面试真题

【美团】讲一下你熟悉垃圾回收器？

**1. Serial GC：**

* **工作原理**：Serial GC使用单线程进行垃圾回收，适用于单核环境或资源受限的场景。它的回收过程会在应用线程停顿时进行，可能导致较长的GC停顿时间。
* **适用场景**：对于内存较小、CPU资源有限的应用适用。

**2. Parallel GC：**

* **工作原理**：Parallel GC使用多个线程并行进行垃圾回收，主要通过分代回收的方式优化回收过程。回收时，应用线程会暂停，称为Stop-the-World。
* **适用场景**：适用于多核服务器，能够充分利用多核的优势来加速垃圾回收。

**3. CMS GC (Concurrent Mark-Sweep GC)：**

* **工作原理**：CMS GC旨在减少垃圾回收的停顿时间，它将标记阶段与清理阶段并行进行，应用线程在此期间可以继续执行，减少了停顿时间。回收时会有多个线程并行执行，优先考虑响应时间。
* **优缺点**：
  + **优点**：较低的停顿时间。
  + **缺点**：可能出现碎片问题（Full GC的频率较高，特别是老年代的内存碎片化严重时）。
* **适用场景**：对响应时间要求较高的应用，如Web服务器、实时系统。

**4. G1 GC (Garbage First GC)：**

* **工作原理**：G1 GC是一种低延迟垃圾回收器，采用了区域划分的方式（称为Region），不同的区域大小可以进行独立回收。G1 GC的目标是控制停顿时间，使其达到可预测性。在回收过程中，G1会尝试优先回收那些垃圾最多的区域。
* **优缺点**：
  + **优点**：适用于大内存、高延迟要求的应用，能够提供可控的停顿时间。
  + **缺点**：较为复杂，调优较困难，且初期性能可能不如Parallel GC。
* **适用场景**：大内存、大型Web应用、长时间运行的服务器等。

**GC选择及调优：**

* **GC的选择**：选择合适的垃圾回收器取决于应用的内存使用特点和对响应时间的要求。大多数情况下，Parallel GC适用于大多数应用，但对于要求低延迟的应用，G1或CMS GC会是更好的选择。
* **调优**：通过JVM的参数配置，如-XX:+UseG1GC、-Xms、-Xmx等，可以选择垃圾回收器并进行内存优化。常用的监控工具有jstat、VisualVM等，可以帮助分析和优化GC行为。

**延伸问答：**

1. **G1 GC的优缺点**：
   * 优点：更稳定的停顿时间，适合大内存应用。
   * 缺点：较为复杂，初期可能导致停顿时间过长，需要仔细调优。
2. **如何优化JVM的垃圾回收？**
   * 选择合适的垃圾回收器。
   * 调整堆内存大小，避免频繁的Full GC。
   * 通过分析应用的对象分配特征，调整年轻代和老年代的比例。
3. **什么是Full GC？如何减少Full GC的发生？**
   * Full GC是指JVM回收整个堆内存，包括年轻代和老年代，通常会导致较长的停顿。可以通过调整堆内存大小、增加老年代的空间或使用更高效的垃圾回收器来减少Full GC。

【美团】新生代和老年代的占比有了解吗？

**简要回答**

**新生代与老年代的占比：**

* **新生代**：通常占堆内存的 **1/3** 左右，存放新创建的对象。
* **老年代**：通常占堆内存的 **2/3** 左右，用于存放经过多次GC的存活对象。

**详细回答**

**1. 新生代 (Young Generation)：**

* **包含区域**：
  + **Eden区**：新对象大部分会在此分配。
  + **Survivor区 (S0和S1)**：存放从Eden区中幸存下来的对象。两块Survivor区交替使用，避免频繁的GC。
* **占比**：堆内存的 **1/3**。此部分较小，主要用于存放生命周期短、较为频繁的对象。对象被创建后，大部分会在新生代中回收。

**2. 老年代 (Old Generation)：**

* **包含区域**：此区域存储的是经过多次垃圾回收、仍然存活的对象。老年代的回收通常比较少发生。
* **占比**：堆内存的 **2/3**。老年代存储的是长生命周期对象，GC次数相对较少，回收时间较长，但停顿时间较长。
* **对象晋升**：从新生代晋升到老年代的对象通常具有较长的生命周期。

**3. 新生代和老年代占比的调整：**

* 这些默认占比（1:2 或 1/3 : 2/3）是基于JVM的默认设置，实际应用中可以通过JVM启动参数调整。
  + **-Xms**：初始化堆大小
  + **-Xmx**：最大堆大小
  + **-XX:NewRatio**：控制新生代与老年代的比例。比如设置为 -XX:NewRatio=2 会让老年代占总堆内存的 **2/3**，新生代占 **1/3**。

**延申问答**

**1. 如何调整新生代和老年代的比例？**

* **-XX:NewRatio=<ratio>**：设置新生代和老年代的比例，例如 -XX:NewRatio=3 会使得新生代占堆内存的 **1/4**，老年代占 **3/4**。

**2. 新生代与老年代的垃圾回收有什么不同？**

* 新生代GC（通常是Minor GC）比较频繁，但每次回收的时间较短。老年代GC（Major GC或Full GC）较少发生，但回收时间较长，会造成更大的停顿。

**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应用（如Web应用），可以增加新生代的内存，减少老年代。
* 对于长期运行、长生命周期对象较多的应用（如大数据处理），可以增加老年代的内存。

知识框架