- NETTY

精尽 Netty 源码解析 —— Buffer 之 ByteBufAllocator（二）UnpooledByteBufAllocator

1. 概述

本文，我们来分享 UnpooledByteBufAllocator ，**普通**的 ByteBuf 的分配器，**不基于内存池。**

2. ByteBufAllocatorMetricProvider

`io.netty.buffer.ByteBufAllocatorMetricProvider` ，ByteBufAllocator Metric 提供者接口，**用于监控 ByteBuf 的 Heap 和 Direct 占用内存的情况。**代码如下：

```
public interface ByteBufAllocatorMetricProvider {

    /**
```

文章目录


```
ByteBufAllocator}.
```

PooledByteBufAllocator 。

接口。代码如下：

```
@link ByteBufAllocator} or {@code -1} if

    /**
     * Returns the number of bytes of direct memory used by a {@link ByteBufAllocator} or {@code -1} i
     *
```

```
     * 已使用 Direct 占用内存大小
     */
    long usedDirectMemory();

}
```

ByteBufAllocatorMetric 有两个子类：UnpooledByteBufAllocatorMetric 和 PooledByteBufAllocatorMetric 。

3.1 UnpooledByteBufAllocatorMetric

UnpooledByteBufAllocatorMetric ，在 UnpooledByteBufAllocator 的**内部静态类**，实现 ByteBufAllocatorMetric 接口，UnpooledByteBufAllocator Metric 实现类。代码如下：

```
/**
 * Direct ByteBuf 占用内存大小
 */
final LongCounter directCounter = PlatformDependent.newLongCounter();
/**
 * Heap ByteBuf 占用内存大小
 */
final LongCounter heapCounter = PlatformDependent.newLongCounter();

@Override
public long usedHeapMemory() {
    return heapCounter.value();
}
```

文章目录

象。代码如下：

```
 for the current platform.
```

dder ，JDK <7 使用
写多读少，所以 LongAdder 比
。

4. UnpooledByteBufAllocator

`io.netty.buffer.UnpooledByteBufAllocator` ，实现 ByteBufAllocatorMetricProvider 接口，继承 AbstractByteBufAllocator 抽象类，**普通**的 ByteBuf 的分配器，**不基于内存池。**

4.1 构造方法

```
/**
 * Metric
 */
private final UnpooledByteBufAllocatorMetric metric = new UnpooledByteBufAllocatorMetric();
/**
 * 是否禁用内存泄露检测功能
 */
private final boolean disableLeakDetector;
/**
 * 不使用 `io.netty.util.internal.Cleaner` 释放 Direct ByteBuf
 *
 * @see UnpooledUnsafeNoCleanerDirectByteBuf
 * @see InstrumentedUnpooledUnsafeNoCleanerDirectByteBuf
 */
private final boolean noCleaner;

public UnpooledByteBufAllocator(boolean preferDirect) {
    this(preferDirect, false);
}

public UnpooledByteBufAllocator(boolean preferDirect, boolean disableLeakDetector) {
    this(preferDirect, disableLeakDetector, PlatformDependent.useDirectBufferNoCleaner() /** 返回 true
```

文章目录


```
hould try to allocate a direct buffer rat

ion should be disabled completely for thi
f the user just want to depend on the GC
t released.
@link PlatformDependent#allocateDirectNoC

 disableLeakDetector, boolean tryNoCleane

 /** 返回 true **/
tructor() /** 返回 true **/ ;
```

- 默认为 `false` 。
- `noCleaner` 属性，是否不使用 `io.netty.util.internal.Cleaner` 来释放 Direct ByteBuf 。
  - 默认为 `true` 。

- 详细解析，见「5.5 InstrumentedUnpooledUnsafeNoCleanerDirectByteBuf」。

4.2 newHeapBuffer

```
@Override
protected ByteBuf newHeapBuffer(int initialCapacity, int maxCapacity) {
    return PlatformDependent.hasUnsafe() ?
            new InstrumentedUnpooledUnsafeHeapByteBuf(this, initialCapacity, maxCapacity) :
            new InstrumentedUnpooledHeapByteBuf(this, initialCapacity, maxCapacity);
}
```

- 创建的是以 `"Instrumented"` 的 Heap ByteBuf 对象，因为要结合 Metric 。详细解析，见「5. Instrumented ByteBuf」。

4.3 newDirectBuffer

```
@Override
protected ByteBuf newDirectBuffer(int initialCapacity, int maxCapacity) {
    final ByteBuf buf;
    if (PlatformDependent.hasUnsafe()) {
        buf = noCleaner ? new InstrumentedUnpooledUnsafeNoCleanerDirectByteBuf(this, initialCapacity,
                new InstrumentedUnpooledUnsafeDirectByteBuf(this, initialCapacity, maxCapacity);
    } else {
        buf = new InstrumentedUnpooledDirectByteBuf(this, initialCapacity, maxCapacity);
    }
    return disableLeakDetector ? buf : toLeakAwareBuffer(buf);
```

文章目录

ic 。详细解析，见「5. Instrumented

```
nts) {
maxNumComponents);
;
```

```
@Override
public CompositeByteBuf compositeDirectBuffer(int maxNumComponents) {
    CompositeByteBuf buf = new CompositeByteBuf(this, true, maxNumComponents);
    return disableLeakDetector ? buf : toLeakAwareBuffer(buf);
}
```

- 结合了 `disableLeakDetector` 属性。

4.6 isDirectBufferPooled

```
@Override
public boolean isDirectBufferPooled() {
    return false;
}
```

4.7 Metric 相关操作方法

```
@Override
public ByteBufAllocatorMetric metric() {
    return metric;
}

void incrementDirect(int amount) { // 增加 Direct
    metric.directCounter.add(amount);
}
```

文章目录

r 的**内部静态类**，继承

```
private static final class InstrumentedUnpooledUnsafeHeapByteBuf extends UnpooledUnsafeHeapByteBuf {

    InstrumentedUnpooledUnsafeHeapByteBuf(UnpooledByteBufAllocator alloc, int initialCapacity, int max
        super(alloc, initialCapacity, maxCapacity);
```

```
    }

    @Override
    protected byte[] allocateArray(int initialCapacity) {
        byte[] bytes = super.allocateArray(initialCapacity);
        // Metric ++
        ((UnpooledByteBufAllocator) alloc()).incrementHeap(bytes.length);
        return bytes;
    }

    @Override
    protected void freeArray(byte[] array) {
        int length = array.length;
        super.freeArray(array);
        // Metric --
        ((UnpooledByteBufAllocator) alloc()).decrementHeap(length);
    }

}
```

- 在原先的基础上，调用 Metric 相应的增减操作方法，得以记录 Heap 占用内存的大小。

5.2 InstrumentedUnpooledHeapByteBuf

InstrumentedUnpooledHeapByteBuf ，在 UnpooledByteBufAllocator 的**内部静态类**，继承 UnpooledHeapByteBuf 类。代码如下：

文章目录


```
tends UnpooledHeapByteBuf {

alloc, int initialCapacity, int maxCapaci

tes.length);

ngth);

}
```

- 在原先的基础上，调用 Metric 相应的增减操作方法，得以记录 Heap 占用内存的大小。

5.3 InstrumentedUnpooledUnsafeDirectByteBuf

InstrumentedUnpooledUnsafeDirectByteBuf ，在 UnpooledByteBufAllocator 的**内部静态类**，继承 UnpooledUnsafeDirectByteBuf 类。代码如下：

```
private static final class InstrumentedUnpooledUnsafeDirectByteBuf extends UnpooledUnsafeDirectByteBuf
    InstrumentedUnpooledUnsafeDirectByteBuf(
            UnpooledByteBufAllocator alloc, int initialCapacity, int maxCapacity) {
        super(alloc, initialCapacity, maxCapacity);
    }

    @Override
    protected ByteBuffer allocateDirect(int initialCapacity) {
        ByteBuffer buffer = super.allocateDirect(initialCapacity);
        // Metric ++
        ((UnpooledByteBufAllocator) alloc()).incrementDirect(buffer.capacity());
        return buffer;
    }

    @Override
    protected void freeDirect(ByteBuffer buffer) {
        int capacity = buffer.capacity();
        super.freeDirect(buffer);
        // Metric --
        ((UnpooledByteBufAllocator) alloc()).decrementDirect(capacity);
    }
```

文章目录

内存的大小。

yteBuf 类。代码如下：

```
extends UnpooledDirectByteBuf {

ty, int maxCapacity) {

{
ity);

buffer.capacity());

    }

    @Override
```

```
    protected void freeDirect(ByteBuffer buffer) {
        int capacity = buffer.capacity();
        super.freeDirect(buffer);
        // Metric --
        ((UnpooledByteBufAllocator) alloc()).decrementDirect(capacity);
    }

}
```

- 在原先的基础上，调用 Metric 相应的增减操作方法，得以记录 Direct 占用内存的大小。

5.5 InstrumentedUnpooledUnsafeNoCleanerDirectByteBuf

InstrumentedUnpooledDirectByteBuf 的**内部静态类**，继承 UnpooledUnsafeNoCleanerDirectByteBuf 类。代码如下：

```
private static final class InstrumentedUnpooledUnsafeNoCleanerDirectByteBuf
        extends UnpooledUnsafeNoCleanerDirectByteBuf {

    InstrumentedUnpooledUnsafeNoCleanerDirectByteBuf(
            UnpooledByteBufAllocator alloc, int initialCapacity, int maxCapacity) {
        super(alloc, initialCapacity, maxCapacity);
    }

    @Override
    protected ByteBuffer allocateDirect(int initialCapacity) {
        ByteBuffer buffer = super.allocateDirect(initialCapacity);
        // Metric --
        ((UnpooledUnsafeNoCleanerDirectByteBuf) alloc()).incrementDirect(buffer.capacity());
    }

    @Override
    protected void freeDirect(ByteBuffer buffer, int capacity) {
        super.freeDirect(buffer, capacity);
        // Metric --
        ((UnpooledUnsafeNoCleanerDirectByteBuf) alloc()).decrementDirect(buffer.capacity() - capacity);
    }

}
```

文章目录

- 在原先的基础上，调用 Metric 相应的增减操作方法，得以记录 Heap 占用内存的大小。

5.5.1 UnpooledUnsafeNoCleanerDirectByteBuf

`io.netty.buffer.UnpooledUnsafeNoCleanerDirectByteBuf` ，继承 UnpooledUnsafeDirectByteBuf 类。代码如下：

```
class UnpooledUnsafeNoCleanerDirectByteBuf extends UnpooledUnsafeDirectByteBuf {

    UnpooledUnsafeNoCleanerDirectByteBuf(ByteBufAllocator alloc, int initialCapacity, int maxCapacity)
        super(alloc, initialCapacity, maxCapacity);
    }

    @Override
    protected ByteBuffer allocateDirect(int initialCapacity) {
        // 反射，直接创建 ByteBuffer 对象。并且该对象不带 Cleaner 对象
        return PlatformDependent.allocateDirectNoCleaner(initialCapacity);
    }

    ByteBuffer reallocateDirect(ByteBuffer oldBuffer, int initialCapacity) {
        return PlatformDependent.reallocateDirectNoCleaner(oldBuffer, initialCapacity);
    }

    @Override
    protected void freeDirect(ByteBuffer buffer) {
        // 直接释放 ByteBuffer 对象
        PlatformDependent.freeDirectNoCleaner(buffer);
    }

    @Override
    public ByteBuf capacity(int newCapacity) {
```

文章目录


```
pacity);

ity ，避免越界

        setByteBuffer(newBuffer, false);
        return this;
    }
```

```
}
```

> FROM 《Netty源码分析（一） ByteBuf》
>
> 和 UnpooledUnsafeDirectByteBuf 最大区别在于 UnpooledUnsafeNoCleanerDirectByteBuf 在 allocate的时候通过反射构造函数的方式创建DirectByteBuffer，这样在DirectByteBuffer中没有对应的Cleaner函数(通过ByteBuffer.allocateDirect的方式会自动生成Cleaner函数，Cleaner用于内存回收，具体可以看源码)，内存回收时，UnpooledUnsafeDirectByteBuf通过调用DirectByteBuffer中的Cleaner函数回收，而UnpooledUnsafeNoCleanerDirectByteBuf直接使用UNSAFE.freeMemory(address)释放内存地址。

666. 彩蛋

😈 小水文一篇。铺垫铺垫，你懂的。

文章目录