# NIO API

# 2 Buffer

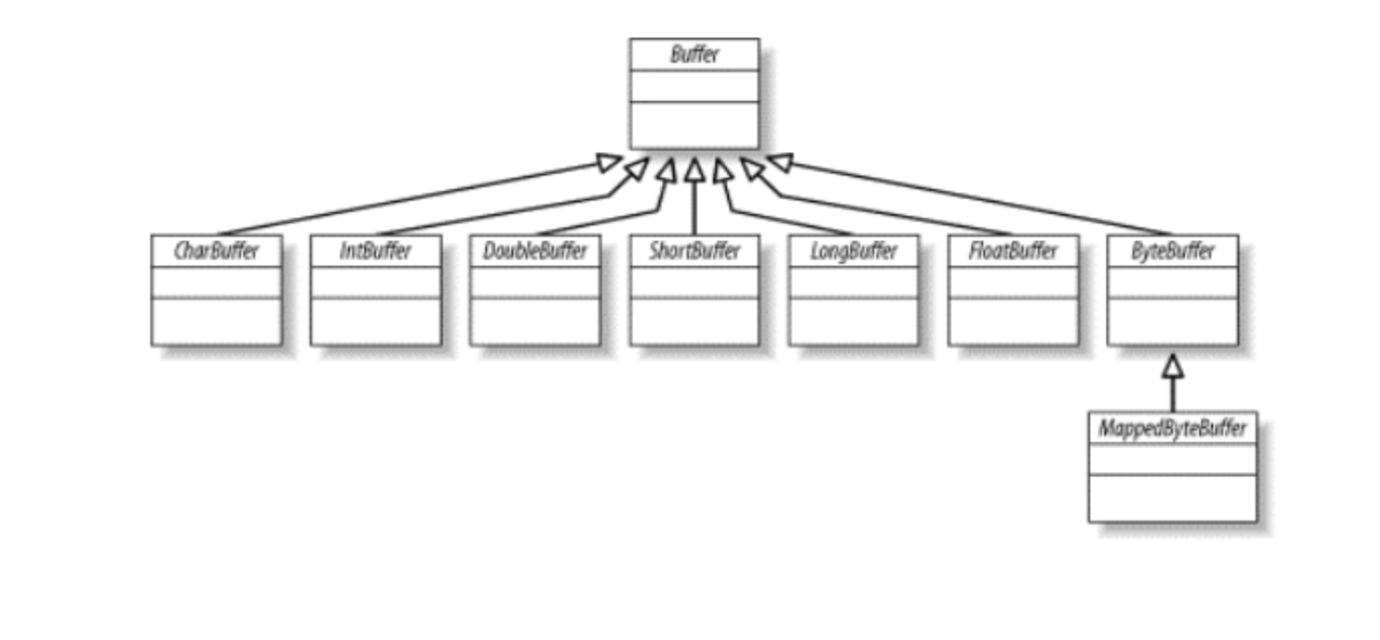
本质上，缓冲区是就是一个数组。所有的缓冲区都具有四个属性来提供关于其所包含的数组的信息。它们是：

1.容量（Capacity） 缓冲区能够容纳的数据元素的最大数量。容量在缓冲区创建时被设定，并且永远不能被改变。

2.上界（Limit） 缓冲区里的数据的总数，代表了当前缓冲区中一共有多少数据。

3.位置（Position） 下一个要被读或写的元素的位置。Position会自动由相应的 get( )和 put( )函数更新。

4.标记（Mark） 一个备忘位置。用于记录上一次读写的位置。一会儿，我会通过reset方法来说明这个属性的含义。



## 2.1 缓冲区基础

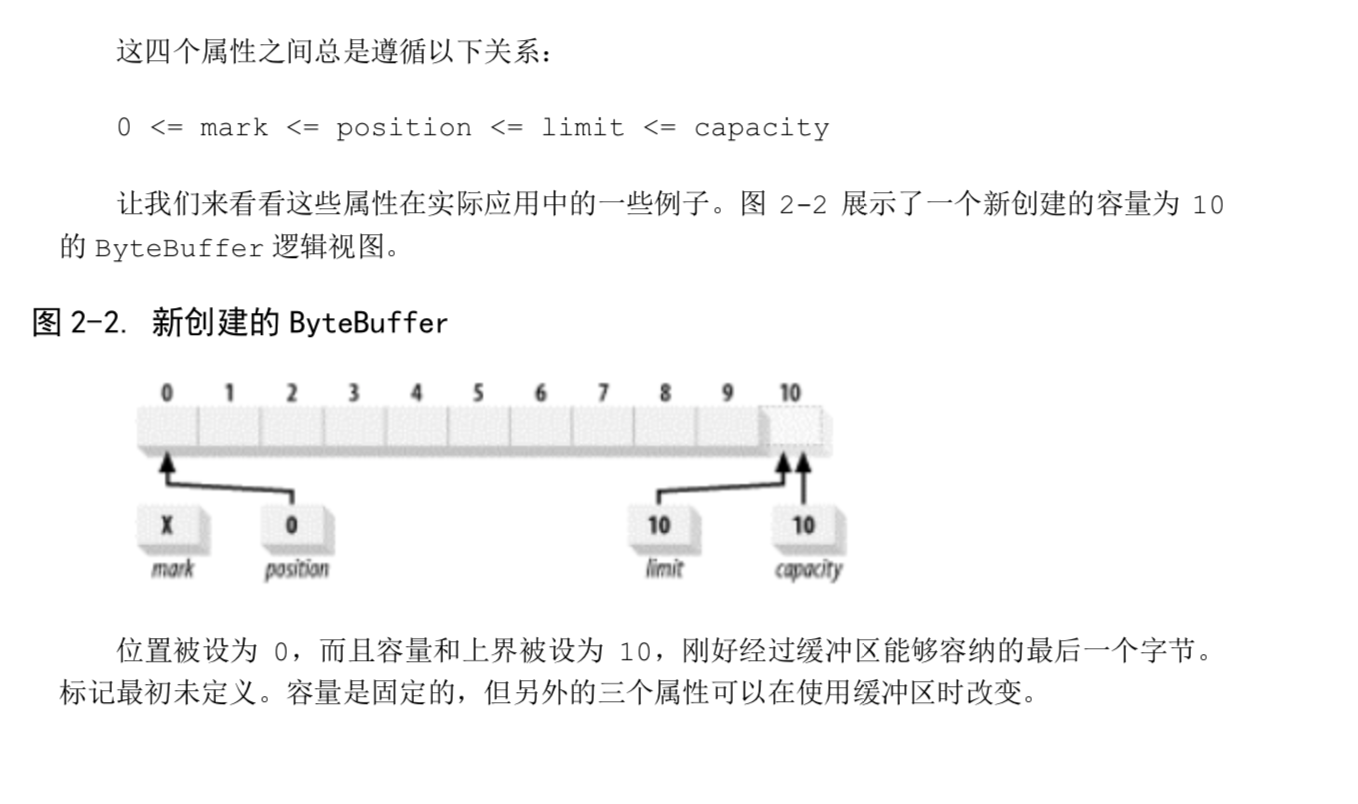
### 2.1.1 属性

容量(Capacity)：

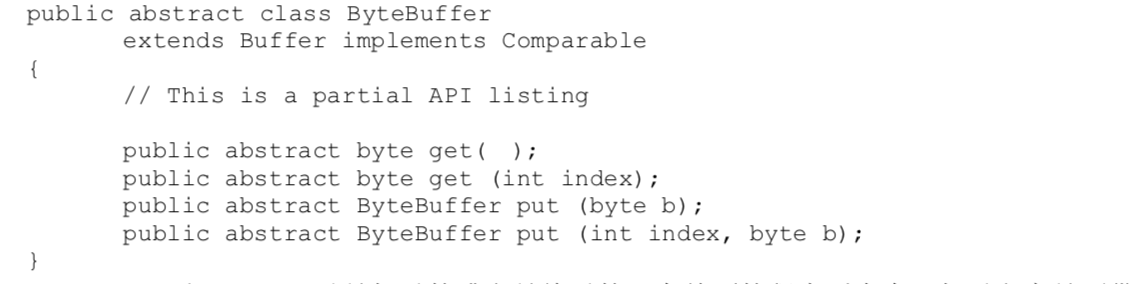
上界(Limit)：

位置(Position):

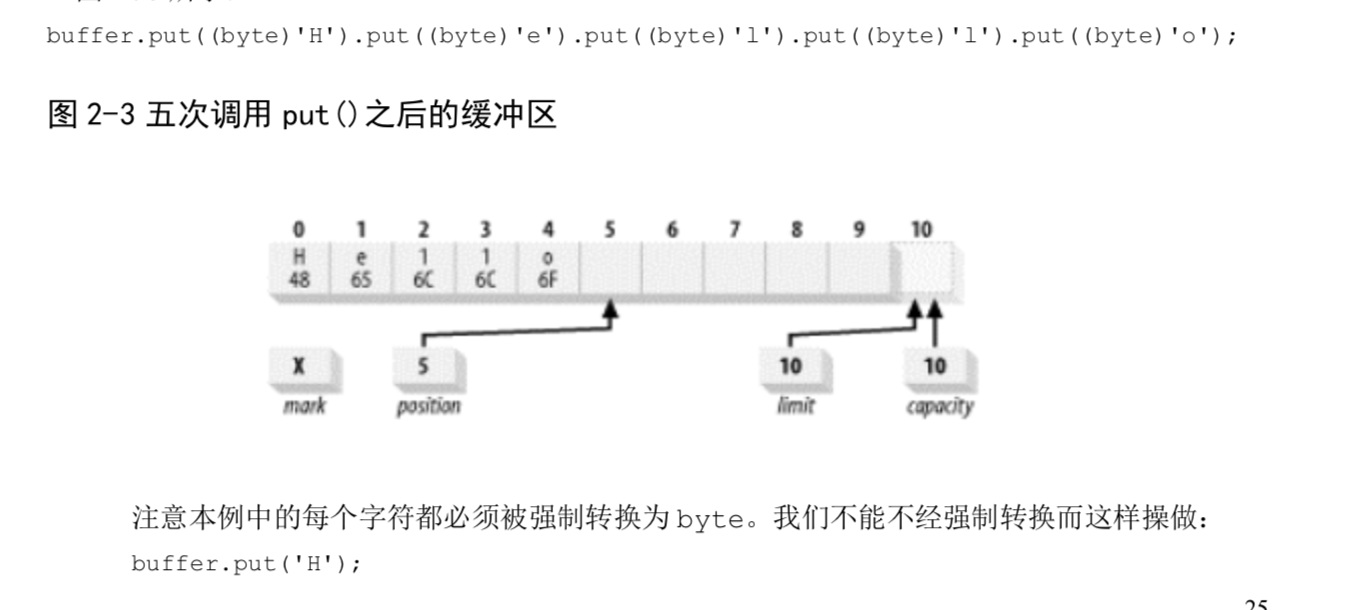
标记（Mark）：

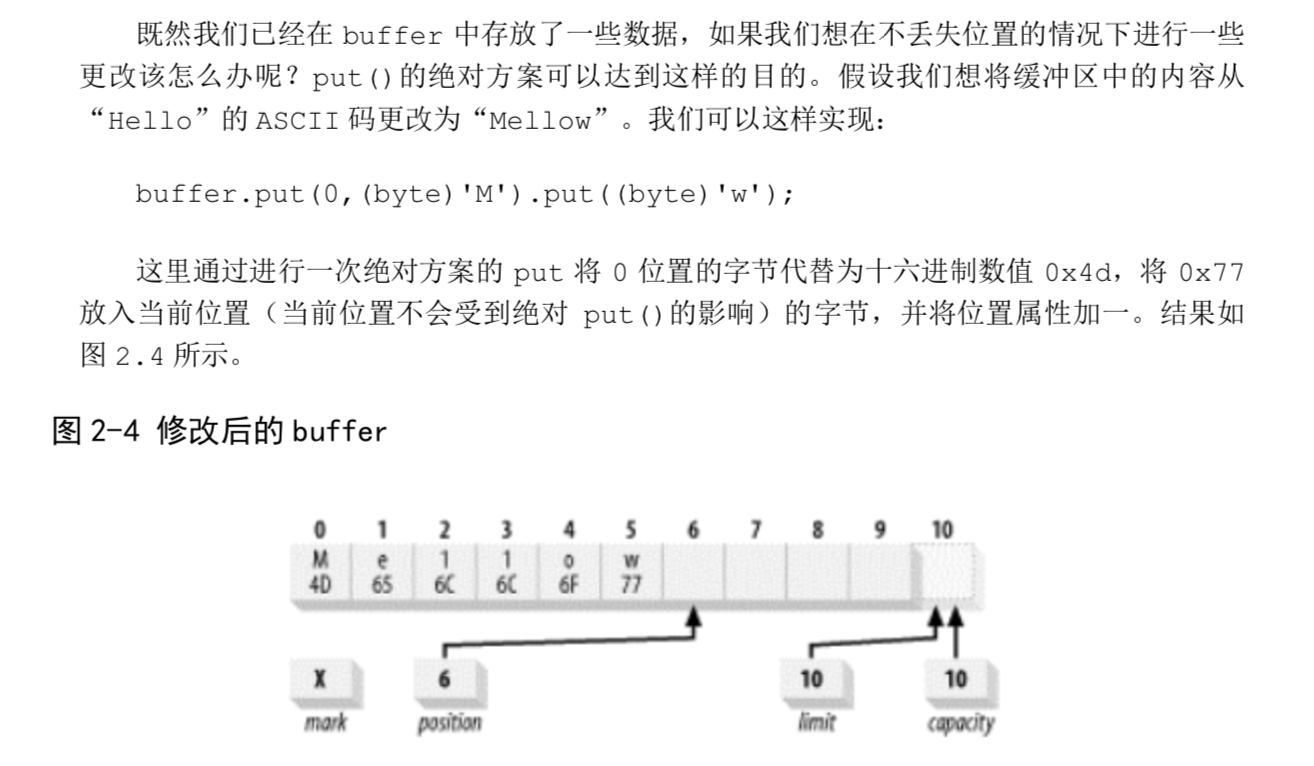


### 2.1.3 存取

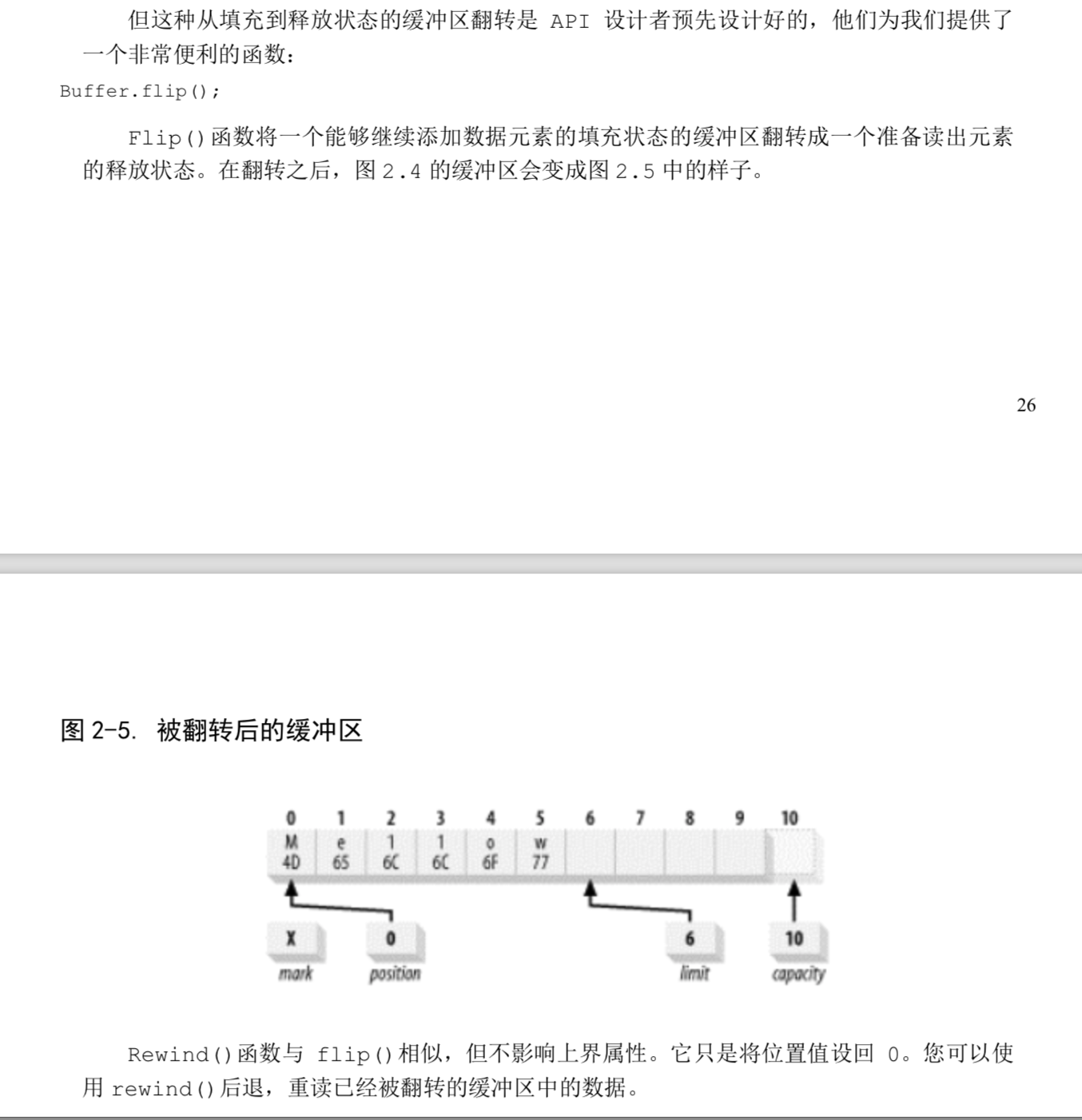


### 2.1.4 填充





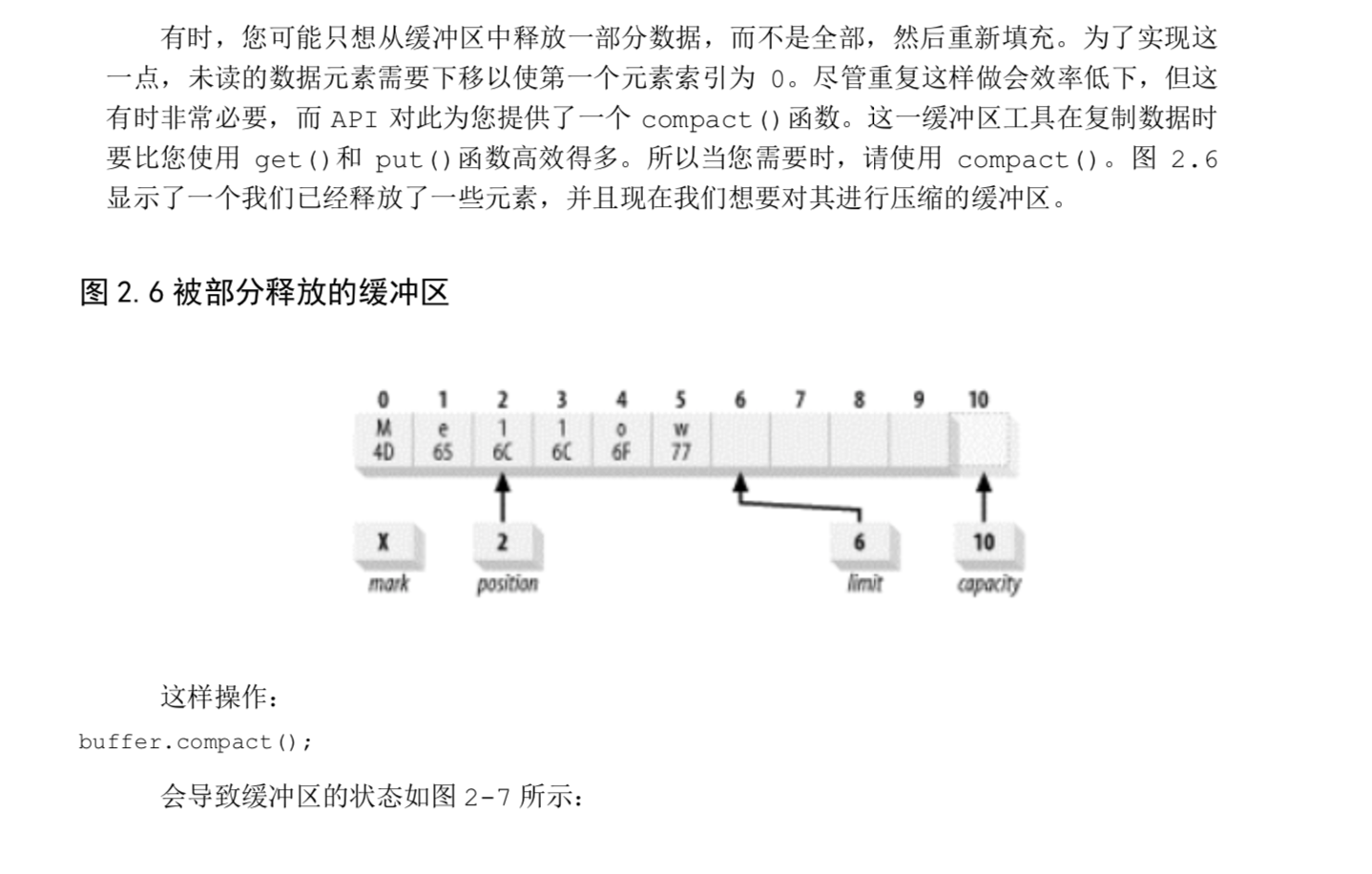
### 2.1.5 翻转

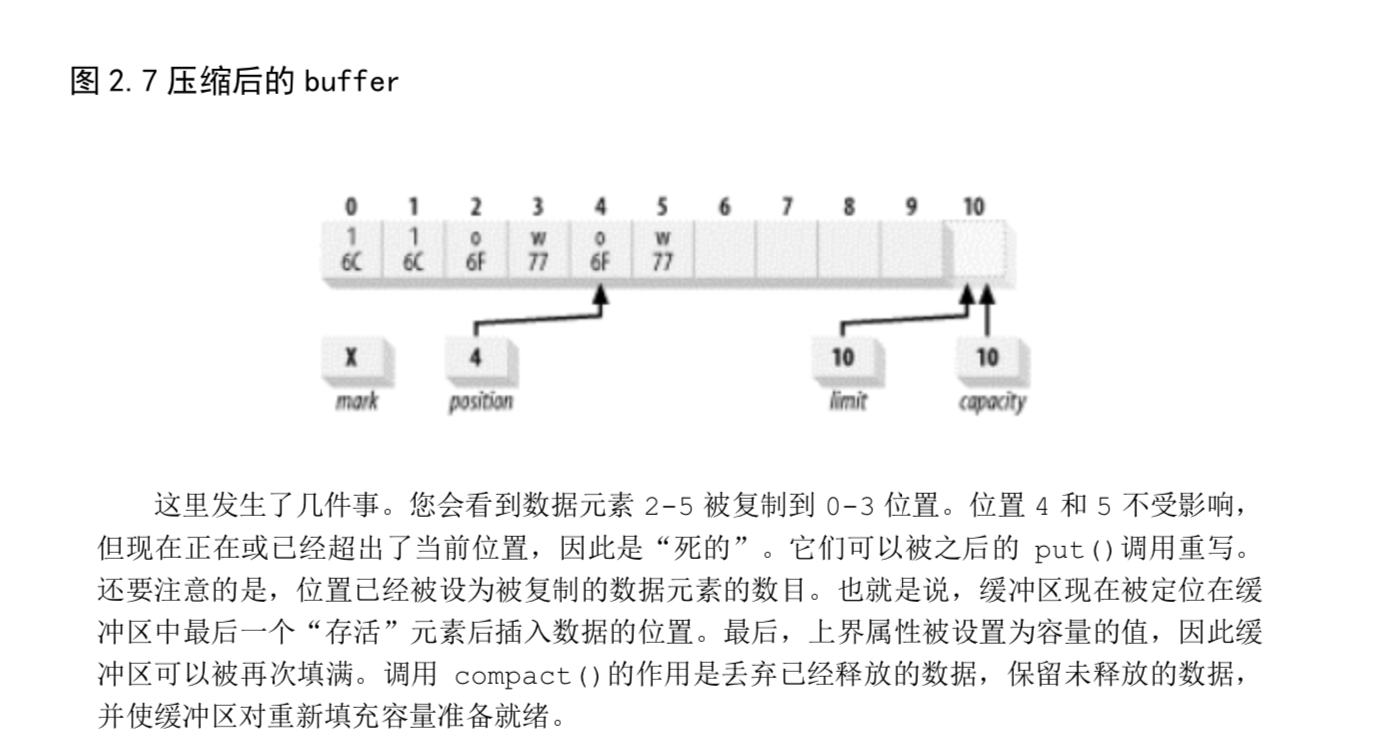


### 2.1.6 释放

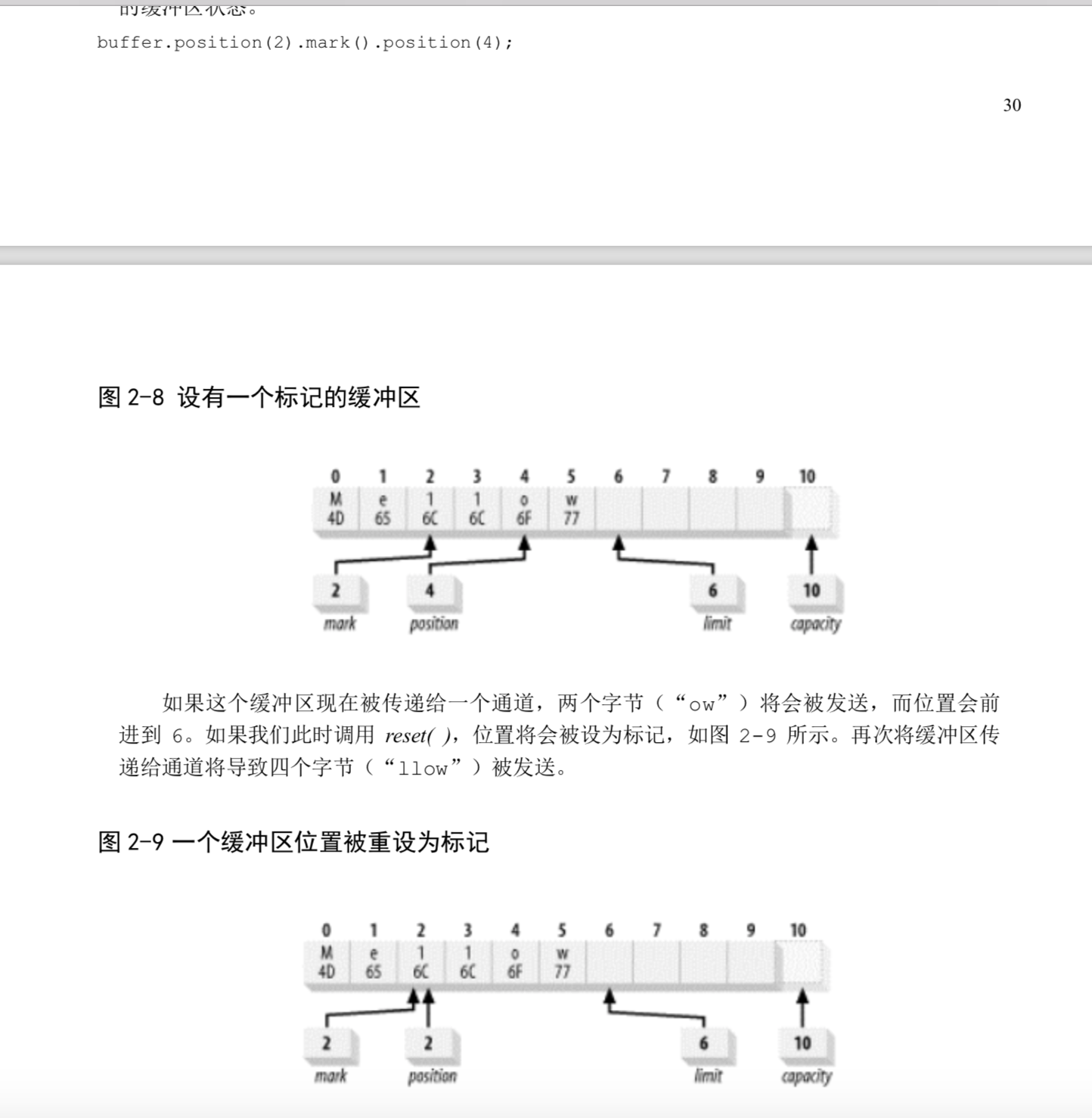
package buffer;  
  
import java.nio.CharBuffer;  
  
public class BufferFillDrain {  
  
 private static int *index* = 0;  
  
 private static String[] *strings* = {  
 "A random string value",  
 "The product of an infinite number of monkeys",  
 "Hey hey we're the Monkees",  
 "Opening act for the Monkees: Jimi Hendrix",  
 "'Scuse me while I kiss this fly '",  
 "Hellp Me! Help Me!"  
 };  
  
 public static void main(String[] args) {  
  
 CharBuffer buffer = CharBuffer.*allocate*(100);  
  
 while (*fillBuffer*(buffer)) {  
 buffer.flip();  
 *drainBuffer*(buffer);  
 buffer.clear();  
 }  
  
  
 }  
  
 */\*\*  
 \* 填充  
 \*  
 \** ***@param*** *buffer  
 \** ***@return*** *\*/* private static boolean fillBuffer(CharBuffer buffer) {  
 if (*index* >= *strings*.length) {  
 return false;  
 }  
 String string = *strings*[*index*++];  
 for (int i = 0; i < string.length(); i++) {  
 buffer.put(string.charAt(i));  
  
 }  
 return true;  
 }  
  
 */\*\*  
 \* 释放输出  
 \*  
 \** ***@param*** *buffer  
 \*/* private static void drainBuffer(CharBuffer buffer) {  
 while (buffer.hasRemaining()) {  
 System.*out*.print(buffer.get());  
 }  
 System.*out*.println("");  
 }  
  
  
}

### 2.1.7 压缩

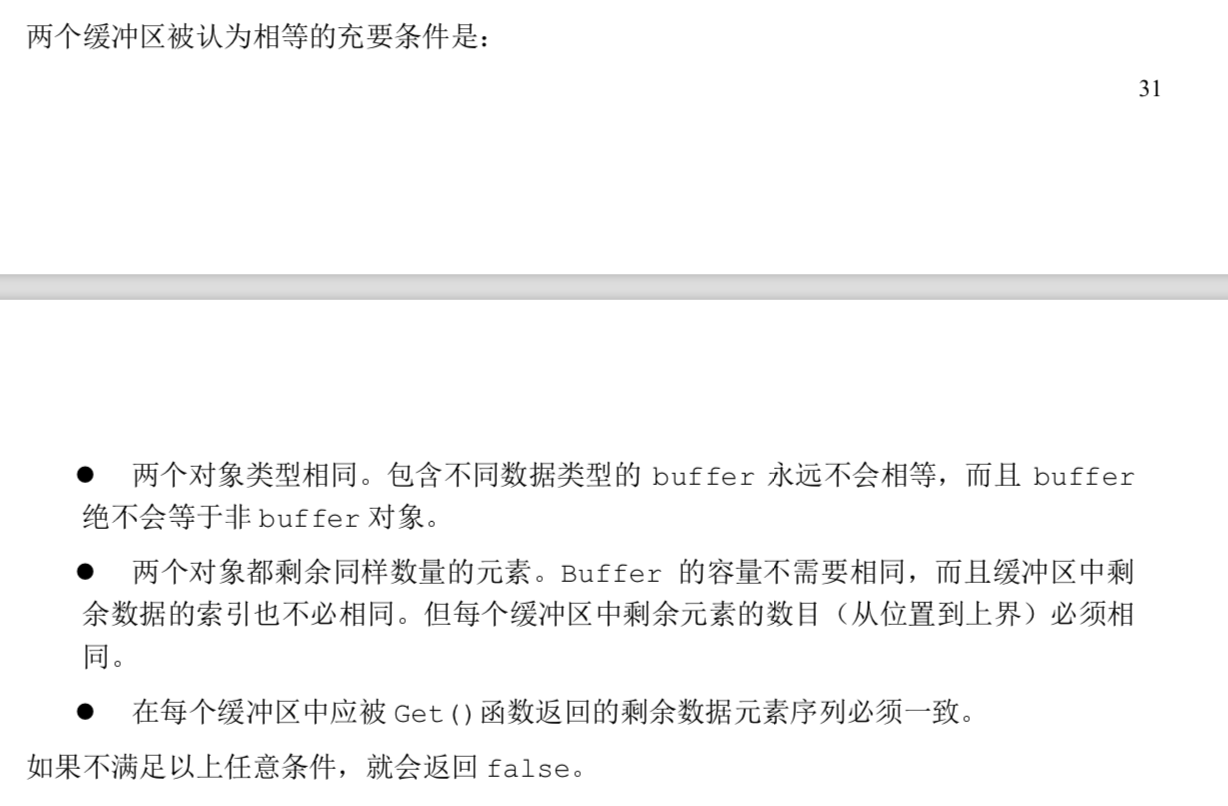




### 2.1.8 标记



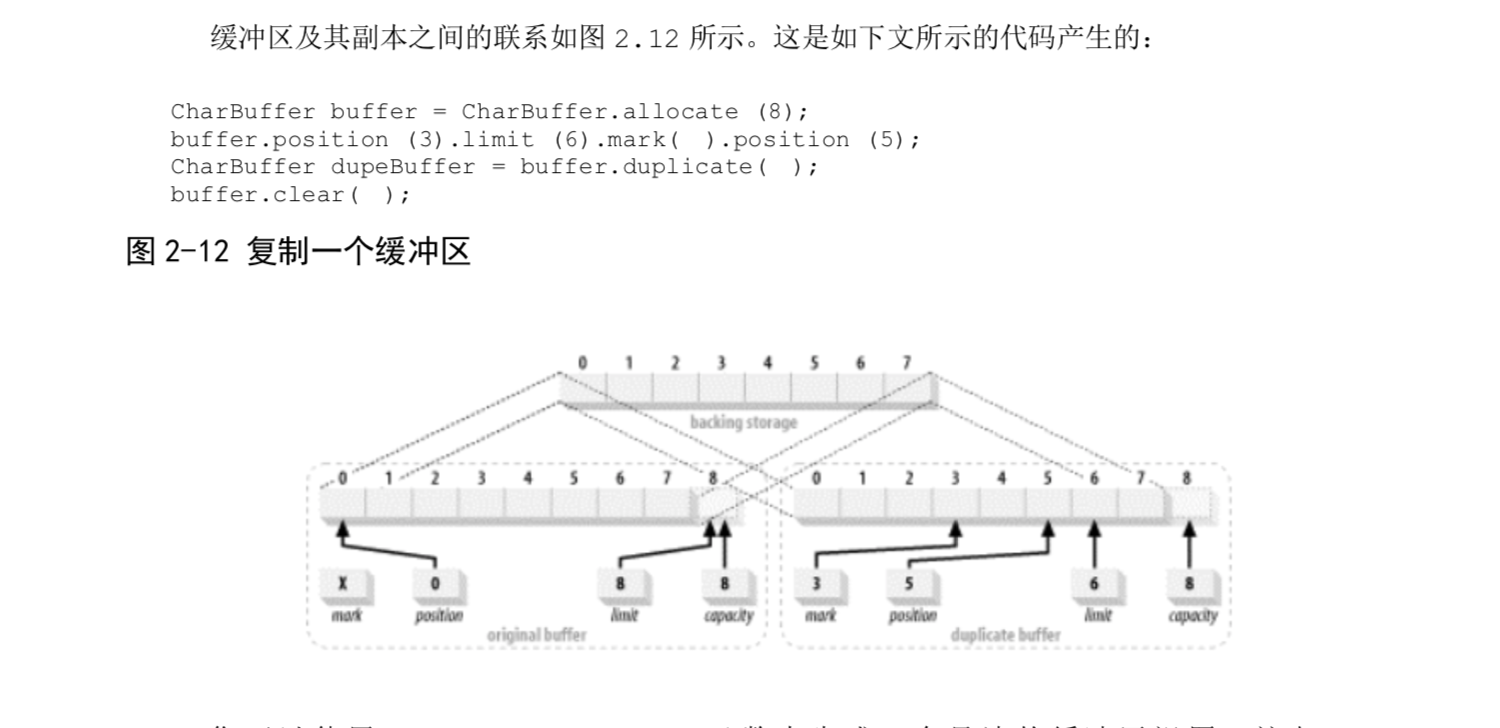
### 2.1.9 比较

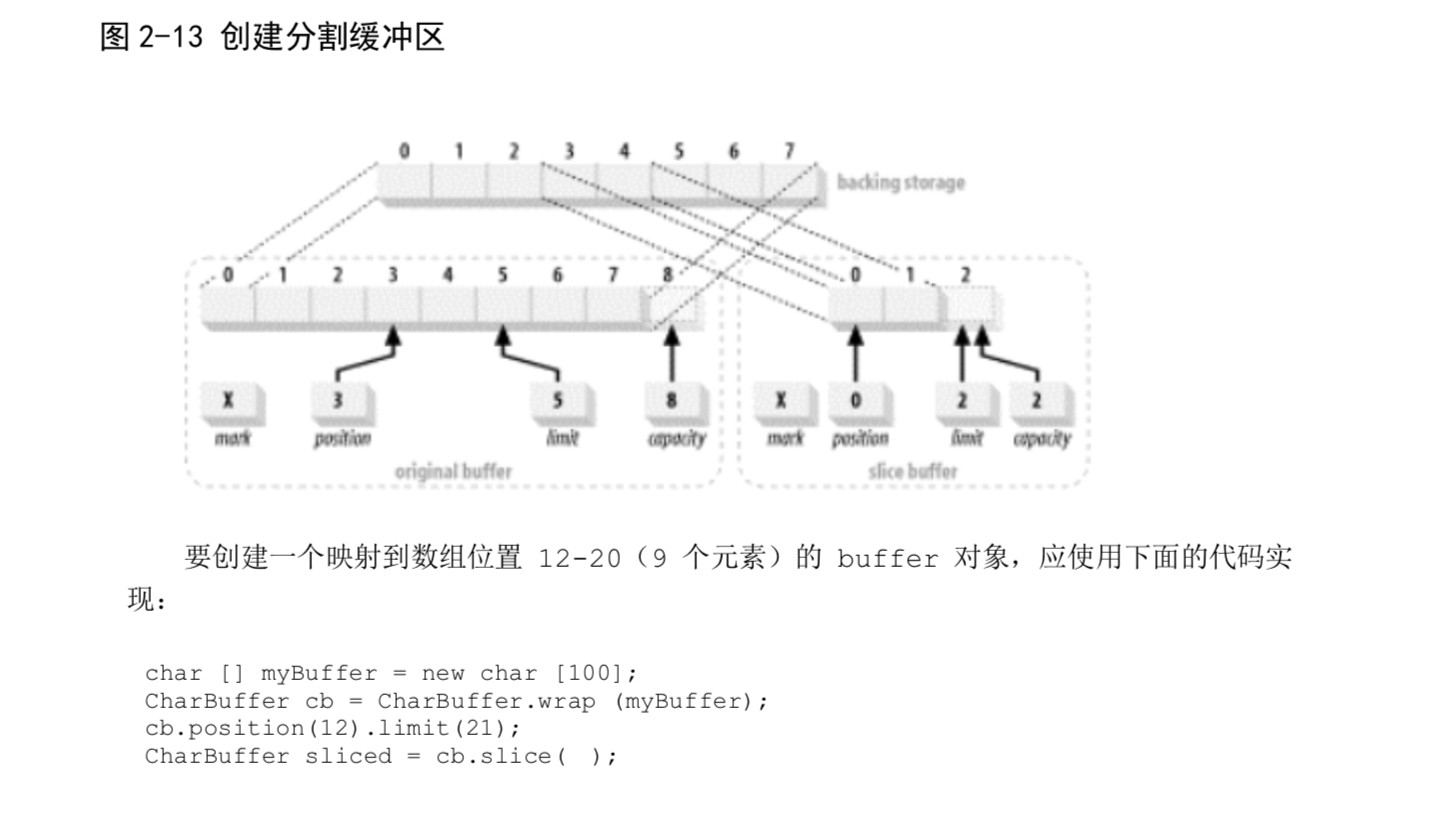


### 2.1.10 批量移动

## 2.2 创建缓冲区

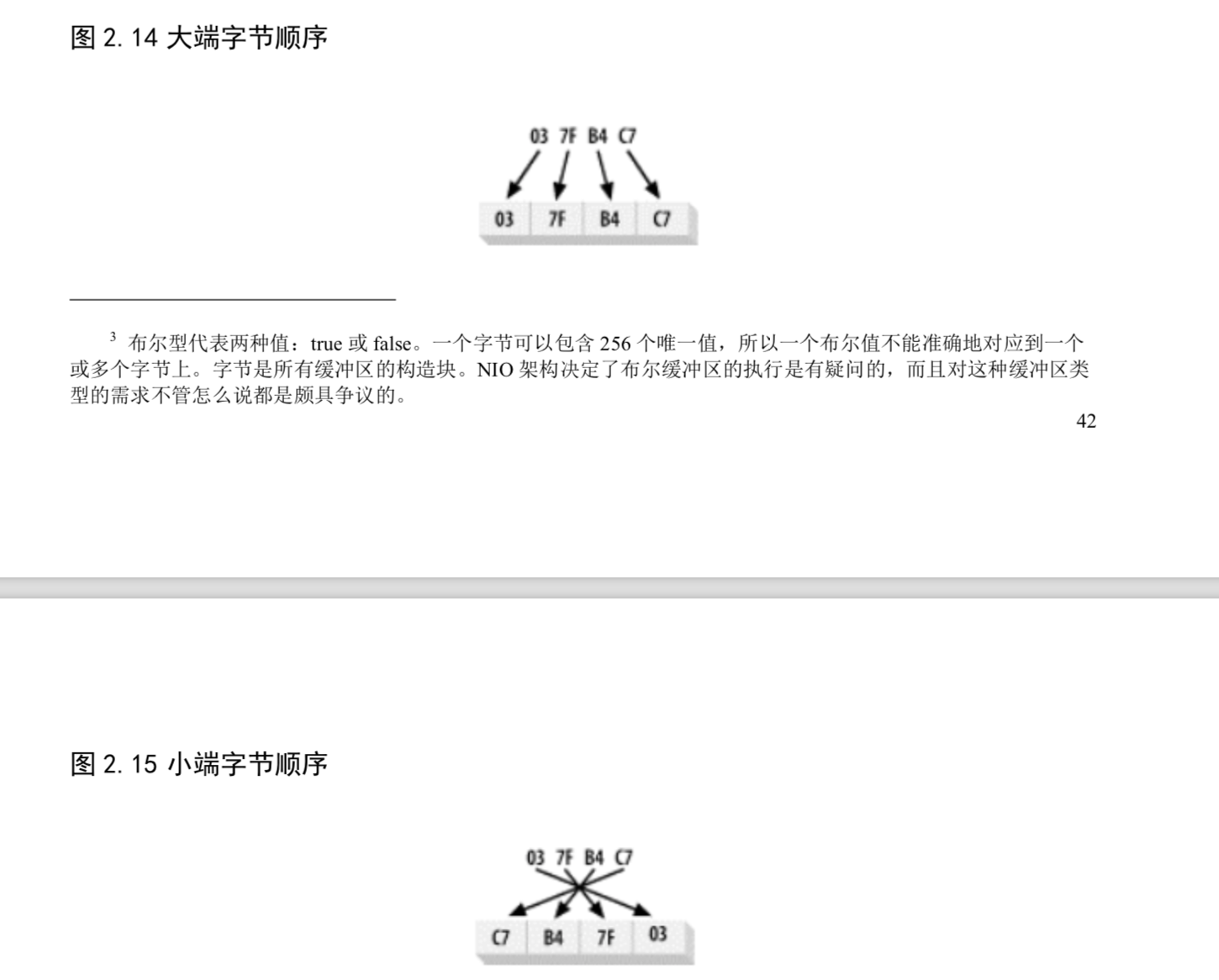
## 2.3 复制缓冲区





## 2.4 字节缓冲区

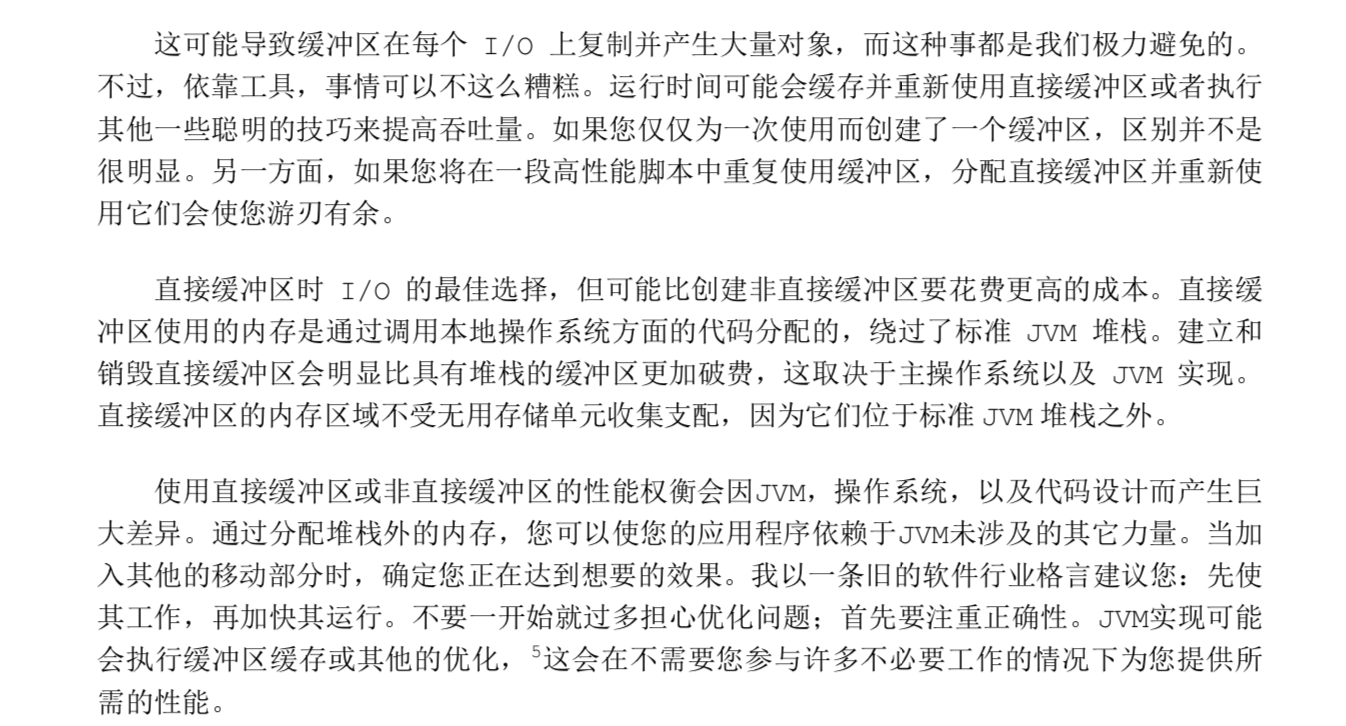
### 2.4.1 字节顺序

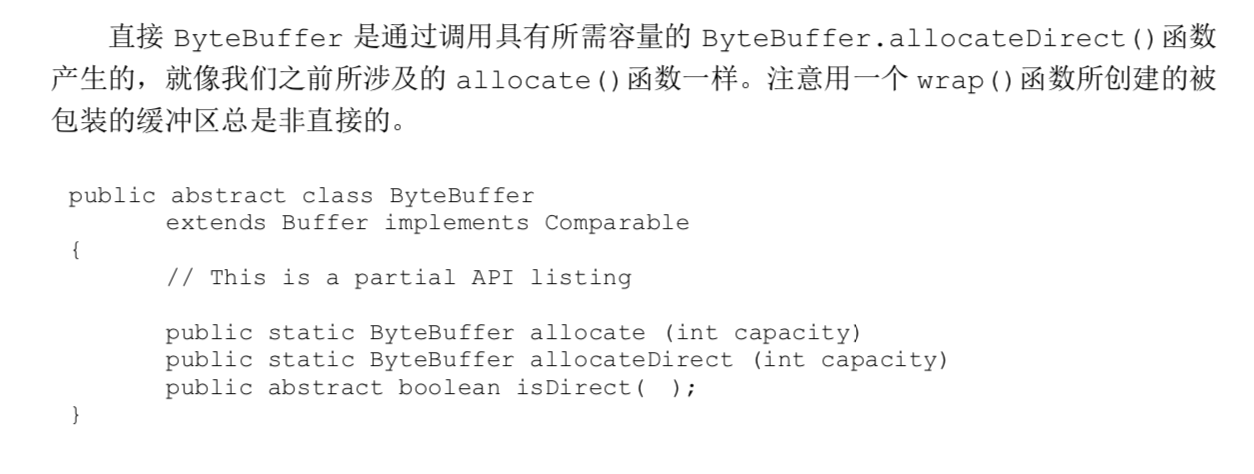




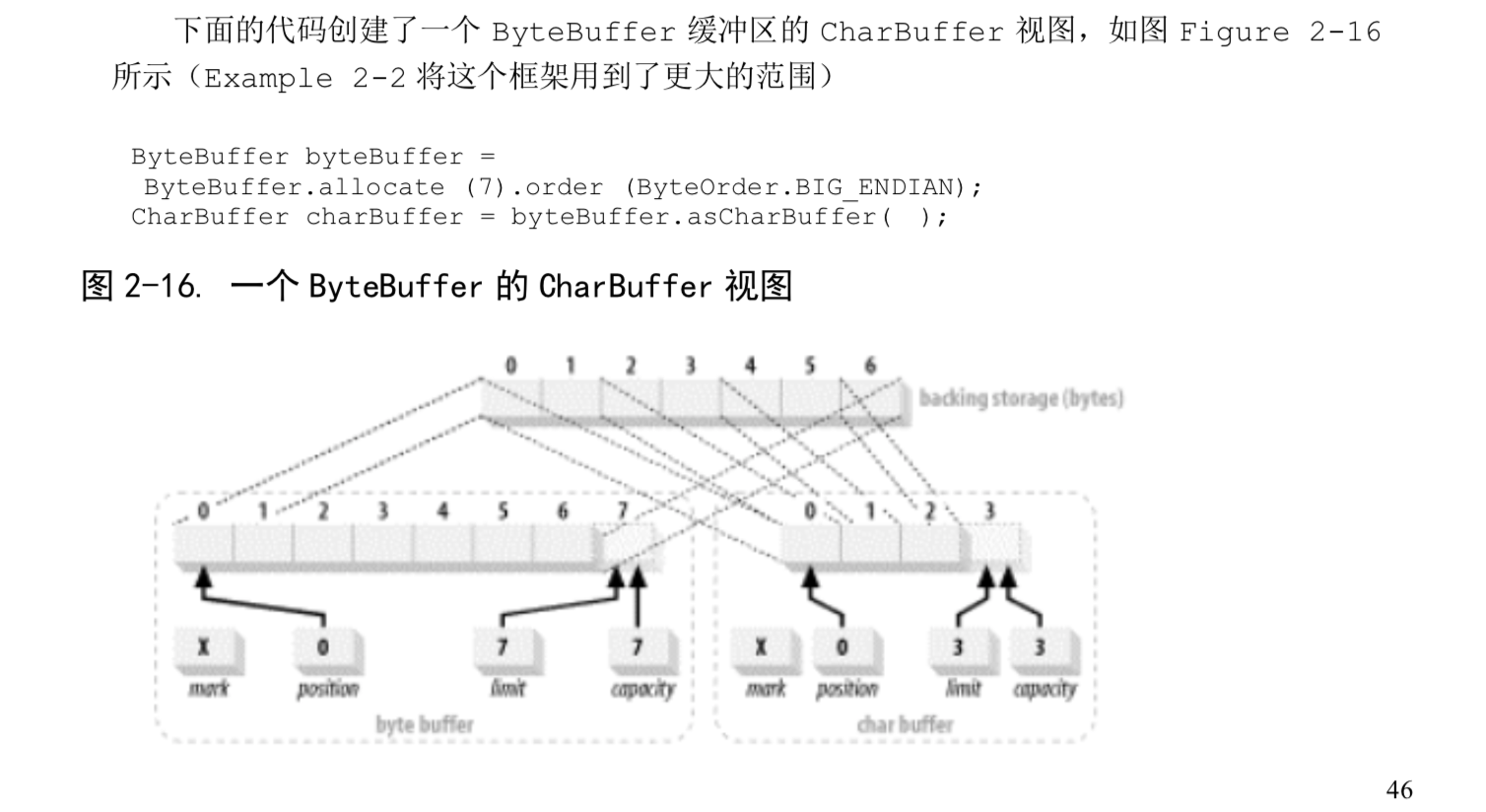
### 2.4.2 直接缓冲区



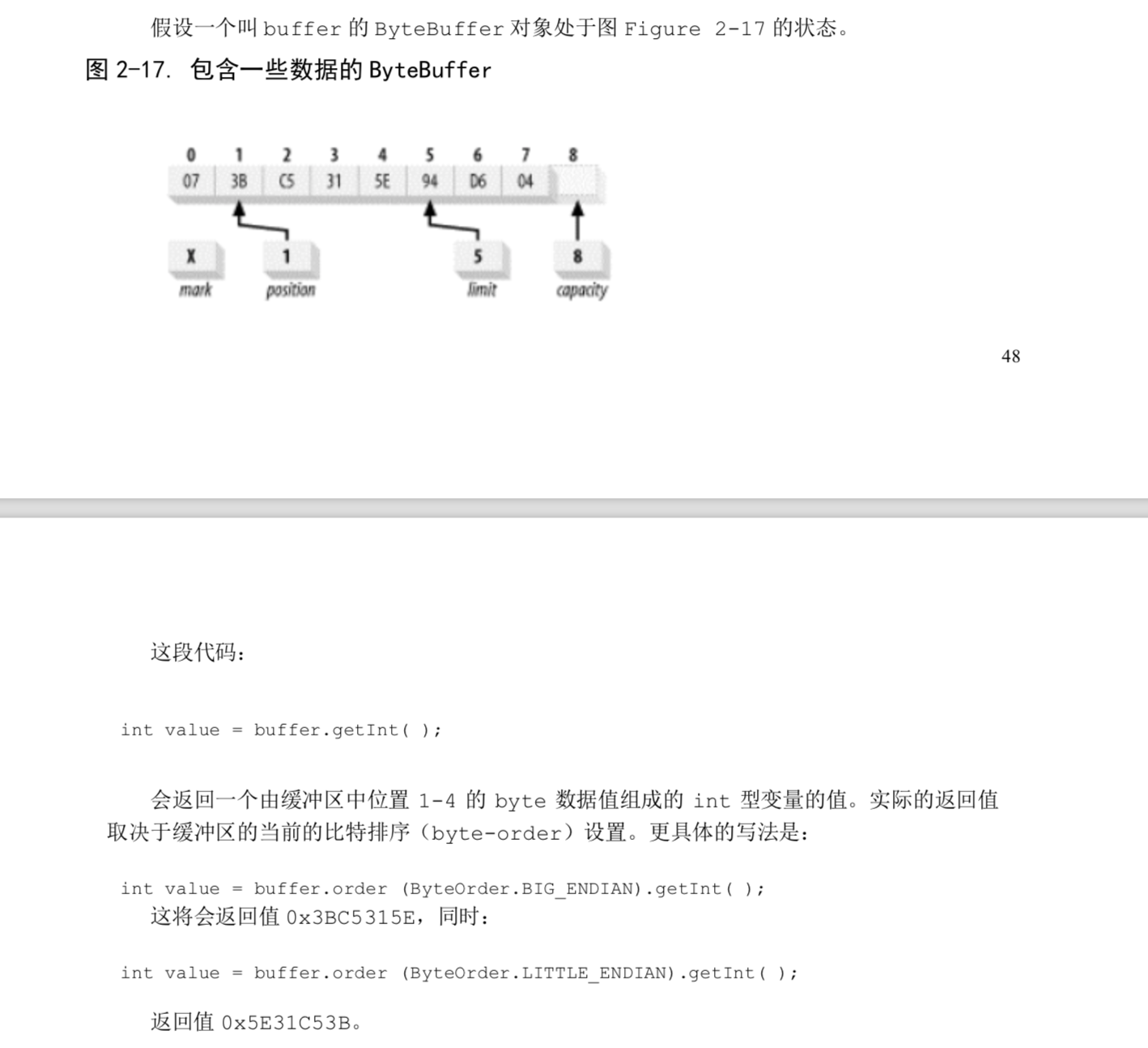




### 2.4.3 视图缓冲区

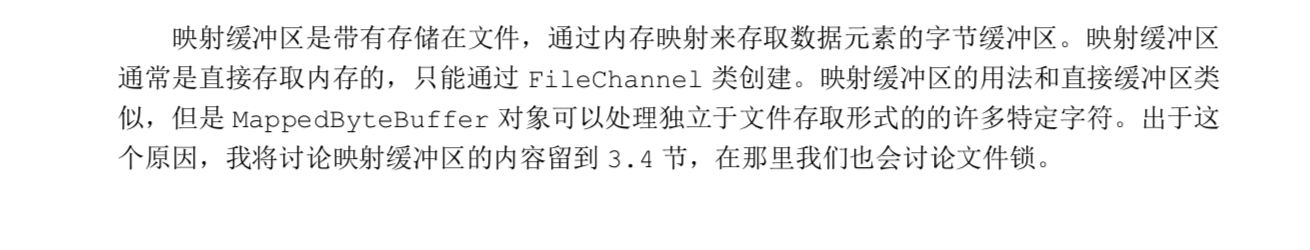


### 2.4.4 数据元素视图

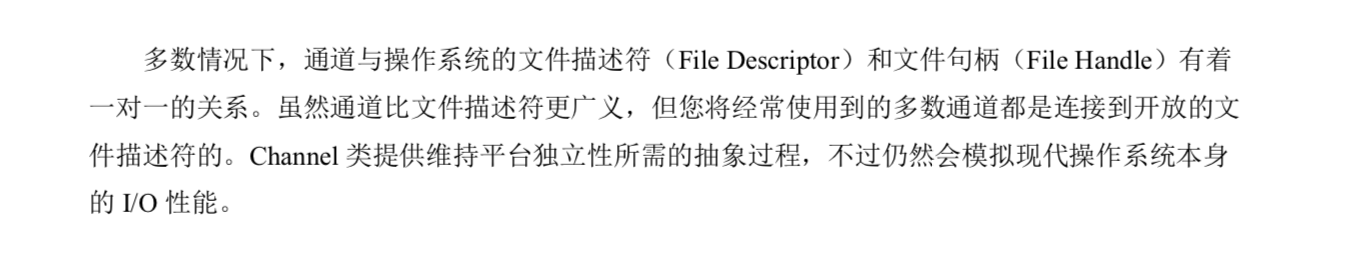


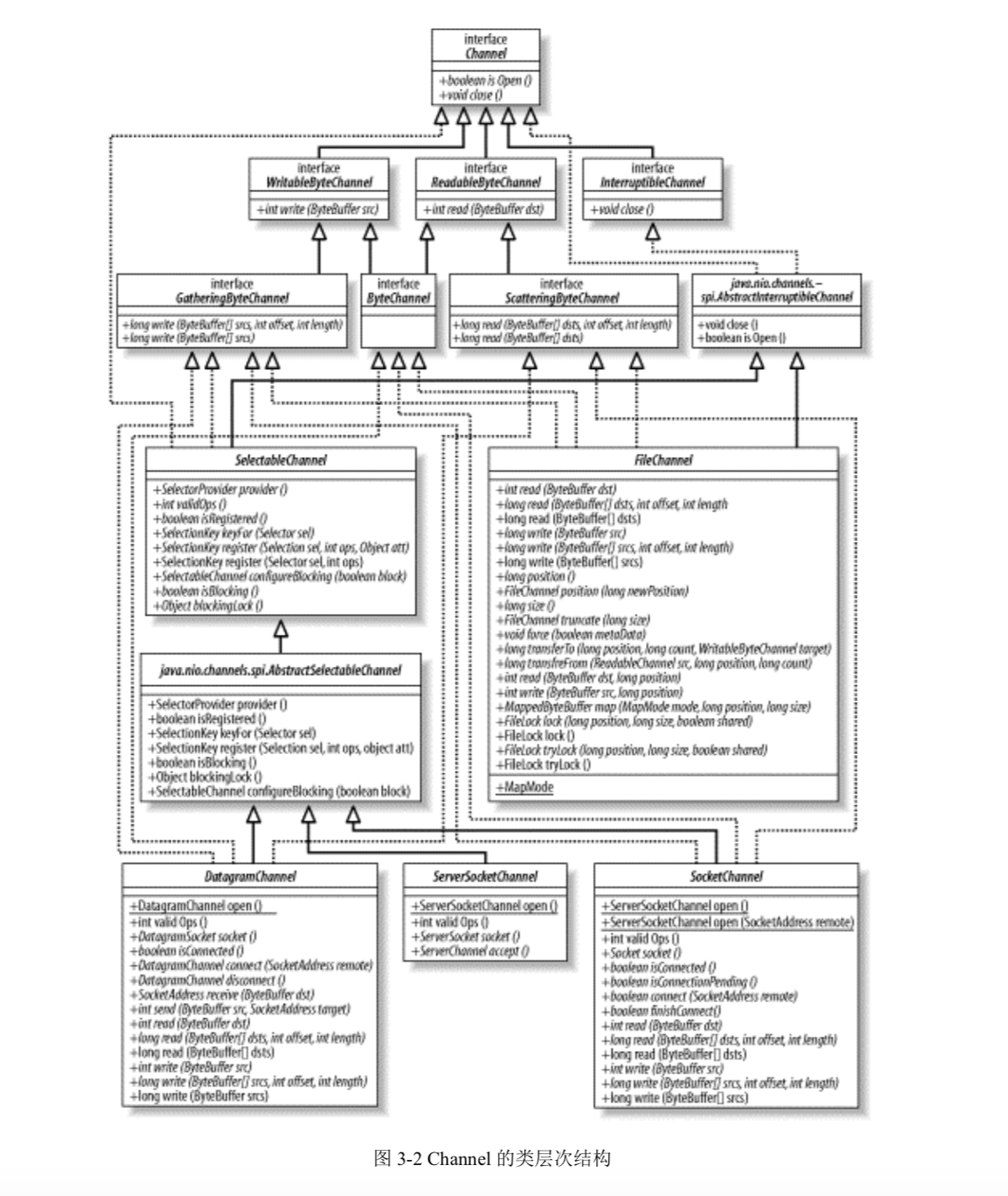
### 2.4.5 存取无符号数据

### 2.4.6 内存映射缓冲区



# 3 通道





## 3.1 通道基础

### 3.1.1 打开通道



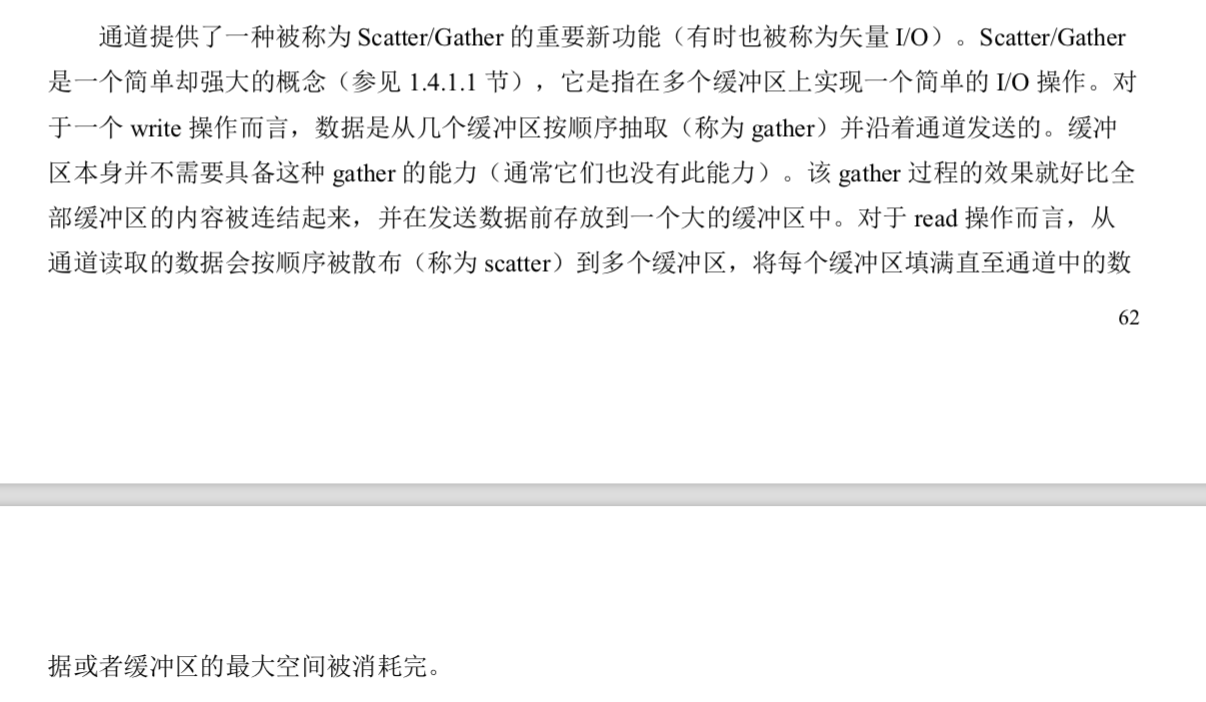
### 3.1.2 使用通道

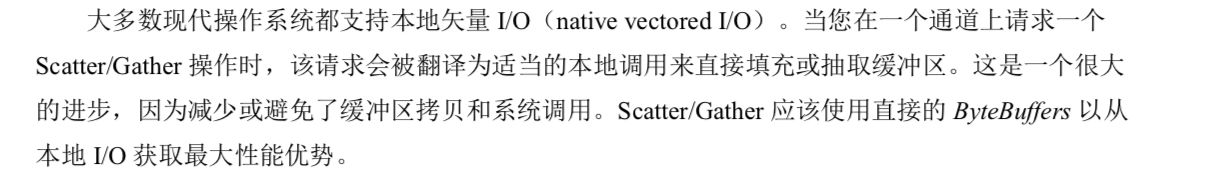
|  |
| --- |
| package channel;  import java.io.IOException; import java.nio.ByteBuffer; import java.nio.channels.Channel; import java.nio.channels.Channels; import java.nio.channels.ReadableByteChannel; import java.nio.channels.WritableByteChannel;  public class ChannelCopy {   public static void main(String[] args) throws IOException {   ReadableByteChannel source = Channels.*newChannel*(System.*in*);  WritableByteChannel dest = Channels.*newChannel*(System.*out*);  *channelCopy1*(source, dest); // channelCopy2(source, dest);  source.close();  dest.close();    }   private static void channelCopy2(ReadableByteChannel source, WritableByteChannel dest) throws IOException {  ByteBuffer buffer = ByteBuffer.*allocate*(16 \* 1024);   while (source.read(buffer) != -1) {  buffer.flip();  while (buffer.hasRemaining()) {  dest.write(buffer);  }  buffer.clear();  }  }   private static void channelCopy1(ReadableByteChannel source, WritableByteChannel dest) throws IOException {   ByteBuffer buffer = ByteBuffer.*allocate*(16 \* 1024);   while (source.read(buffer) != -1) {  buffer.flip();   dest.write(buffer);   buffer.compact();  }   buffer.flip();   while (buffer.hasRemaining()) {  dest.write(buffer);  }    }   } |

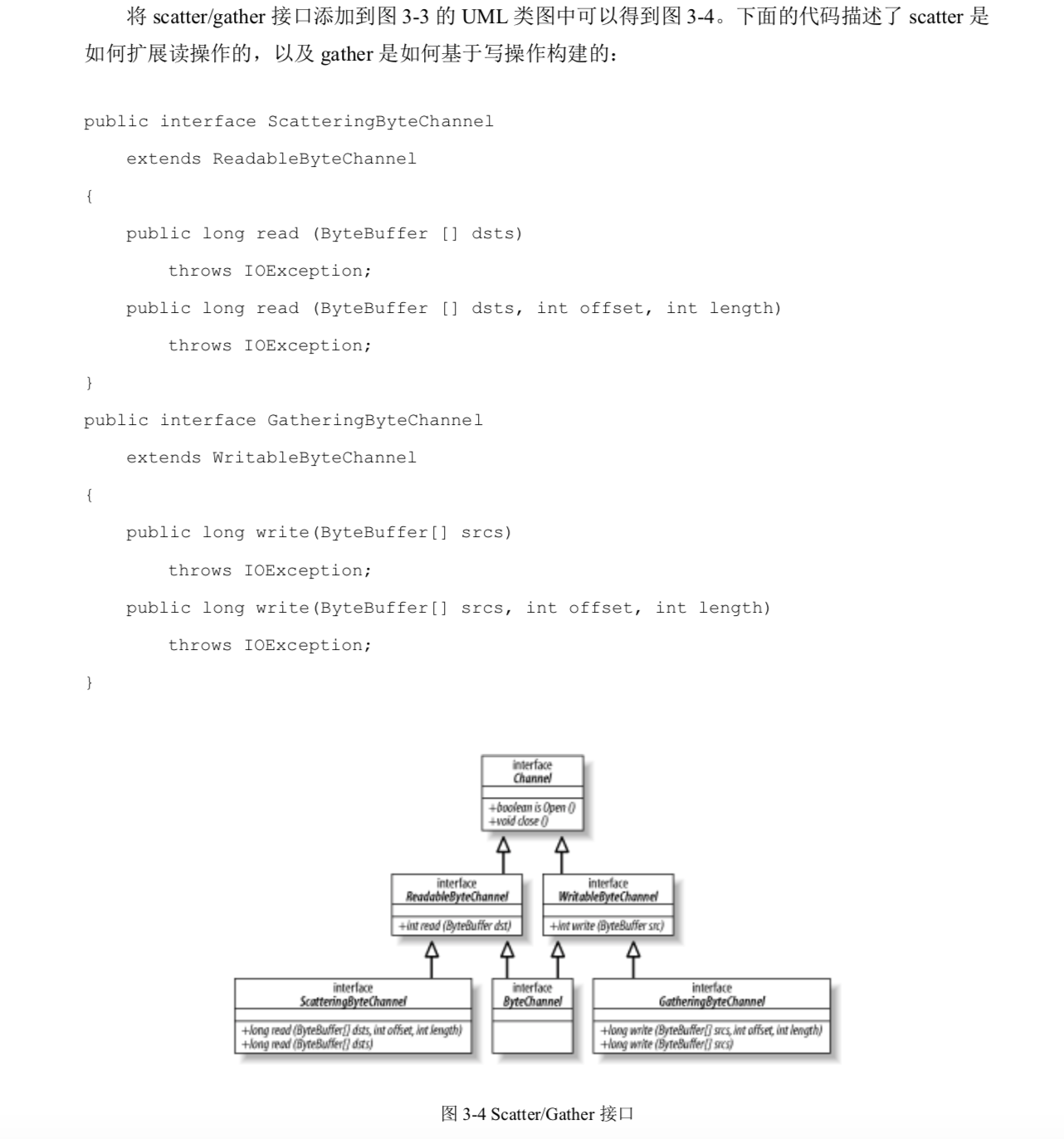


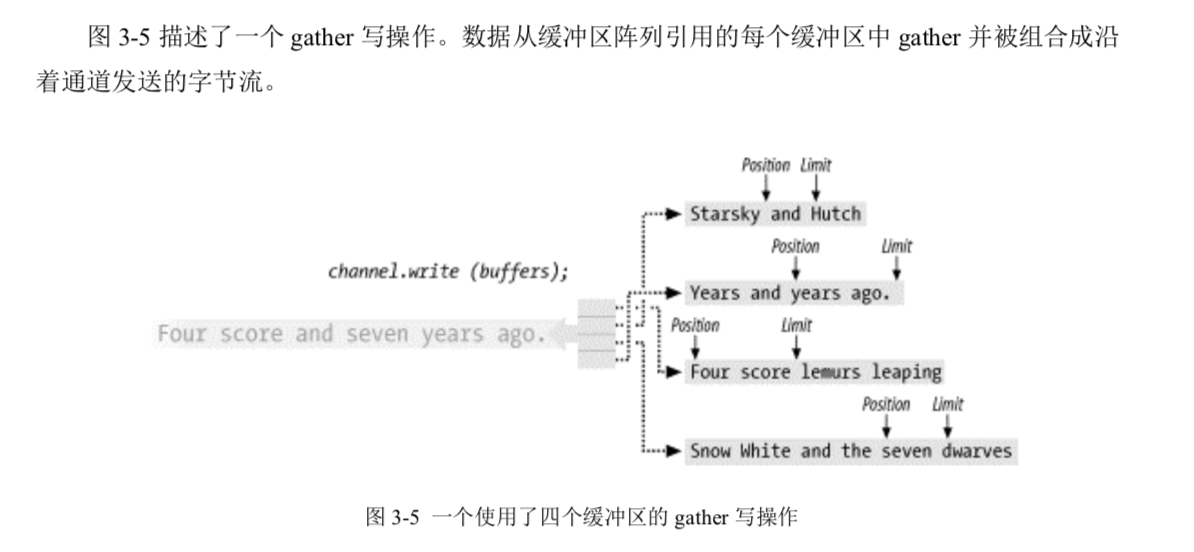
### 3.1.3 关闭通道

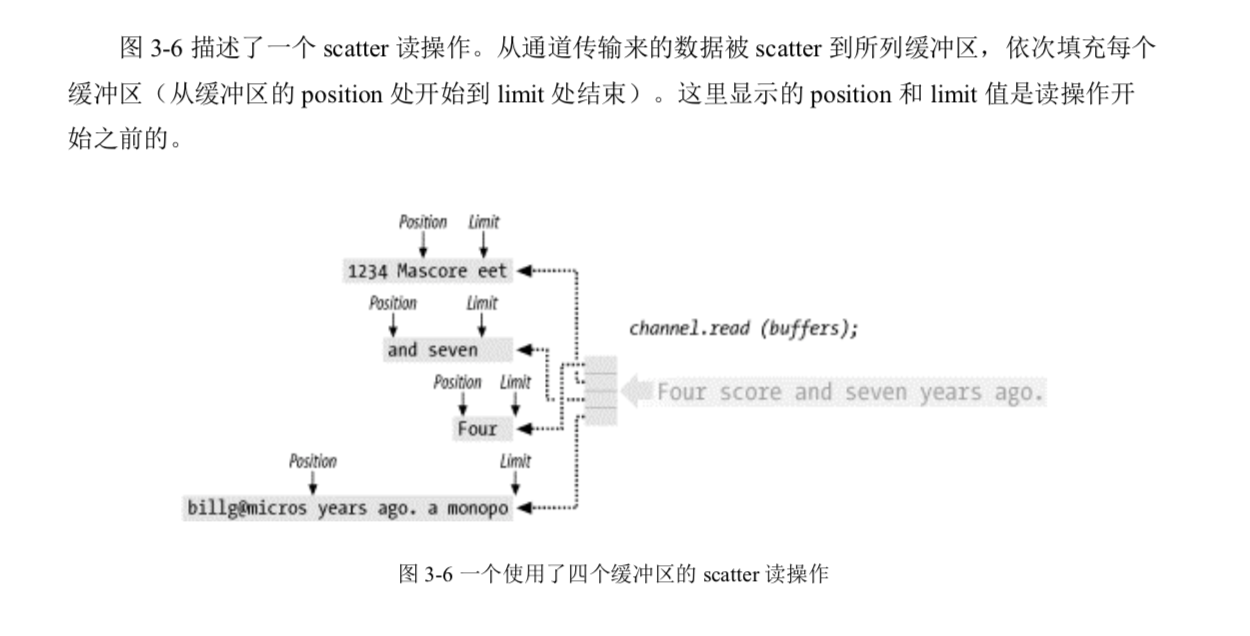
## 3.2 Scatter/Gather(矢量I/O)



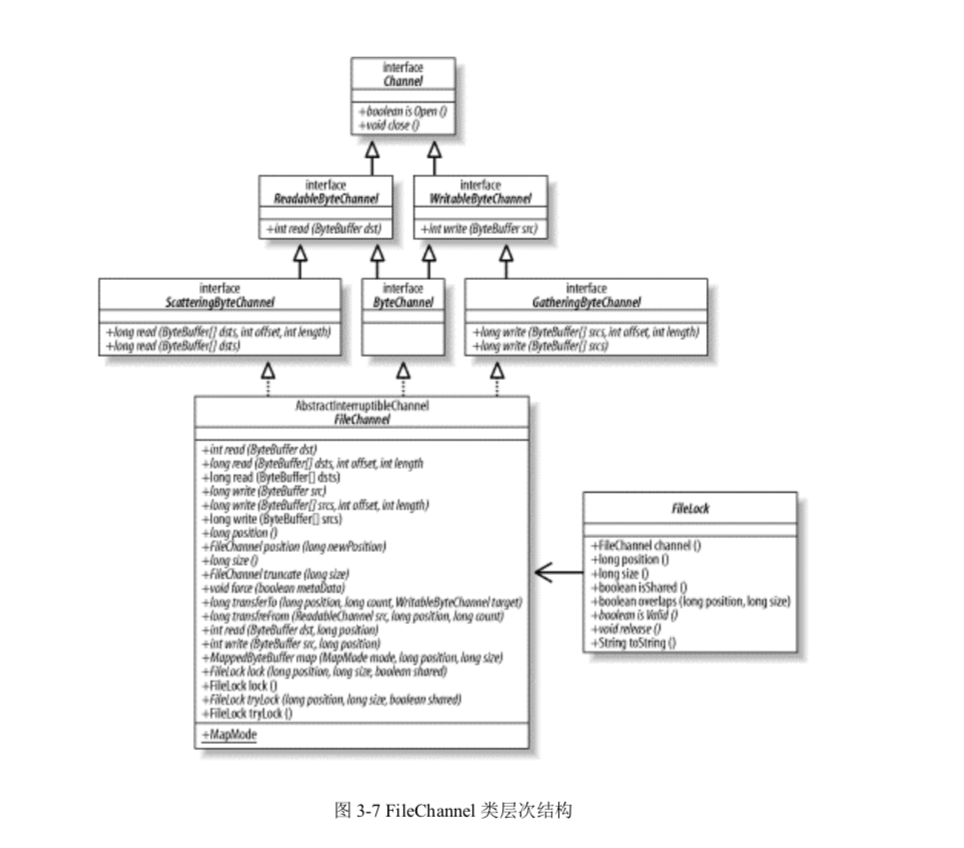


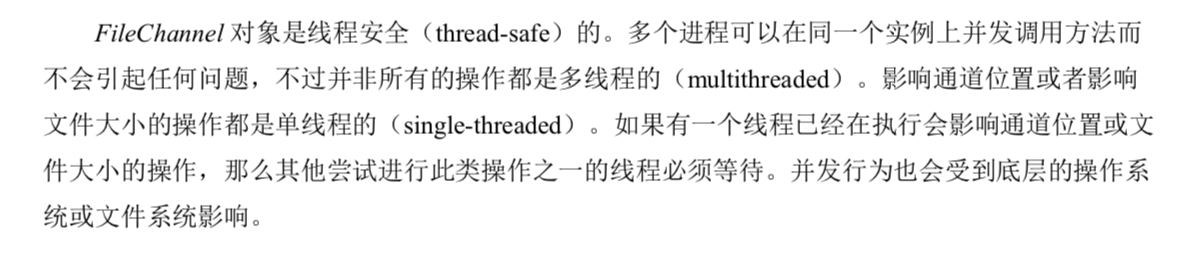






## 3.2 文件通道





### 3.3.1 访问文件

### 3.3.2 文件锁定