx设置相同？

如何根据GC日志调整年轻代与老年代比例？

类加载的过程分为哪几个阶段？

1. **加载**：读取字节码到方法区，生成Class对象。
2. **验证**：检查字节码合法性（如魔数、语法）。
3. **准备**：为静态变量分配内存并赋初始值（0或null）。
4. **解析**：将符号引用转为直接引用。
5. **初始化**：执行静态代码块和变量赋值（触发<clinit>）。

哪些操作会触发类的初始化？（如new、反射、访问静态变量）

如何实现一个热部署的类加载机制？

内存管理与垃圾回收

Minor GC、Major GC、Full GC 的区别及触发条件？

* **Minor GC**：回收年轻代（Eden + Survivor），触发条件为Eden区满。速度快，频率高。
* **Major GC**：回收老年代（部分回收器如CMS单独回收老年代）。
* **Full GC**：回收整个堆（年轻代+老年代+方法区），触发条件包括老年代不足、System.gc()调用、晋升失败等。  
  **误区**：Major GC ≠ Full GC（如CMS的并发回收不包含年轻代）。

CMS为什么需要两次Stop-the-World（初始标记、重新标记）？

什么情况下频繁Minor GC会导致Full GC？

标记-清除、复制算法、标记-压缩的区别及适用场景？

1. **标记-清除**：标记存活对象后清除垃圾，产生内存碎片。适用于老年代（如CMS）。
2. **复制算法**：将存活对象复制到Survivor区，无碎片但浪费空间。适用于年轻代（Eden→Survivor）。
3. **标记-压缩**：标记后压缩存活对象到内存一端，消除碎片但耗时。适用于老年代（如Serial Old）。

为什么年轻代默认用复制算法？（对象存活率低，效率高）

G1如何结合三种算法？（分Region标记-压缩+局部复制）

CMS、G1、ZGC的核心区别及适用场景？

* **CMS**：并发标记清除，低停顿但碎片多。适用老年代，对CPU敏感（已废弃）。
* **G1**：分Region回收，可预测停顿（MaxGCPauseMillis），兼顾吞吐和延迟。
* **ZGC**：亚毫秒级停顿（<10ms），基于颜色指针和读屏障，适用大堆低延迟场景（如金融交易）。

ZGC如何实现并发整理？（颜色指针+多映射）

G1的Mixed GC是什么？（回收部分老年代Region）

如何排查JVM内存泄漏？常用工具有哪些？

**步骤**：

1. **现象**：频繁Full GC但堆内存持续增长，OOM。
2. **工具**：
   * jmap -histo查看对象数量分布。
   * **VisualVM**：堆Dump分析Dominator Tree（定位大对象引用链）。
   * **MAT（Memory Analyzer）**：分析泄漏路径（如ThreadLocal未释放）。
3. **代码**：检查静态集合、未关闭资源（连接池、流）、监听器未注销。

如何用WeakReference解决缓存泄漏？

OOM时自动生成Dump的参数？（-XX:+HeapDumpOnOutOfMemoryError）

如何从GC日志分析性能问题？优化方向有哪些？

**日志关键指标**：

* **吞吐量**：1 - (GC时间/总时间) > 90%。
* **停顿时间**：单次Young GC < 50ms，Full GC < 1s。
* **晋升速率**：对象过快进入老年代（检查Survivor区大小）。

**优化方向**：

* **年轻代**：增大Eden（减少Minor GC频率）或调整SurvivorRatio。
* **老年代**：避免过早晋升（-XX:MaxTenuringThreshold）。
* **GC选择**：高吞吐选Parallel，低延迟选G1/ZGC。

如何通过日志判断内存碎片问题？（Full GC后老年代可用空间仍不足）

G1的Humongous对象如何处理？（单独分配Humongous Region）

三、性能调优

* JVM性能瓶颈：如何检测JVM性能瓶颈，使用工具如JProfiler、JVM自带的JVM工具（jstat、jmap、jstack等）分析性能。
* JVM调优的常见方法：如何根据具体的应用场景（如Web应用、高并发应用）进行JVM调优，例如调整堆内存大小、优化GC策略等。
* HotSpot优化：如何进行HotSpot虚拟机的调优，JIT编译优化，代码热替换等

四、JVM与多线程

* JVM中的线程模型：线程的生命周期、状态转换等。
* JVM中的并发问题：如何避免内存可见性问题、竞态条件和死锁。
* JVM的锁优化：Synchronized关键字和锁优化（如偏向锁、轻量级锁、重量级锁等）。
* JVM如何管理线程：JVM如何管理线程的栈空间，线程的优先级和调度。

五、JVM与操作系统

* JVM与操作系统的关系：JVM如何与操作系统交互，包括内存分配、文件IO等。
* 操作系统对JVM性能的影响：操作系统的内存管理策略、进程调度对JVM性能的影响。

六、高级话题

* JVM的类卸载机制：如何理解和实现类卸载，类卸载的触发条件。
* 逃逸分析与锁消除：JVM中的逃逸分析、锁消除、内联等优化技术。
* JVM与网络通信：JVM在处理大规模分布式系统中的表现，JVM对高并发请求的优化

七、常见面试题

* 内存溢出（OutOfMemoryError）的类型：不同类型的内存溢出（如Java Heap Space、PermGen Space、Direct Buffer Memory等）的原因及解决方法。
* JVM如何处理堆外内存：直接内存（Direct Memory）的使用、Native Memory Tracking（NMT）的启用。
* 如何进行JVM调优：如果你的应用程序性能很差，如何分析JVM并做调优？有哪些工具和方法可以用来优化JVM的性能？
* CMS和G1的区别：CMS和G1垃圾回收器的工作原理、优缺点，适用场景。
* JVM调优中的常见误区：例如，过多的堆内存设置、频繁Full GC等常见的调优错误。