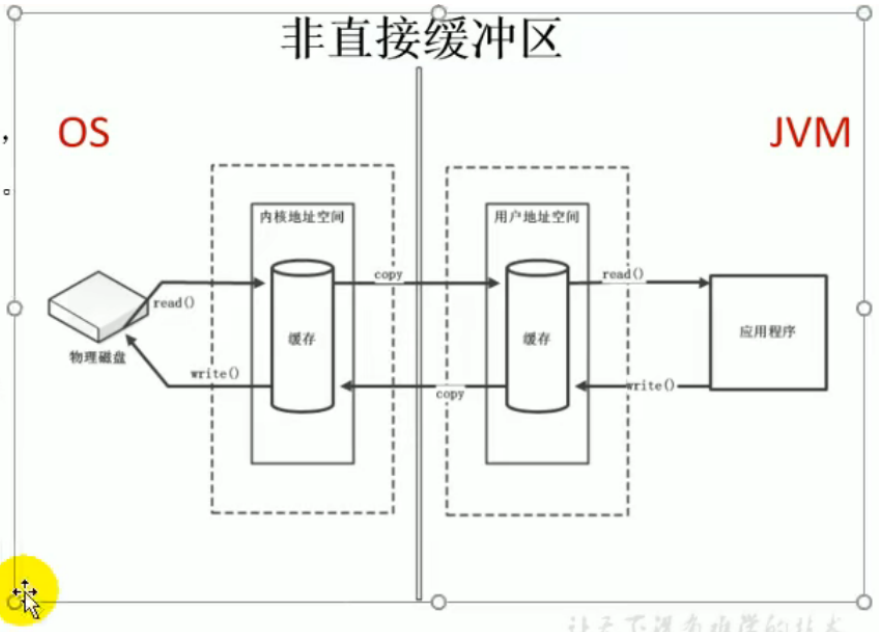
1、概述

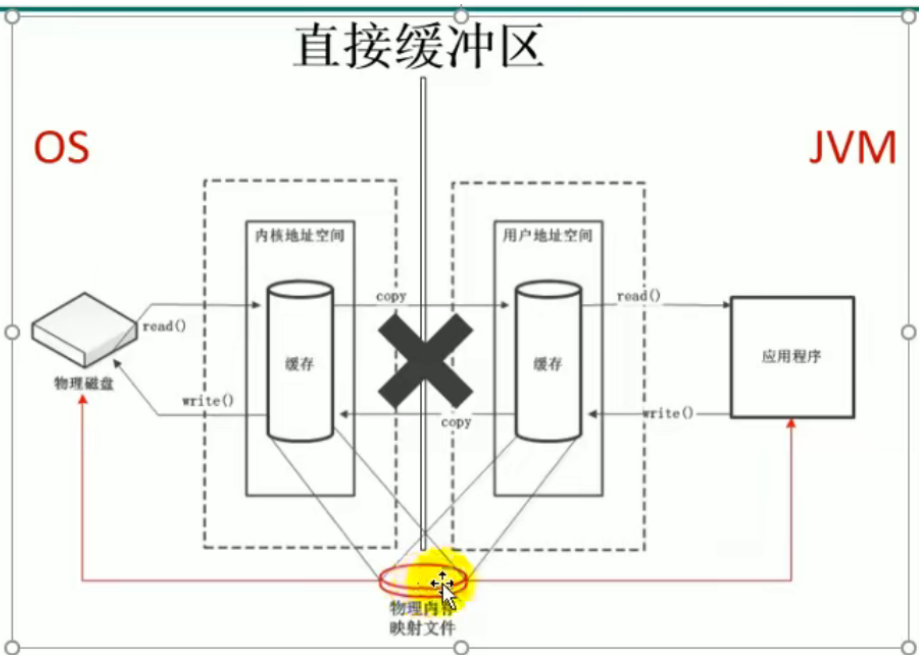
- 不是虚拟机运行时数据区的一部分，也不是《Java虚拟机规范》中定义的内存区域。
- 直接内存是在Java堆外的、直接向系统申请的内存区间。
- 来源于NIO，通过存在堆中的DirectByteBuffer操作Native内存
- 通常，访问直接内存的速度会优于Java堆。即读写性能高。
  - 因此出于性能考虑，读写频繁的场合可能会考虑使用直接内存。
  - Java的NIO库允许Java程序使用直接内存，用于数据缓冲区

读写文件，需要与磁盘交互，需要由用户态切换到内核态。在内核态时，需要内存如右图的操作。

使用IO,见右图。这里需要两份内存存储重复数据，效率低。



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
package com.qy;

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

/**
 * @author 千祎来了
 * @date 2022-06-12 10:31
 */
public class BufferTest {
 private static final int BUFFER = 1024 * 1024* 1024; // 1GB

 public static void main(String[] args) {
 // 通过allocateDirect分配直接内存
 ByteBuffer byteBuffer = ByteBuffer.allocateDirect(BUFFER);
 System.out.println("直接内存分配完毕，请求指示！");

 Scanner scanner = new Scanner(System.in);
 // 接受，将程序阻塞
 scanner.next();

 System.out.println("直接内存开始释放！");
 byteBuffer = null;
 System.gc();
 }
}
```

```java
package com.qy;

import javax.xml.transform.sax.SAXTransformerFactory;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;

/**
 * @author 千祎来了
 * @date 2022-06-12 10:53
 */
public class BufferTest1 {

 private static final int _100Mb = 1024 * 1024 * 100;

 public static void main(String[] args) {
 long sum = 0;
 String src = "E:\\test\\coffee.jpg";
 for (int i = 0; i < 3; i ++) {
 String dest = "E:\\test\\coffeebackup" + i + ".jpg";
 sum += io(src, dest); // 花费385
// sum += directBuffer(src, dest); // 花费256
 }
 System.out.println("花费时间: " + sum);
 }

 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 (Exception e) {
e.printStackTrace();
}
long end = System.currentTimeMillis();
return end - start;
}

// 直接内存
private static long directBuffer(String src, String dest) {
long start = System.currentTimeMillis();

FileChannel inChannel = null;
FileChannel outChannel = null;

try {
inChannel = new FileInputStream(src).getChannel();
outChannel = new FileOutputStream(dest).getChannel();

ByteBuffer byteBuffer = ByteBuffer.allocateDirect(_100Mb);
while (inChannel.read(byteBuffer) != -1) {
byteBuffer.flip(); // 修改为读数据模式
outChannel.write(byteBuffer);
byteBuffer.clear(); // 清空
}
} catch (Exception e) {
e.printStackTrace();
} finally {
if (inChannel != null) {
try {
inChannel.close();
} catch (IOException e) {
e.printStackTrace();
}
}
```

```
80  if (outChannel != null) {
81  try {
82  outChannel.close();
83  } catch (IOException e) {
84  e.printStackTrace();
85  }
86  }
87  }
88
89  long end = System.currentTimeMillis();
90  return end - start;
91  }
92 }
```

- 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