第二章：java内存区域与内存溢出

2.2 运行时数据区



2.3 hotspot虚拟机对象探秘

2.3.1 对象的创建

对象所需内存大小在类加载完成就已经确定。Java堆内存是规整的就用“指针碰撞”分配内存，不规整的就用“空闲列表”分配。

在并发情况下，为了防止分配内存指针指向错误，有两种解决方式：一种是对分配内存空间的动作进行同步处理—采用CAS配上失败重试保证更新操作原子性；二是把内存分配的动作按照线程划分到不同的空间，即每个线程在java堆中预先分配一小块内存，称为本地线程分配缓冲（TLAB）。

对象创建之后，只有<inti>方法执行，对于程序来说，对象才真正创建完成。

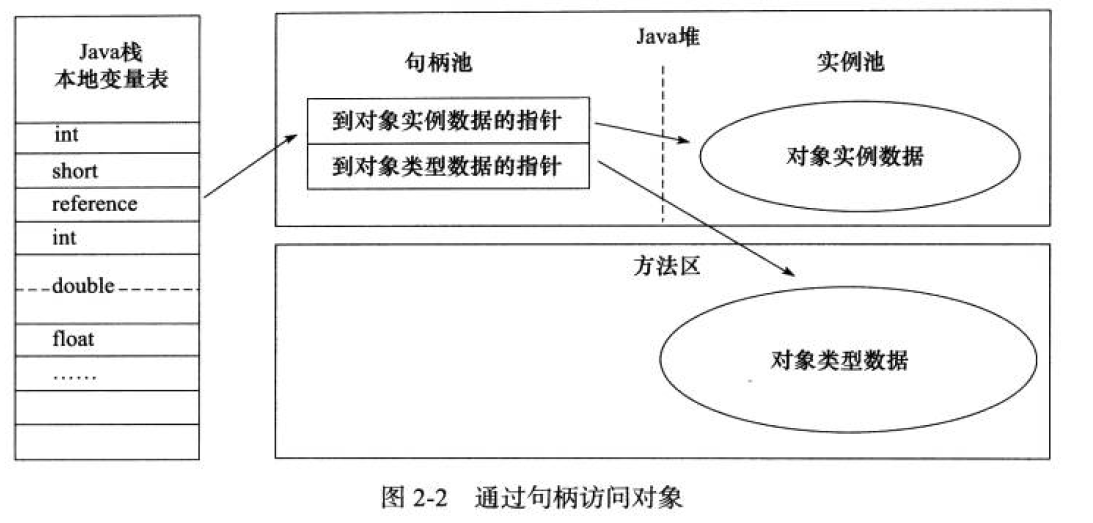
2.3.2 对象的内存布局

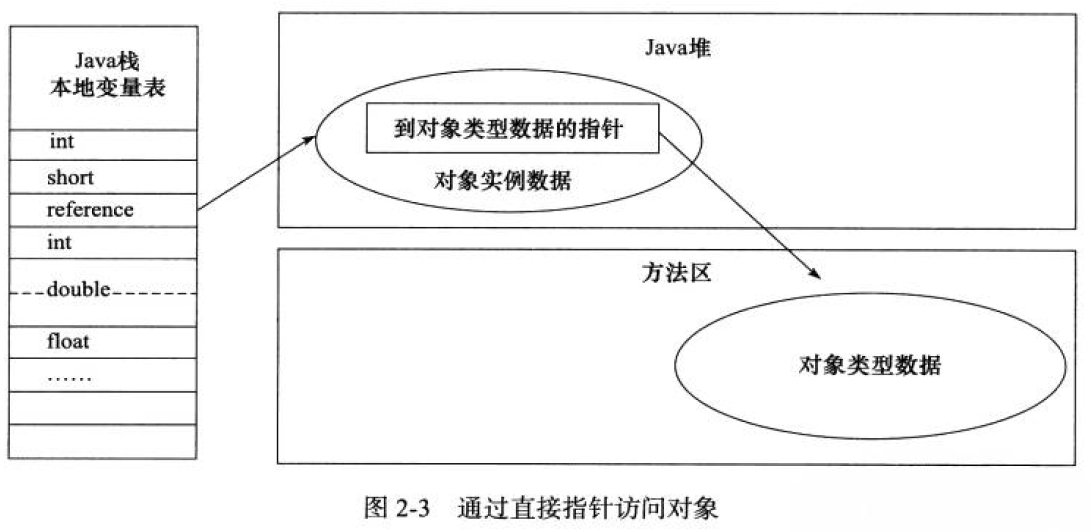
对象在内存中存储分为三块区域：对象头，实例数据，对齐填充。



2.3.3 对象的访问定位

Java对象需要栈上的reference数据来操作堆上的对象，reference只规定了对象的引用，对象的访问方式有两种：使用句柄和直接指针。





第三章 垃圾收集器与内存分配策略

对于java运行时数据区，虚拟机栈，本地方法栈，程序计数器都是线程私有，随线程的消亡而消亡，栈上的栈帧随着方法随着方法的进入和退出进行着入栈和出栈，每个栈帧分配的内存在编译期已经确定，所以这部分不用进行垃圾回收。主要集中在堆和方法区上面。

判断对象是否存活的方法：

1.引用计数算法

给对象添加一个引用计数器，有地方引用，就+1，引用失效，就-1，当这个值为0，就可以进行垃圾回收。但是它不能解决对象之间循环调用的问题。

2.可达性分析算法（主流方法）

以“GC Roots”对象为起点开始往下搜索，搜索所走过的路径称为引用链（reference chain），当一个对象到“GC Roots”没有任何引用链，说明这个对象不可达，就可以进行回收。

Java中可以作为”GC Roots”的有：

1.虚拟机栈（栈帧中的局部变量表）中引用的对象

2.方法区中类静态属性引用的对象

3.方法区中常量引用的对象

4.本地方法栈JNI（native方法）引用的对象

3.2.3 java引用的理解

Jdk1.2之前，java中引用的定义是：如果reference类型中存储的数据代表另外一块内存起始地址，就称这块内存代表一个引用。

Jdk1.2之后，扩展为四种

强引用：new等，只要强引用存在，垃圾回收永远不会回收

软引用：softReference，用来描述一些有用但是非必须的对象，在系统即将发生内存溢出之前，会将这些引用进行二次回收，如果回收之后还是没有足够内存，会报内存溢出

弱引用：weakReference，下一次垃圾回收，就会回收掉。

虚引用：phantomReference，存在的唯一意义：在对象回收之前收到一个系统通知。

3.2.4 生存还是死亡

对于标记为不可达的对象，垃圾回收不会立即处理。先判断是否执行Finalize方法，如果执行，在这方法中如果调用this成功拯救自己就可以不用回收。如果没有，就会被回收掉

Finalize方法只能被系统调用一次。

3.2.5 方法区的回收

主要回收废弃常量和无用的类

3.3 垃圾收集算法

3.3.1 标记--清除算法

不足：1.标记和清除效率都不高；

2.垃圾回收会产生大量内存碎片，可能导致后续无法分配大内存而再次出发垃圾回收

3.3.2 复制算法

可用内存分为相等两块，每次使用一块，这块使用完，把存活的对象放到另外一块，然后再把使用过的空间清理掉。每次只对半区进行回收，也就不用考虑内存碎片情况，只要移动堆顶指针，按顺序分配即可。

优点：实现简单，运行高效

缺点：太浪费空间

新生代采用复制算法，eden，survivor0，survivor1按照8：1：1划分，每次用eden和一个survivor区域，回收时，全部放入另外一块survivor区域。

3.3.3 标记—整理算法

标记，然后把存活对象向一端移动，然后清理掉边界以外的内存。

3.5 垃圾收集器

第七章：虚拟机类加载机制

1、类加载时机

类从被加载到虚拟机内存，到卸载出内存。整个生命过程包括：加载(loading)，验证(verification)，准备(preparation)，解析(resolution)，初始化(initialization)，使用(using)，卸载(unload)。



有且只有五种情况，上述顺序的初始化最先执行

1. 遇到new，getstatic，putstatic，invokestatic四条字节码指令的时候，如果类没有进行初始化，则初始化最先触发。场景：new实例化对象、读取或设置一个类的静态字段（被final修饰，已在编译器放入常量池除外）、调用一个类的静态方法
2. 使用java.lang.reflect进行反射的时候，如果类没有进行初始化，会触发初始化
3. 初始化一个类，如果发现父类没有进行初始化，先触发父类的初始化
4. 当虚拟机启动时，用户需要指定一个要执行的主类，虚拟机会初始化这个主类
5. 当使用jdk1.7动态语言支持，如果一个java.lang.invoke.MethodHandle实例最后解析结果ref\_getStatic,ref\_putStatic,ref\_invokeStatic的方法句柄，并且这个方法没有初始化，会触发初始化

上述五种叫主动引用，其他情况都不会触发初始化，称为被动引用。