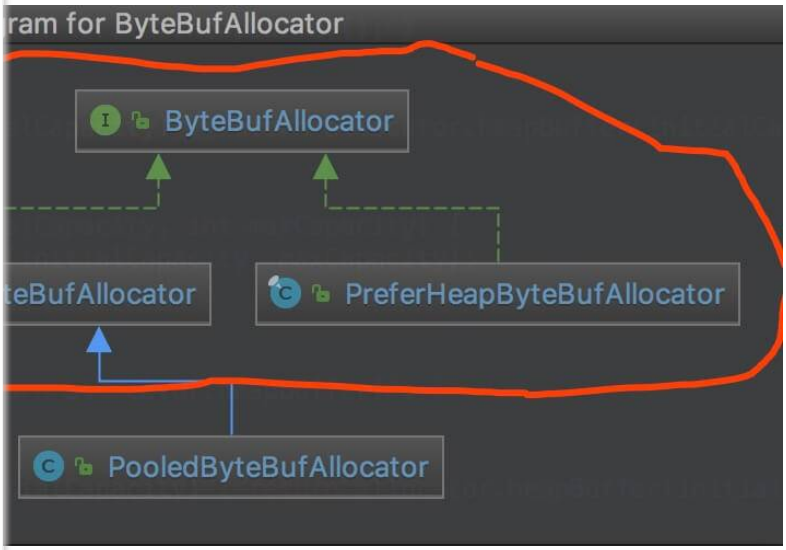
文章目录

Buffer 之 ByteBufAllocator（一）简

的分配器，负责创建 ByteBuf 对象。它的子类类图如下：



- PreferHeapByteBufAllocator ，倾向创建 **Heap** ByteBuf 的分配器。
- PooledByteBufAllocator ，基于**内存池**的 ByteBuf 的分配器。
- UnpooledByteBufAllocator ，**普通**的 ByteBuf 的分配器。

本文分享上面类图红框部分，后面两篇文章再分别分享 UnpooledByteBufAllocator 和 PooledByteBufAllocator 。

2. ByteBufAllocator

`io.netty.buffer.ByteBufAllocator` ，ByteBuf 分配器**接口**。

还是老样子，我们逐个来看看每个方法。

2.1 DEFAULT

```
ByteBufAllocator DEFAULT = ByteBufUtil.DEFAULT_ALLOCATOR;
```

- 默认 ByteBufAllocator 对象，通过 `ByteBufUtil.DEFAULT_ALLOCATOR` 中获得。代码如下：

```
static final ByteBufAllocator DEFAULT_ALLOCATOR;

static {
    // 读取 ByteBufAllocator 配置
    String allocType = SystemPropertyUtil.get("io.netty.allocator.type", PlatformDependent.isAndro
    allocType = allocType.toLowerCase(Locale.US).trim();

    // 读取 ByteBufAllocator 对象
    ByteBufAllocator alloc;
    if ("unpooled".equals(allocType)) {
        alloc = UnpooledByteBufAllocator.DEFAULT;
        logger.debug("-Dio.netty.allocator.type: {}", allocType);
    } else if ("pooled".equals(allocType)) {
        alloc = PooledByteBufAllocator.DEFAULT;
        logger.debug("-Dio.netty.allocator.type: {}", allocType);
    } else {
        alloc = PooledByteBufAllocator.DEFAULT;
        logger.debug("-Dio.netty.allocator.type: pooled (unknown: {})", allocType);
    }
```

文章目录

ocator 作为默认 ByteBufAllocator 对象。

ocator 作为默认 ByteBufAllocator 对象。因为 Android 客户端的内

体创建的是 Heap ByteBuf 还是 Direct ByteBuf ，由实现类决定。

```
/**
 * Allocate a {@link ByteBuf}. If it is a direct or heap buffer
 * depends on the actual implementation.
 */
ByteBuf buffer();
ByteBuf buffer(int initialCapacity);
ByteBuf buffer(int initialCapacity, int maxCapacity);
```

2.2.1 ioBuffer

`#ioBuffer(...)` 方法，创建一个用于 IO 操作的 ByteBuf 对象。倾向于 Direct ByteBuf ，因为对于 IO 操作来说，性能更优。

```
/**
 * Allocate a {@link ByteBuf}, preferably a direct buffer which is suitable for I/O.
 */
ByteBuf ioBuffer();
ByteBuf ioBuffer(int initialCapacity);
ByteBuf ioBuffer(int initialCapacity, int maxCapacity);
```

2.2.2 heapBuffer

文章目录

对象。代码如下：

```
y)
y, nt maxCapacity);
```

ffer 对象。代码如下：

```
wi the given initial capacity.
it ;
it int maxCapacity);
```

osite ByteBuf 对象。具体创建的是 Heap ByteBuf 还是 Direct ByteBuf，由实现类决定。

```
/**
 * Allocate a {@link CompositeByteBuf}.
 * If it is a direct or heap buffer depends on the actual implementation.
 */
CompositeByteBuf compositeBuffer();
CompositeByteBuf compositeBuffer(int maxNumComponents);
```

2.3.1 compositeHeapBuffer

`#compositeHeapBuffer(...)` 方法，创建一个 Composite Heap ByteBuf 对象。代码如下：

```
/**
 * Allocate a heap {@link CompositeByteBuf}.
 */
CompositeByteBuf compositeHeapBuffer();
CompositeByteBuf compositeHeapBuffer(int maxNumComponents);
```

2.3.2 compositeDirectBuffer

`#compositeDirectBuffer(...)` 方法，创建一个 Composite Direct ByteBuf 对象。代码如下：

```
/**
 * Allocate a direct {@link CompositeByteBuf}.
 */
CompositeByteBuf compositeDirectBuffer();
CompositeByteBuf compositeDirectBuffer(int maxNumComponents);
```


ect ByteBuf 对象池。代码如下：

```
ir  ByteBuf}'s are pooled
```

, int maxCapacity) 方法，在 ByteBuf 扩容时，计算新的容量，该容
围内。代码如下：

```
li  ByteBuf} that is used when a {@link ByteBuf} needs to expand
 m  Capacity} as upper-bound.

pa  ty, int maxCapacity);
```

3. AbstractByteBufAllocator

`io.netty.buffer.AbstractByteBufAllocator` ，实现 ByteBufAllocator 接口，ByteBufAllocator 抽象实现类，为 PooledByteBufAllocator 和 UnpooledByteBufAllocator 提供公共的方法。

3.1 构造方法

```
/**
 * 是否倾向创建 Direct ByteBuf
 */
private final boolean directByDefault;
/**
 * 空 ByteBuf 缓存
 */
private final ByteBuf emptyBuf;

/**
 * Instance use heap buffers by default
 */
```

```
protected AbstractByteBufAllocator() {
    this(false);
}

/**
 * Create new instance
 *
 * @param preferDirect {@code true} if {@link #buffer(int)} should try to allocate a direct buffer rat
 *                     a heap buffer
```

文章目录


```
n preferDirect) {
tformDependent.hasUnsafe(); // 支持 Unsafe
;
```

Dir ByteBuf 。有一个前提是需要支持 Unsafe 操作。

用于 #buffer() 等方法，创建**空** ByteBuf 对象时。

```
ci ) {

pa ty);
    return heapBuffer(initialCapacity);
}
@Override
public ByteBuf buffer(int initialCapacity, int maxCapacity) {
    if (directByDefault) {
        return directBuffer(initialCapacity, maxCapacity);
    }
    return heapBuffer(initialCapacity, maxCapacity);
}
```

- 根据 directByDefault 的值，调用 #directBuffer(...) 方法，还是调用 #heapBuffer(...) 方法。

3.2.1 ioBuffer

```
/**
 * 默认容量大小
 */
static final int DEFAULT_INITIAL_CAPACITY = 256;
```

```java
@Override
public ByteBuf ioBuffer() {
    if (PlatformDependent.hasUnsafe()) {
        return directBuffer(DEFAULT_INITIAL_CAPACITY);
    }
    return heapBuffer(DEFAULT_INITIAL_CAPACITY);
}

@Override
  ByteBuf ioBuffer(int initialCapa  ty) {
                                   )
                                   pa  ty);

                                   );



                                   pa  ty, int maxCapacity) {
                                   )
                                   pa  ty, maxCapacity);

                                   ,   xCapacity);
```

di ctBuffer(...) 方法，还是调用 #heapBuffer(...) 方法。

文章目录

```
/**
 * 默认最大容量大小，无限。
 */
static final int DEFAULT_MAX_CAPACITY = Integer.MAX_VALUE;

@Override
public ByteBuf heapBuffer() {
    return heapBuffer(DEFAULT_INITIAL_CAPACITY, DEFAULT_MAX_CAPACITY);
}
```

文章目录

`Ca city, int maxCapacity)` **抽象**方法，创建 Heap ByteBuf 对象。代

`the given initialCapacity and maxCapacity.`

`er(int initialCapacity, int maxCapacity);`

- 因为是否基于对象池的方式，创建 Heap ByteBuf 对象的实现会不同，所以需要抽象。

3.2.3 directBuffer

```
@Override
public ByteBuf directBuffer() {
    return directBuffer(DEFAULT_INITIAL_CAPACITY, DEFAULT_MAX_CAPACITY);
}

@Override
public ByteBuf directBuffer(int initialCapacity) {
    return directBuffer(initialCapacity, DEFAULT_MAX_CAPACITY);
}

@Override
public ByteBuf directBuffer(int initialCapacity, int maxCapacity) {
    // 空 ByteBuf 对象
    if (initialCapacity == 0 && maxCapacity == 0) {
        return emptyBuf;
```

```
    }
    validate(initialCapacity, maxCapacity); // 校验容量的参数
    // 创建 Direct ByteBuf 对象
    return newDirectBuffer(initialCapacity, maxCapacity);
}
```

- 最终调用 `#newDirectBuffer(int initialCapacity, int maxCapacity)` **抽象**方法，创建 Direct ByteBuf 对象。代码如下：


```
/**
 w  the given initialCapacity and maxCapacity.
 :t  ffer(int initialCapacity, int maxCapacity);
B  Buf 对象的实现会不同，所以需要抽象。
(
I  er  {
 );
(
I  er  nt maxNumComponents) {
 ma  umComponents);
 Co  onents);
```

- 根据 `directByDefault` 的值，调用 `#compositeDirectBuffer(...)` 方法，还是调用 `#compositeHeapBuffer(...)` 方法。

3.3.1 compositeHeapBuffer

```
/**
 * Composite ByteBuf 可包含的 ByteBuf 的最大数量
 */
static final int DEFAULT_MAX_COMPONENTS = 16;

@Override
public CompositeByteBuf compositeHeapBuffer() {
    return compositeHeapBuffer(DEFAULT_MAX_COMPONENTS);
}

@Override
public CompositeByteBuf compositeHeapBuffer(int maxNumComponents) {
```

```
    return toLeakAwareBuffer(new CompositeByteBuf(this, false, maxNumComponents));
}
```

- 创建 CompositeByteBuf 对象，并且方法参数 `direct` 为 `false` ，表示 Heap 类型。
- 调用 `#toLeakAwareBuffer(CompositeByteBuf)` 方法，装饰成 LeakAware 的 ByteBuf 对象。

3.3.2 compositeDirectBuffer


```
@Override
ct ffer() {
Ul MAX_COMPONENTS);



ct ffer(int maxNumComponents) {
os eByteBuf(this, true, maxNumComponents));
```

数 irect 为 true ，表示 Direct 类型。

te f) 方法，装饰成 LeakAware 的 ByteBuf 对象。

Buf 三）内存泄露检测》 中的 「3.1 创建 LeakAware ByteBuf 对象」 小节，

```
static final int CALCULATE_THRESHOLD = 48576 * 4; // 4 MiB page
```

```
 1: @Override
 2: public int calculateNewCapacity(int minNewCapacity, int maxCapacity) {
 3:     if (minNewCapacity < 0) {
 4:         throw new IllegalArgumentException("minNewCapacity: " + minNewCapacity + " (expected: 0+)
 5:     }
 6:     if (minNewCapacity > maxCapacity) {
 7:         throw new IllegalArgumentException(String.format(
 8:                 "minNewCapacity: %d (expected: not greater than maxCapacity(%d)",
 9:                 minNewCapacity, maxCapacity));
10:     }
11:     final int threshold = CALCULATE_THRESHOLD; // 4 MiB page
12:
13:     // <1> 等于 threshold ，直接返回 threshold 。
14:     if (minNewCapacity == threshold) {
15:         return threshold;
16:     }
17:
18:     // <2> 超过 threshold ，增加 threshold ，不超过 maxCapacity 大小。
```

```
19:     // If over threshold, do not double but just increase by threshold.
20:     if (minNewCapacity > threshold) {
21:         int newCapacity = minNewCapacity / threshold * threshold;
22:         if (newCapacity > maxCapacity - threshold) { // 不超过 maxCapacity
23:             newCapacity = maxCapacity;
24:         } else {
25:             newCapacity += threshold;
26:         }
27:         return newCapacity;
    }
```

文章目录

64 开始两倍计算，不超过 4M 大小。
e to 4 MiB, starting from 64.

ap ity) {

m Capacity);

成 3 种情况： <1> / <2> / <3> 。代码比较简单，胖友自己看注释。

cator

lo tor ，实现 ByteBufAllocator 接口，**倾向创建 Heap ByteBuf** 的分配器。
.. 和 #compositeBuffer(...) 方法，创建的都是 Heap ByteBuf 对象。

to

```
public PreferHeapByteBufAllocator(ByteBufAllocator allocator) {
    this.allocator = ObjectUtil.checkNotNull(allocator, "allocator");
}

@Override
public ByteBuf buffer() {
    return allocator.heapBuffer();
}

@Override
public ByteBuf ioBuffer() {
    return allocator.heapBuffer();
}

@Override
public CompositeByteBuf compositeBuffer() {
    return allocator.compositeHeapBuffer();
}
```

其它方法，就是调用 `allocator` 的对应的方法。

666. 彩蛋

😈 小水文一篇。铺垫铺垫，你懂的。

文章目录