sun.misc.Cleaner实现堆外内存回收

原创

逆水行-周 于 2020-04-02 15:42:58 发布 854 收藏

简介

项目中采用了java+c的混合开发，通过jni进行了底层结构体的内存分配，将指针返回给java层保存，随后则可以通过传递指针值来操作底层代码。在java中，仍然需要手动释放jni分配出来的内存的。

如何让GC来自动管理jni内存

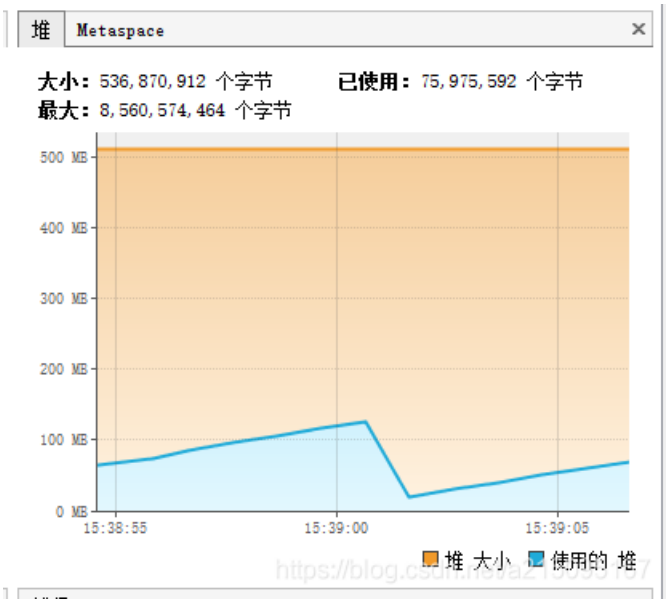
sun.misc.Cleaner可以做到！
话不多说，直接看例子。

示例

```java
import sun.misc.Cleaner;

public class Main {
    public static class FreeMemoryTask implements Runnable {
        private long address;

        public FreeMemoryTask(long address) {
            this.address = address;
        }

        @Override
        public void run() {
            // 有地址可以进行回收
            System.out.println("垃圾回收触发" + address);
        }
    }

    public static class MyObject {
        public static int a = 0;
        private long id = 0;
        private byte[] array;

        public MyObject() {
            id = ++a;
            array = new byte[1024 * 1024 * 10];
            System.out.println("创建了对象" + id);
        }

        public static void main(String[] args) throws Exception {

            while (true) {
//                System.gc();
                MyObject object = new MyObject();
                // Cleaner.create 检测对象回收，回收时触发FreeMemoryTask
                Cleaner.create(object, new FreeMemoryTask(object.id));
                Thread.sleep(1000);
            }
        }
    }
}
```

运行效果

```
创建了对象1
创建了对象2
创建了对象3
创建了对象4
创建了对象5
创建了对象6
创建了对象7
创建了对象8
创建了对象9
创建了对象10
垃圾回收触发1
垃圾回收触发2
垃圾回收触发7
垃圾回收触发6
垃圾回收触发5
垃圾回收触发3
垃圾回收触发4
垃圾回收触发10
垃圾回收触发9
垃圾回收触发8
创建了对象11
创建了对象12
创建了对象13
创建了对象14
创建了对象15
创建了对象16
创建了对象17
```



说明

在发生GC时，就会触发FreeMemoryTask。只需要将指针地址交给它，就可以进行垃圾回收。

优点

1. 比finalizer更轻量更好用。
2. 遇到该对象共享给多个线程，大部分时间只读，少部分时间更新时，释放需要用读写锁确保安全。但采用Cleaner则无需锁，更加高效。

netty中的DirectByteBuffer

通过一系列的源码跟踪，最终会发现，其回收相关的代码如下：

```
DirectByteBuffer(int cap) {                   // package-private
    ……
    Cleaner.create(this, new Deallocator(base, size, cap));
    ……
}

private static class Deallocator implements Runnable
{

    private static Unsafe unsafe = Unsafe.getUnsafe();

    private long address;
    private long size;
    private int capacity;

    private Deallocator(long address, long size, int capacity) {
        assert (address != 0);
        this.address = address;
        this.size = size;
        this.capacity = capacity;
    }

    public void run() {
        if (address == 0) {
            // Paranoia
            return;
        }
        unsafe.freeMemory(address);
        address = 0;
        Bits.unreserveMemory(size, capacity);
    }

}
```

从这段源码来看，也就是说，即使忘记手动释放资源，在对象能被GC的情况下，应该也是能够通过正常的GC回收掉堆外内存的，这点暂不确认。