对象生命周期与垃圾回收 GC 机制

- ❓如果一直 new 对象，内存会不会爆？

> ✅ 可能不会

- Java 有 **垃圾回收机制 garbage collection，会自动回收不再使用的对象**
- 只要不是无限创建**还一直有引用指着它们**，JVM 会自己清理

👉 所以：**爆不爆，取决于有没有把旧对象“断引用”**

❓对象的内存什么时候被回收？

> ✅谁也不知道, JVM 说了算

- JVM 会**在合适的时机**自动触发垃圾回收
- 可以用 `System.gc()` 建议 GC 执行，但不能强制

❓对象的内存如何被回收？

> ✅ 不用管，JVM 帮你干

- Java 最大的优点之一就是：**自动垃圾回收**
- 只要保证不再引用某个对象，GC 迟早会处理它

```
Student s = new Student();
s = null;  // 原来的 Student 对象就没人用了，可以被回收
```

❓JVM 怎么知道“哪个对象没人用了”？

✅通过引用链（GC Root）判断

▼ 🔗 GC Root

JVM 会维护一组特殊的“根对象”，叫做 **GC Roots**，它们永远不会被回收

- 方法栈中的局部变量（如 `main` 方法里的对象）
- 类的静态成员引用的对象
- JNI（本地代码）引用的对象

> 只要某个对象从 GC Root 出发“能找到”，就活着；找不到，就可以被回收

这叫做 👉 **可达性分析算法（Reachability Analysis）**

```
public class Demo {
    public static void main(String[] args) {
        Dog d = new Dog(); // 这个 d 是 GC Root 的一部分
    }
}
```

- `d` 是一个局部变量，是 GC Root
- 所以它引用的 `Dog` 对象也算“活着”
- 当 `main` 方法结束，`d` 不再存在，对象就失去引用，准备被 GC 回收

▼ **分代回收算法 Generational Garbage Collection**

> 分代回收算法是 JVM 垃圾回收器的一种策略：
>
> 把内存中的对象按照“存活时间”分成几代，**不同代用不同方式回收**，提高效率

JVM 把堆内存标准来说是分为 3 代

名称	特点	举例
新生代（Young Generation）	放“刚创建的对象”，**回收频繁**	99%的小对象
老年代（Old Generation）	放“长时间没死的对象”，**回收少**	例如：缓存、大数组
永久代/元空间（Perm/Metaspace）	存类的定义、方法区，不存对象本体	只在类加载时存在

> Java 8 开始：永久代被移除，改成了元空间（Metaspace）

🧪 为什么要分代？

因为绝大多数 Java 对象**很快就死了**

```
String name = scanner.nextLine();
```

- `scanner` 是局部变量，一用完就回收
- 很多对象都只活一会儿

所以：如果一视同仁、全堆一起回收，会非常浪费性能

✅ 分代的核心策略：

▼ 🟩 新生代（Young Gen）

- 分为：
  - Eden 区（伊甸园）
  - Survivor 区（幸存区）两个：S0 和 S1
- 创建对象 → 先放 Eden
- GC 时 → 活下来的对象搬去 S0/S1
- 如果对象经历了**多次 GC 仍然活着**，就被提升到：

▼ 🟨 老年代（Old Gen）

- 存放“老而不死”的对象
- 只在必要时才做 GC，称为 **Major GC 或 Full GC**
- 比较慢，但不经常发生

房间类型	对应代	清理频率	原因
当天退房的客房	新生代	天天清理	很快就空了
长住套房	老年代	几天一次	住客不走，清理频率低
酒店登记处/员工资料	元空间	不清理	这些数据很稳定，系统级别的

名词	解释
Minor GC	清理新生代（频繁，快）
Major GC / Full GC	清理老年代（不频繁，慢）
晋升（Promotion）	新生代对象“活太久”后移入老年代
对象年龄（Age）	每经历一次GC +1，达到一定值被晋升