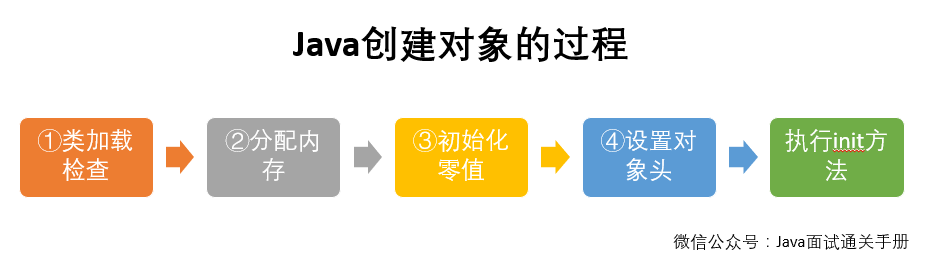
通过上面的介绍大概知道了虚拟机的内存情况，下面详细的了解下HotSpot虚拟机在Java堆中对象分配、布局和访问的全过程。

### 3.1对象的创建

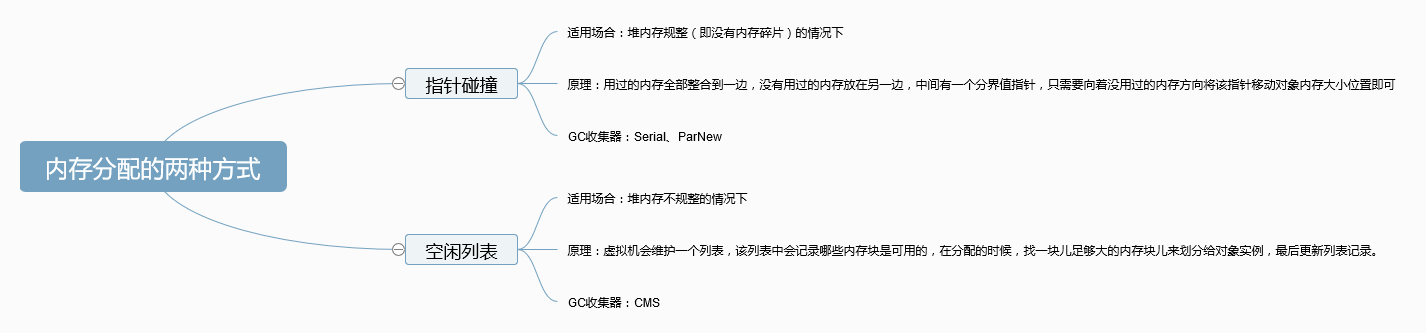
下图便是Java对象的创建过程，建议最好能默写下来，并且要掌握每一步在做什么。



1. 类加载检查：虚拟机遇到一条new指令时，首先将去检查这个指令的参数是否能在常量池中定位到这个类的符号引用，并检查这个符号引用代表的类是否已被加载过、解析和初始化过。如果没有，那必须先执行相应的类加载过程。
2. 分配内存：在类加载检查通过后，接下来虚拟机将为新生对象分配内存。对象所需的内存大小在类加载完成后便可确定，为对象分配空间的任务等同于把一块确定大小的内存从Java堆中划分出来。分配方式有“指针碰撞”和“空闲列表”两种，选择哪种分配方式由Java堆是否规整决定，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。

内存分配的两种方式：（补充内容，需要掌握）

选择以上两种方式中的哪一种，取决于Java堆内存是否规整。而Java堆内存是否规整，取决于GC收集器的算法是“标记-清除”，还是“标记-整理”（也称作“标记-压缩”），值得注意的是，复制算法内存也是规整的。



内存分配并发问题（补充内容，需要掌握）

在创建对象的时候有一个很重要的问题，就是线程安全，因为在实际开发过程中，创建对象是很繁琐的事情，作为虚拟机来说，必须要保证线程是安全的，通常来讲，虚拟机采用两种方式来保证线程安全：

* CAS+失败重试：CAS是乐观锁的一种实现方式。所谓乐观锁就是，每次不加锁而是假设没有冲突而去完成某项操作，如果因为冲突失败就重试，直到成功为止。虚拟机采用CAS配上失败重试的方式保证更新操作的原子性。
* TLAB：为每一个线程预先在Eden区分配一块儿内存，JVM在给线程中的对象分配内存时，首先在TLAB分配，当对象大于TLAB中的剩余内存或TLAB的内存已用尽时，再采用上述的CAS进行内存分配。

1. 初始化零值：内存分配完成后，虚拟机需要将分配到的内存空间都初始化为零值（不包括对象头），这一步操作保证了对象的实例字段在Java代码中可以不赋初始值就直接使用，程序能访问到这些字段的数据类型所对应的零值。
2. 设置头对象：初始化零值完成之后，虚拟机要对对象进行必要的设置，例如这个对象是哪个类的实例、如何才能找到类的元数据信息、对象的哈希码、对象的GC分代年龄等信息。这些信息存放在对象头中。另外，根据虚拟机当前运行状态的不同，如是否启用偏向锁等，对象头会有不同的设置方式。
3. 执行init方法：在上面工作都完成之后，从虚拟机的视角来看，一个新的对象已经产生了，但从Java程序的视角来看，对象的创建才刚开始，<init>方法还没有执行，所有的字段都还为零。所以一般来说，执行new指令之后会接着执行<init>方法，把对象按照程序员的意愿进行初始化，这样一个真正可用的对象才算完成生产出来。

### 3.2对象的内存布局

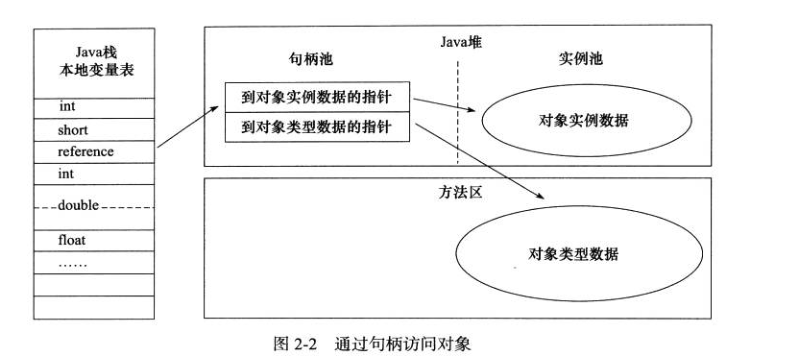
在HotSpot虚拟机中，对象在内存中的布局可以分为3块区域：对象头、实例数据和对齐填充。

1. HotSpot虚拟机的对象头包括两部分信息，第一部分用于存储对象自身的自身运行时数据（哈希码、GC分代年龄、锁状态标志等等），另一部分是类型指针，即对象指向它的类元数据的指针，虚拟机通过指针来确定这个对象是哪个类的实例。
2. 实例数据部分是对象真正存储的有效信息，也是在程序中所定义的各种类型的字段内容。
3. 对齐填充部分不是必然存在的，也没什么特别的含义，仅仅起占位作用。因为HotSpot虚拟机的自动内存管理系统要求对象起始地址必须是8字节的整数倍，换句话说就是对象的大小必须是8字节的整数倍。而对象头部分正好是8字节的倍数（1倍或2倍），因此，当对象实例数据部分没有对齐时，就需要通过对齐填充来补全。

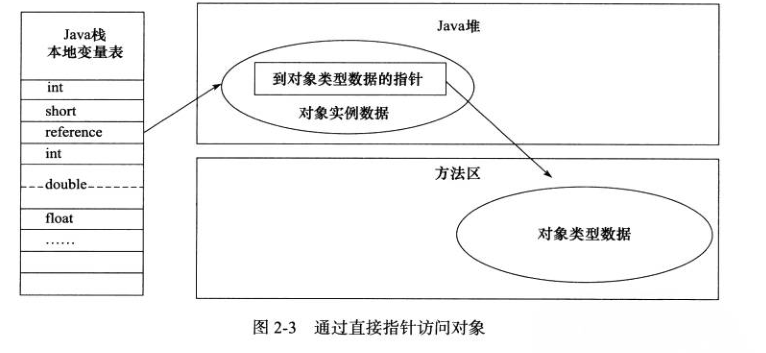
### 3.3对象的访问定位

建立对象就是为了使用对象，我们的Java程序通过栈上的reference数据来操作堆上的具体对象。对象的访问方式由虚拟机实现而定，目前主流的访问方式有使用句柄和直接指针两种：

1. 句柄：如果使用句柄的话，那么Java堆中将会划分出一块内存来作为句柄池，reference中存储的就是对象的句柄，那么Java堆中将会划分出一块内存来作为句柄池，reference中存储的就是对象的句柄地址，而句柄中包含了对象实例数据与类型数据各自的具体地址信息。



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



这两种对象访问方式各有优势。使用句柄来访问的最大好处是reference中存储的是稳定的句柄地址，在对象被移动时只会改变句柄中的实例数据指针，而reference本身不需要修改。使用直接指针访问方式最大的好处就是速度快，它节省了一次指针定位的时间开销。