DocValues是在Lucene4.0引入的新特性，属于正向索引。它存储文档编号到字段值正向关系的索引，意在取代FieldCache在搜索时所发挥的作用，消除搜索时需要加载倒排索引构建FieldCache而引起的性能问题。相当于将FieldCache的构建下推至索引时，以空间换时间，从而获得更高的搜索性能。倒排索引是搜索的核心，而正向索引则为搜索结果的排序和统计等搜索结果加工过程提供了有力帮助。

> 倒排索引，也称反向索引，它是通过Term（字段值）召回相关的文档编号。
> DocValues则通过文档编码号召回字段值。

简单理解DocValues的话，它是一个以DocID为键，以Value为值的Map。它存储DocID到文档的正向关系，在排序或者统计计算时，通过DocID可以迅速取字段的值进行二次计算。

DocValues存储结构

开始之前，有必要先来看一下DocValues存储上的一些细节，如针对不同的数据特点采用不同的压缩方案，根据数据集分布情况选择合适的存储格式。整个DocValues索引文件中虽说只是存储了DocID与Values之间的映射关系，但需要支持的数据类型繁多。当然必不可少是DocID和Values，此外为了能维护二者之间的关系还需要Address信息。针对多值的情况，则有TermsDict以及TermsIndex两种数据，Values还包含Numeric和Binary两种类型。

Numeric存储格式

构建DocValues过程中有多处数据集的数据是数值类型的，Lucene也针对各种数值集的数据特征有多种压缩方式。除了DocIDSet之外，还有如下几种方式，但是它们的原理都是一样的，其它都是变种。

DocValues文件构建过程有多种类型的数据需要存储，其中很大一部分是数值类型的数据，它们用到的压缩类型主要为两种：DirectWriter，DirectMonotonicWriter。DocIDs虽然也是数值，但因它特殊而重要的存在地位，Lucene对它做了单独的优化。

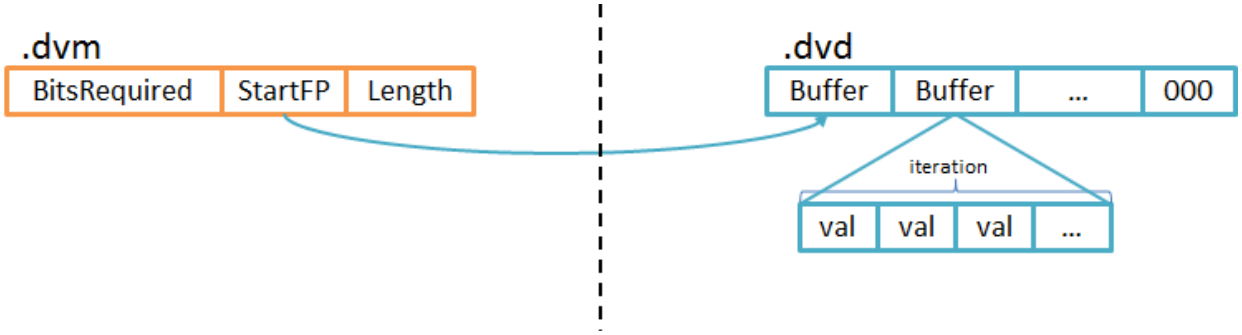
DirectWriter

DirectWriter是Lucene为整型数组重编码成字节数组的工具，它的底层包含一系列编码器，将整型数组的所有元素按固定位长度的位存储。它按Bit存储，预留长度过长会浪费空间，短了会因为截断导致错误。因此需要在数组中查找最大值，由它的长度作为存储的长度。

假设有一组数据{3, 16, 7, 12}，它们会用二进制表示是{101, 10000, 111, 1100}。占用有效位最长的是10000（5个bit），因此需要用5个bits来表示一个数值，得到如下结果。

需要注意的是，DirectWriter存储的最小单位是bit，为了充分使用Byte中每个bit会出现如下图情况，相当于把byte[]的位展开了成bit[]。

DirectWriter的Buffer是限制内存使用，避免OOM的手段，Lucene默认Buffer大小是1024Bytes。它包含压缩的long[]和压缩后的byte[]，它们两者占用内存不大于1024字节，一旦达到限制条件会将Buffer的数据编码输出。

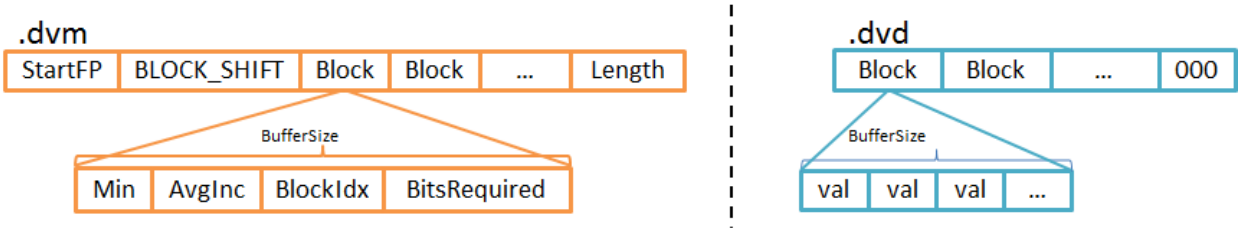


DirectWriter用重编码方式进行数组压缩的功能，它在整个数组的所有元素都不大的情况下能带来不错的压缩效果。

DirectMonotonicWriter

DirectMonotonicWriter是DirectWriter的扩展结构，它在DirectWriter之上加入分组的功能。数据分片是为了让每个分片内的数据分布平稳，即标准差比较小、数据波动幅度更平缓。

它不是通用方案，**它仅适用于单调递增的数组**。它通过计算两者之间的增量，让所有元素迅速缩小。所以这是非常适合存储文件地址之类比较连续的数据。比如{100, 102, 103, 105}，最终会变成{100, 2, 1, 2}。如果将第一个元素存到.dvm文件，则变成{0, 2, 1, 2}，**仅需要一个字节即可**。



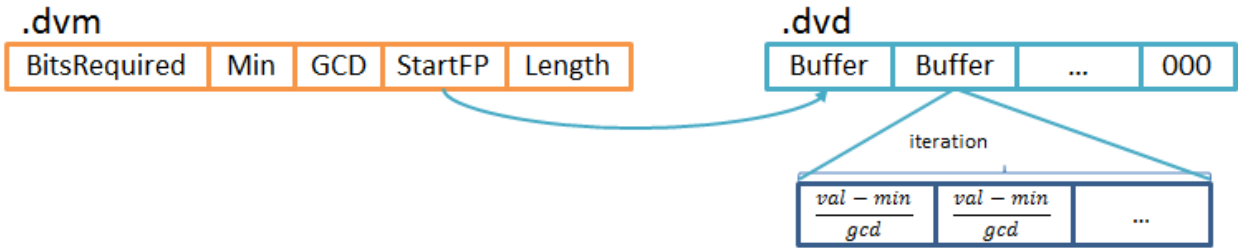
StartFP是数据写入在.dvd文件的起始位置，BLOCK_SHIFT决定每个Block的大小，BlockIdx指向具体的Block位置。 每个Block都是一个独立的DirectWriter，它们有各自的元数据信息。每个Block内部是一个DirectWriter结构，这里没有展开来。

DirectMonotonicWriter的每个Block实现上是由DirectWriter编码，它还为每个Block创建索引并且保存在.dvm文件中。写入过程会记录整个Block的平均值和最小值。

使用DirectMonotonicWriter的前提是数据必须从小到大排序的，在增长平缓情况下能够达到非常不错的压缩效果。

GCD-Compression

GCD-Compression是DirectWriter扩展，底层结构与DirectWriter完全一样，只是写入的值是加工过的。GCD与DirectMonotonicWriter不一样，实质上它算不上是扩展，只是将数据写入之前做一次预计算，核心工作还是由DirectWriter完成的。下面还会提及Table-Compression，它跟GCD的原理完全一样，就是计算方式不同。



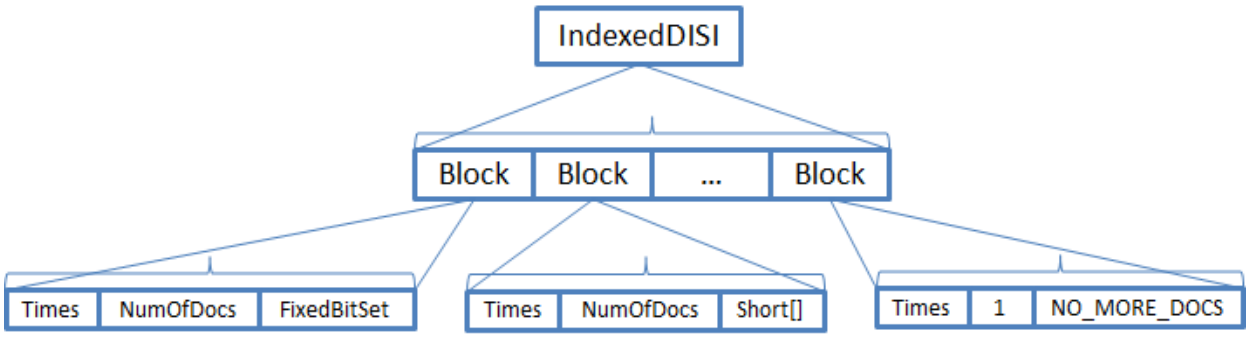
Lucene为了保证此计算可逆在.dvm记下方程的两个参数（gcd和min）的值。需要先求出整个数组的最大公约数，通过公式将所有元素缩小。比如，$\{9, 6, 12, 33\}$，它的最大公约数是3，最小值是6，将数组缩小之后得$\{1, 0, 2, 9\}$。原数组用DirectWriter存储需要3个字节，缩小后仅需要2个字节，**显然这种方式可以有效缩小每个元素的大小从而获得更高压缩比。**尤其在数据集比较大，分布离散的数据集，NumericField的值恰好满足这些特点。

2. IndexedDISI存储格式

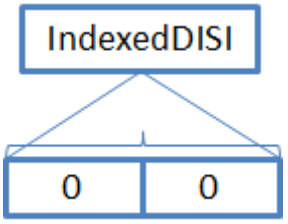
存储格式是根据数据集的分布而设定的存储方案，对于DocIDSet这种特殊的结构，Lucene设计了IndexedDISI结构，它充分利用了数据集的稀稠性特点。

IndexedDISI按65535的倍数为界将DocIDSet分组，故第n组的所有DocIDs必须在65535*(n-1)~65535*n范围内。当有些文档的字段无值时，便会出现某些组DocID的数量不满65535，**当它小于4096时，Lucene将它视为稀疏结构用short[]存储；反之则是稠密结构用BitSet存储。**当然所有的DocID都存在，则称为全量，那也没必要存储DocIDSet了，仅写一个Flag来表示即可。

BitSet的存储空间复杂度由它的最大值唯一决定，那么数据集比较小而最大值比较大时，这种方案存储代价会比较高的。而对short[]它的存储复杂度是随数量的增长呈正相关，而4096这个数值是BitSet与short[]存储复杂度的分水岭。小于则是稀疏的short[]结构，否则则是稠密的BitSet结构。



注：这里画的示意图并不准确，因为每个Block可能是稀疏的，也可能是稠密的。这里仅是为了表示稀疏和稠密的Block的结构，并不代表真正的存储结构。最后一个Block用于代表没有更多的文档，这里的Times表示第N个Block。



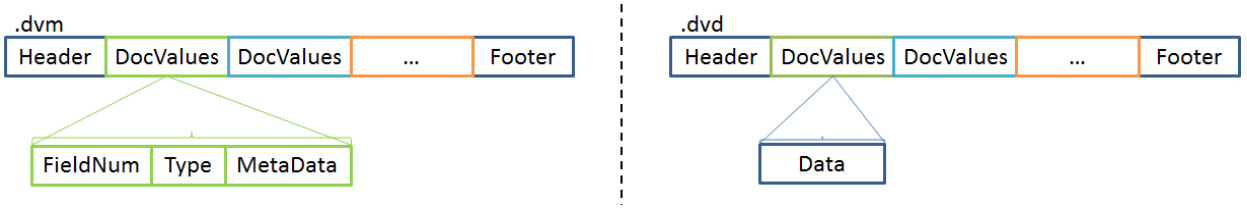
需要注意的是DocIDSet的所有DocID都存在时，DocIDSet可以省略，通过在Meta文件写入一个Flag值表示全量。因此这种情况不需要在data文件上写入任何内容。最终在.dvm文件会是如上图所示情景，此时.dvd不需要再记录DocIDSet的相关信息。

DocValues类型

Lucene当前版本（Lucene 7.5）DocValues共支持五种字段的值类型，且针对每种字段值的类型有不同的编码策略，以适应它们的数据特征。DocValues如今还不支持分词字段类型，将来可能会支持。

不管是哪种字段值类型，Lucene都是用.dvd文件存储DocValues的数据；用.dvm文件存储DocValues的元数据，用于解析数据文件。每种字段类型都涉及到这样的一对文件，下面我们挖掘一下每种类型的存储结构。

与Lucene其它的索引文件不一样的是，Lucene的文档基本没有介绍DocValues索引文件的存储结构，所以我们需要通过源代码来勾绘它的结构示意图。如Document有多个DocValues字段的话，每个字段的数据文件将是存储在同一个索引文件.dvd上，同样元数据文件也是。



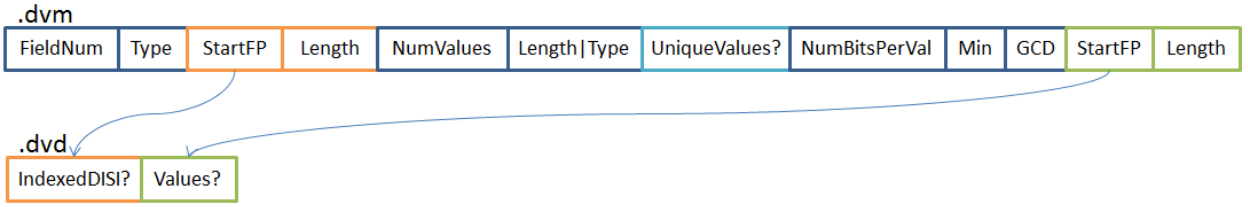
所有的DocValues类型中，.dvm文件的结构远比.dvd文件复杂。.dvm记录整个DocValues字段的各种元数据，通过.dvm文件才能将.dvd的数据还原。Lucene将DocValues的DocIDSet和Values分开存储在.dvd文件上，而且两者之间并没有强关联，全凭.dvm来维护它们之间的关系。 虽然在字段层面上.dvd文件的大体结构.dvm相差不多，而且走进字段内部结构会有天壤之别。

> Solr已经弱化了DocValues值的类型，对用户完全屏蔽的DocValues的具体类型。实际上它在Lucene是强类型，每种类型的存储结构也不尽相同。

1. Numeric

Numeric是针对数值的DocValues类型，它仅能处理单值的字段。NumericField/SortedNumericField都没有直接支持浮点型，但我们可以通过重编码的方式将Float转成Integer，将Double转成Long的方式曲线达成支持浮点型的目标。

Numeric类型的结构比较简单，画出来的结构示意如下：



1. Type是DocValues，这里值为Lucene70DocValuesFormat.NUMERIC。
2. 第一个StartFP存储IndexedDISI在.dvd文件起始位置的地址。当DocIDSet为空或者全量时，Lucene不需要记录IndexedDISI，仅在.dvm写入StartFP特殊标

记的值，随后的Length为-1（表示.dvd文件中没有IndexedDISI，它原意是指IndexedDISI在占用.dvd多大空间）。

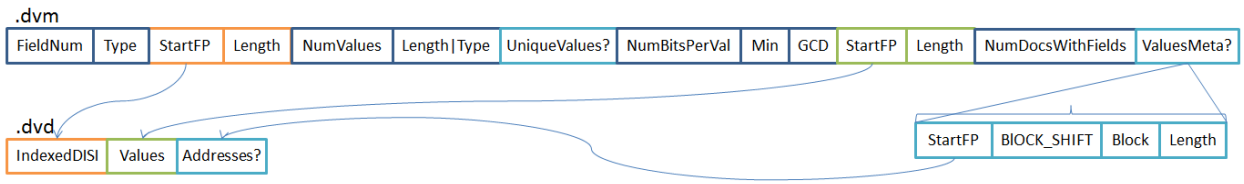
当字段唯一值个数不超256个时，会触发Table-Compressed压缩。一旦启用Table-Compressed压缩，Lucene将会把所有值去重并排序之后写入.dvm文件。而后，.dvd文件的Values部分内容改记为每个值在排序之后的新次序。

优化后Values的每个元素都不会大于256，直接采用DirectWriter编码写在.dvm文件中。那么它下标与Value即是Table的数据结构了，在写Values的时候，将Value通过这个Table获取下标写入.dvd文件完成DocIDSet与Values的映射。更多细节内容可参考Lucene官方文档介绍的Table-Compressed压缩方式。

如果Values没条件启用Table-Compressed压缩，它将会是以GCD-Compressed方式压缩，所以它会在.dvm文件记下DirectWriter编码成多少个Bit，最小值以及GCD值。

2. SortedNumeric

SortedNumeric是Numeric类型的升级版，它支持**多值**。如果所有文档都不超过1个值时，它们的存储结构基本类似（就是在Numeric结构的后面加一个NumDocsWithField来说明字段有多个文档）。只不过此时会在.dvm文件后加NumDocsWithField来标明是否为多值字段。



SortedNumeric的存储结构仅比Numeric多了NumDocsWithFields和Addresses。由于IndexedDISI与Values分开存储的，从示意图上可以知道它们之前没有直接关系。对于单值的情况，DocValues将DocID和Value写入顺序相同，即是IndexedDISI的第n个DocID对应第n个Value。

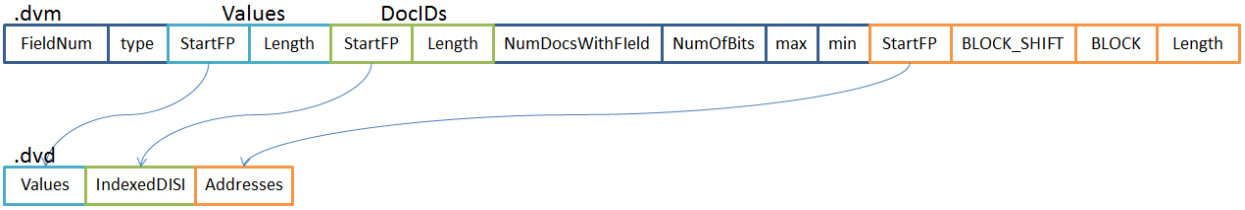
但是在多值场景下，这种方式就失去它的功能了。因为Document的值的个数无法确定，因此需要额外记录每个文档有几个值。这就是图中Addresses部分的内容，它采用DirectMonotonicWriter编码，它的结构跟DirectMonotoincWriter的完全一样。

Addresses有什么用呢？ Address是衔接DocId与Values映射的桥梁，通过Address能让DocID快速找到DocID对应的Values，它可能有多个Values，可能是Numeric类型，也可

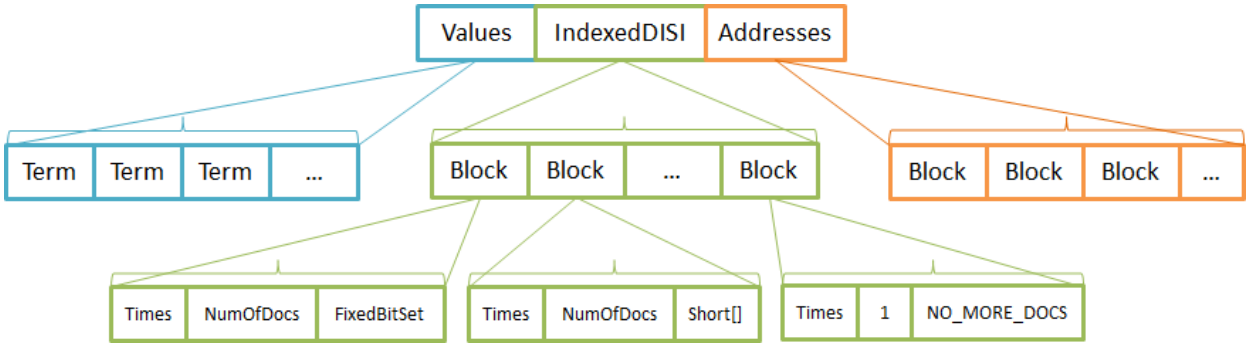
能是Binary。对于Numeric而言，由于它的长度是已知的（NumBitsRequired)，所以它记录的Values的个数。而对于Binary而言，它的长度是未知的，所以它需要记录每个值的长度。

3. Binary

Binary类型支持byte[]的DocValues，它的长度不能超过32766 Bytes且必须是单值。实际上StringField字段类型有值长度的要求，Binary作为StringField对应的DocValues类型，跟StringField有相同的要求。



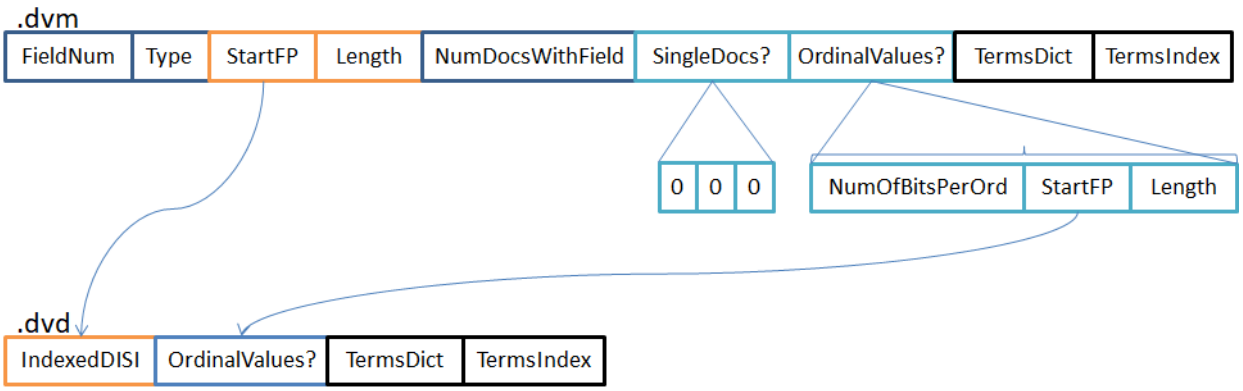
Binary类型的结构与SortedNumeric类似，比较简单。将.dvd文件存储结构展示如下图:



Binary的记录的Addresses与SortedNumeric略有不同，SortedNumeric记的是每个文档有几个值，而Binary则是记每个Term的长度。

4. Sorted

Sorted是实现了排序的Binary类型，它也是单值，此外它先预将byte[]排序之后再写到文件。它在Values部分记录的并不是真正的Value，而是记录Value的次序（ordinal，去重排序后的下标），这是与Binary的不同之处。OrdinalValues是Value的次序，基于DirectWriter编码存储。



如果是SingleDocs（文档数<=1）的情况下，元数据中OrdinalValues的NumerOfBitsPerOrd,StartFP和Length三个值均为零。此时也表示.dvd也没记录OrdinalValues的信息。

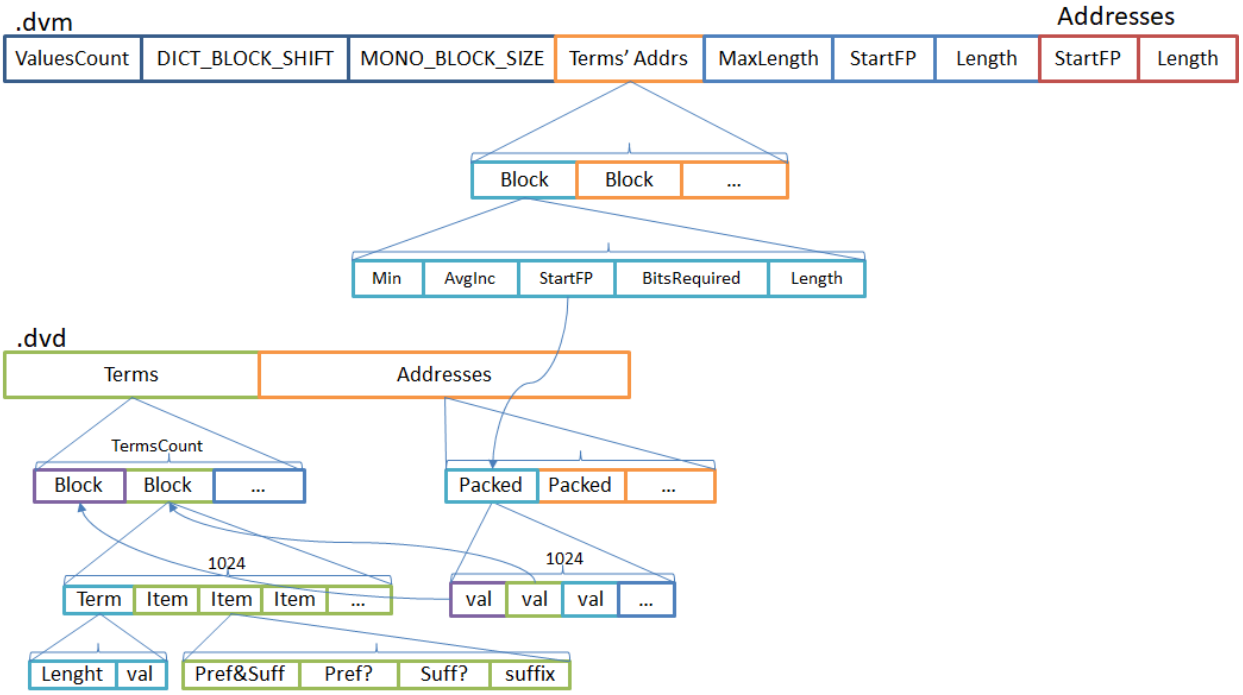
Sorted出现两个新的结构TermsDict和TermsIndex，故名思义TermsDict是Terms字典，作为字典是不会有重复的词元的；TermsIndex是TermsDict索引。它们的意义在于TermsDict是去重之后存储的，所以它在一定程序上能够节减空间开销；另外排序之后Values与IndexedDISI存储在.dvd文件中下标不一致，需要额外映射表来串联它们的关系。

TermsIndex并不是TermsDict的元数据，它会同时出现在.dvm和.dvd两个文件中。

TermsDict算是实现了映射表的作用，TermsDict每个Term的下标等同于Ordinal，因此通过Ordinal便能获取TermsDict的位置了。

TermsDict

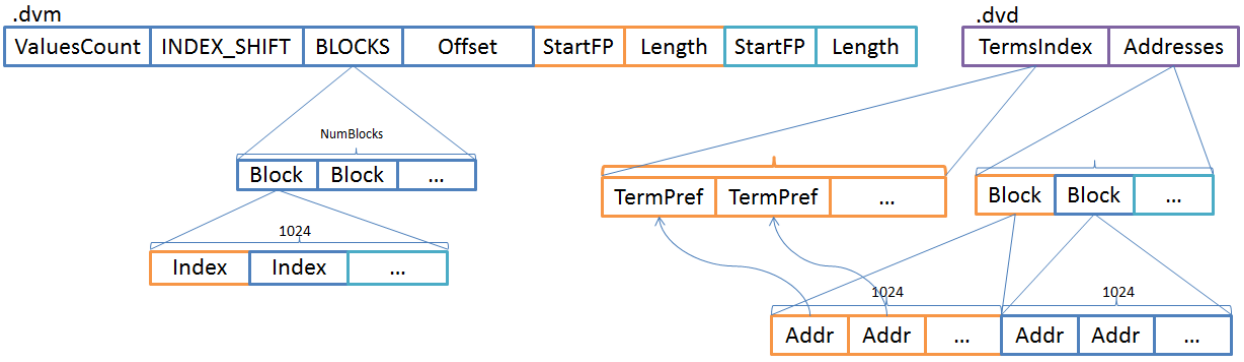
TermsDict采用了分组存储的方式，每1024个文档为一个组。分组存储的好处是方便构建索引，其次能够实现起到压缩前缀的作用。如下图，每个Block只有第一元素是直接存储的，之后每个元素都跟前一元素共享共同前缀（如果有的话）。通常来说，Term的长度不会太长，所以Lucene在这里又做了一个小优化，一个字节的前4个位来存储共同前缀的长度，后4个位存储后缀的长度。如果4位表示不了，则会以VInt的格式写在后面。



每1024个Term构成一个Block，每个Block会在Addresses中记文件地址索引，Addresses采用DirectMonotonicWriter编码。而DirectMonotonicWriter也会将1024个Address打成Packed，每个Packed也会它记的文件地址索引，不过是记在.dvm文件上。需要注意的是这里的Terms是TermsDict，Term不重复。dvm文件的Addresses索引是为了让Lucene能够成功解析.dvd文件的Terms的，并不是让我们用它来检索的。

TermsIndex

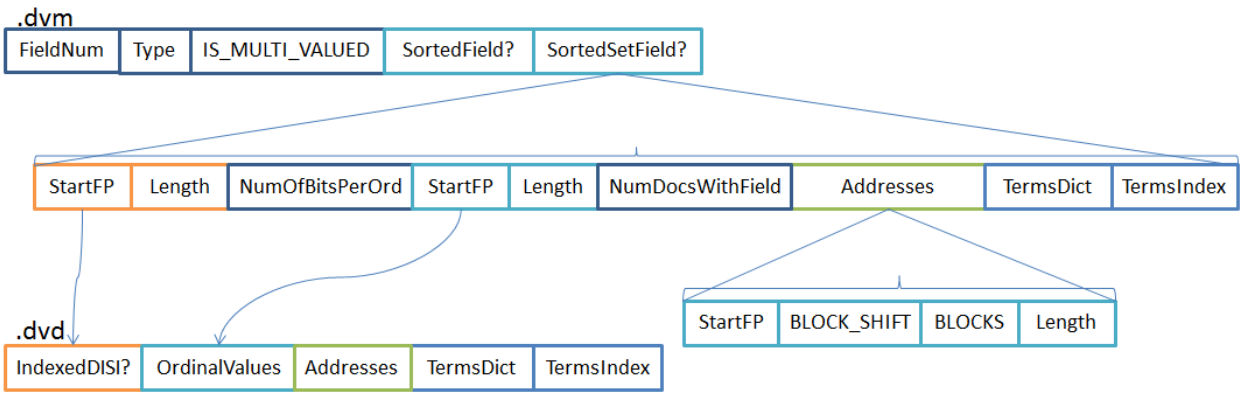
TermsIndex结构与TermsDict基本一样，它将Terms中每个Block的第一个Term写到.dvd文件中，将它的位置写在Addresses中，所以还会按Addresses的Block生成一个索引记在.dvm文件。



5. SortedSet

SortedSet是支持多值且有序的Binary类型，它是Sorted类型的加强版。单值是采用Sorted结构，多值的结构如下图所示。结构上跟Sorted非常相似，只是多加一个Addresses结构记

录每个Doc有多少个值，跟SortedNumeric的做法一样。



结语

在Lucene的官方文档中，关于DocValues的介绍可谓是少之又少，所以只能通过剖析代码来理解它的实现细节。本文先梳理了DocValues所涉及到的几种编码方式，而后介绍了各种数据类型的结构，从Numeric到Binary，从单值到多值。借助此文，希望能让你领略DocValues设计全貌，洞悉每种类型的设计异同。