1. 同步容器
2. 使用

austen-concurrency-> com.austen.syncContainer(Demo\*)

1. Queue
   1. Queue
   2. BrokingQueue

LinkedBlockingQueue

ArrayBlockingQueue

DelayQueue

SynchronousQueue

* 1. TransferQueue

LinkedTransferQueue

二、线程

1）interrupt();

参考：austen-concurrency->com.austen.threadExample.interrupt

--用来优雅停止线程的函数

--jvm不推荐直接停止一个线程，一定要让一个线程执行完

--假设现在在阻塞->解阻塞

--假设线程是while(flag)->flag=false

三、线程池、

参考：austen-concurrency->com.austen.threadPool

四、synchronized

参考：austen-concurrency->com.austen.lockExample.sync

1）使用（demo\*）

2）synchronized锁的证明

1.6之前，所有的synchronized都是重量级锁

--偏向锁、轻量级锁、重量级锁

--证明方式。

1、修改linux源码-glibc（test.\*）

java中使用public native void gettid()获取jvm的线程ID

gettid使用c语言编写

--重量级锁：借助于OS函数来实现的锁。存在互斥场景中

jvm->lock()//C++

如：linux

pthread\_mutex\_t mt;//int

pthread\_mutex\_init(mt);

pthread\_mutex\_lock(mt);//1

pthread\_mutex\_unlock(mt);//0

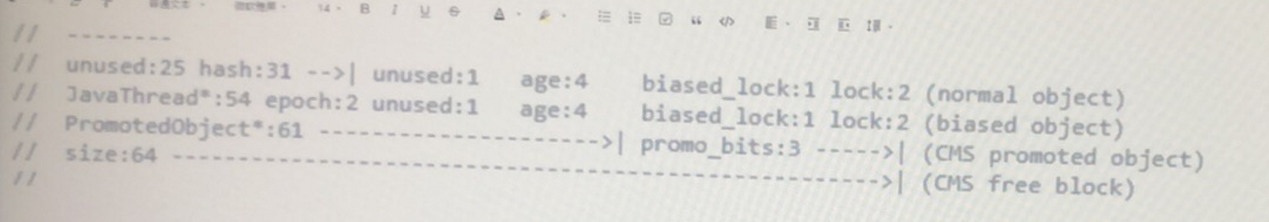
--轻量级锁：线程必须线程安全，但不一定有互斥。不会调用OS函数。存在没有资源竞争的情况下

--偏向锁：

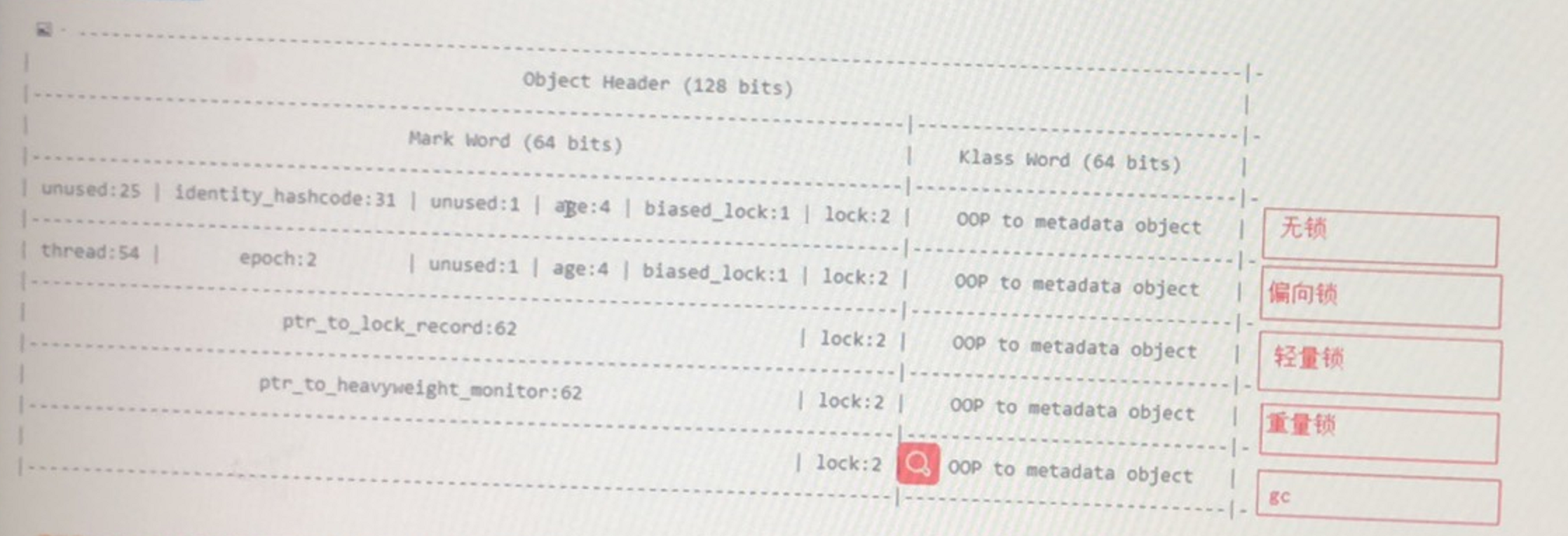
2、看对象头 （layout.\*）

使用openjdk.jol

Hotspot源码：



转换成可读表格：



object header,mark word,kclass pointer参考：<http://openjdk.java.net/groups/hotspot/docs/HotSpotGlossary.html>

分析结果：

Jvm源码当中得知：mark word ----64bit ==8byte

Jvm规范得知，对象头一共包含两个部分----mark word + klass word（4byte 32bit） 做了指针压缩，所以klass word就只有32bit

无锁：unused:25 hash:31 unused:1 age:4 0 01

偏向锁：javaThread\*:54 epoch:2 unused:1 age:4 1 01

轻量级锁：ptr\_to\_lock\_record:62 00

重量级锁：ptr\_to\_lock\_record:62 10

3）偏向锁批量重偏向(阈值20)

参考：austen-concurrency->com.austen.lockExample.sync.layout.JOLExample9

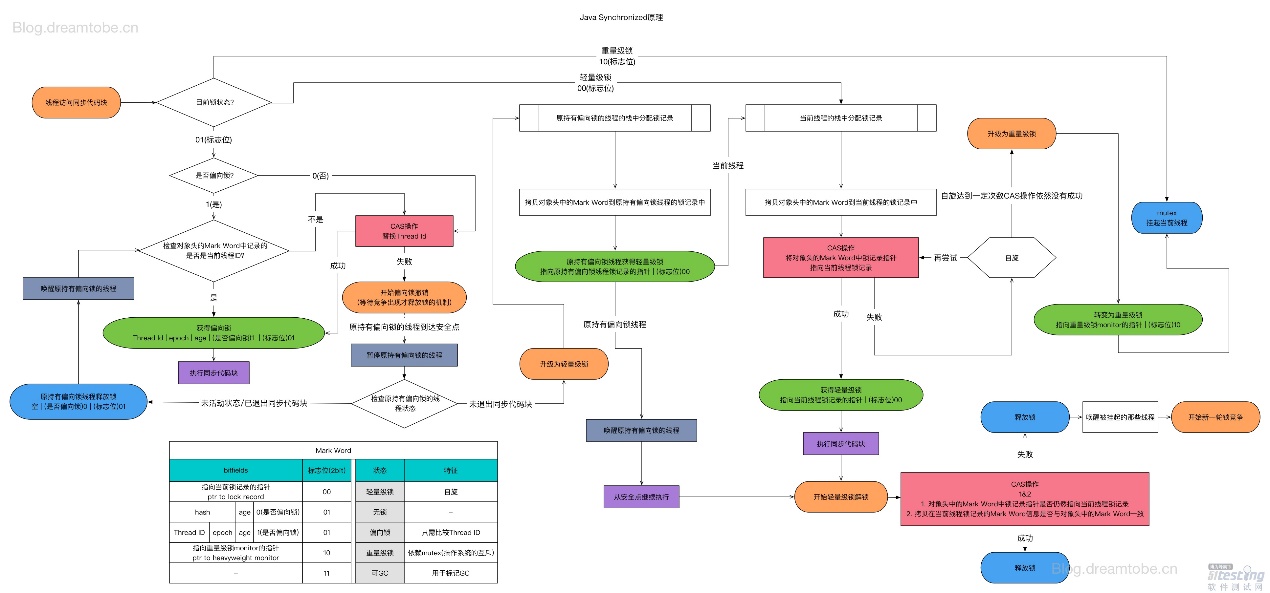
t1实例化多个对象(同一个类)并且同步了这些对象，t2也同步了这些对象，因为锁要升级，多次撤销偏向锁，jvm会认为接下来的对象需要批量重偏向，那么接下来的对象都是偏向不再是轻量。

4）批量撤销(阈值40)

参考：austen-concurrency->com.austen.lockExample.sync.layout.JOLExample11

5）锁的膨胀过程

参考：austen-concurrency->com.austen.lockExample.sync.layout1.\*



五、AQS

参考：austen-concurrency->com.austen.lockExample.aqs

1）使用场景（demo）