虚引用与堆外内存研究(基于DirectByteBuffer源码)

背景：

堆外内存是相对于堆内内存的一个概念。堆内内存是由JVM所管控的Java进程内存。那么堆外内存就是存在于JVM管控之外的一块内存区域，因此它是不受JVM的管控。

DirectByteBuffer是通过虚引用(Phantom Reference)来实现堆外内存的释放的。

> g关于虚引用的作用
> PhantomReference 是所有“弱引用”中最弱的引用类型。不同于软引用和弱引用，虚引用无法通过 get() 方法来取得目标对象的强引用从而使用目标对象，观察源码可以发现 get() 被重写为永远返回 null。
> 那虚引用到底有什么作用？其实虚引用主要被用来 跟踪对象被垃圾回收的状态，通过查看引用队列中是否包含对象所对应的虚引用来判断它是否 即将被垃圾回收，从而采取行动。它并不被期待用来取得目标对象的引用，而目标对象被回收前，它的引用会被放入一个 ReferenceQueue 对象中，从而达到跟踪对象垃圾回收的作用。

堆内存

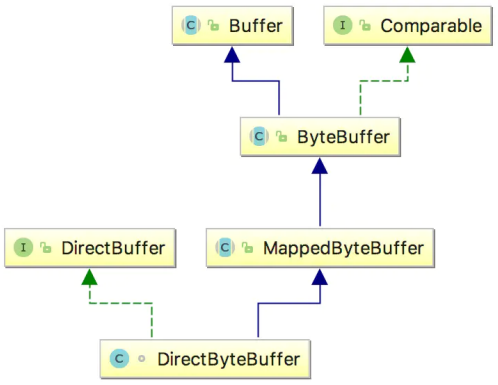
```
ByteBuffer buffer = ByteBuffer.allocate(1024);
```

字节数组保存数据HeapByteBuffer

堆外内存

```
// 注意这里是 allocateDirect 方法而不是allocate，返回值是DirectByteBuffer
ByteBuffer directByteBuffer = ByteBuffer.allocateDirect(1024);

//另外 FileChannel的map方法返回一个MappedByteBuffer类型的对象，实际上
该对象也是一个DirectByteBuffer类型的直接内存对象。
```



Buffer类有个：long address;用于保存堆外空间地址

```
// Used only by direct buffers
// NOTE: hoisted here for speed in JNI GetDirectBufferAddress
//仅用于直接缓冲区
//在JNI的GetDirectBufferAddress中提升速度
//注意这个Buffer类的address属性明确 注明了 该属性只会被DirectBuffer使用
//也就是直接内存（对外内存）
long address;
```

另外DirectByteBuffer中有两个重要的属性：

```
// Cached unsafe-access object 缓存unsafe-access对象
protected static final Unsafe unsafe = Bits.unsafe();
private final Cleaner cleaner;
```

其构造器实现如下：

```
// Primary constructor
//
DirectByteBuffer(int cap) { // package-private

 super(-1, 0, cap, cap);
 boolean pa = VM.isDirectMemoryPageAligned();
 int ps = Bits.pageSize();
 long size = Math.max(1L, (long)cap + (pa ? ps : 0));
 Bits.reserveMemory(size, cap);

 long base = 0;
 try {
 //这里使用unsafe分配内存
 base = unsafe.allocateMemory(size);
 } catch (OutOfMemoryError x) {
 Bits.unreserveMemory(size, cap);
 throw x;
 }
 unsafe.setMemory(base, size, (byte) 0);
 //记录基准地址 address
 if (pa && (base % ps != 0)) {
 // Round up to page boundary
 address = base + ps - (base & (ps - 1));
 } else {
 address = base;
 }
 //这里创建cleaner对象
 cleaner = Cleaner.create(this, new Deallocator(base, size, cap));
 att = null;


}
```

unsafe来分配以及最后回收空间

构建Cleaner对象，继承虚引用，将当前堆外内存以及垃圾清理线程对象传递过去，GC发生后，调用Deallocator的clean方法，内部有调用unsafe来回收堆外内存。

DirectByteBuffer和虚引用之间的关系

Cleaner类继承自PhantomReference

```
package sun.misc;

import ...

public class Cleaner extends PhantomReference<Object> {
    private static final ReferenceQueue<Object> dummyQueue = new ReferenceQueue();
    private static Cleaner first = null;
    private Cleaner next = null;
```

那么我们就来分析一下PhantomReference和Clear类

PhantomReference

```
public class PhantomReference<T> extends Reference<T> {

    Returns this reference object's referent. Because the referent of a phantom reference is
    always inaccessible, this method always returns null.
    Returns: null
    public T get() { return null; }

    Creates a new phantom reference that refers to the given object and is registered with the
    given queue.
    It is possible to create a phantom reference with a null queue, but such a reference is
    completely useless: Its get method will always return null and, since it does not have a
    queue, it will never be enqueued.
    Params: referent - the object the new phantom reference will refer to
            q - the queue with which the reference is to be registered, or null if registration
            is not required
    public PhantomReference(T referent, ReferenceQueue<? super T> q) { super(referent, q); }

}
```

PhantomReference继承自Reference，值得注意的是其get方法没有返回Reference内部包装的引用的目标对象，而是永远返回了null。也就是说我们无法通过PhantomReference的get方法获取到目标对象的强引用。具体为什么另做分析

Cleaner

参考源码文档注释