为对象分配内存

类加载完成后，接着会在Java堆中划分一块内存分配给对象。内存分配根据Java 堆是否规整，有两种方 式：

- 指针碰撞：如果Java堆的内存是规整，即所有用过的内存放在一边，而空闲的的 放在另一边。分配 内存时将位于中间的指针指示器向空闲的内存移动一段与对象大小 相等的距离，这样便完成分配内 存工作。
- 空闲列表：如果Java堆的内存不是规整的，则需要由虚拟机维护一个列表来记录 那些内存是可用 的，这样在分配的时候可以从列表中查询到足够大的内存分配给对 象，并在分配后更新列表记录。

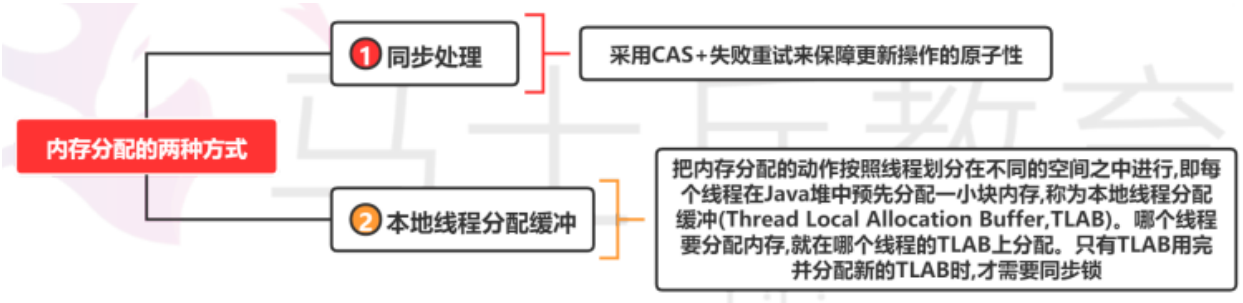
选择哪种分配方式是由 Java 堆是否规整来决定的，而 Java 堆是否规整又由所 采用的垃圾收集器是否带 有压缩整理功能决定。

处理并发安全问题

对象的创建在虚拟机中是一个非常频繁的行为，哪怕只是修改一个指针所指向的 位置，在并发情况下也是不安全的，可能出现正在给对象 A 分配内存，指针还 没来得及修改，对象 B 又同时使用了原来的指针来分配内存的情况。解决这个 问题有两种方案：

对分配内存空间的动作进行同步处理（采用 CAS + 失败重试来保障更新操作的 原子性）；

把内存分配的动作按照线程划分在不同的空间之中进行，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（Thread Local Allocation Buffer, TLAB）。哪个线程要分配内存，就在哪个线程的 TLAB 上分配。只有 TLAB 用完并 分配新的 TLAB 时，才需要同步锁。通过-XX:+/-UserTLAB参数来设定虚拟机是否使 用TLAB



对象的访问定位

Java程序需要通过 JVM 栈上的引用访问堆中的具体对象。对象的访问方式取决 于 JVM 虚拟机的实现。

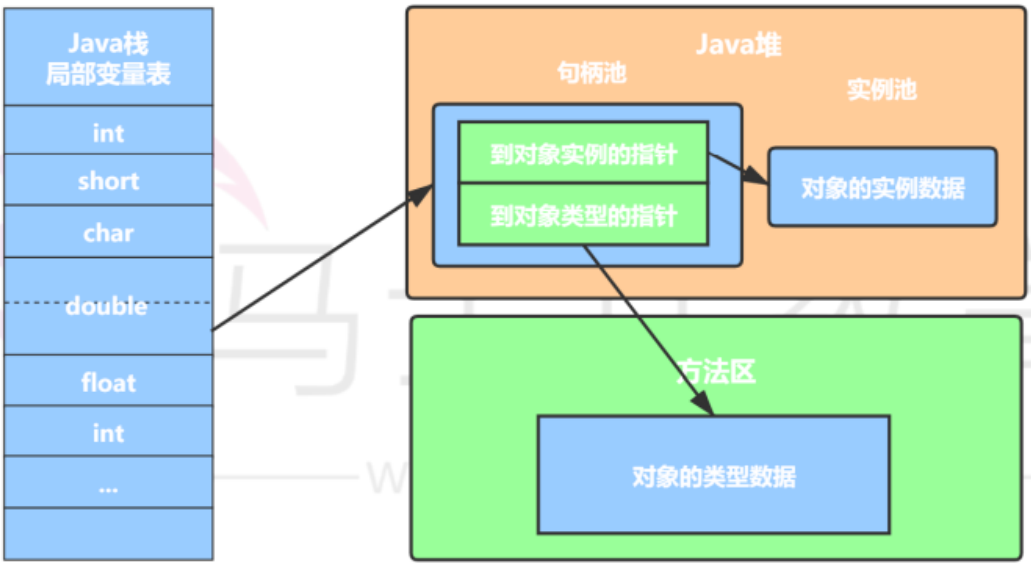
目前主流的访问方式有 **句柄** 和 **直接指针** 两种方式。

指针： 指向对象，代表一个对象在内存中的起始地址。

句柄： 可以理解为指向指针的指针，维护着对象的指针。句柄不直接指向对象，而是 指向对象的指针（句柄不发生变化，指向固定内存地址），再由对象的指针指向对象的 真实内存地址。

句柄访问

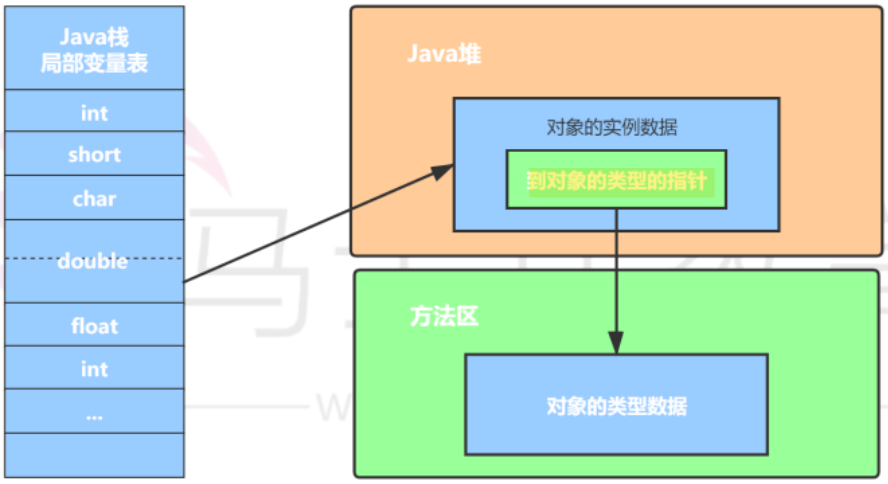
Java堆中划分出一块内存来作为句柄池，引用中存储对象的句柄地址，而句柄中 包含了对象实例数据与 对象类型数据各自的具体地址信息，具体构造如下图所 示：



优势：引用中存储的是稳定的句柄地址，在对象被移动（垃圾收集时移动对象是 非常普遍的行为）时只会改变句柄中的实例数据指针，而引用本身不需要修改。

直接指针

如果使用直接指针访问，引用 中存储的直接就是对象地址，那么Java堆对象内 部的布局中就必须考虑如何放置访问类型数据的相关信息。



优势：速度更快，节省了一次指针定位的时间开销。由于对象的访问在Java中非 常频繁，因此这类开销积少成多后也是非常可观的执行成本。HotSpot 中采用 的就是这种方式。