버퍼의 속성

|  |  |
| --- | --- |
| 속성 | 설명 |
| Position | 버퍼에서 현재 읽거나 쓸 위치 값이다 limit보다 큰 값을 가질수 없다 . 즉 limit와 같은 값이면 더 이상 읽거나 쓰지 못하고 의미고 만약 읽기나 쓰기 작업을 할 경우 런타임 예외가 발생한다. 버퍼생성햇을때 초기값은 0이다. |
| Limit | 버퍼에서 읽거나 쓸수 있는 한계값이다 버퍼에서 실제 어디까지를 사용할지 지정하는 속성값으료 capacity보다 클수 없다 버퍼를 생성했을 때 초기값음 capacity와같다 |
| Capacity | 버퍼의 크기를 나타낸다 즉 메모리 크기라고 생각하면된다 이값은 버퍼를 생성할 때 파라미터로 주어진다 한번 생성되면 크기를 변경할수 없으므로 버퍼의 크기를 신중 결정해야한다 |
| Mark | Mark()메서드로 현재의 position을 표시해둘 때 사용한다 나중에 reset() 메서드를 호출해서 mark위치로 position을 바꿀수 있다.  Mark는 초기 -1값을가진다 그러므로 mark() 함수를 쓰지않는상태에서 reset을하게되면 예외발생한다. |

0<=mark<=positon<=limit<=capacity

API

|  |  |
| --- | --- |
| Position() | 각각값 리턴 |
| Limit() | 각각값 리턴 |
| Capacity() | 각각값 리턴 |
| Position(int newPosition) | 포지션셋팅 |
| Limit(int newLimit) | 리미트셋팅 |
| Mark() | 현재포지션을 mark속성값으로 셋팅 |
| Reset() | 포지션을 마크값으로이동 |
| Remaining() | Limit - position같다는뜻 즉 현재 position을 기준으로 해서 버퍼에서 읽을수 있는 데이터가 몇 개나 남았는지 알려주는 메서드 |
| hasRemaining() | Limit-position>0을 의미  포지션 기준으로 버퍼를 읽을수 있는 데이터가 있는지 없는지 블리언으로 리턴 |
| isReadOnly() | 리드온니인지 ? |

**상대적으로 읽고 쓰기**

Buf.get(); //포지션에서 읽는다. 하나 그다음 자동으로 포지션이 이동한다

Buf.put(byte)//포지션에 넣는다 그다음 자동으로 포지션을 이동한다

//Limit , capacity는 변화가없다 콜바이벨류

**절대적인 위치에서 읽고 쓰기**

Buf.get(int)

Buf.put(int,byte)

//Position , Limit , capacity는 변화가없다 콜바이벨류

**배열이용 한꺼번에 많은 데이터 읽고 쓰기**

Buf.get(byte[])

Buf.get(byte[],int offset,int length)

Buf.put(byte[])

Buf.put(byte[],int offset,int length)

//Limit , capacity는 변화가없다 콜바이벨류

**버퍼자체를 put으로 넣기**

Buf.put(ByteBuffer src);

이렇게하면 src 인자값의 버퍼의 position값이 맨끝으로 이동한다

//Limit , capacity는 변화가없다 콜바이벨류

유틸리트

clear메서드

position을 0으로 limit를 capacity 값으로 이동한다 즉 초기화된다

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | position |  | limit |  |  | Capacity |
| position |  |  |  |  |  | Limit  Capacity |

마크역시 초기값 -1로 이동

rewind메서드

position 초기화

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | position |  | limit |  |  | Capacity |
| position |  |  | limit |  |  | Capacity |

마크역시 초기값 -1로 이동

flip()메서드  
position 위치를 limit로 설정후 position은 0으로 초기화

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | position |  | limit |  |  | Capacity |
| position | limit |  |  |  |  | Capacity |

마크역시 초기값 -1로 이동

compact()

position과 limit 사이에 남아있는 데이터를 버퍼의 맨앞에서부터 순서대로 쓴다. 그만큼 position을 이동시키고 limit는 버퍼거 생성되는 시점으로 capacity와 동일하게 만든다.

마크역시 초기값 -1로 이동

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [a] | [b] | [c] | **[d]**  position | **[e]** | [f]  limit | [g]  Capacity |
| **[d]** | **[e]** | [c]  position | [d] | [e] | [f] | [g]  Capacity  limit |

**Duplicate()메서드**

원본과 같은 주소를 갖는 버퍼를 만든다 즉 **같은 주소를 갖는 버퍼를생성**

**속성값은 따로가진다.**

Bytebuffer byf =Bytebuffer.allocate(10);

Bytebuffer byf =buf.duplicate();

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [a] | [b] | [c] | [d]  position | [e] | [f]  limit | [g]  Capacity |
| [a] | [b] | [c] | [d]  position | [e] | [f]  limit | [g]  Capacity |

**asReadOnlyBuffer() 메서드**

Bytebuffer byf =Bytebuffer.allocate(10);

Bytebuffer byf =buf.asReadOnlyBUffer();

원본과 같은 주소를 갖는 버퍼를 만든다 즉 같은 주소를 갖는 버퍼를생성

읽기전용 버퍼 복사본. 리턴

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [a] | [b] | [c] | [d]  position | [e] | [f]  limit | [g]  Capacity |
| [a] | [b] | [c] | [d]  position | [e] | [f]  limit | [g]  Capacity |

[ReadOnly]

slice() 버퍼의 일부분만을 복사하는 메서드

**position에서limit** 까지 떼어내서 만든다.

Bytebuffer byf =Bytebuffer.allocate(10);

Bytebuffer byf =buf.slice();

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [a] | [b] | [c] | [d]  position | [e] | [f]  limit | [g]  Capacity |
| [d]  position | [e] | [f]  Limit  Capacity |  |  |  |  |

Byte[] my = new byte[1024];

ByteBuffer buf = byTeBUffer.wrap(my);

이렇게하면 1024 크기를 가지는 버퍼가 생성된다 하지만 내용만 참조하는것이므로

My가 변경되면 안쪽도 변경된다

Byte[] my = new byte[1024];

ByteBuffer buf = byTeBUffer.wrap(my,10,100); 이렇게하면 포지션10 리미트100이된다

arrayOffset()메서드로 보조 배열 내의 있는 버퍼의 최초 요소의 오프셋(시작위치)알수있다.

**바이트버퍼 다른형으로 읽기/쓰기.**

**차례대로 읽으면 포지션이 4씩쩜푸!!**

System.*out*.println();

System.*out*.println(buf.getInt());

System.*out*.println(buf.toString());

System.*out*.println(buf.getInt());

System.*out*.println(buf.toString());

00 00 00 00 00 00 00 00

00 00 00 09 00 00 00 0C

9

java.nio.DirectByteBuffer[pos=4 lim=8 cap=8]

12

java.nio.DirectByteBuffer[pos=8 lim=8 cap=8]

**buf.getInt() 이렇게하면 4바이트씩 이동해서 자기가 뽑아서준다 인트형으로**

**buf.getShort() 이렇게하면 2바이트씩 이동해사 자기가 뽑아서준다**

**인덱스 값을줄때는 그 인덱스(바이트 인덱스)로 가서 getshort() 숏으로 getInt() 인트로.**

**쓰기**

**Buf.putInt(10); 이렇게하면 바이트4바이트잡고 인설트된다. 똑같다 위와.**

**이동해서 읽을려면 그쪽 4바이트에서부터 하나 인트형으로 읽어줘**

System.*out*.println();

System.*out*.println(buf.getInt(0));

System.*out*.println(buf.toString());

System.*out*.println(buf.getInt(4));

System.*out*.println(buf.toString());

buf.rewind();

System.*out*.println(buf.toString());

java.nio.DirectByteBuffer[pos=0 lim=8 cap=8]

12

java.nio.DirectByteBuffer[pos=0 lim=8 cap=8]

java.nio.DirectByteBuffer[pos=0 lim=8 cap=8]

**버퍼자체를 파라미터로 받고 쓰기**

Put에 바이트배열을 넣으면 그렇게된다 포지션이동됨.

**Wrap을 이용한 바이트배열에서 바이트버퍼로..**

**byte** [] a = {1,2,3,4,5,6,7};

ByteBuffer bbb = ByteBuffer.*wrap*(a);

a[1]=9;

**for** (**int** i = bbb.position(); i < bbb.limit(); i++)

{

System.*out*.format("%d",bbb.get(i));

}

100

1934567

**주의 바이트배열을 변경하면 바이트버퍼도 수정이된다 즉 주소만 가지고 들어간다.!!!**

**바이트 오더링 리틀인디안 빅인디안**

인풋할때도 오더해주면 그렇게 바뀌어서들어간다

buf.order(ByteOrder.*LITTLE\_ENDIAN*);

System.*out*.println(buf.order());

**for** (**int** i = 0; i < buf.limit(); i++)

{

System.*out*.format("%02X \t",buf.get(i));

}

System.*out*.println();

System.*out*.println(buf.getInt(0));

System.*out*.println(buf.getInt(4));

buf.rewind();

System.*out*.println(buf.toString());

System.*out*.println();

buf.order(ByteOrder.*BIG\_ENDIAN*);

System.*out*.println(buf.order());

**for** (**int** i = 0; i < buf.limit(); i++)

{

System.*out*.format("%02X \t",buf.get(i));

}

System.*out*.println();

System.*out*.println(buf.getInt(0));

System.*out*.println(buf.getInt(4));

buf.rewind();

System.*out*.println(buf.toString());

LITTLE\_ENDIAN

00 00 00 09 00 00 00 0C

150994944

201326592

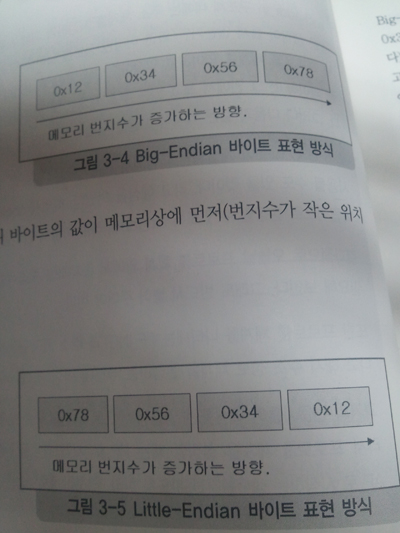
java.nio.DirectByteBuffer[pos=0 lim=8 cap=8]

BIG\_ENDIAN

00 00 00 09 00 00 00 0C

9

12

java.nio.DirectByteBuffer[pos=0 lim=8 cap=8]****

**CharBuffer**

CharBuffer ch = CharBuffer.*allocate*(10);

ch.put("가나다라");

System.*out*.println(ch.position());

4

//하지만 이건 CharBuffer로 입력 한거에만.. 그냥 바이트버퍼면 깨진다.

CharBuffer cbuf = buf.asCharBuffer();

String a = Cbuf.toString();

//////////////////////////

**ByteBuffer bv = ByteBuffer.*allocateDirect*(8);**

**bv.put("aaaaaa".getBytes());**

**bv.flip();**

**CharBuffer cv = bv.asCharBuffer();**

**System.*out*.println(cv.toString());**

**???**

**깨짐.**

**/////////////////////**

스트링으로빼온다

CharBuffer Wrap

CharBuffer c =CharBuffer.*wrap*("abcdefg");

**while** (c.hasRemaining())

{

System.*out*.println(c.get());

}

a

b

c

d

e

f

g

**뷰버퍼**

ByteBuffer buf = byTeBUffer.allocate(10);

IntBuffer ib =buf.asIntBUffer()

버퍼의 크기를 4바이트씩…읽고쓰고한다 ib는

**앞에서부터 채워진다.**

ByteBuffer buf = ByteBuffer.*allocateDirect*(10);

**for** (**int** i = 0; i < buf.limit(); i++)

{

System.*out*.format("%02X \t",buf.get(i));

}

System.*out*.println();

IntBuffer inbuf = buf.asIntBuffer();

inbuf.put(9);

inbuf.put(12);

**for** (**int** i = 0; i < buf.limit(); i++)

{

System.*out*.format("%02X \t",buf.get(i));

}

|  |
| --- |
| **ByteBuffer.*allocateDirect*(10);**  00 00 00 00 00 00 00 00 00 00  00 00 00 09 00 00 00 0C 00 00 |
| **ByteBuffer.*allocateDirect*(8);**  00 00 00 00 00 00 00 00  00 00 00 09 00 00 00 0C |

바이트 오더 및 컨버팅

// CONVERTING

**private** **static** **byte**[] intToByteArray(**final** **int** integer) {

ByteBuffer buff = ByteBuffer.*allocate*(Integer.*SIZE* / 8);

buff.putInt(integer);

buff.order(ByteOrder.*BIG\_ENDIAN*);

**return** buff.array();

}

**private** **static** **int** byteArrayToInt(**byte**[] bytes) {

**final** **int** size = Integer.*SIZE* / 8;

ByteBuffer buff = ByteBuffer.*allocate*(size);

**final** **byte**[] newBytes = **new** **byte**[size];

**for** (**int** i = 0; i < size; i++) {

**if** (i + bytes.length < size) {

newBytes[i] = (**byte**) 0x00;

} **else** {

newBytes[i] = bytes[i + bytes.length - size];

}

}

buff = ByteBuffer.*wrap*(newBytes);

buff.order(ByteOrder.*BIG\_ENDIAN*);

**return** buff.getInt();

}

/////////////////////////////////////

**public** **static** **byte**[] floatToByteArray(**float** f) {

ByteBuffer buff = ByteBuffer.*allocate*(Float.*SIZE* / 8);

buff.putFloat(f);

buff.order(ByteOrder.*BIG\_ENDIAN*);

**return** buff.array();

}

**public** **static** **float** byteArrayToFloat(**byte** bytes[]) {

**final** **int** size = Integer.*SIZE* / 8;

ByteBuffer buff = ByteBuffer.*allocate*(size);

**final** **byte**[] newBytes = **new** **byte**[size];

**for** (**int** i = 0; i < size; i++) {

**if** (i + bytes.length < size) {

newBytes[i] = (**byte**) 0x00;

} **else** {

newBytes[i] = bytes[i + bytes.length - size];

}

}

buff = ByteBuffer.*wrap*(newBytes);

buff.order(ByteOrder.*BIG\_ENDIAN*);

**return** buff.getFloat();

}