去重编码(dedupAndEncode)

去重编码是Lucene中对int类型数据的一种压缩存储方式，在FacetsConfig类中用到此方法来处理int类型数据。其优点在于，存储一个原本需要固定4个字节空间大小的int类型的数据，最好的情况下只要1个字节，最差的情况下需要5个字节。

处理过程

去重编码的过程主要分三步：

1. 排序
2. 去重
3. 差值存储

关系图

根据int数值的大小，在去重编码后存储该数值所需要的字节大小关系图如下

数值范围（指数）	数值范围（10进制）	字节大小
0 ~ (2^7 - 1)	0 ~ 127	1
2^7 ~ (2^14 - 1)	128 ~ 16383	2
2^14 ~ (2^21 - 1)	16384 ~ 2097151	3
2^21 ~ (2^28 - 1)	2097152 ~ 268435455	4
2^28 ~ *	268435456 ~ *	5

去重编码中最重要的一点是差值存储，从上图可以看出，我们在存储一组有序的数值时，除第一个数值外，其他的数值如果只存储跟它前面数值的差值，那么可以使得达到最大的压缩比。这种方式在存储大数值时的有点更明显。

> 例如我们有一组数据：`{17832，17842，17844}`，如果我们直接对3个数值进行存储(不存储差值)，那么最终需要9个字节才能存储这三个数值，而如果我们进行差值存储，那么我们需要存储的数据就变为：`{17832，10，2}`，其中10是17842跟17832的差值，2是17844跟17842的差值，那么最终只需要5个字节存储即可。

去重编码源码

相比较源码，删除了代码中的IntsRef类，便于理解

encode

```
public class ConvertIntToByteRef {
public static BytesRef dedupAndEncode(int[] ordinals) {
    // 对 ordinal[]数组排序，目的是为了去重跟计算差值
    Arrays.sort(ordinals, 0, ordinals.length);
    // 先给每一个int类型分配5个字节大小的空间，每个字节中只有7位是有效字节(描述数值),最高位是个定界符，所以一个int类型最多要5个字节
    byte[] bytes = new byte[5*ordinals.length];
    // 记录上次处理的值，用于去重判断
    int lastOrd = -1;
    int upto = 0;
    // 遍历处理每一个int数值
    for(int i=0;i<ordinals.length;i++) {
      int ord = ordinals[i];
      // ord could be == lastOrd, so we must dedup:
      // 去重操作，当前处理的数值跟上一个如果一样的话，skip
      if (ord > lastOrd) {
        // 存储差值的变量
        int delta;
        if (lastOrd == -1) {
          // 处理第一个值，只能储存原始的数值
          delta = ord;
        } else {
          // 处理非第一个值，就可以储存这个值与前一个值的差值
          delta = ord - lastOrd;
        }
        // if语句为真说明delta是 0~(2^7 - 1)内的值，需要1个字节存储
        if ((delta & ~0x7F) == 0) {
          // 注意的是第8位是0(位数从0计数)，一个byte的最高位如果是0，表示是数值的最后一个byte
          bytes[upto] = (byte) delta;
          upto++;
          // if语句为真说明delta是 2^7 ~ (2^14 - 1)内的值，需要2个字节存储
        } else if ((delta & ~0x3FFF) == 0) {
          // 这个字节的最高位是1，表示下一个byte字节和当前字节属于同一个int类型的一部分
          bytes[upto] = (byte) (0x80 | ((delta & 0x3F80) >> 7));
          // 这个字节的最高位是0，表示表示是数值的最后一个byte
          bytes[upto + 1] = (byte) (delta & 0x7F);
          upto += 2;
          // if语句为真说明delta是 2^14 ~ (2^21 - 1)内的值，需要3个字节存储
        } else if ((delta & ~0x1FFFFF) == 0) {
          // 这个字节的最高位是1，表示下一个byte字节和当前字节属于同一个int类型的一部分
          bytes[upto] = (byte) (0x80 | ((delta & 0x1FC000) >> 14));
          // 这个字节的最高位是1，表示下一个byte字节和当前字节属于同一个int类型的一部分
          bytes[upto + 1] = (byte) (0x80 | ((delta & 0x3F80) >> 7));
          // 这个字节的最高位是0，表示表示是数值的最后一个byte
          bytes[upto + 2] = (byte) (delta & 0x7F);
          upto += 3;
          // if语句为真说明delta是 2^21 ~ (2^28 - 1)内的值，需要4个字节存储
        } else if ((delta & ~0xFFFFFFF) == 0) {
```

```
          bytes[upto] = (byte) (0x80 | ((delta & 0xFE00000) >> 21));
          bytes[upto + 1] = (byte) (0x80 | ((delta & 0x1FC000) >> 14));
          bytes[upto + 2] = (byte) (0x80 | ((delta & 0x3F80) >> 7));
          bytes[upto + 3] = (byte) (delta & 0x7F);
          upto += 4;
          // delta是 2^28 ~ *内的值, 需要5个字节存储
        } else {
          bytes[upto] = (byte) (0x80 | ((delta & 0xF0000000) >> 28));
          bytes[upto + 1] = (byte) (0x80 | ((delta & 0xFE00000) >> 21));
          bytes[upto + 2] = (byte) (0x80 | ((delta & 0x1FC000) >> 14));
          bytes[upto + 3] = (byte) (0x80 | ((delta & 0x3F80) >> 7));
          bytes[upto + 4] = (byte) (delta & 0x7F);
          upto += 5;
        }
        // 这里将ord保存下来是为了去重
        lastOrd = ord;
      }
    }
    return new BytesRef(bytes, 0, upto);
  }

  public static void main(String[] args) {
    int[] array = {3, 2, 2, 8, 12};
    BytesRef ref = ConvertIntToByteRef.dedupAndEncode(array);
    System.out.println(ref.toString());
  }
}
```

最终结果用BytesRef对象表示，上面的输出结果：[2 1 5 4]

decode

```
public class ConvertByteRefToInt {
  public static void decode(BytesRef bytesRef){
    byte[] bytes = bytesRef.bytes;
    int end = bytesRef.offset + bytesRef.length;
    int ord = 0;
    int offset = bytesRef.offset;
    int prev = 0;
    while (offset < end) {
      byte b = bytes[offset++];
      // if语句为真：byte字节的最高位是0，decode结束
      if (b >= 0) {
        // ord的值为差值，所以(真实值 = 差值(ord) + 前面一个值(prev))
        prev = ord = ((ord << 7) | b) + prev;
        // 输出结果
        System.out.println(ord);
        ord = 0;
        // decode没有结束，需要继续拼接
```

```
      } else {
        // 每次处理一个byte
        ord = (ord << 7) | (b & 0x7F);
      }
    }
  }
  public static void main(String[] args) {
    int[] array = {3, 2, 2, 8, 12};
    // 去重编码
    BytesRef ref = ConvertIntToByteRef.dedupAndEncode(array);
    // 解码
    ConvertByteRefToInt.decode(ref);
  }
}
```

结语

去重编码(dedupAndEncode)是Lucene中的压缩存储的方式之一，还有VInt，VLong等数据类型都是属于压缩存储，在后面的博客中会一一介绍。demo看这里：https://github.com/luxugang/Lucene-7.5.0/tree/master/LuceneDemo/src/main/java/lucene/compress/dedupAndEncodeTest

点击下载Markdown文件