**Garbage Collection(heap)**

1. 引用类型

强引用strong reference：常用的，只要引用，永不GC

软引用SoftReference：内存溢出异常时(OOM)，才纳入GC

弱引用WeakReference：只能生存至下次GC之前

虚引用PhantomReference：被GC时，收到一个系统通知

2. 回收标记策略

a. 引用计数

问题：循环引用，被引用次数一直非零，导致不标记回收

a.next = b;

b.next = a;

a = null;

b = null;

b. 可到达性分析(reachability analysis)

以GC root为起点，能遍历到的节点，不标记回收；不可到达的节点，标记回收

3. 内存清除策略

a. 标记-清除

问题：内存空间碎片化，难以为大对象分配连续的空间

b. 复制-清除

划分区域A和B，只使用A，若对A进行GC，则把A中存活的复制至B，清除整个A

问题：存活率高的场景，效率降低

c. 标记-整理

d. 分代

根据不同场景，对应不同策略。如：新生代、老年代

**ByteCode(.class)**

1. 数据：没有间隔符的无符号数表格(二进制字节流)

|  |  |
| --- | --- |
| Magic Number | 被JVM识别的标记 |
| minor version | jdk次版本号 |
| major version | jdk主版本号 |
| constant pool | 常量池：包括literal和symbolic references |
| access\_flag | 访问标志，区分ENUM/CLASS/INTERFACE/ABSTRACT等 |
| this\_class | 当前类索引 |
| super\_class | 父类索引 |
| interfaces | 接口索引列表 |
| field\_info | 变量字段表：access\_flags、name\_index、descriptor\_index、attributes\_count、attributes |
| method\_info | 方法表：同上 |
| attribute\_info | 属性表：  Code代码区  LineNumberTable源码行号与字节码行号的映射表  LocalVariableTable局部变量与源码的映射表  Signature类型擦除实现伪泛型的参数 |

2. 指令集：逻辑运算、类型转换、对象创建与访问、方法调用与返回、读写操作数栈、流程控制、异常处理、线程同步等

**Class Loader**

0. 基本过程

加载、验证、准备、解析、初始化、使用、卸载

1. loading

a. 通过全限定名获取类的二进制字节流，如: class, jar, war等

b. 二进制字节流->运行时数据结构

c. 在内存中生成这个类的java.lang.class对象，作为该类的访问入口

2. verification：各种检查

3. preparation：类变量分配内存，设置为0，如：static

4. resolution：symbolic reference(内存无关) -> direct reference(内存相关)

5. initialization

**调优分析**

1. javap -verbose Dao.class分析字节码结构

---------TODO----------------------------------------------------------

内存管理

1. 运行时数据区

程序计数器：指令取址，线程私有

虚拟机栈：局部变量空间，for java method，线程私有

本地方法栈：局部变量空间，for C++ native function，线程私有

堆heap：动态分配的变量空间，被所有线程共享的GC空间

方法区：类信息、静态变量、常量池等，线程共享

2. 常见异常

StackOverflowError：栈帧溢出，一般是函数调用过深

OutOfMemoryError：内存溢出，一般是数据量过大

3. 常用参数

-Xms 堆的最小值

-Xmx 堆的最大值

-Xmn 新生代大小

-Xss 栈大小