# 概述

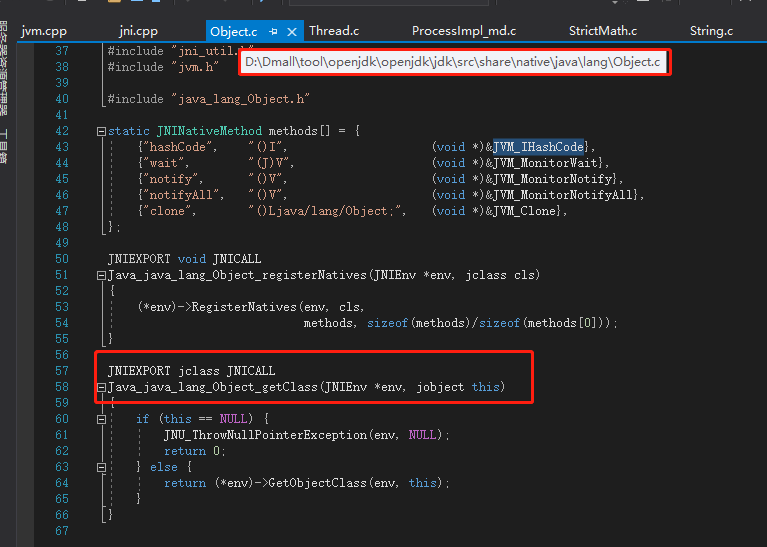
本文档主要记录以下几点：

* 学习Object类需要掌握各个方法作用；
* 关键字，比如native等

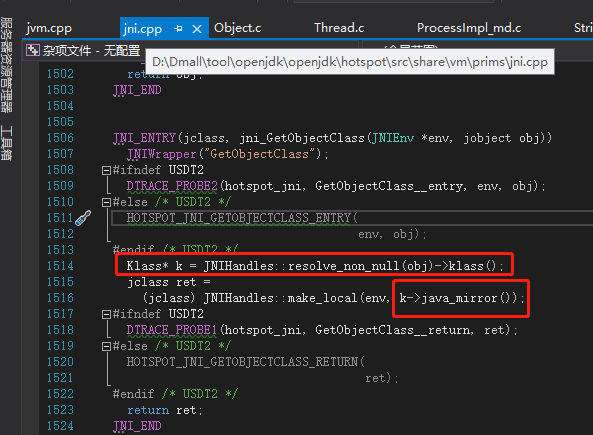
# 主要方法

## getClass

getClass方法返回对象所属类（直接类，并非父类）的Class对象。该方法是native方法，需要查看对应的jdk源码。源码参照如下，最核心的是jni.cpp中的GetObjectClass方法。

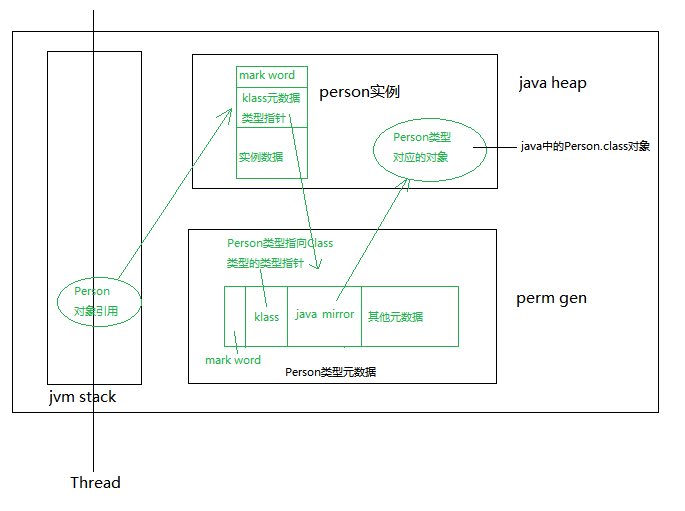


GetClassObject首先获取对象所对应类型的元数据klass，这个元数据存放位置正是对象头MarkWord之后的4个字节（32位系统）或者8个字节（64位系统）。然后通过这个klass指针获取对应的Class对象。



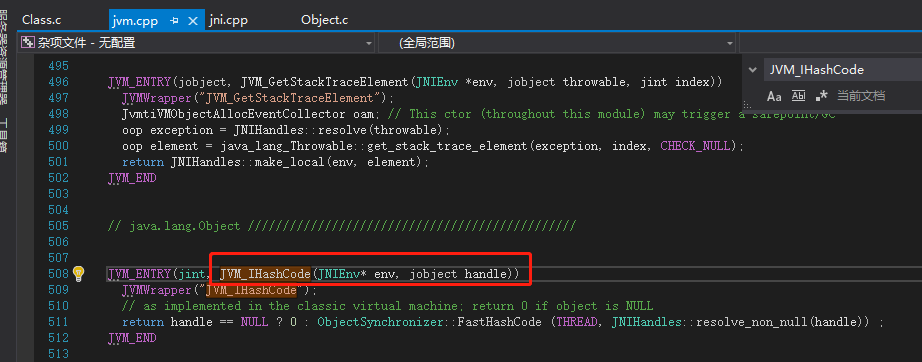
在内存中的映射关系如下



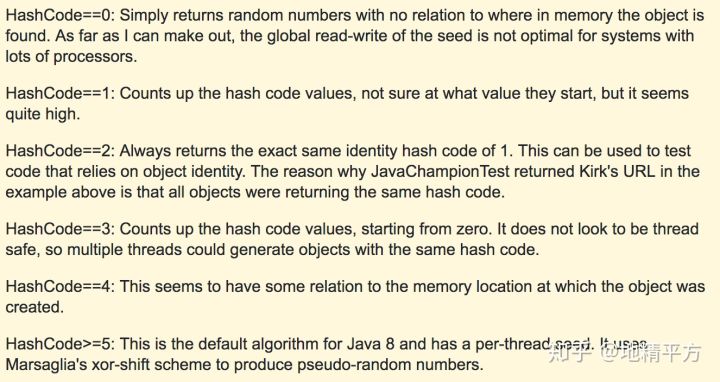


## hashCode

该方法对应的native实现在Object.c文件，而对应的具体实现是jvm.cpp中的JVM\_IhashCode方法。



openJdk内部提供多种hashcode计算算法，可以通过XX:hashCode参数来设置具体使用哪种算法。比如如果设置XX:hashCode=2，那如果调用了native的hashcode方法，则该方法始终返回固定值1。



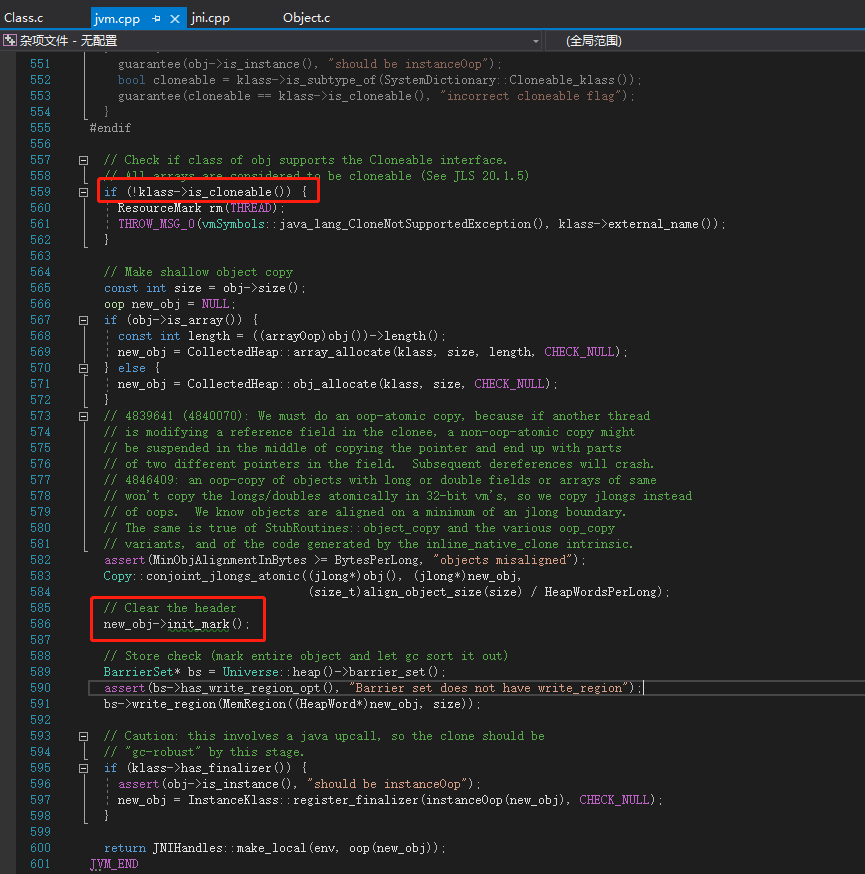
## clone

此方法是从已有对象复制一份完全一致的对象（浅拷贝）。通过native代码可以得知如下两个点：

* 被clone对象的类必须实现了Cloneable接口
* clone的新对象头部markWord是被clear了的

通过如上第二点可知，如果一个对象用于synchronized并获取到锁，在释放锁之前该对象clone了一份新的对象，该新对象的markword并不会被标记为获取锁状态，而是clear为非锁状态；

如此看来，一个对象执行clone时并不是完完全全的拷贝，拷贝的只是对象实体数据。



## wait

wait方法的native入口在jvm.cpp#JVM\_MonitorWait方法，其实质调用的是objectMonitor.cpp#wait方法。该方法主要思路是首先获取对应的ObjectMonitor，然后将线程封装为node节点并插入到对应的双向循环链表上（头插，这一步是加锁操作的），最后调用park释放该线程的cpu。

## notify

ObjectSynchronizer::notify方法,调用ObjectSynchronizer::inflate,object的对象中找到ObjectMonitor对象 ,然后调用方法ObjectMonitor::notify,,调用ObjectMonitor::DequeueWaiter 摘除第一个ObjectWaiter对象从\_WaitSet 的队列中,并把这个ObjectWaiter对象放入\_EntryList中,\_EntryList 存放的是ObjectWaiter的对象列表，列表的大小就是那些所有在等待这个对象锁的线程数。注意这里并没有调用ObjectMonitor::exit释放锁。

## notifyAll

# 参考