

PackedInts (二)

本文承接[PackedInts \(一\)](#)，继续介绍剩余的内容。

压缩实现

在上一篇文章中，我们介绍了Lucene 7.5.0中PackedInts提供的几种压缩实现，如下所示：

表1：

数据分布	是否有填充bit	是否单block单值	实现类
一个block	否	是	Direct8 Direct16 Direct32 Direct64
	是	否	Packed64SingleBlock1 Packed64SingleBlock2 Packed64SingleBlock3 Packed64SingleBlock4 Packed64SingleBlock5 Packed64SingleBlock6 Packed64SingleBlock7 Packed64SingleBlock8 Packed64SingleBlock9 Packed64SingleBlock10 Packed64SingleBlock12 Packed64SingleBlock16 Packed64SingleBlock21 Packed64SingleBlock32
两个block	否	否	Packed64
三个block	否	-	Packed8ThreeBlocks Packed16ThreeBlocks

我们接着介绍如何选择这些压缩实现：

在源码中Lucene会根据使用者提供的三个参数来选择其中一种压缩实现，即PackedInts类中的[getMutable\(int valueCount, int bitsPerValue, float acceptableOverheadRatio\)](#)方法，参数如下所示：

- valueCount：描述待处理的数据集的数量
- bitsPerValue：描述待处理的数据集中的最大值，它的有效数据占用的bit个数
- acceptableOverheadRatio：描述可接受的开销

什么是可接受的开销acceptableOverheadRatio:

- bitsPerValue描述了每一个数值占用的bit位数， acceptableOverheadRatio则是每一个数值额外的空间开销的比例，允许使用比bitsPerValue更多的bit位，我们称之为maxBitsPerValue，来存储每一个数值，计算公式如下所示：

```
1      int maxBitsPerValue = bitsPerValue + (int)(bitsPerValue *  
      acceptableOverheadRatio)
```

例如我们有以下的数据集：

数组一：

```
1      long[] values = {3, 8, 7, 12, 18};
```

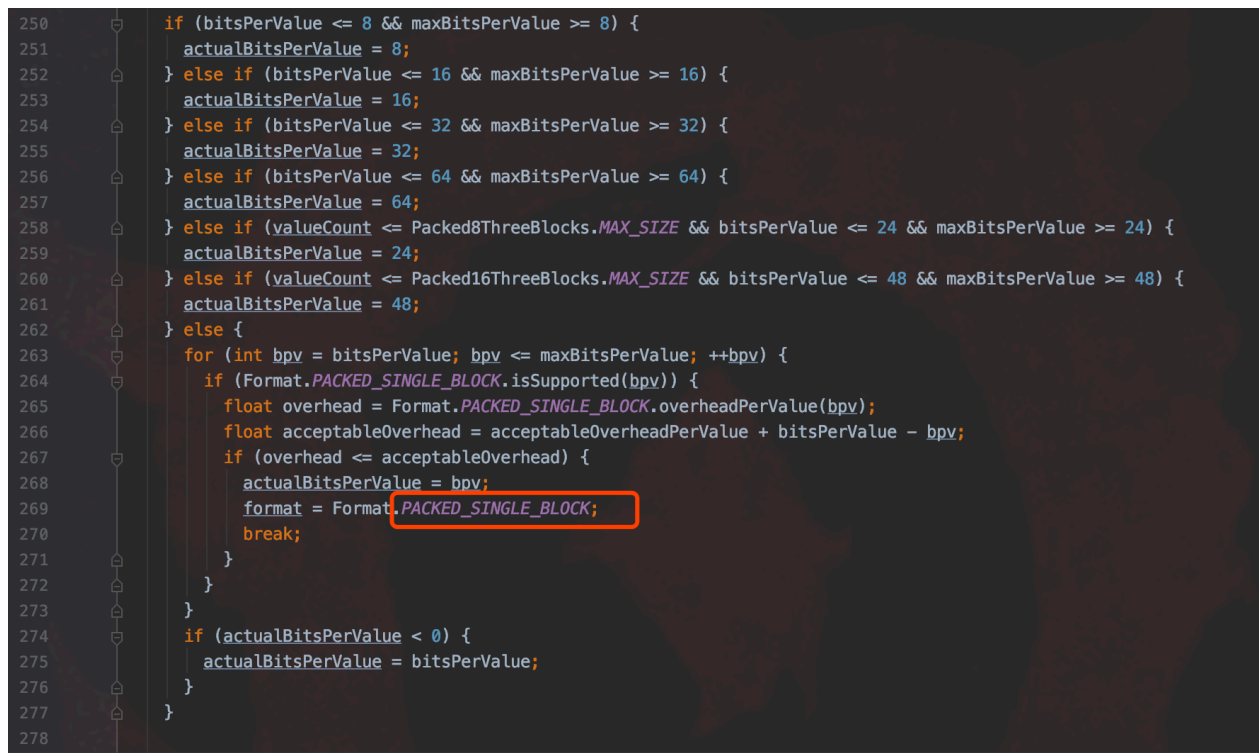
该数组的bitsPerValue为5，如果此时acceptableOverheadRatio的值为7，那么maxBitsPerValue = 5 + 5*7 = 40，即允许使用40个bit位来存储数组一。

当然Lucene并不会真正的使用40个bit来存储一个数值，**maxBitsPerValue**只是用来描述使用者可接受的额外开销的程度。

为什么要使用acceptableOverheadRatio:

使得在使用者可接受的额外开销前提下，尽量使用读写性能最好的压缩来处理，我们先看下源码中的部分代码截图：

图1:



```
250      if (bitsPerValue <= 8 && maxBitsPerValue >= 8) {  
251          actualBitsPerValue = 8;  
252      } else if (bitsPerValue <= 16 && maxBitsPerValue >= 16) {  
253          actualBitsPerValue = 16;  
254      } else if (bitsPerValue <= 32 && maxBitsPerValue >= 32) {  
255          actualBitsPerValue = 32;  
256      } else if (bitsPerValue <= 64 && maxBitsPerValue >= 64) {  
257          actualBitsPerValue = 64;  
258      } else if (valueCount <= Packed8ThreeBlocks.MAX_SIZE && bitsPerValue <= 24 && maxBitsPerValue >= 24) {  
259          actualBitsPerValue = 24;  
260      } else if (valueCount <= Packed16ThreeBlocks.MAX_SIZE && bitsPerValue <= 48 && maxBitsPerValue >= 48) {  
261          actualBitsPerValue = 48;  
262      } else {  
263          for (int bpv = bitsPerValue; bpv <= maxBitsPerValue; ++bpv) {  
264              if (Format.PACKED_SINGLE_BLOCK.isSupported(bpv)) {  
265                  float overhead = Format.PACKED_SINGLE_BLOCK.overheadPerValue(bpv);  
266                  float acceptableOverhead = acceptableOverheadPerValue + bitsPerValue - bpv;  
267                  if (overhead <= acceptableOverhead) {  
268                      actualBitsPerValue = bpv;  
269                      format = Format.PACKED_SINGLE_BLOCK;  
270                      break;  
271                  }  
272              }  
273          }  
274          if (actualBitsPerValue < 0) {  
275              actualBitsPerValue = bitsPerValue;  
276          }  
277      }  
278  }
```

先说明下上图的一些内容，actualBitsPerValues的值在后面的逻辑中用来选择对应的压缩实现，actualBitsPerValues与压缩实现的对应关系如下：

- 8: Direct8

- 16: Direct16
- 32: Direct32
- 64: Direct64
- 24: Packed8ThreeBlocks
- 48: Packed16ThreeBlocks
- 随后先考虑是否能用Packed64SingleBlock*（红框表示），最后才考虑使用Packed64

在第250行的if语句判断中，如果bitsPerValues的值小于等于8，并且maxBitsPerValue大于等于8，那么就使用Direct8来处理，在文章[PackedInts \(一\)](#)中我们知道，Direct*的压缩实现是读写性能最好的，可以看出来acceptableOverheadRatio是空间换时间的设计思想，并且压缩实现的选择优先级如下所示：

1 | Direct* > Packed*ThreeBlocks > Packed64SingleBlock* > Packed64

acceptableOverheadRatio的取值范围是什么：

Lucene提供了以下几种取值：

表2：

acceptableOverheadRatio	源码中的描述
7	At most 700% memory overhead, always select a direct implementation.
0.5	At most 50% memory overhead, always select a reasonably fast implementation
0.25	At most 25% memory overhead
0	No memory overhead at all, but the returned implementation may be slow.

acceptableOverheadRatio的值为7

如果acceptableOverheadRatio的值为7，那么不管bitsPerValue是区间[1, 64]中的哪一个值，总是会选择Direct*压缩实现。例如bitsPerValue的值为1，那么maxBitsPerValue = 1 + 1*7 = 8，那么根据图1中第250行的判断，就会使用Direct8来处理，意味着每一个数值使用8个bit位存储，由于每一个数值的有效数据的bit位个数为1，那么每个数值的额外开销为7个bit，即表2中描述的At most 700% memory overhead。

acceptableOverheadRatio的值为0.5

如果acceptableOverheadRatio的值为0.5，那么总能保证选择除了Packed64的其他任意一个压缩实现，它们是比较快（reasonably fast）的实现。

acceptableOverheadRatio的值为0.25

相对acceptableOverheadRatio的值为0的情况获得更多的非Packed64的压缩实现。

acceptableOverheadRatio的值为0

没有任何额外的空间开销，虽然读写性能慢，但是因为使用了固定位数按位存储，并且没有填充bit（见[PackedInts \(一\)](#)），所以有较高的压缩率。

Packed64的实现

表4中的所有压缩实现，除了Packed64，其他的实现逻辑由于过于简单就不通过文章介绍了，而Packed64的实现核心是BulkOperation，BulkOperation中根据bitsPerValue从1到64的不同取值一共有64种不同的逻辑，但他们的实现原理是类似的，故感兴趣的同学可以看文章[BulkOperationPacked](#)来了解其中的一个实现。

结语

无

[点击](#)下载附件