Buffer缓冲区

# Buffer缓冲区介绍

**Buffer抽象类及其子类**存在于**java.nio**包中，一种9个Buffer。另外，有一个ByteOrder是辅助**ByteBuffer**的。

有一个**Buffer抽象类**，其中有7个基本数据类型(除Boolean之外)对应的**XxxBuffer**子类，又ByteBuffer比较特殊，还有两个**MappedByteBuffer**和**ByteOrder**两个类。

由于Buffer是一个抽象类，集中了所有Buffer子类的共性，因此Buffer抽象类的学习很关键。

# Buffer抽象类

## Buffer类的简单介绍

public **abstract** class **Buffer** extends Object



Direct Known Subclasses:(7个基本数据类型的子类**XxxBuffer**)

ByteBuffer, CharBuffer, DoubleBuffer, FloatBuffer, IntBuffer, LongBuffer, ShortBuffer

## 重要的属性：capacity、limit、position

**Buffer抽象类**---A container for data of a specific primitive type.

A buffer is **a linear,** **finite sequence of elements of a specific primitive type**. **Aside from** its content, the essential properties of a buffer are **its capacity, limit, and position**:

### A **buffer’s capacity**

A buffer's capacity is the number of elements it contains. The capacity of a buffer is never negative and **never changes**.

### A **buffer's limit**

A buffer's limit is the index of the first element that should not be read or written. A buffer's limit is never negative and is never greater than its capacity.

### A **buffer's position**

A buffer's position is the index of the next element to be read or written. A buffer's position is never negative and is never greater than its limit.

There is one subclass of this class for **each non-boolean primitive type.**

## 特性

总结：有些是只读的、线程不安全的。

### **Transferring data传输数据**

Each subclass of this class defines two categories of **get and put operations**:

**Relative operations** **read or write** one or more elements starting at the current position and then increment the position by the number of elements transferred. If the requested transfer exceeds the limit then a relative **get** operation throws a BufferUnderflowException and a relative **put** operation throws a BufferOverflowException; in either case, no data is transferred.

**Absolute operations** **take** an **explicit** element index and do not affect the position. Absolute get and put operations throw an IndexOutOfBoundsException if the index argument exceeds the limit.

Data may also, of course, be transferred in to or out of a buffer by the **I/O operations** of an appropriate **channel**, which are always relative to the current position.

### **Marking and resetting**

**A buffer's mark** is the index to which its position will be reset when the **reset** method is invoked. The mark is not always defined, but when it is defined it is never negative and is never greater than the position. If the **mark** is defined then it is discarded when the position or the limit is adjusted to a value smaller than the mark. If the mark is not defined then invoking the reset method causes an InvalidMarkException to be thrown.

### Invariants

The following invariant holds for the mark, position, limit, and capacity values:

**0 <= mark <= position <= limit <= capacity**

A newly-created buffer always has a position of **zero** and a mark that is undefined. The initial limit may be zero, or it may be some other value that depends upon the type of the buffer and the manner in which it is constructed. Each element of a newly-allocated buffer is initialized to zero.

### **Clearing, flipping, and rewinding**

In addition to methods for accessing the position, limit, and capacity values and for marking and resetting, this class also defines the following operations upon buffers:

**clear()** makes a buffer ready for a new sequence of **channel-read** or relative **put** operations: It sets the limit to the capacity and the position to zero.

**flip()** makes a buffer ready for a new sequence of **channel-write** or relative **get** operations: It sets the limit to the current position and then sets the position to zero.

**rewind()** makes a buffer ready for **re-reading the data** that it already contains: It leaves the **limit** unchanged and sets the position to zero.

### Read-only buffers：

**Every buffer is readable**, but not every buffer is writable. The **mutation** methods of each buffer class are specified as optional operations that will throw a ReadOnlyBufferException when invoked upon a read-only buffer. A read-only buffer does not allow its content to be changed, but its mark, position, and limit values are mutable. Whether or not a buffer is read-only may be determined by invoking its **isReadOnly** method.

### Thread safety：线程不安全的

Buffers are not safe for use by multiple concurrent threads. If a buffer is to be used by more than one thread then access to the buffer should be controlled **by appropriate synchronization**.

### Invocation chaining:调用链

类似于StringBuilder和StringBuffer

Methods in this class that do not otherwise have a value to return are specified to return the buffer upon which they are invoked. **This allows method invocations to be chained**; for example, the sequence of statements

b.flip(); b.position(23); b.limit(42); can be replaced by the single, more compact statement **b.flip().position(23).limit(42);**

# Buffer抽象类的方法介绍

## capacity、limit、position、remaining

### capacity

int capacity() Returns this buffer's capacity.

### limit

int limit() ： Returns this buffer's limit.

**Buffer** limit(int newLimit)： Sets this buffer's limit.

### position

int position() ： Returns this buffer's position.

Buffer position(int newPosition)： Sets this buffer's position.

### remaining

int remaining()

Returns the number of elements between the current position and the limit.

## mark和reset

Buffer **mark()** Sets this buffer's mark at its position.

Buffer **reset()** Resets this buffer's position to the previously-marked position.

## clear, flip, and rewind

Buffer clear() Clears this buffer.

Buffer flip() Flips this buffer.

Buffer rewind() Rewinds this buffer.

## 抽象方法

abstract Object **array**()

Returns the array that backs this buffer (optional operation).

abstract int **arrayOffset**()

Returns the **offset** within this buffer's backing array of the first element of the buffer (optional operation).

abstract boolean **hasArray**()

Tells whether or not this buffer is backed by an accessible array.

boolean **hasRemaining**()

Tells whether there are any elements between the current position and the limit.

abstract boolean **isDirect**()：判断是否是直接缓冲区

Tells whether or not this buffer is direct.

abstract boolean **isReadOnly**()

Tells whether or not this buffer is read-only.

# XxxBuffer的共性

## 实例创建

创建方式：调用**allocate(int capacity)或wrap()**静态方法。

## 重要方法get 和 put 实现数据传输

注意：有些get和put方法返回本身，可以实现链式调用(Invocation chaining)。

## 都实现了Comparable<XxxBuffer>接口，因此都具有compareTo(XxxBuffer)方法。

## 都可以转换为数组，因此具有array方法；还一般可以复制，具有duplicate方法。

## direct vs non-direct buffer

通过wrap创建的一定是non-direct buffer,利用ShortBuffer为例：

Like a byte buffer, a short buffer is either **direct or non-direct**.

A short buffer created via the wrap methods of this class will be non-direct.

A short buffer created as a view of a byte buffer will be direct if, and only if, the byte buffer itself is direct.

Whether or not a short buffer is direct may be determined by invoking the **isDirect** method.

# ByteBuffer字节缓冲区

## 简单介绍

public **abstract** class **ByteBuffer** extends **Buffer** implements **Comparable<ByteBuffer>**

Direct Known Subclasses: **MappedByteBuffer**

**抽象类**

## 功能介绍

This class defines **six categories** of operations upon byte buffers:(六类操作)

• **Absolute and relative get and put** methods that read and write single bytes;

• **Relative bulk get methods** that transfer contiguous sequences of bytes from this buffer into an array;

• **Relative bulk put methods** that transfer contiguous sequences of bytes from a byte array or some other byte buffer into this buffer;

• **Absolute and relative get and put methods** that read and write values of other primitive types, translating them to and from sequences of bytes in a particular byte order;

• **Methods for creating view buffers**, which allow a byte buffer to be viewed as a buffer containing values of some other primitive type; and

• **Methods for compacting, duplicating, and slicing a byte buffer**.

Byte buffers can be created either by **allocation**, which allocates space for the buffer's content, or by **wrapping** an existing byte array into a buffer.

## Direct vs. non-direct buffers

A byte buffer is either direct or non-direct. Given a direct byte buffer, the Java virtual machine will **make a best effort to perform native I/O operations directly upon it**. That is, it will attempt to avoid copying the buffer's content to (or from) an intermediate buffer before (or after) each invocation of one of the underlying operating system's native I/O operations.

**A direct byte buffer** may be created by invoking the **allocateDirect** **factory** method of this class. The buffers returned by this method typically have somewhat higher allocation and deallocation costs than non-direct buffers. The contents of direct buffers may reside(居住，属于) outside of the normal **garbage-collected heap**, and so their impact upon the memory footprint of an application might not be obvious. **It is therefore recommended that direct buffers be allocated primarily for large, long-lived buffers that are subject to the underlying system's native I/O operations**. In general it is best to allocate direct buffers only when they yield a measureable gain in program performance.

**A direct byte buffer** may also be created by mapping a region of a file directly into memory. An implementation of the Java platform may optionally support the creation of direct byte buffers from native code via JNI. If an instance of one of these kinds of buffers refers to an inaccessible region of memory then an attempt to access that region will not change the buffer's content and will cause an unspecified exception to be thrown either at the time of the access or at some later time.

Whether a byte buffer is direct or non-direct may be determined by invoking its **isDirect** method. This method is provided so that explicit buffer management can be done in performance-critical code.

## Access to binary data

This class defines methods for **reading and writing values** of all other primitive types, except boolean. Primitive values are translated to (or from) sequences of bytes according to the buffer's **current byte order**, which may be retrieved and modified via the order methods. Specific byte orders are represented by instances of **the ByteOrder class**. The initial order of a byte buffer is always BIG\_**ENDIAN**.

For access to **heterogeneous(不同的，异类的)** binary data, that is, sequences of values of different types, this class defines a family of **absolute and relative get and put methods** for each type. For 32-bit floating-point values, for example, this class defines:

float getFloat()

float getFloat(int index)

void putFloat(float f)

void putFloat(int index, float f)

Corresponding methods are defined for the **types char, short, int, long, and double**. The index parameters of the absolute get and put methods are in terms of bytes rather than of the type being read or written.

For access to **homogeneous(同种的，同类的) binary data**, that is, sequences of values of the same type, this class defines methods that can create views of a given byte buffer. **A view buffer** is simply another buffer whose content is backed by the byte buffer. Changes to the byte buffer's content will be visible in the view buffer, and vice versa; the two buffers' position, limit, and mark values are independent. The asFloatBuffer method, for example, creates an instance of the FloatBuffer class that is backed by the byte buffer upon which the method is invoked. Corresponding view-creation methods are defined for the types char, short, int, long, and double.

View buffers have three important advantages over the families of **type-specific** get and put methods described above:

A view buffer is indexed not in terms of bytes but rather in terms of the type-specific size of its values;

A view buffer provides relative bulk get and put methods that can transfer contiguous sequences of values between a buffer and an array or some other buffer of the same type; and

A view buffer is potentially much more efficient because it will be direct if, and only if, its backing byte buffer is direct.

The byte order of a view buffer is fixed to be that of its byte buffer at the time that the view is created.

## Invocation chaining:调用链

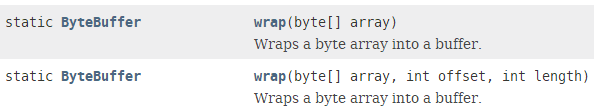
Methods in this class that do not otherwise have a value to return are specified to return the buffer upon which they are invoked. This allows method invocations to be chained. The sequence of statements

bb.putInt(0xCAFEBABE); bb.putShort(3); bb.putShort(45); can, for example, be replaced by the single statement **bb.putInt(0xCAFEBABE).putShort(3).putShort(45);**

## 创建ByteBuffer实例

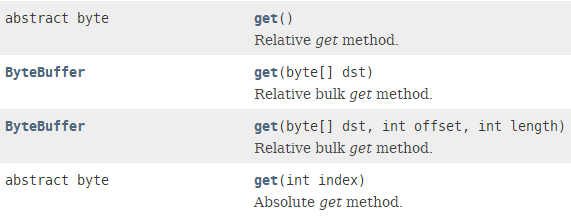
allocate 和 wrap方法





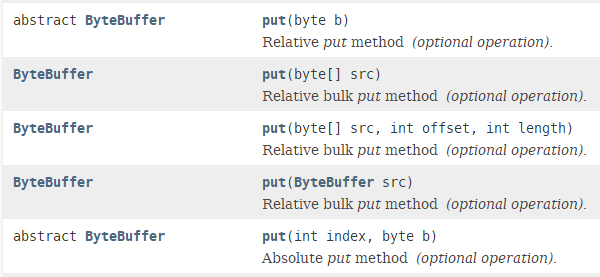
## get方法与put方法

### get方法



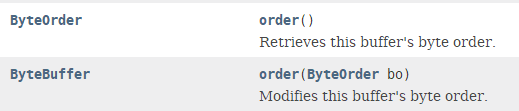
还有：getChar、getShort、getInt、getLong、getDouble

### put方法



与getXxx对象，有putChar、putShort、putInt、putLong、putDouble

## order



# ByteOrder介绍

## 简单介绍

public **final** class **ByteOrder** extends Object

----A typesafe **enumeration** for byte orders.



## 两个静态属性

static **ByteOrder** **BIG\_ENDIAN**

Constant denoting **big-endian** byte order. In this order, **the bytes of a multibyte value** are ordered from most significant to least significant.

static **ByteOrder** **LITTLE\_ENDIAN**

Constant denoting little-endian byte order. In this order, the bytes of a multibyte value are ordered **from least significant to most significant**.

## 静态方法nativeOrder

static **ByteOrder** **nativeOrder**()

Retrieves **the native byte order** of the underlying platform.

This method is defined so that **performance-sensitive Java code** can allocate direct buffers with the same byte order as the hardware. **Native code libraries** are often more efficient when such buffers are used.

## toString

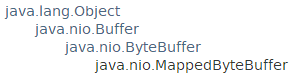
String toString()

Constructs a string describing this object.

This method returns the string "**BIG\_ENDIAN**" for BIG\_ENDIAN and "**LITTLE\_ENDIAN**" for LITTLE\_ENDIAN.

# MappedByteBuffer

## 简单介绍

public **abstract** class **MappedByteBuffer** extends **ByteBuffer** 

## 功能介绍

**A direct byte buffer** whose content is **a memory-mapped region** of a file.

Mapped byte buffers are created via **the FileChannel.map method**. This class extends the ByteBuffer class with operations that are specific to **memory-mapped file regions.**

A mapped byte buffer and the file mapping that it represents remain valid until the buffer itself is **garbage-collected**.

The content of a mapped byte buffer can change at any time, for example if the content of the corresponding region of the mapped file is changed by this program or another. Whether or not such changes occur, and when they occur, is operating-system dependent and therefore unspecified.

All or part of a mapped byte buffer may become inaccessible at any time, for example if the mapped file is truncated. An attempt to access an inaccessible region of a mapped byte buffer will not change the buffer's content and will cause an unspecified exception to be thrown either at the time of the access or at some later time. It is therefore strongly recommended that appropriate precautions be taken to avoid the manipulation of a mapped file by this program, or by a concurrently running program, except to read or write the file's content.

**Mapped byte buffers otherwise behave no differently than ordinary direct byte buffers**.

## 特有方法

**MappedByteBuffer** **force**()

Forces any changes made to this buffer's content to be written to the storage device containing the mapped file.

**boolean** **isLoaded**()

Tells whether or not this buffer's content is resident in physical memory.

**MappedByteBuffer** **load**()

Loads this buffer's content into physical memory.

# CharBuffer

## 简单介绍

public **abstract** class **CharBuffer** extends Buffer

implements **Comparable<CharBuffer>, Appendable, CharSequence, Readable**

## 特性

A char buffer.

This class defines four categories of operations upon char buffers:

**Absolute and relative** **get and put** methods that read and write single chars;

Relative **bulk get** methods that transfer contiguous sequences of chars from this buffer into an array; and

Relative **bulk put** methods that transfer contiguous sequences of chars from a char array, a string, or some other char buffer into this buffer; and

Methods for **compacting, duplicating, and slicing** a char buffer.

创建CharBuffer实例：allocate和wrap方法

**Char buffers** can be created either by **allocation**, which allocates space for the buffer's content, by **wrapping** an existing char array or string into a buffer, or by creating a view of an existing byte buffer.

direct和non-direct

Like a byte buffer, a char buffer is either direct or non-direct. A char buffer created via the wrap methods of this class will be non-direct. A char buffer created as a view of a byte buffer will be direct if, and only if, the byte buffer itself is direct. Whether or not a char buffer is direct may be determined by invoking the isDirect method.

This class implements the **CharSequence** interface so that character buffers may be used wherever character sequences are accepted, for example in the **regular-expression** package ***java.util.regex***.

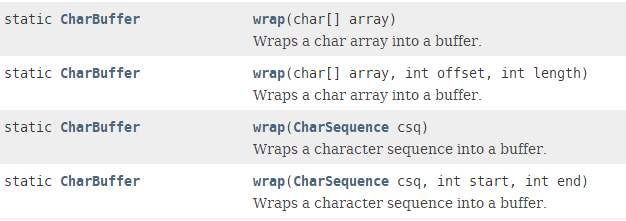
链式调用

Methods in this class that do not otherwise have a value to return are specified to return the buffer upon which they are invoked. This allows method invocations to be chained. The sequence of statements cb.put("text/"); cb.put(subtype); cb.put("; charset="); cb.put(enc); can, for example, be replaced by the single statement **cb.put("text/").put(subtype).put("; charset=").put(enc);**

## 创建CharBuffer实例

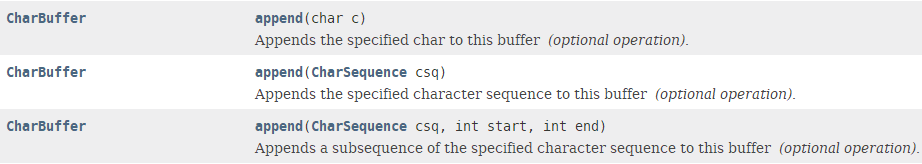
allocate、wrap





## 特有方法

### append：实现了Appendable接口



### 重要方法get和put

### compareTo



### duplicate方法



### read方法



# IntBuffer

## 简单介绍

public **abstract** class **IntBuffer** extends **Buffer** implements **Comparable<IntBuffer>**

**IntBuffer也是一个抽象类。**

## 功能介绍

This class defines four **categories** of operations upon int buffers:

Absolute and relative **get** and **put** methods that read and write single ints;

Relative **bulk get** methods that transfer contiguous sequences of ints from this buffer into an array; and Relative **bulk put** methods that transfer contiguous sequences of ints from an int array or some other int buffer into this buffer; and Methods for compacting, duplicating, and slicing an int buffer.

**Int buffers** can be created either by allocation, which allocates space for the buffer's content, by wrapping an existing **int array** into a buffer, or by creating a view of an existing byte buffer.

**Like a byte buffer, an int buffer is either direct or non-direct.** A int buffer created via the **wrap** methods of this class will be **non-direct**. An int buffer created as a view of **a byte buffer** will be **direct** if, and only if, the byte buffer itself is direct. Whether or not an int buffer is direct may be determined by invoking the **isDirect** method.

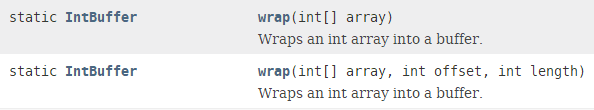
Methods in this class that do not otherwise have a value to return are specified to return the buffer upon which they are invoked. This allows method invocations to be chained.

## 静态方法:创建IntBuffer实例

allocate(int capacity)



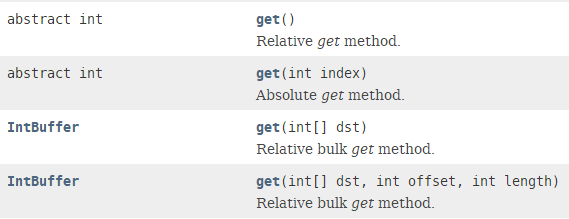
wrap==>non-direct



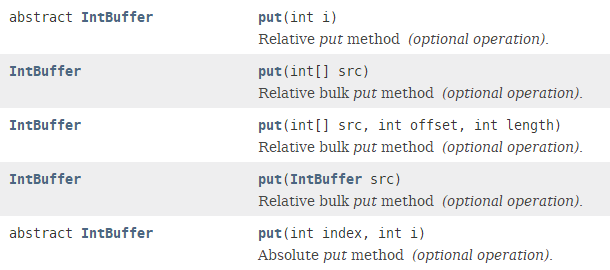
Note：A int buffer created via the **wrap** methods of this class will be **non-direct**. An int buffer created as a view of **a byte buffer** will be **direct** if, and only if, the byte buffer itself is direct.

## 一般方法

### get方法



### put方法



# ShortBuffer

public **abstract** class **ShortBuffer** extends Buffer implements **Comparable<ShortBuffer>**

A short buffer.

This class defines four categories of operations upon short buffers:

**Absolute and relative get and put methods** that read and write single shorts;

Relative bulk get methods that transfer contiguous sequences of shorts from this buffer into an array; and

**Relative bulk put** methods that transfer contiguous sequences of shorts from a short array or some other short buffer into this buffer; and

Methods for compacting, duplicating, and slicing a short buffer.

Short buffers can be created either by allocation, which allocates space for the buffer's content, by wrapping an existing short array into a buffer, or by creating a view of an existing byte buffer.

Like a byte buffer, **a short buffer is either direct or non-direct**. A short buffer created via the wrap methods of this class will be non-direct. A short buffer created as a view of a byte buffer will be direct if, and only if, the byte buffer itself is direct. Whether or not a short buffer is direct may be determined by invoking the isDirect method.

Methods in this class that do not otherwise have a value to return are specified to return the buffer upon which they are invoked. This allows method **invocations to be chained.**

# LongBuffer

public abstract class LongBuffer extends Buffer implements **Comparable<LongBuffer>**

与ShortBuffer类似。

# FloatBuffer

public abstract class **FloatBuffer** extends Buffer implements **Comparable<FloatBuffer>**

# DoubleBuffer

public abstract class **DoubleBuffer** extends Buffer implements **Comparable<DoubleBuffer>**