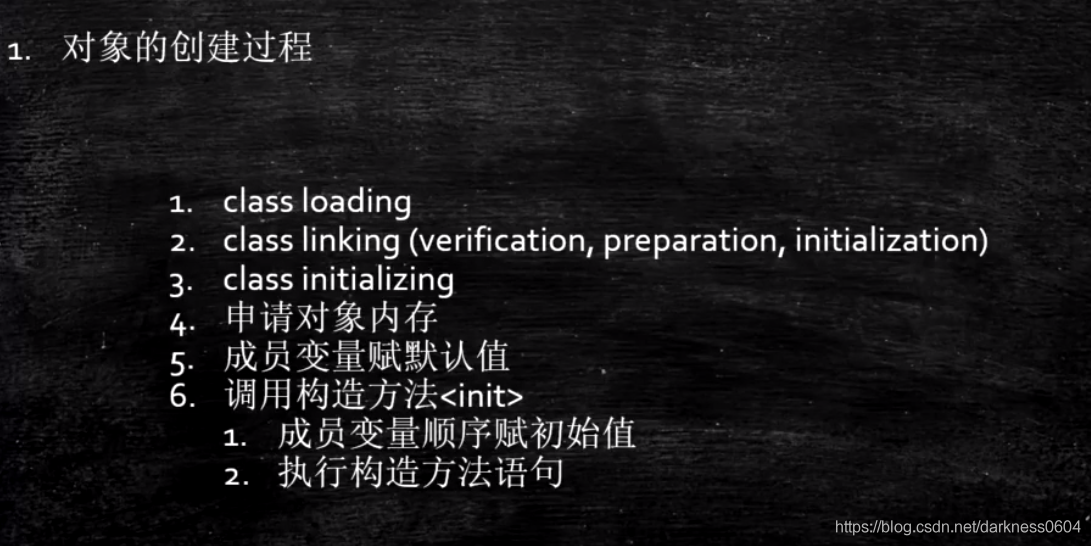
JVM第八天-JMM之对象的内存分布

**1、对象的创建过程**



**2、对象在内存中的存储布局**

这个要分为两种情况来看：

**1、普通对象**

一个一般对象的构成，分成这么四个部分：

1、对象头：在hotspot的实现中叫markword，占用 8个字节

2、ClassPointer指针：指向当前对象所属Class的一个指针，大小会根据JVM启动参数而定，-XX:+UseCompressedClassPointers 为4字节 不开启为8字节

3、实例数据：也就是一个对象里存放的那些具体的各个成员属性了。其中的引用类型变量的字节大小会根据JVM启动参数而定：-XX:+UseCompressedOops 为4字节 不开启为8字节。Oops： Ordinary Object Pointers

4、一般计算机单次读取内存的大小是成页的，不是说你用多少就只读多少，因此一般来说成某种倍数的大小更加方便一次性读取。在上面3项计算完字节大小后，会看一下字节大小离最近的8的倍数差多少字节，使用Padding对齐补全，使得大小变成8的倍数。

**2、数组对象**

1、对象头：在hotspot的实现中叫markword，占用 8个字节

2、ClassPointer指针：指向当前对象所属Class的一个指针，大小会根据JVM启动参数而定，-XX:+UseCompressedClassPointers 为4字节 不开启为8字节

3、数组长度：占用4个字节

4、数组数据：也就是一个数组对象里存放的那些具体的各个元素了。其中的引用类型变量的字节大小会根据JVM启动参数而定：-XX:+UseCompressedOops 为4字节 不开启为8字节。Oops： Ordinary Object Pointers

5、同普通对象一样，使用Padding对齐补全，使得大小变成8的倍数。

**Hotspot开启内存压缩的规则（64位机）**

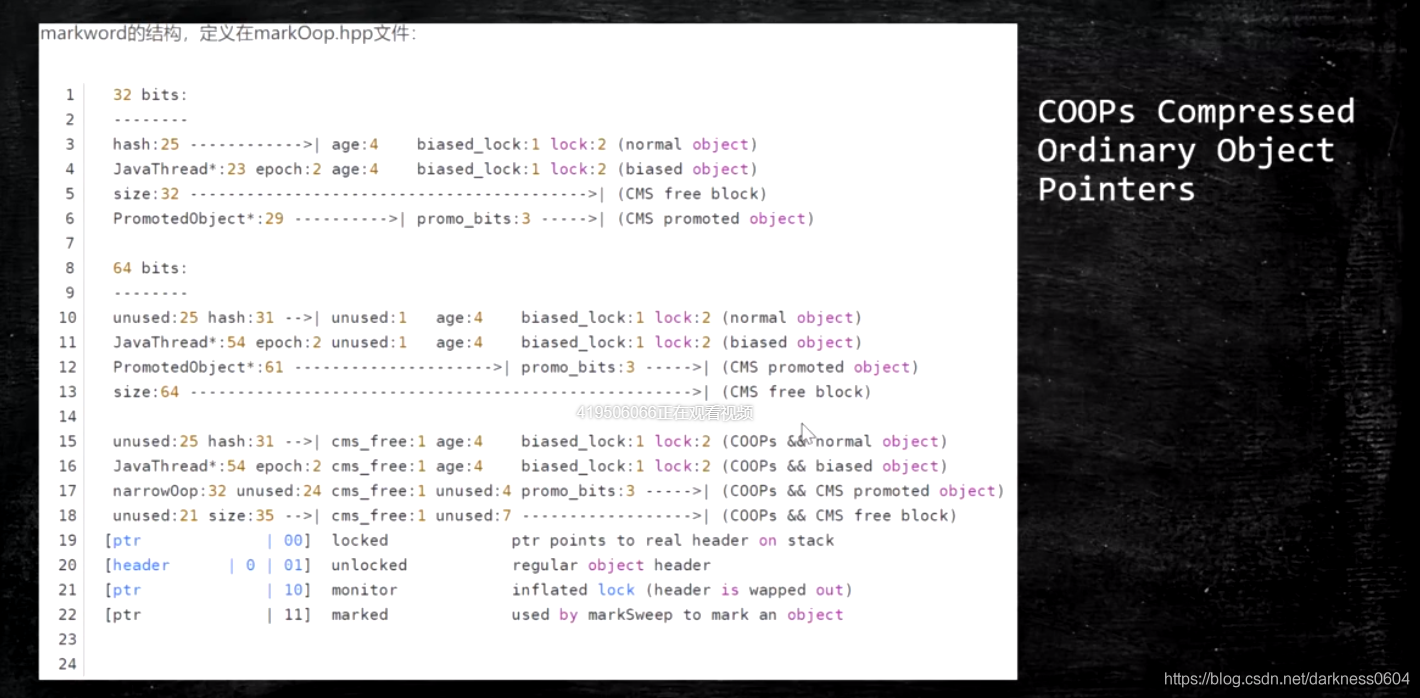
4G以下，直接砍掉高32位

4G - 32G，默认开启内存压缩 ClassPointers Oops

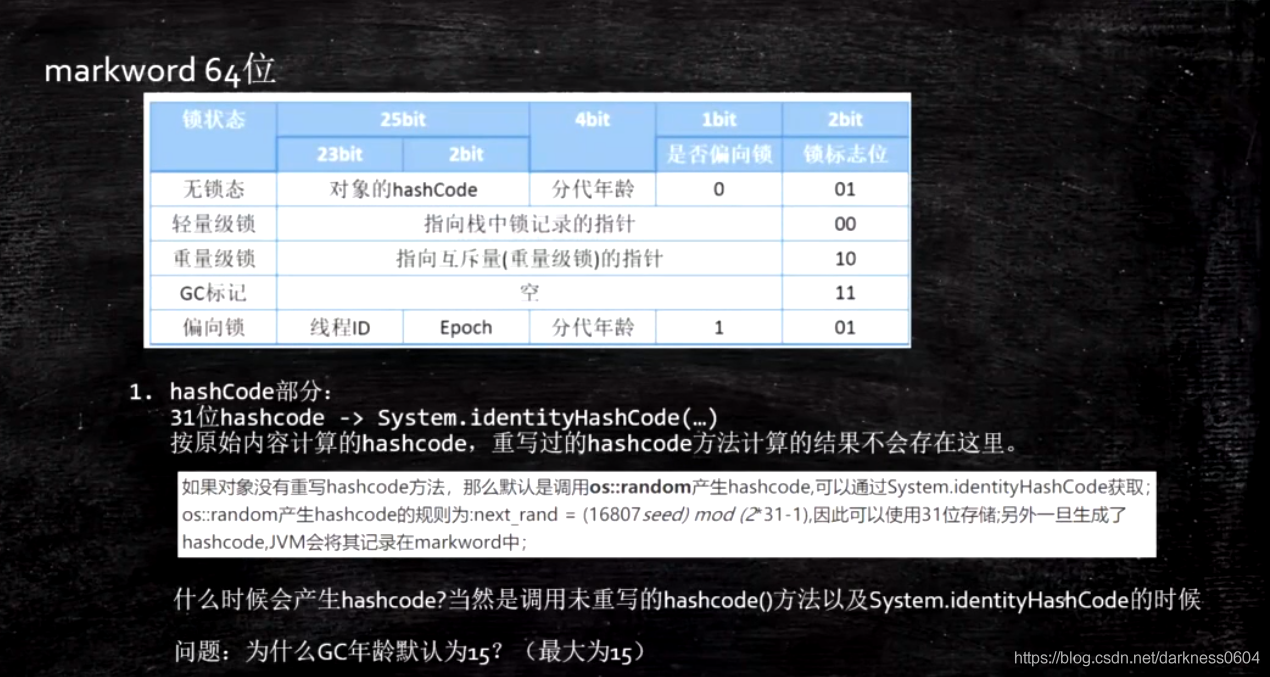
32G，压缩无效，使用64位

所以也不是内存越大，就越好。

**3、对象头具体包含内容**



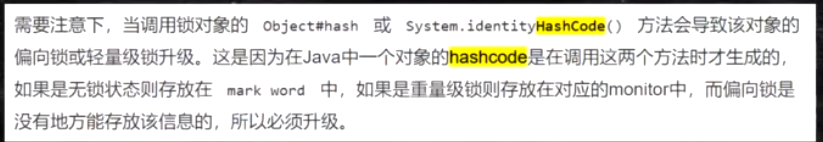
对象头比较复杂，不同的jdk，对象头里的内容都可能不同。



使用2位来标识锁的状态。（当然准确来说算上偏向锁的标识，应该是3位）

GC分代年龄用4位标识，4位能表示的最大数就是15，因此GC最大年龄是15。





当一个对象计算过identityHashCode之后，不能进入偏向锁状态，因为这个hashcode会存储在markwrd中，导致不能再放得下偏向锁的全部信息。

参考文章：

https://cloud.tencent.com/developer/article/1480590

https://cloud.tencent.com/developer/article/1484167

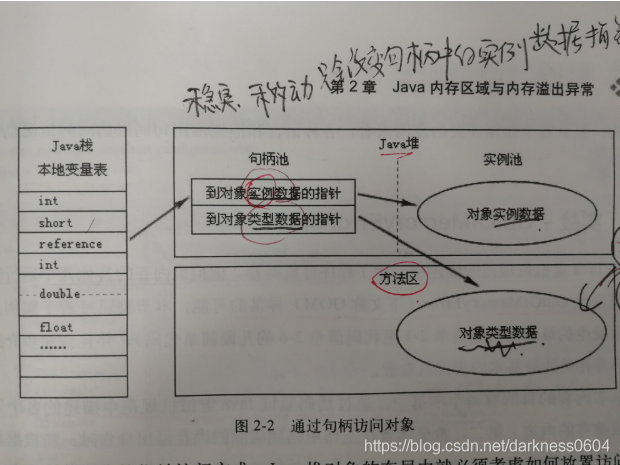
https://cloud.tencent.com/developer/article/1485795

https://cloud.tencent.com/developer/article/1482500

**对象定位**

对象定位分为两种：

**1、句柄池**

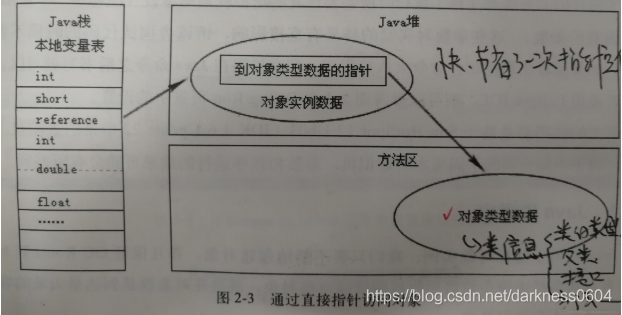


先通过句柄池进行寻找，再根据句柄池中的对应实例的指针找到真正的对象地址。

更利于GC垃圾回收。

但寻找对象要多一次寻址。

**2、直接指针**



直接一次寻址就可以找到对象。

但GC相较于第一种方式会稍微差一些。

参考文章：

https://blog.csdn.net/clover\_lily/article/details/80095580