堆外内存（直接内存）

1、堆外内存定义

内存对象分配在Java虚拟机的堆以外的内存，这些内存直接受操作系统管理（而不是虚拟机），这样做的结果就是能在一定程度上减少垃圾回收对应用程序造成的影响。使用未公开的Unsafe和NIO包下ByteBuffer来创建堆外内存。

2、为什么使用堆外内存

1、减少了垃圾回收
使用堆外内存的话，堆外内存是直接受操作系统管理（而不是虚拟机）。这样做的结果就是能保持一个较小的堆内内存，以减少垃圾收集对应用的影响。

2、提升复制速度
堆内内存由JVM管理，属于“用户空间”；而堆外内存由OS管理，属于“内核空间”。如果从堆内向磁盘写数据时，数据会被先复制到堆外内存，即“内核空间”，然后再由OS写入磁盘，使用堆外内存避免这个操作

3、堆外内存申请

JDK的ByteBuffer类提供了一个接口allocateDirect（int capacity）进行堆外内存的申请，底层通过unsafe.allocateMemory(size)实现。Netty、Mina等框架提供的接口也是基于ByteBuffer封装的。

```
DirectByteBuffer(int cap) {

        super(-1, 0, cap, cap);
        //内存是否按页分配对齐
        boolean pa = VM.isDirectMemoryPageAligned();
        //获取每页内存大小
        int ps = Bits.pageSize();
        //分配内存的大小，如果是按页对齐方式，需要再加一页内存的容量
        long size = Math.max(1L, (long)cap + (pa ? ps : 0));
        //用Bits类保存总分配内存(按页分配)的大小和实际内存的大小
        Bits.reserveMemory(size, cap);

        long base = 0;
        try {
           //在堆外内存的基地址，指定内存大小
            base = unsafe.allocateMemory(size);
        } catch (OutOfMemoryError x) {
            Bits.unreserveMemory(size, cap);
            throw x;
```

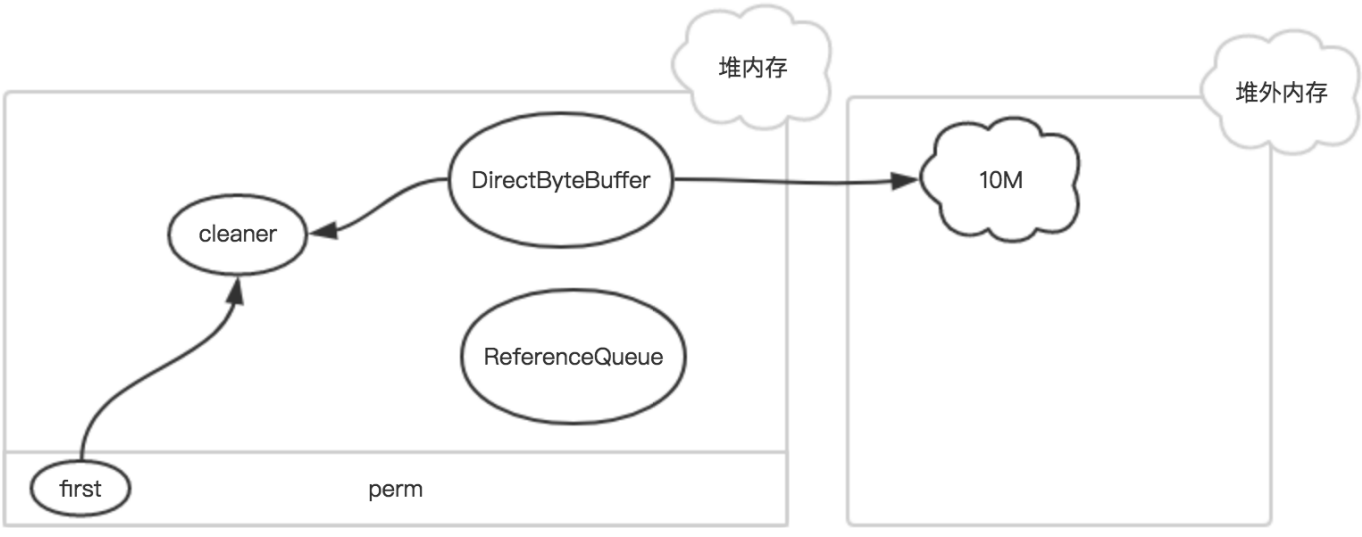
```
        }
        unsafe.setMemory(base, size, (byte) 0);
        //计算堆外内存的基地址
        if (pa && (base % ps != 0)) {
            // Round up to page boundary
            address = base + ps - (base & (ps - 1));
        } else {
            address = base;
        }
        cleaner = Cleaner.create(this, new Deallocator(base, size, cap));
        att = null;
    }
```

注：unsafe.allocateMemory(size)最底层是通过malloc方法申请的，但是这块内存需要进行手动释放，JVM并不会进行回收，幸好Unsafe提供了另一个接口freeMemory可以对申请的堆外内存进行释放。

4、堆外内存释放

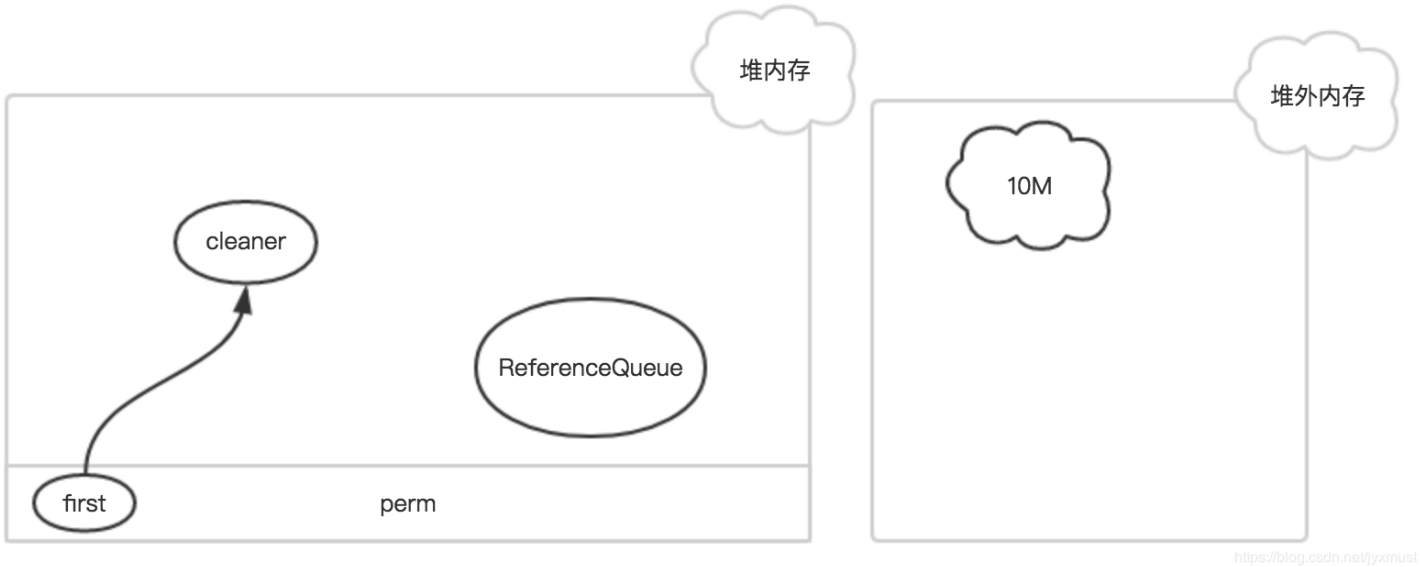
JDK中使用DirectByteBuffer对象来表示堆外内存，每个DirectByteBuffer对象在初始化时，都会创建一个对应的Cleaner对象，这个Cleaner对象会在合适的时候执行unsafe.freeMemory(address)，从而回收这块堆外内存。

当初始化一块堆外内存时，对象的引用关系如下：



其中first是Cleaner类的静态变量，Cleaner对象在初始化时会被添加到Clener链表中，和first形成引用关系，ReferenceQueue是用来保存需要回收的Cleaner对象。

如果该DirectByteBuffer对象在一次GC中被回收了



此时，只有Cleaner对象唯一保存了堆外内存的数据（开始地址、大小和容量），在下一次FGC时，把该Cleaner对象放入到ReferenceQueue中，并触发clean方法。

Cleaner对象的clean方法主要有两个作用：
1、把自身从Clener链表删除，从而在下次GC时能够被回收
2、释放堆外内存

如果JVM一直没有执行FGC的话，无效的Cleaner对象就无法放入到ReferenceQueue中，从而堆外内存也一直得不到释放，内存岂不是会爆？

其实在初始化DirectByteBuffer对象时，如果当前堆外内存的条件很苛刻时，会主动调用System.gc()强制执行FGC。

参考链接：https://www.jianshu.com/p/35cf0f348275