概述

元空间使用的直接内存;

关于内存空间的称呼变化：JDK 1.4 之前的称呼 native heap 转为现在的称呼 directory memory 。之所以 heap 前加 native 来修饰，是因为要让其和虚拟机规范中的内存 heap，而现在称呼 directory memory 是因为我们能够直接通过引用访问对象，消除了拷贝操作。

- 不是虚拟机运行时数据区的一部分, 也不是 <<java虚拟机规范>>中定义的内存区域;
- 直接内存是在Java堆外的、直接向系统申请的内存区间;
- **来源于NIO, 通过存在堆中的DirectByteBuffer操作Native内存;**
- 通常, 访问直接内存的速度会优于Java堆, 即读写性能高;

  因此处于性能考虑, 读写频繁的场合可能会考虑使用直接内存;

  Java的NIO库允许Java程序使用直接内存, 用于数据缓冲区;

代码示例:

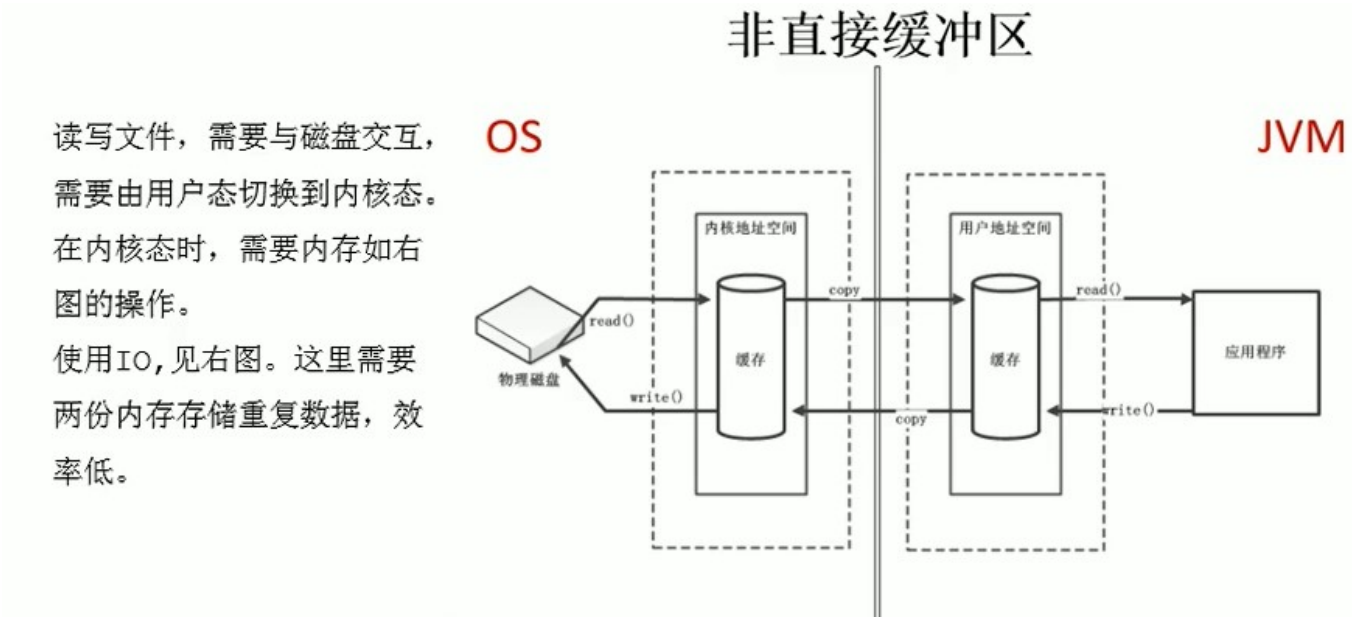
```
import java.nio.ByteBuffer;
import java.util.Scanner;

/**
 *  IO                  NIO (New IO / Non-Blocking IO)
 *  byte[] / char[]     Buffer
 *  Stream              Channel
 *
 * 查看直接内存的占用与释放
 * @create 2020  0:22
 */
public class BufferTest {
    private static final int BUFFER = 1024 * 1024 * 1024;//1GB

    public static void main(String[] args){
        //直接分配本地内存空间
```

```
17          ByteBuffer byteBuffer = ByteBuffer.allocateDirect(BUFFER);
18          System.out.println("直接内存分配完毕，请求指示！");
19
20          Scanner scanner = new Scanner(System.in);
21          scanner.next();
22
23          System.out.println("直接内存开始释放！");
24          byteBuffer = null;
25          System.gc();
26          scanner.next();
27      }
28  }
```

使用本地内存读写数据的测试

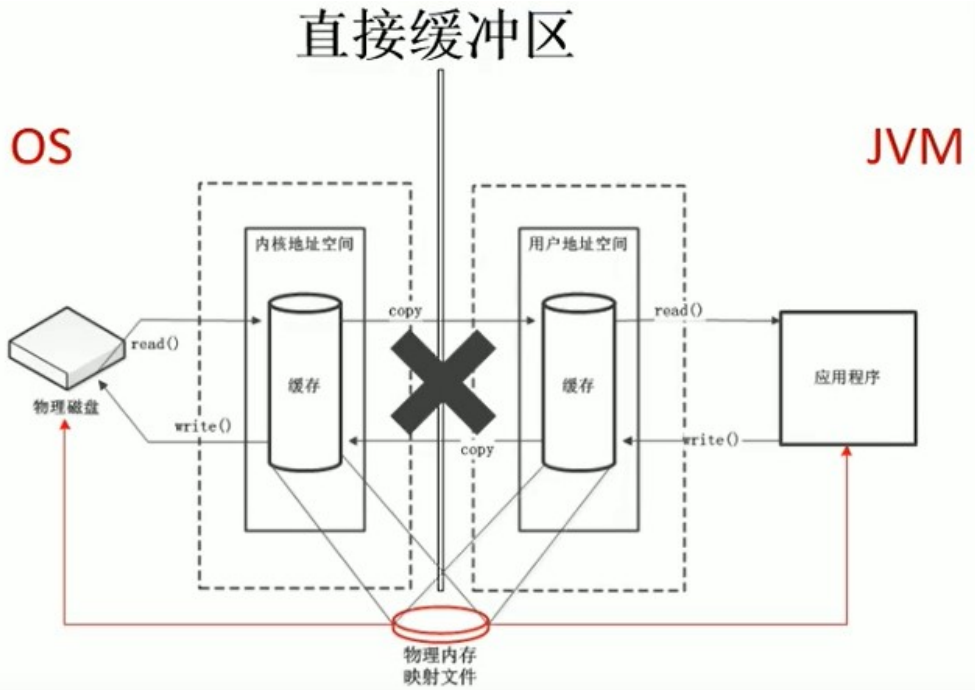
也可以认为 IO 和 NIO 的区别;

非直接缓冲区



直接缓冲区

使用NIO时，如右图。操作系统划出的直接缓存区可以被java代码直接访问，只有一份。NIO适合对大文件的读写操作。




```java
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;

/**
 * @author shkstart  shkstart@126.com
 * @create 2020  0:04
 */
public class BufferTest1 {

    private static final String TO = "F:\\test\\异界BD中字.mp4";
    private static final int _100Mb = 1024 * 1024 * 100;

    public static void main(String[] args) {
        long sum = 0;
        String src = "F:\\test\\异界BD中字.mp4";
        for (int i = 0; i < 3; i++) {
            String dest = "F:\\test\\异界BD中字_" + i + ".mp4";
//            sum += io(src,dest);//54606
            sum += directBuffer(src,dest);//50244
        }
```

```
24
25          System.out.println("总花费的时间为：" + sum );
26      }
27
28      private static long directBuffer(String src,String dest) {
29          long start = System.currentTimeMillis();
30
31          FileChannel inChannel = null;
32          FileChannel outChannel = null;
33          try {
34              inChannel = new FileInputStream(src).getChannel();
35              outChannel = new FileOutputStream(dest).getChannel();
36
37              ByteBuffer byteBuffer = ByteBuffer.allocateDirect(_100Mb);
38              while (inChannel.read(byteBuffer) != -1) {
39                  byteBuffer.flip();//修改为读数据模式
40                  outChannel.write(byteBuffer);
41                  byteBuffer.clear();//清空
42              }
43          } catch (IOException e) {
44              e.printStackTrace();
45          } finally {
46              if (inChannel != null) {
47                  try {
48                      inChannel.close();
49                  } catch (IOException e) {
50                      e.printStackTrace();
51                  }
52
53              }
54              if (outChannel != null) {
55                  try {
56                      outChannel.close();
57                  } catch (IOException e) {
58                      e.printStackTrace();
59                  }
60
61              }
62          }
63
```

```
        long end = System.currentTimeMillis();
        return end - start;

    }

    private static long io(String src,String dest) {
        long start = System.currentTimeMillis();

        FileInputStream fis = null;
        FileOutputStream fos = null;
        try {
            fis = new FileInputStream(src);
            fos = new FileOutputStream(dest);
            byte[] buffer = new byte[_100Mb];
            while (true) {
                int len = fis.read(buffer);
                if (len == -1) {
                    break;
                }
                fos.write(buffer, 0, len);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }

            }
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }

            }
```

```
        }


        long end = System.currentTimeMillis();

        return end - start;
    }
}
```

直接内存的OOM和内存设置大小

- 也可能导致 OutOfMemoryError 异常;
- 由于直接内存在Java堆外, 因此它的大小不会直接受限于-Xmx指定的最大堆大小, 但是系统内存是有限的, Java堆和直接内存的总和依然受限于操作系统能给出的最大内存;
- 缺点

  分配回收成本较高

  不受JVM内存回收管理
- 直接内存大小可以通过 MaxDirectMemorySize设置
- 如果不指定, 默认与堆的最大值 -Xmx参数值一致;

代码示例:

```
import java.nio.ByteBuffer;
import java.util.ArrayList;

/**
 * 本地内存的OOM:  OutOfMemoryError: Direct buffer memory
 *
 * @author shkstart  shkstart@126.com
 * @create 2020  0:09
 */
public class BufferTest2 {
    private static final int BUFFER = 1024 * 1024 * 20;//20MB
```

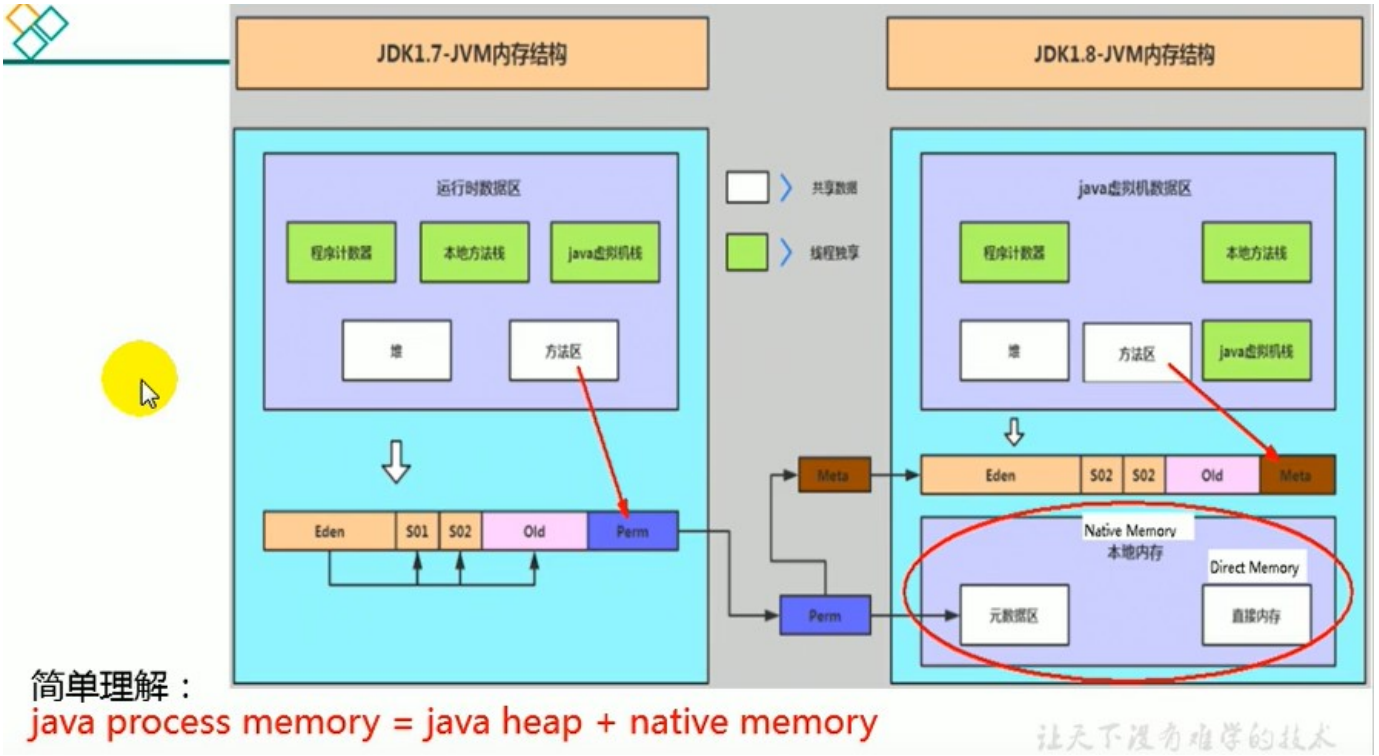
```
12
13      public static void main(String[] args) {
14          ArrayList<ByteBuffer> list = new ArrayList<>();
15
16          int count = 0;
17          try {
18              while(true){
19                  ByteBuffer byteBuffer = ByteBuffer.allocateDirect(BUFFER);
20                  list.add(byteBuffer);
21                  count++;
22                  try {
23                      Thread.sleep(100);
24                  } catch (InterruptedException e) {
25                      e.printStackTrace();
26                  }
27              }
28          } finally {
29              System.out.println(count);
30          }
31
32      }
33  }
```

```
181
Exception in thread "main" java.lang.OutOfMemoryError: Direct buffer memory
    at java.nio.Bits.reserveMemory(Bits.java:693)
    at java.nio.DirectByteBuffer.<init>(DirectByteBuffer.java:123)
    at java.nio.ByteBuffer.allocateDirect(ByteBuffer.java:311)
    at com.atguigu.java.BufferTest2.main(BufferTest2.java:20)

Process finished with exit code 1
```

unsafe 类这个例子 直接反射拿的;

```
1  import sun.misc.Unsafe;
2
3  import java.lang.reflect.Field;
4
5  /**
```

```
 * -Xmx20m -XX:MaxDirectMemorySize=10m
 * @author shkstart  shkstart@126.com
 * @create 2020  0:36
 */
public class MaxDirectMemorySizeTest {
    private static final long _1MB = 1024 * 1024;

    public static void main(String[] args) throws IllegalAccessException {
        Field unsafeField = Unsafe.class.getDeclaredFields()[0];
        unsafeField.setAccessible(true);
        Unsafe unsafe = (Unsafe)unsafeField.get(null);
        while(true){
            unsafe.allocateMemory(_1MB);
        }

    }
}
```



简单理解;