ICS 35.040

L 71

T/AI XXX—XXXX

|  |
| --- |
|  |

信息技术 基因数据压缩

第1部分：测序数据

Information technology - Genomic Data Compression -

Part 1: Sequencing Data

（征求意见稿）

XXXX - XX - XX发布

XXXX - XX - XX实施

中关村视听产业技术创新联盟 发布

团体标准

1. 前言

本文件按照GB/T 1.1—2020《标准化工作导则 第1部分：标准化文件的结构和起草规则》的规定起草。

本文件是T/AI XXX《信息技术 基因数据压缩》的第1部分。

请注意本文本的某些内容可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

本文件由数字音视频编解码技术标准工作组提出。

本文件由中关村视听产业技术创新联盟归口。

本文件起草单位：鹏城实验室、深圳大学、深圳华大生命科学研究院、哈尔滨工业大学、西安电子科技大学、深圳华大智造科技股份有限公司、华为技术有限公司、香港城市大学。

本文件主要起草人：朱泽轩、张勇、徐讯、刘贤明、牛毅、陈毓新、李胜康、王荣杰、王诗淇、谢少辉、马明明、周倩、江俊君、孙杰、黎宇翔、古圣昌、郭良顺、谢寅龙、曾文君、魏晓峰、单日强、刘元盛、曾湘祥、黄铁军、高文。

1. 引言

T/AI XXX《信息技术 基因数据压缩》是为了适应基因测序、基因序列比对、分析等应用中对基因数据压缩技术的需求而制定的。T/AI XXX采用一系列技术来达到基因数据的高效率压缩编码包括有参考碱基序列压缩、基于自组装参考序列的碱基压缩、基于上下文的质量分数压缩等，拟由以下五个部分构成：

——第1部分：测序数据。目的在于确立适应于基因测序数据中碱基序列、元数据、质量分数等的压缩编解码。

——第2部分：比对数据。目的在于确立基因数据比对结果中参考基因组序列、高通量碱基序列、比对位置、错配信息等的压缩编解码。

——第3部分：分析结果数据。目的在于确立基因测序数据变异检测结果中参考序列、变异位置、变异结果等的压缩编解码。

——第4部分：参考软件。包含了T/AI XXX第1、2、3等部分所定义的工具集的模拟软件。该软件由在T/AI XXX制定过程中的一系列验证模型演变而来。

——第5部分：符合性测试。定义了如何测试验证编码位流和解码器是否满足T/AI XXX所规定的要求。

信息技术 基因数据压缩

第1部分：测序数据

* 1. 范围

本文件给出了高通量测序产生的FASTQ和FASTA数据的压缩表示，包括基因组序列、测序碱基序列、元数据、质量分数等的编解码。

本文件适用于基因测序数据产生、识别、分析等过程中涉及数据存储和传输的相关应用。

* 1. 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 29859 生物信息学术语

GB/T 35890 高通量测序数据格式规范

* 1. 术语和定义

下列术语和定义适用于本文件。

参考基因组 reference genome

通过对一个或多个个体的DNA进行测序而组装得到的一组核酸序列，是一个物种的理想个体生物中一组基因的代表实例。

测序 sequencing

测定氨基酸或者核苷酸序列的过程。

[来源：GB/T 29859-2013,2.4]

测序序列 reads

高通量测序平台产生的序列片段，包含元数据、碱基序列和质量分数信息。

[来源：GB/T 35890-2018,3.2，有修改]

错配 mismatch

序列比对过程中碱基序列和参考基因组或参考序列不匹配的碱基。

EBML元素 EBML elements

EBML的基本单元，包含元素ID、元素数据大小和元素数据三部分。

反向互补 reverse complement

基因序列反向互补是指呈现相反方向的两个互补碱基序列。 具体来说，就是 A-T 和 C-G 的配对关系在两个互补序列上呈现相反的方向。例如ACCTGAG的反向互补序列是CTCAGGT。

FASTA格式 FASTA format

基于文本的、保存生物序列(通常是DNA序列，也可以是蛋白质序列)的、每N行表示一条序列的标准格式。

FASTQ格式 FASTQ format

基于文本的、保存生物测序序列的、每四行表示一条序列的标准格式。

[来源：GB/T 35890-2018, 3.9，有修改]

简并碱基 ambiguous bases

根据密码子的兼并性，用一个符号代替某两种或者更多种碱基，具体对应规则应符合附录A的规定。

碱基序列 base sequence

测序片段中记录碱基排列的字符串，碱基序列中每个碱基应使用大写字母（A、T、C、G和N）或小写字母（a、t、c、g和n），其中字母A和a表示腺嘌呤，字母T和t表示胸腺嘧啶，字母C和c表示胞嘧啶，字母G和g表示鸟嘌呤，字母N和n表示未测定的碱基。

[来源：GB/T 35890-2018, 3.6]

长读长测序 long-read sequencing

单分子测序技术,测序时不需要经过聚合酶链式反应扩增，实现了对每一条DNA分子的单独测序，测序长读可达几十到几万碱基，错误率较高。

序列比对 sequence alignment

将碱基序列和参考基因组或者参考序列的某一区域进行字符串匹配，从而得到两者之间的编辑距离，包括碱基插入、删除、替换等信息。

元数据 metadata

FASTA或FASTQ格式中每一条序列的第一行，以>或@开头，在FASTA格式中一般后接染色体信息，在FASTQ格式中一般后接测序序列属性信息如ID、测序平台、读长等。

质量分数 quality scores

FASTQ格式中每一条测序序列的第四行，与第二行碱基序列的等长的ASCII字符串，每个字符代表对应碱基的测序质量。常见的质量分数体系有Phred+33和 Phred+64, Phred+33体系质量分数0对应ASCII码33用!表示，Phred+64体系质量分数0对应ASCII码64，用@表示。

[来源：GB/T 35890-2018, 3.8,有修改]

自组装参考序列 self-assembled reference sequence

不借助外部参考基因组，通过寻找原始碱基序列之间的重叠得到的参考序列。

* 1. 缩略语

下列缩略语适用于本文件。

ACO:自适应编码顺序（Adaptive coding order）

AMGC:基于匹配的自适应基因压缩(Adaptive match-based genomic compression)

ASCII：美国标注信息交换码（American standard code for information interchange）

BSC：块排序压缩器（Block sorting compressor）

CRC32: 循环冗余校验(A cyclic redundancy check 32)

DNA：脱氧核糖核酸 (Deoxyribonucleic acid)

EBML：可扩展二进制元语言(Extensible binary meta language)

GZ：GNU压缩（GNU zip）

ID：唯一标识号 (Identity)

LZMA：Lempel-Ziv-Markov链算法（Lempel-Ziv-Markov chain-algorithm）

MD5：消息摘要算法第五版(Message-digest algorithm 5)

METC:基于最高位平面和流道列上下文的元数据编码(Meta encoding based on max bit plane and tile context)

XXH3：非加密哈希算法（xxHASH-XXH3）

VINT：可变长整数（Variable-length integer）

* 1. 约定
     1. 概论

本部分中使用的数学运算符和优先级与C语言使用的类似，但对整型除法和算术移位操作进行了特定定义。除特别说明外，约定编号和计数从0开始。

* + 1. 算术运算符

算术运算符定义见表 1。

表 1算术运算符定义

| 算术运算符 | 定义 |
| --- | --- |
|  | 加法运算 |
| - | 减法运算（二元运算符）或取反（一元前缀运算符） |
| × | 乘法运算 |
| *ab* | 幂运算，表示*a*的*b*次幂。也可表示上标 |
| / | 整除运算，沿向0的取值方向截断。  例如，7/4和-7/-4截断至1，-7/4和7/-4截断至-1 |
|  | 除法运算，不做截断或四舍五入 |
|  | 除法运算，不做截断或四舍五入 |
|  | 自变量*i*取由*a*到*b*（含*b*）的所有整数值时，函数的累加和 |
| *a* % *b* | 模运算，*a*除以*b*的余数，其中*a*与*b*都是正整数 |

* + 1. 逻辑运算符

逻辑运算符定义见表 2。

表 2逻辑运算符定义

| 逻辑运算符 | 定义 |
| --- | --- |
| a && b | a和b之间的与逻辑运算 |
| a || b | a和b之间的或逻辑运算 |
| ! | 逻辑非运算 |

* + 1. 关系运算符

关系运算符定义见表 3。

表 3关系运算符定义

| 关系运算符 | 定义 |
| --- | --- |
| > | 大于 |
| >= | 大于或等于 |
| < | 小于 |
| <= | 小于或等于 |
| == | 等于 |
| != | 不等于 |

* + 1. 赋值

赋值运算定义见表 4。

表 4赋值运算符定义

| 赋值运算符 | 定义 |
| --- | --- |
| = | 赋值运算符 |
| ++ | 递增，*x*相当于*x*  *x*  1。当用于数组下标时，在自加运算前先求变量值 |
| -- | 递减，*x--*相当于*x*  *x* – 1。当用于数组下标时，在自减运算前先求变量值 |
| += | 自加指定值，例如*x* += 3相当于*x* = x + 3，*x* += (-3)相当于*x* = *x* + (-3) |
| -= | 自减指定值，例如*x* –= 3相当于*x* = *x* – 3,，*x* –= (-3)相当于*x* = *x* – (-3) |

* 1. 比特流语法规则
     1. 描述方法

比特流语法描述方法类似C语言。比特流的语法元素使用粗体字表示，每个语法元素通过名字（用下划线分割的英文字母组，所有字母都是小写）、语法和语义来描述。语法表和正文中语法元素的值用常规字体表示。

某些情况下，可在语法表中应用从语法元素导出的其他变量值，这样的变量在语法表或正文中用不带下划线的小写字母和大写字母混合命名。大写字母开头的变量用于解码当前以及相关的语法结构，也可用于解码后续的语法结构。小写字母开头的变量只在它们所在的小节内使用。

语法元素值的助记符和变量值的助记符与它们的值之间的关系在正文中说明。在某些情况下，二者等同使用。助记符由一个或多个使用下划线分隔的字母组表示，每个字母组以大写字母开始，也可包括多个大写字母。

比特串的长度是4的整数倍时，可使用十六进制符号表示。十六进制的前缀是‘0x’，例如‘0x1a’表示比特串‘0001 1010’。

条件语句中0表示FALSE，非0表示TRUE。

语法表描述了所有符合本部分的比特流语法的超集,附加的语法限制在相关小节中说明。

表 5给出了描述语法的伪代码例子。当语法元素出现时，表示从比特流中读一个数据单元。

表 5语法描述的伪代码

|  | 描述符 |
| --- | --- |
| /\*语句是一个语法元素的描述符，或者说明语法元素的存在、类型和数值，下面给出两个例子。\*/ |  |
| **syntax\_element** | u(2) |
| conditioning statement |  |
|  |  |
| /\*花括号括起来的语句组是复合语句，在功能上视作单个语句。\*/ |  |
| { |  |
| statement |  |
| statement |  |
| … |  |
| } |  |
|  |  |
| /\*“while”语句测试condition是否为TRUE，如果为TRUE，则重复执行循环体，直到condition不为TRUE。\*/ |  |
| while ( condition ) |  |
| statement |  |
|  |  |
| /\*“do … while”语句先执行循环体一次，然后测试condition是否为TRUE，如果为TRUE，则重复执行循环体，直到condition不为TRUE。\*/ |  |
| do |  |
| statement |  |
| while ( condition ) |  |
|  |  |
| /\*“if … else”语句首先测试condition，如果为TRUE，则执行primary语句，否则执行alternative语句。如果alternative语句不需要执行，结构的“else”部分和相关的alternative语句可忽略。\*/ |  |
| if ( condition ) |  |
| primary statement |  |
| else |  |
| alternative statement |  |
|  |  |

表5 语法描述的伪代码（续）

|  |  |
| --- | --- |
|  | 描述符 |
| /\*“for”语句首先执行initial语句，然后测试condition，如果conditon为TRUE，则重复执行primary语句和subsequent语句直到condition不为TRUE。\*/ |  |
| for ( initial statement; condition; subsequent statement ) |  |
| primary statement |  |

解析过程和解码过程用文字和类似C语言的伪代码描述。

* + 1. 描述符

描述符表示不同语法元素的解析过程见表 6。

表 6描述符说明

| 描述符 | 说明 |
| --- | --- |
| u(n) | *n*位无符号整数。如果n是'v'，其比特数由其他语法元素值确定。 |
| uc(n) | *n*位无符号char。如果n是'v'，其比特数由其他语法元素值确定。 |
| vi | 变长无符号整形。数据结构包含三部分即前缀、终止比特和实际数值。前缀为连续’0’串代表数据长度，记’0’比特位数为*k*则代表整个数据长度为*k*个字节。终止比特为第一个’1’比特。实际数值记录在终止比特后面7*k*-1位。 |
| eb | 由两个vi和一个uc组成的EBML元素。解析过程为，解析第一个vi得到数值记为*n*表示该eb元素ID，解析第二个vi得到数值*m*表示后续uc的长度，最后解析uc(*m*)得到该eb实际数值。 |
| ac(k,n) | 使用阶数为k、取值范围为*n*的自适应算术编码器编码码流，其中k表示在编码当前字符时，使用其前序k个字符作为上文对当前字符概率进行预测。解析过程为，初始化一个范围为*n*的解码器，以码流为输入，根据前k阶上下文逐个解码出原始元素，直到码流结束，或者元素数量达到目标，具体在各个应用该描述符的变量的解码过程中进行说明。如果k或n是'v'，其数值由其他语法元素值确定。 |
| rc(k,n) | 使用阶数为k、取值范围为*n*的自适应区间编码器编码码流，其中k表示在编码当前字符时，使用其前序k个字符作为上文对当前字符概率进行预测。解析过程为，初始化一个范围为*n*的解码器，以码流为输入，根据前k阶上下文逐个解码出原始元素，直到码流结束，或者元素数量达到目标，具体在各个应用该描述符的变量的解码过程中进行说明。如果k或n是'v'，其数值由其他语法元素值确定。 |
| src(n) | 自适应区间编码器编码码流，一个src(n)维护一个独有的阶数为0、范围为n的计数器，同个码流内多个字段共用同一区间编码器输出码流，各字段基于各自计数器将当前待编码字符输入结果码流，编码顺序与各字段在编码时出现顺序一致，对于同一层级或逻辑分支中的各个字段，其编码顺序与其在表格中出现的顺序一致。解析过程为，初始化一个区间解码器，以码流为输入，根据各字段的逻辑出现顺序，逐个解码出原始元素，直到码流结束，或者元素数量达到目标，具体在各个应用该描述符的变量的解码过程中进行说明。 |
| qs(n) | 用*n*阶上下文模型驱动的多值算术编码器对一个字符的编码码流，*n*阶上下文模型的构建见7.1.6.5　。 |
| lz | LZMA编码码流，按LZMA解码器解码 |
| bs | 采用块排序压缩器BSC编码的码流 |

* + 1. 函数
       1. 概述

以下函数用于语法描述。函数由函数名及左右圆括号内的参数构成。函数也可没有参数。

* + - 1. num\_chars( str )

返回字符串str中字符的个数。

* + - 1. num\_tokens( str )

以所有非字母（ASCII码从65到90以及97到122）或非数字（ASCII码从48到57）的字符为分隔符将字符串str进行分割，返回分割后子串数量。

1. str=“@ERR966765.1 HS12\_14113:3:2308:5268:18887#4/1”,分割后得到10个子串 ["ERR966765", "1", "HS12", "14113", "3", "2308", "5268", "18887", "4", "1"]，函数返回10。
   * + 1. token( str，i )

以所有非字母（ASCII码从65到90以及97到122）或非数字（ASCII码从48到57）的字符为分隔符将字符串str进行分割，返回分割后第i个子串。

1. str=“@ERR966765.1 HS12\_14113:3:2308:5268:18887#4/1”,分割后得到10个子串 ["ERR966765", "1", "HS12", "14113", "3", "2308", "5268", "18887", "4", "1"]， token(str，2)返回"HS12"。
   * + 1. is\_number( str )

判断字符串str是否由全数字字符构成，是则返回TRUE，否则返回FALSE。

* + - 1. next\_ebml\_id()

返回码流当前位置后续embl元素ID（ebml解析过程见表 6）。

* + - 1. ebml\_value( e )

返回embl元素e的实际数值（ebml解析过程见表 6）。

* + - 1. ebml\_length( e )

返回embl元素e的长度，即第二个vi的实际数值（ebml解析过程见表 6）。

* + - 1. cardinality (str)

返回二进制码流str中‘1’的个数。

* 1. 测序数据编码
     1. 编码框架
        1. 概述

测序数据编码主要由三部分构成，包括头部码流、压缩数据码流和尾部码流。其中头部码流包含魔数、基础信息、压缩信息等。压缩数据码流针对FASTA和FASTQ格式采用分块方式编码，包括了各个数据块的码流。尾部码流主要包含各个数据块码流的信息。总体的码流概况见图 1。

图示

描述已自动生成

图 1码流概览

* + - 1. 系统编码
         1. 头部码流

头部码流按照排列顺序分成两部分，第一部分是魔数，用于鉴别码流是否为本文件所规定的格式；第二部分是头数据。头数据中包含码流基础信息和压缩信息。

码流基础信息

码流基础信息的内容依次包含原始输入类型、标准版本号、编码器标识符、原始文件名、原始文本字节大小、原始GZ字节大小共六类信息，如图 2所示。

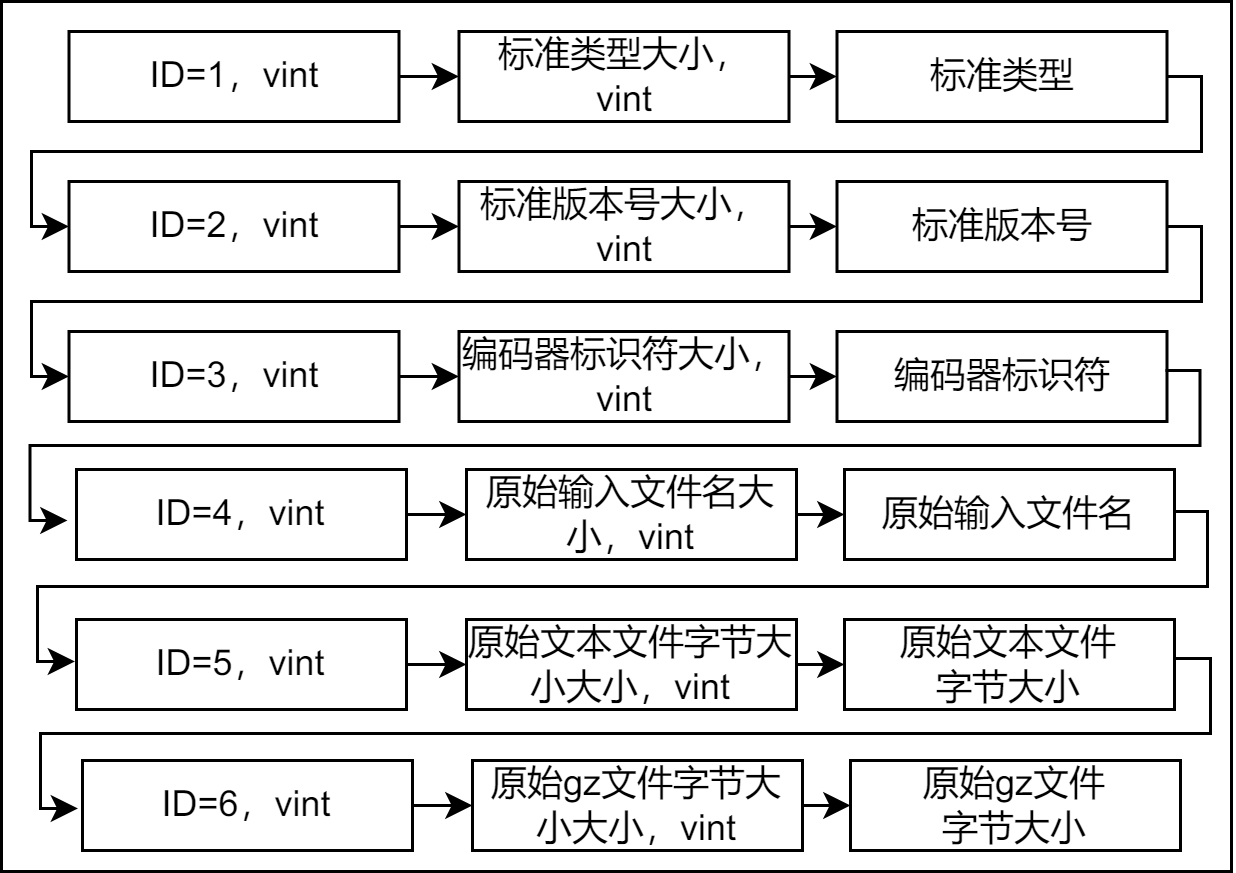


图 2 码流基础信息码流结构

压缩信息

压缩信息的内容依次包含原始输入类型、原始输入第三行是否单字符、是否长读长序列数据、校验算法、原始输入文本校验码、原始输入压缩校验码，如图 3所示。

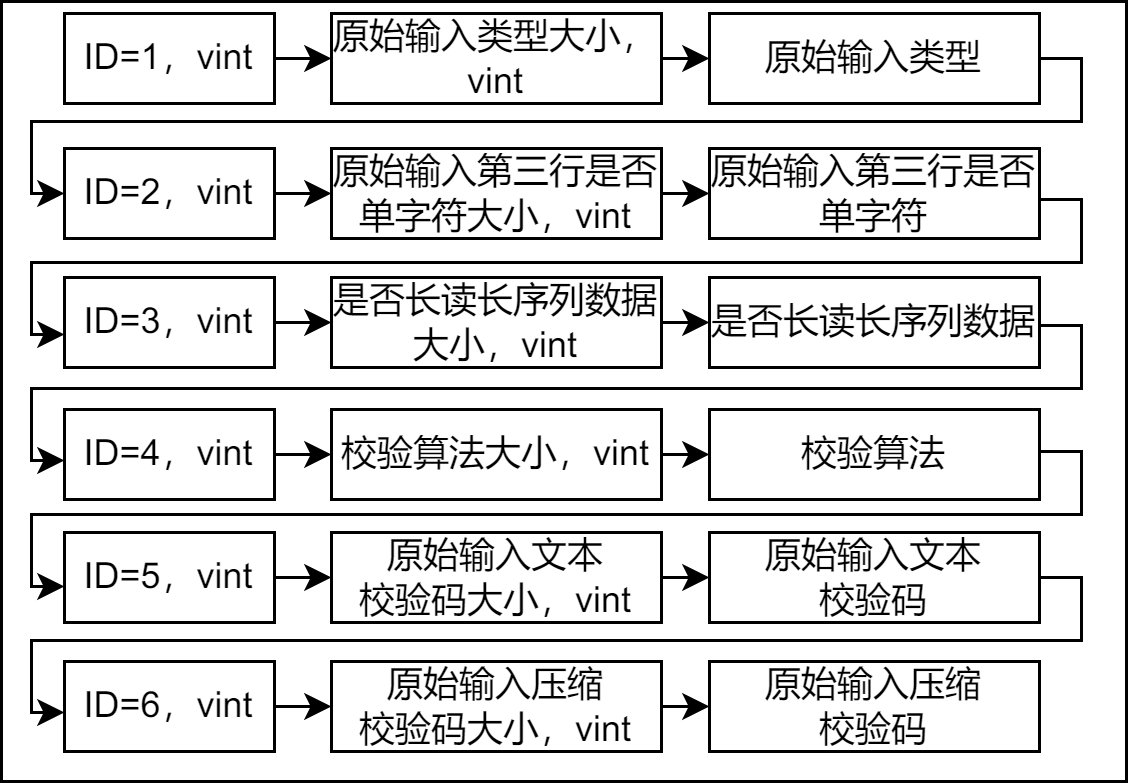


图 3 压缩信息码流结构

* + - * 1. 压缩数据码流

概述

压缩数据码流包含若干个FASTQ数据块码流和一个FASTA码流。单个FASTQ数据块码流如图 4所示，包括数据块基础信息、元数据文本校验码、元数据数据、序列长度校验码、序列长度数据、碱基序列校验码、碱基序列数据、质量分数文本校验码、质量分数数据、数据块自定义信息。FASTA压缩数据码流包括FASTA数据。

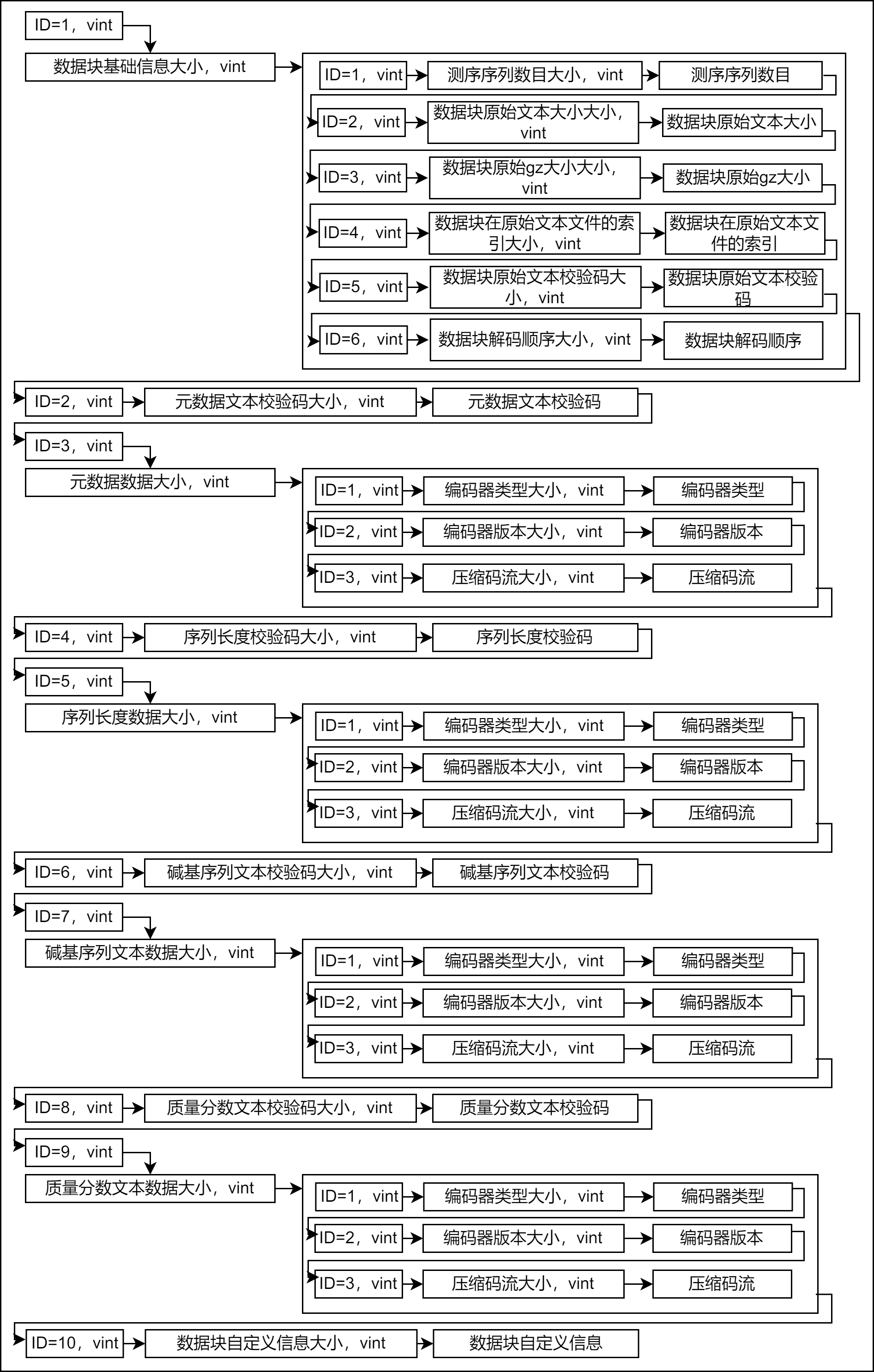


图 4 单个FASTQ数据块码流结构

数据块基础信息

数据块基础信息存储内容依次包含测序序列数目、数据块原始文本大小、数据块原始GZ大小、数据块在原始文本索引、数据块原始文本校验码、数据块解码顺序。

元数据文本校验码

元数据文本校验码，记录元数据的文本的校验码，类型为无符号整型。校验算法与前述校验算法字段(7.1.2.1.2　)一致。

元数据数据

元数据数据存储内容依次包含编码器、编码器版本、压缩码流。

序列长度校验码

序列长度校验码，记录序列长度的校验码。

序列长度数据

序列长度存储内容依次包含编码器、编码器版本、压缩码流。

碱基序列文本校验码

碱基序列文本校验码，记录碱基序列的文本的校验码。

碱基序列数据

碱基序列数据存储内容依次包含编码器、编码器版本、压缩码流。

质量分数文本校验码

质量分数文本校验码，记录质量分数的校验码。

质量分数数据

质量分数数据存储内容依次包含编码器、编码器版本、压缩码流。

数据块自定义信息

数据块自定义信息与编码器标识符相关，由编码器控制写入。

FASTA数据

FASTA编码码流包括3个元素为FASTA压缩码流的编码器类型、编码器版本和码流，如图 5所示。

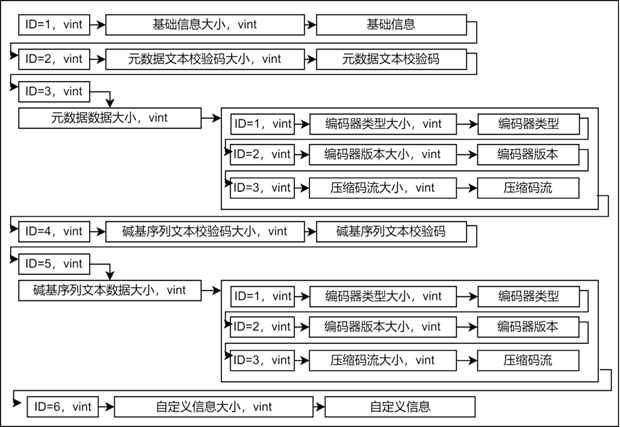


图 5 FASTA压缩数据码流结构

* + - * 1. 尾部码流

尾部码流的外层结构见图 1。尾部码流包含3个元素，分别为数据块信息、压缩方法参数、自定义信息。

数据块信息码流结构可见图 6，其尾部存储信息依次包含8种信息，数据块总数、表示单个数据块字节大小的比特数、最大数据块文本字节大小、各数据块的原始文本字节大小列表、各数据块的压缩后字节大小列表、表示单个偏移量的比特数、各数据块在原始文本起始偏移量列表、各数据块在压缩码流中起始偏移量列表。

压缩方法参数码流结构可见图 6，其存储信息依次包含4种信息，数据块总数、表示单个数据块字节大小的比特数、最大数据块文本字节大小、各数据块的原始文本字节大小列表。

自定义信息码流结构可见图 6，其尾部存储信息由编码器自行定义。

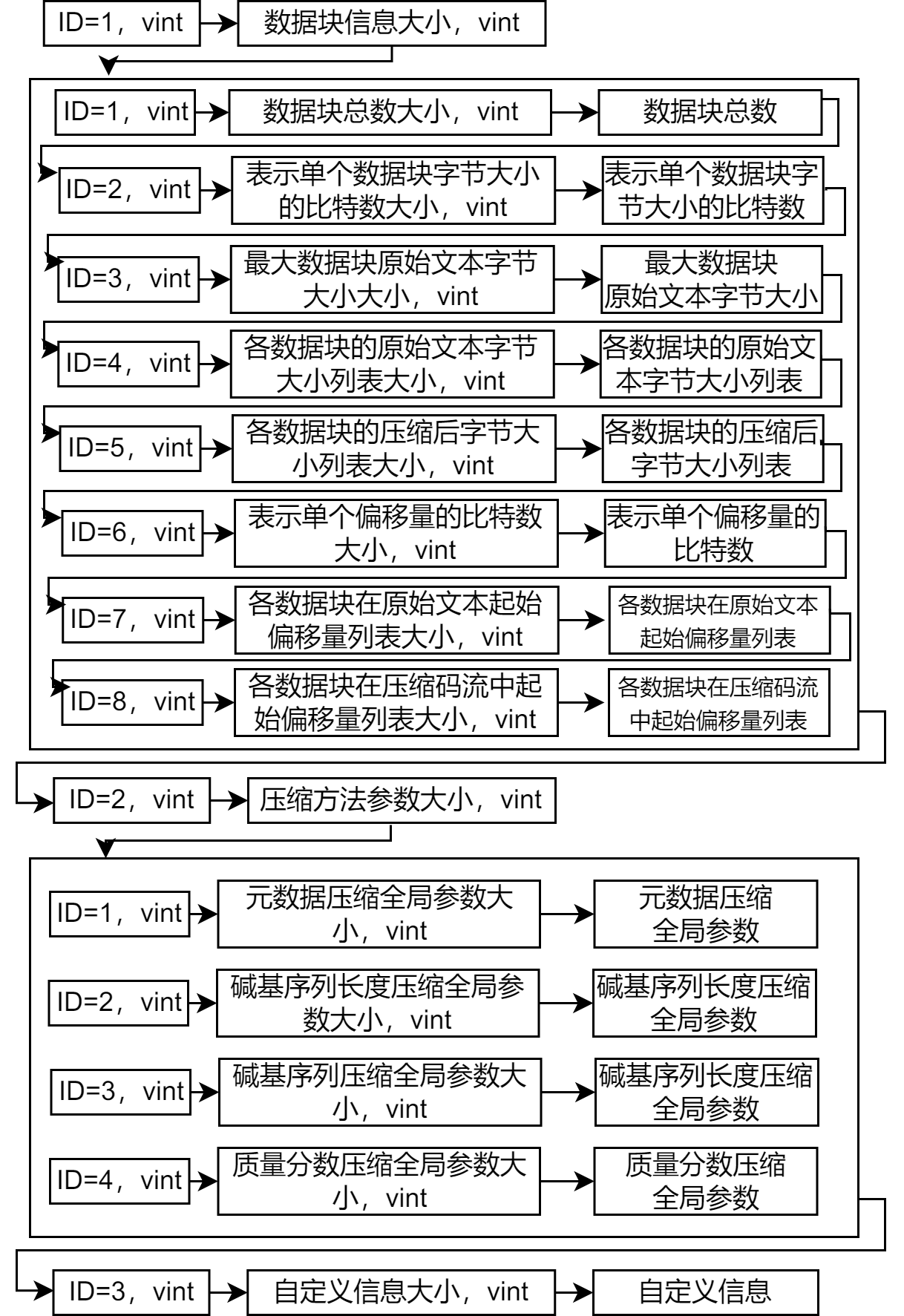


图 6 尾部数据码流结构

* + - 1. 元数据编码
         1. 基于窗口划分的编码

概述

元数据编码以符号（所有不为字母或数字的字符）为分隔标志、把首个元数据拆分成各个组分，基于每个组分的特点，对后续元数据的各个组分进行针对性编码，编码由两种数据流组成，分别是：组分信息、同组分元数据，其中，同组分元数据的数量大于等于1。

组分信息

组分信息以8字节为一个单位，假设有n个元数据组分，则组分信息有8n个字节，对于每个单位，前4个字节用于表示本组分元数据数量，后4个字节用于表示本组分首个元数据长度。

同组分元数据

概述

在一个同组分元数据的编码内，所有元数据的组分格式都与首个元数据的组分格式相同。假如一个同组分元数据的首个元数据的组分个数为M，则一个同组分元数据的编码由M+1个编码组成，分别是：首个元数据，M个组分。根据组分格式，组分可以分为纯数字组分和字母及数字组分，他们对格式相同的定义也不同：纯数字组分的格式相同认定标准为，两个元数据在进行组分划分后，相同顺位的组分必须同为纯数字组分；字母及数字组分的格式相同认定标准为，两个元数据在进行组分划分后，相同顺位的组分必须同为字母及数字组分。

首个元数据内容

首个元数据逐位编码，编码器范围为所有ASCII字符（也就是区间大小等于128），编码对象为首个元数据字符串。

纯数字组分

对于纯数字组分，根据以下判定继续细分为四类：

* 1. 若纯数字组分的长度大于8，或出现前缀0，则将其作为字符串处理，即处理逻辑与字母数字组分一致，可参考字母数字组分部分的处理逻辑；
  2. 若纯数字组分的长度小于等于8，且其前一个组分为字母和数字组分且内容为“length”，则将其作为序列长度处理，即无需额外编码，可通过序列长度进行恢复；
  3. 若纯数字组分非前述两种情况，且其前一个字符为“/”，且组分长度为1，则认定其为单端测序尾部信息，即恒定为1，无需额外编码；
  4. 若纯数字组分非前述三种情况，则将其视为一般纯数字组分进行编码。一般纯数字组分由以下两个部分组成：
* 差值类型:记录了在当前元数据的该纯数字组分与上一个元数据的该纯数字组分的差值的类型，有四个取值可能：0表示二者差值为0；1表示二者差值为1；2表示二者差值不为0或1且为小于4096的正数；3表示二者差值不为0或1且为绝对值小于4096的负数。差值类型用编码范围为4的编码器编码。差值类型的码流包含实际内容长度和实际内容，实际内容长度中记录实际内容的字节数。若出现四种取值可能之外的情况，则将该纯数字组分作为字符串类型处理，处理逻辑与字母与数字组分一致。
* 差值：字母及数字组分记录了在当前元数据的该纯数字组分与上一个元数据的该纯数字组分的差值的绝对值（不为0或1时在该编码中记录，否则已在差值类型的编码中记录）。差值包括一个十六进制长度和对应数值。差值长度使用范围为*n*的编码器进行编码，差值数值使用*n*个范围为16的编码器对各个位上的数字进行编码。

字母及数字组分

假设某字母及数字组分的长度为L，则该字母及数字组分由L个部分组成，分别对应各个位置的编码，每个部分为该位置的字符编码。对于这L个编码的其中任意一个都包含以下两个部分：

1. 标识符：该标识符取值范围为4，0为内容相同，1为内容不同但长度相同，2为当前组分长度大于首个元数据对应组分，3为当前组分长度小于首个元数据对应组分；该标识符用范围为4的编码器编码。当出现情况2和3时，会编码额外的长度差值。当出现情况2时，会增加额外的编码码流。
2. 不相同字符：不相同字符的取值范围为64（52个字母、10个数字、2个预留的终止符），因此不相同字符用范围为64的编码器编码。
   * + - 1. 基于最高位平面嵌入和流道列上下文的编码

概述

在7.1.3.1基于窗口划分的编码基础上，元数据按照窗口划分的各部分具有明确的物理意义，元数据通常以 ‘@’为开头，以‘.’、‘\_’、‘:’、‘#’、‘/’这五个符号作为间隔符进行划分，第一部分表示FASTQ文件名，第二部分表示第几个组分的指示，第三部分为固定的序列，即测序所用机器的型号和编号，第四部分为流道列，第六部分和第七部分为测序时的物理坐标X、Y，最后一部分指示本数据是否为单双端测序，以及若为双端测序，属于哪一端。在这八部分中，流道列和物理坐标X、Y列具有乱序性，针对这三部分可进一步采用最高位平面嵌入和流道列上下文的编码方法。

流道列和物理坐标编码

基于最高位平面嵌入和流道列上下文的编码过程如下：

1. 流道列数量是可枚举的，共有96种，将千位数和百位数做上下文，将十位数和个位数联合起来采用4比特组合16值算术编码。流道列的十位数和个位数联合后的取值范围为[1,16]；
2. 根据流道列数据对物理坐标X、Y列进行分组，分组方式为基于流道列十位数和个位数的值为依据（即0~16），分为16组。同时为了捕捉X、Y列分组后的线性信息，采用组内的第*n*+1个组分的数值与第*n*个组分的数值做差值的方式处理数据，将差值作为信息采用4比特组合16值算术编码。
3. 对于分组后的X、Y列差值数据，计算差值的最高位平面，将最高位平面嵌入4比特组合16值算术编码，采用联合编码形式（见图 7）。同时对于剩余位平面部分，将最高位平面用作上下文，捕捉X、Y列潜在的排序信息，使用4比特组合16值算术编码器编码；
4. 由步骤a)和b）得到X、Y列上下文模型的序号的计算公式：序号 = 组号×位平面×最高位平面。最终的上下文个数为16×10×16=2560个，位平面预设的最大值为10，但实际上位平面的范围分布在[3,6]之间，因此实际使用的上下文应该是1024个，编码器根据上下文模型序号进行遍历编码。

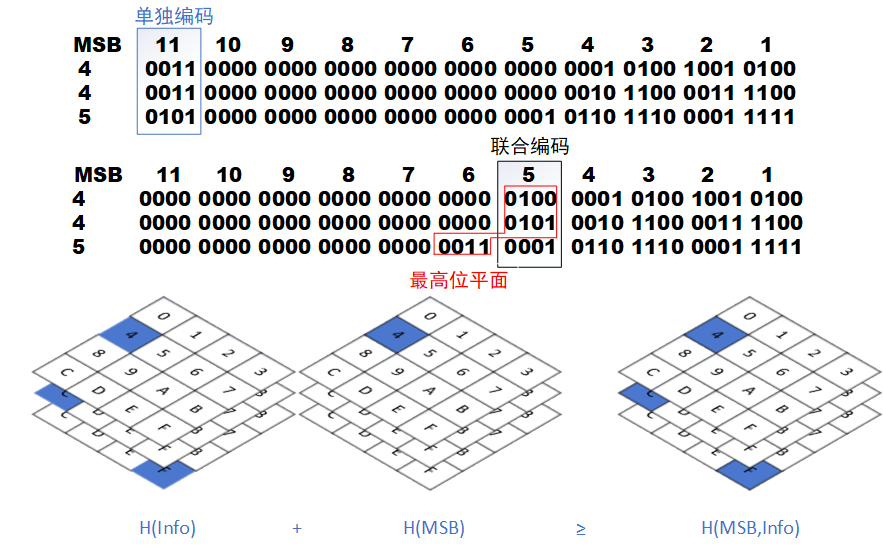


图 7最高位平面嵌入联合编码示意图

* + - 1. 序列长度编码
         1. 字节分割编码

概述

序列长度即各个碱基序列与质量分数串的长度，为一个整数列表，将各个长度与其前一个长度进行比较。若数据为短读长序列（二代测序序列），则分离得到三部分数据，即等长标识符、次高字节、最高字节；若数据为长读长序列（三代测序序列），则分离得到五部分数据，即等长标识符、末字节、次末字节、次高字节、最高字节。

等长标识符

等长标识符表示当前序列是否与上一条序列等长，为布尔型，进行二值编码。

末字节

若当前序列与上一条序列不等长，编码当前序列长度最低8位的字节。

次末字节

若当前序列与上一条序列不等长，编码当前序列长度次低8位的字节。

次高字节

若当前序列与上一条序列不等长，且序列为长读长，编码当前序列长度次高8位的字节。

最高字节

若当前序列与上一条序列不等长，且序列为长读长，编码当前序列长度最高8位的字节。

* + - 1. 碱基序列编码
         1. 基于输入性参考序列的容错配编码

简并碱基编码

概述

简并碱基编码利用简并碱基普遍质量分数偏低的特点，将简并碱基在低质量碱基中的位置编码为码流，编码由多个数据流组成，分别是简并碱基存在标识符、简并碱基字符、简并碱基最大质量分数、低质量碱基中非简并碱基个数、非简并碱基间的简并碱基数量。

简并碱基存在标识符

简并碱基存在标识符表示当前序列是否包含简并碱基。

简并碱基字符

简并碱基字符表示各个简并碱基的字符。

简并碱基最大质量分数

简并碱基最大质量分数表示当前序列中所有简并碱基对应的质量分数中的最大值。

低质量碱基中非简并碱基个数

低质量碱基中非简并碱基个数表示在所有质量分数小于等于简并碱基最大质量分数的碱基中，非简并碱基的个数。

非简并碱基间的简并碱基数量

非简并碱基间的简并碱基数量表示在所有质量分数小于等于简并碱基最大质量分数的碱基中，当前非简并碱基与上一个非简并碱基之间的简并碱基个数。

有无参考标记

该码流包含有参考压缩与无参考压缩，且二者都以整条序列为目标，因此，当序列编码同时启用这两种方式时，需要有码流记录各个序列是用哪种方式进行编码，也就是有无参考顺序。这部分只有一个码流，取值范围为2。

有参考编码

概述

有参考碱基序列编码包含序列在参考序列上的比对信息，包括比对位置、比对方向、错配个数、错配位置、错配字符。存在简并碱基的位置已经被记录，默认为比对成功。

比对位置

直接位置编码

直接记录序列比对到参考序列上的位置信息，若正向比对则为5’端所在位置，若反向互补比对则为3’端所在位置，最大取值范围为参考序列长度。

基于上下文的位平面比对位置编码

碱基测序中存在一定的局部性，即相邻测序序列大概率来自于参考序列的临近片段，在每个编码块内部，比对位置并不是均匀分布在参考序列的所有区域，可利用高比特位作为上下文建模低比特位的条件概率，对比对位置进行编码，具体如下：

* + - * 1. 删除连续相同的比对位置,连续出现几个相同的比对位置仅需保留一个。将每个原始数据中每个比对位置是否与前一个相同用一串二进制序列表示并存储在一个字段中,使用二阶算术编码器编码字该段；
        2. 使用中值滤波差分处理去重复后的比对位置，差分被减数被选择为前三个比对位置的中间值。结合中值滤波的差分将不会受到单个噪点对差分结果的影响。
        3. 将差分结果编码为位平面，这些平面分为最高有效位和剩余比特。最高有效位使用32值算术编码器直接编码，剩余比特部分最高有效位和每个位所在的位平面在算术编码器编码时用作上下文。

比对方向

比对方向表示序列是匹配到正向参考序列的具体位置，还是匹配到其反向互补序列上，有正向比对或反向互补比对两种可能。

基于二叉树分段匹配的碱基比对

碱基序列的前半段测序质量较高，而后半段较低，可以采用分段匹配的方法。比对不上的测序序列会被分段后再比对，直到记录比对信息所需的空间大于直接将子测序序列送入熵编码器编码所占空间，具体过程如下：

预设一个长度下限;

每输入一个测序序列，对该测序序列进行比对，如果成功比对上则比对结束;

如果未比对上且该测序序列的长度大于长度下限，将该测序序列分为前后等长的两段:子测序序列1和子测序序列2,分别重复b），直到长度小于长度下限。

错配个数

错配个数表示当前序列比对到参考序列的比对位置后，两个子串之间的差异碱基个数。

错配位置

直接位置编码

直接记录错配出现在序列上的位置。

基于自适应二值编码的错配位置编码

将每条测序序列的每个碱基是否错配当作二值信号进行编码，以取代当前的“错配数+增量错配位置编码”策略，利用高阶二值上下文建模测序序列中错配趋势的相关性。将碱基是否错配用一串和测序序列等长的二值信号进行编码，用0代表对应位置的碱基匹配上，1代表出错。在顺序编码二值信号时，用当前位置的前10个碱基的出错个数来量化上下文，作为编码字符的概率。

错配字符

错配类型共有12种即四种碱基每种可错配成其它三种。

无参考编码

若某序列无法与输入性参考序列比对成功，则用*k*阶区间编码进行逐个碱基字符的编码。

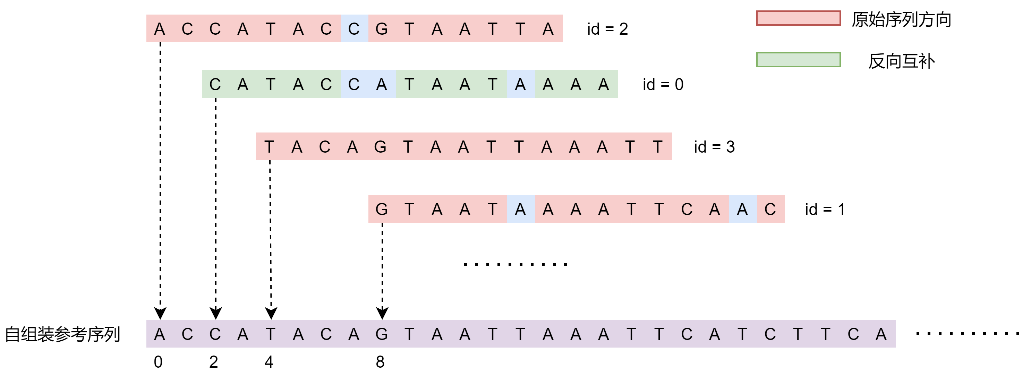
* + - * 1. 基于自组装参考序列的碱基编码

概述

无参考碱基序列压缩编码通过原始碱基数据自组装得到参考序列，再将每条碱基序列比对回参考序列实现编码，编码由多个数据流组成，分别是：自组装参考序列、比对方向、错配碱基及其数量和位置、碱基序列在参考序列的位置或偏移。

自组装参考序列

自组装参考序列采用A,C,G,T的字符数组的形式保存并作为第一个数据流。自组装参考序列应在测序数据种具有尽可能高的代表性，由重复率高的测序碱基序列组成。如图 8所示，自组装参考序列代表多条前后重叠的碱基序列。

图 8 自组装参考碱基压缩编码示例

编码顺序

无参考压缩编码支持两种编码顺序：重排序列和保留序列顺序。重排序列模式按照碱基序列在参考序列的比对先后顺序进行编码，以图 8为例，重排序列模式下，ID为2的碱基序列第一个进行编码，ID为0的碱基序列第二个进行编码，ID为3的碱基序列第三个进行编码，依次类推。保留序列顺序的模式按照碱基序列在FASTQ文件的先后顺序进行编码，ID为0的碱基序列第一个进行编码，ID为1的碱基序列第二个进行编码，ID为2的碱基序列第三个进行编码，依次类推。

碱基序列比对方向

碱基序列比对到参考序列的方向可以是原始序列方向，也可以是其反向互补方向，在编码时只考虑其中一种情况并做标记。

错配碱基

碱基序列与自组装参考序列的比对只考虑碱基替换（即错配）的情况，不考虑插入、删除。每个错配碱基采用8位无符号字符保存，所有碱基序列的错配碱基组成连续的字符数组。

示例：

以**错误!未找到引用源。**为例，ID为2的碱基序列上的错配碱基为C（自组装参考序列上对应的碱基为A），ID为0的碱基序列上的错配碱基为C、A、A，ID为1的碱基序列上的错配碱基为A、A。

错配碱基数量

每条碱基序列的错配碱基数量用8位无符号整数保存，所有碱基序列的错配碱基数量组成连续的整数数组。

错配碱基位置

碱基序列的错配碱基位置（错配碱基相对碱基序列的位置）采用16位无符号整数保存，每条碱基序列上的错配碱基位置根据其数量的不同分配到不同的数组。以图 8为例，ID为2的碱基序列上的错配碱基位置只有7，将位置7按编码顺序放在错配碱基数量为1的错配碱基位置数组中；ID为0的碱基序列上的错配碱基位置有5、6和11，将位置5、6和11按编码顺序放在错配碱基数量为3的错配碱基位置数组中，依次类推。

碱基序列位置

碱基序列在自组装参考序列的位置采用64位无符号整数保存。对于重排序列顺序的模式，碱基序列位置是一个有序的整数数组，需要进一步执行差分编码转为16位无符号整数数组。以图 8为例，如果按照重排序列的编码顺序，ID为2、0、3、1的碱基序列在自组装参考序列的起始位置分别为0、2、4、8，进行差分编码后得到的位置数组为0、2、2、4；如果按照碱基序列在FASTQ文件的先后顺序编码，ID为0、1、2、3的碱基序列在自组装参考序列的起始位置分别为2、8、0、4，不需要其他处理。

* + - 1. 质量分数
         1. 概述

质量分数压缩编码通过对原始质量分数数据进行统计分析，选择适当的压缩模型实现编码，编码包括碱基标识符、均值标识符、质量分数范围、质量分数位流等。

* + - * 1. 碱基标识符

质量分数序列和碱基序列长度相等，碱基与质量分数一一对应，测序仪在测序过程中对每个碱基的测定结果有不同的置信度，而质量分数代表了对应碱基测序结果的正确程度，碱基和质量分数之间有很强的相关性。碱基标识符作为是否采用碱基进行模型构建的判断，在编码时只考虑其中一种情况并做标记。当选择读取碱基时，在构建混合上下文模型进行质量分数编解码过程中会读取每一个质量分数对应的碱基并作为上下文信息；不选择读取碱基时，可直接跳过该过程。

* + - * 1. 均值标识符

每条测序序列的质量分数之间存在相关性，可以通过统计并计算每条测序序列的质量分数平均值来建立混合上下文模型，均值范围与质量分数范围一致。当选择计算均值时，在位流中保存每条质量分数序列的均值并在构建混合上下文模型过程中读取每一条测序序列对应的均值并作为上下文信息；不选择计算均值时，直接跳过该过程。

* + - * 1. 质量分数范围

假设*P*为测序碱基被测序软件识别错误的概率，测序软件根据打分公式计算出该碱基错误率对应的质量分数，错误率*P*越低，则碱基的质量分数得分越高。不同的评分软件有不同的质量分数计算公式，比较常用的有两种格式，第一种是早期测序软件Solexa使用的质量分数计算公式：

()

式中：

Qsolexa——测序软件Solexa产生的质量分数值，取值范围为 [-5,62];

*P* ——测序错误率，取值范围[0,1];

另一种是Sanger软件使用的打分公式：

()

式中：

——测序软件Sange产生的质量分数值，取值范围为 [0,93];

——测序错误率，取值范围[0,1];

通过这两种公式计算的质量分数在*P*较小时几乎一致，*P*越大则差异愈加明显。为了能够使用ASCII码中可打印的字符范围表示质量分数，软件会选择一个偏移量，例如*Q*solexa加上偏移量64即Phred+64体系，最终范围是[59,126]；*Q*sanger加上偏移量33即Phred+33体系，最终范围是[33,126]。

1. 假设一碱基错误概率为0.1%，则按照Sanger软件算出的Qsanger值为30，对应Phred+33体系转换后为ASCII码为66，对应字符‘?’。

不同质量分数范围影响着编码模型的大小。编码时，计算当前编码块中质量分数的最大值和最小值，根据最大值和最小值确定质量分数范围，进而确定模型大小。将最大值和最小值记做质量分数范围数据流进行压缩编码。

* + - * 1. 上下文模型

如图 9所示，对每一个需要编码的质量分数字符构建对应的混合上下文模型。首先选取编码质量分数的四阶上下文模型并进行量化，然后根据是否包含碱基、是否包含均值构建适当的混合上下文模型。采用混合上下文模型如图 9所示，蓝色数据代表当前要进行编码的字符，绿色数据块代表当前字符的前四个字符，红色数据块表示当前字符对应的碱基和上一碱基，黄色字符代表当前质量分数行的均值。对三种不同的上下文信息进行联合建模，采取分布优化的策略达到最优效果。将质量分数较小的符号合并在一起，以同一字符的表现形式进行编码压缩，在其内部进行二次编码，对具体的字符展开编码。

1. 混合上下文模型如图 9所示，蓝色方块表示当前要编码的质量分数字符，绿色方块表示当前编码质量分数字符的前四个字符则q1=35， q2=34， q3=38，q4=35，红色方块表示与当前编码质量分数字符所对应的前两个碱基’A’和’C’，黄色方块表示当前编码质量分数字符所在行的均值M。

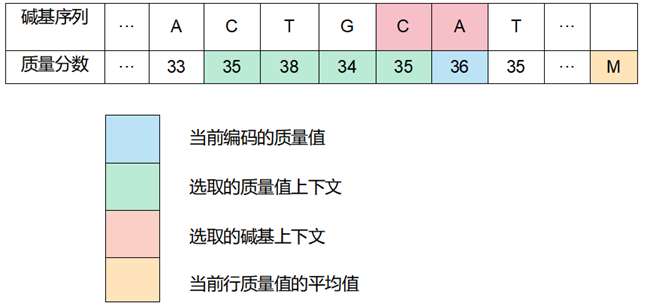


图 9 混合上下文模型示例

* + - * 1. 遍历方式

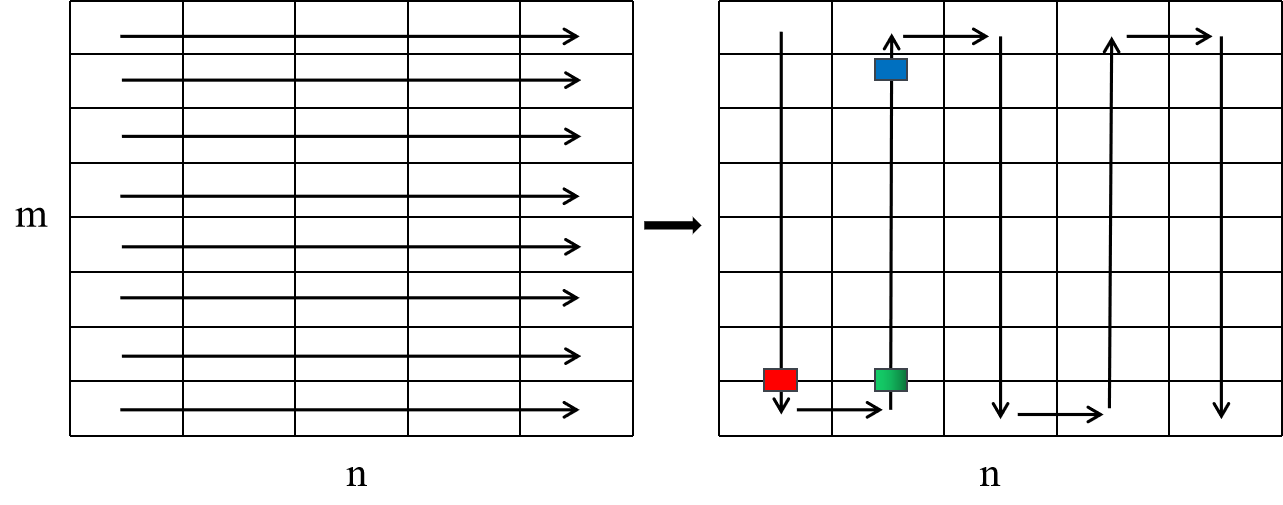
确定好概率预测模型之后，采用蛇形遍历的方式获取数据间的相关性（如图 10所示）。首先向下遍历考虑更多的是行之间的相关性，字符浮动不会过于太大，在编码完红色字符之后没有再从第二列头部重新开始，而是尾尾相连，头头相接。

图 10 常规按行遍历方式和按列蛇形遍历方法

* + - 1. FASTA碱基序列编码
         1. 概述

FASTA数据碱基序列可以采用LZMA或者区间编码直接编码或者采用基于最大匹配的有参考编码。

* + - * 1. 基于最大匹配的有参考编码

先对目标序列按s2采样，将采样到的k长子串存储到布隆过滤器中。再对参考序列进行步长为s1的采样，此处s1和s2互质，在参考序列上采样到的子串在布隆过滤器中查找，如果查找到，则将该子串存入哈希表，否则丢弃。最后再一次采样目标序列，并将采样到的子串与哈希表中的子串匹配，找到则在目标序列和参考序列上左右延伸确定最大精确匹配的长度。得到最大精确匹配后，允许出现3个以内的碱基突变，进而进行双端延伸。配置不同的参数k、s1和s2，检测到的最长精确匹配的长度不同。得到目标序列与参考序列直接的匹配后，对匹配进行编码。如图 11样例所示，下划线部分为目标序列与参考序列的匹配，中间有两个位置的碱基不匹配。目标序列最终编码为{(18,25),TA,18}，最后一个精确匹配部分不需要记录起始位置。



图 11 FASTA碱基最大匹配编码示例

* + 1. 位流语义
       1. 系统位流语义
          1. 主体码流定义

主体码流内容的组成，如表 7所示。

表 7主体码流组成

|  |  |  |
| --- | --- | --- |
| 主体码流定义 | 说明 | 描述 |
| avsg\_stream() { | - | - |
| **header\_magic\_num** | 头部魔数，用4个字节记录的“avsg”字符串信息，用于识别码流是否为本文件规定的码流。 | uc(4) |
| **header\_id** | 数值1，表示接下来内容为头部码流 | vi |
| **header\_len** | 头部码流的长度 | vi |
| header\_stream() | - | - |
| **comp\_data\_id** | 数值2，表示接下来内容为压缩数据 | vi |
| **comp\_data\_len** | 压缩数据码流的长度 | vi |
| comp\_data\_stream() | - | - |
| **tailer\_id** | 数值3，表示接下来内容为尾部码流 | vi |
| **tailer\_len** | 尾部码流的长度 | vi |
| tailer\_stream() | - | - |
| **tailer\_magic\_num** | 尾部魔数，用4个字节记录的“avsg”字符串信息，用于识别码流是否为本文件规定的码流。 | uc(4) |
| } |  |  |

* + - * 1. 头部码流定义

头部码流内容的组成，如表 8所示。

表 8头部码流组成

|  |  |  |
| --- | --- | --- |
| 头部码流定义 | 说明 | 描述 |
| header\_stream () { | - | - |
| **basic\_id** | 数值1，表示接下来内容为基础信息 | vi |
| **basic\_len** | 基础信息长度 | vi |
| basic\_stream() | - | - |
| **comp\_info\_id** | 数值2，表示接下来内容为压缩信息 | vi |
| **comp\_info\_len** | 压缩信息长度 | vi |
| comp\_info\_stream() | - | - |
| } |  |  |

* + - * 1. 码流基础信息定义

码流基础信息内容的组成，如表 9所示。

表 9码流基础信息组成

|  |  |  |
| --- | --- | --- |
| 码流基础信息码流定义 | 说明 | 描述 |
| basic\_stream () { | - | - |
| **std\_type** | ID为1，元素内容为标准类型，元素内容类型为字符串。如内容为两个字节的字符串‘fq’，表示码流为基因测序数据FASTQ。如内部为两个字节的字符串‘fa’，表示码流为基因组数据FASTA。 | eb |
| **std\_version** | ID为2，元素内容为当前被编码数据对应的标准版本号，元素内容类型为无符号整型。本文件规定内容对应版本号为1。 | eb |
| **encoder\_id** | ID为3，元素内容为编码器标识符，表示编码器的种类，元素内容类型为字符串。如使用参考软件写入两个字节的字符串‘ar’，其他编码器可写入自定义信息，以示区别。 | eb |
| if (next\_ebml\_id()==4) | - | - |
| **raw\_name** | ID为4，元素内容为原始输入FASTA/FASTQ文件名，表示编码时原始输入的名称，元素内容类型为字符串。 | eb |
| if (next\_ebml\_id()==5) | - | - |
| **raw\_textbyte** | ID为5，元素内容为原始文本字节大小，表示编码时原始输入文本的字节数，元素内容类型为无符号整型。 | eb |
| if (next\_ebml\_id()==6) | - | - |
| **raw\_gzbyte** | ID为6，元素内容为原始GZ字节大小，表示原始输入文件为GZ格式时，其GZ字节数，元素内容类型为无符号整型。 | eb |
| } |  |  |

* + - * 1. 压缩信息码流定义

压缩信息内容的组成，如表 10所示。

表 10压缩信息码流组成

|  |  |  |
| --- | --- | --- |
| 压缩信息码流定义 | 说明 | 描述 |
| comp\_info\_stream () { | - | - |
| if (next\_ebml\_id()==1) | - | - |
| **rawfile\_type** | ID为1，元素内容为原始输入类型，表示原始输入的类型，元素内容类型为无符号整型。如0表示文本输入，1表示输入为GZ格式，2表示输入为管道。 | eb |

表10　压缩信息码流组成（续）

|  |  |  |
| --- | --- | --- |
| 压缩信息码流定义 | 说明 | 描述 |
| **plussign\_only** | ID为2，元素内容表示FASTQ中每条测序记录的第三行是否只包含单一字符‘+’，元素内容类型为布尔型，0表示不是，1表示是。 | eb |
| **longseq** | ID为3，元素内容为是否长读长序列数据，表示原始输入是否为长读长测序（即三代测序）数据，元素内容类型为布尔型，0表示不是，1表示是。 | eb |
| **checkalgo** | ID为4，元素内容为校验算法，表示本压缩码流中使用的各个校验码的校验算法，元素内容类型为无符号整型。如‘0’表示MD5，‘1’表示CRC32，‘2’表示XXH3。 | eb |
| if (next\_ebml\_id()==5) | - | - |
| **rawtext\_check** | ID为5，元素内容为原始输入文本校验码，表示压缩原始输入的文本内容的校验码，元素内容类型为无符号整型。 | eb |
| if (next\_ebml\_id()==6) | - | - |
| **rawcomp\_check** | ID为6，元素内容为原始输入压缩校验码，表示若输入非文本而是压缩格式（如GZ），整个码流的校验码，元素内容类型为无符号整型。 | eb |
| } |  |  |

* + - * 1. 压缩数据码流定义

压缩数据内容的组成，如表 11所示。

表 11压缩数据码流组成

|  |  |  |
| --- | --- | --- |
| 压缩数据码流定义 | 说明 | 描述 |
| comp\_data\_stream () { | - | - |
| if(ebml\_value(std\_type)== 'fq'){ |  |  |
| for(i=0;i<ebml\_value(block\_num);i++) { | - | - |
| **read\_block\_indiv\_id** | 数值1，表示接下来内容为单个FASTQ数据块码流 | vi |
| **read\_block\_indiv\_len** | 单个FASTQ数据块码流长度 | vi |
| read\_block\_indiv\_stream() |  | - |
| } |  |  |
| } |  |  |
| if(ebml\_value(std\_type)== 'fa'){ |  |  |

表 11压缩数据码流组成（续）

|  |  |  |
| --- | --- | --- |
| 压缩数据码流定义 | 说明 | 描述 |
| **fasta\_id** | 数值2，表示接下来内容为FASTA压缩数据码流 | vi |
| **fasta\_len** | FASTA压缩数据码流长度 | vi |
| fasta\_stream() |  | - |
| } |  |  |
| } |  |  |

* + - * 1. 单个FASTQ数据块码流定义

单个FASTQ数据块码流内容的组成，如表 12所示。

表 12单个FASTQ数据块码流组成

|  |  |  |
| --- | --- | --- |
| 单个FASTQ数据块码流定义 | 说明 | 描述 |
| read\_block\_indiv\_stream () { | - | - |
| **block\_basic\_id** | 数值1，表示接下来内容为数据块基础信息 | vi |
| **block\_basic\_len** | 数据块基础信息码流的长度 | vi |
| block\_basic\_stream() | - | - |
| if (next\_ebml\_id()==2){ | **-** | - |
| **block\_metadata\_check** | ID为2，元素内容为元数据文本校验码，记录元数据的文本的校验码，元素内容类型为无符号整型，校验算法与前述校验算法字段一致 | eb |
| **block\_metadata\_id** | 数值3，表示接下来内容为元数据 | vi |
| **block\_metadata\_len** | 元数据码流的长度 | vi |
| block\_metadata\_stream() | - | - |
| if (next\_ebml\_id()==4) | **-** | - |
| **block\_seqlen\_check** | ID为4，元素内容为序列长度文本校验码，记录序列长度的校验码，元素内容类型为无符号整型，校验算法与前述校验算法字段一致 | eb |
| **block\_seqlen\_id** | 数值5，表示接下来内容为序列长度 | vi |
| **block\_seqlen\_len** | 序列长度码流长度 | vi |
| block\_seqlen\_stream() | - | - |
| if (next\_ebml\_id()==6) | **-** | - |

表12　单个数据块码流组（续）

|  |  |  |
| --- | --- | --- |
| 单个数据块码流定义 | 说明 | 描述 |
| **block\_seq\_check** | ID为6，元素内容为碱基序列文本校验码，记录碱基序列的文本的校验码，元素内容类型为无符号整型，校验算法与前述校验算法字段一致 | eb |
| **block\_seq\_id** | 数值7，表示接下来内容为碱基序列 | vi |
| **block\_seq\_len** | 碱基序列码流的长度 | vi |
| block\_seq\_stream() | - | - |
| if (next\_ebml\_id()==8) | **-** | - |
| **block\_qual\_check** | ID为8，元素内容为质量分数文本校验码，记录质量分数的校验码，元素内容类型为无符号整型，校验算法与前述校验算法字段一致 | eb |
| **block\_qual\_id** | 数值9，表示接下来内容为质量分数 | vi |
| **block\_qual\_len** | 质量分数码流长度 | vi |
| block\_qual\_stream() | - | - |
| if (next\_ebml\_id()==100) | - |  |
| **block\_encode\_custom** | ID为100，数据块自定义信息，与编码器标识符相关，由编码器控制写入，本标准不定义元素内容 | eb |
| } |  |  |

* + - * 1. 数据块基础信息码流定义

数据块基础信息内容的组成，如表 13所示。

表 13数据块基础信息码流组成

|  |  |  |
| --- | --- | --- |
| 数据块基础信息码流定义 | 说明 | 描述 |
| block\_basic\_stream () { | - | - |
| if (next\_ebml\_id()==1) | **-** | - |
| **block\_readcnt** | ID为1，元素内容为测序序列数目，记录当前数据块的序列数目，元素内容类型为无符号整型 | eb |
| if (next\_ebml\_id()==2) | - | - |
| **block\_textbyte** | ID为2，元素内容为数据块原始文本大小，记录当前数据块的原始文本大小，元素内容类型为无符号整型 | eb |
| if (next\_ebml\_id()==3) | - | - |

表13　数据块基础信息码流组成（续）

|  |  |  |
| --- | --- | --- |
| 数据块基础信息码流定义 | 说明 | 描述 |
| **block\_gzbyte** | ID为3，元素内容为数据块原始gz大小，记录当前数据块的原始GZ大小，元素内容类型为无符号整型 | eb |
| if (next\_ebml\_id()==4) | **-** | - |
| **block\_textoffset** | ID为4，元素内容为数据块在原始文本索引，记录当前数据块在原始文本中的偏移量，元素内容类型为无符号整型 | eb |
| if (next\_ebml\_id()==5) | **-** | - |
| **block\_textcheck** | ID为5，元素内容为数据块原始文本校验码，记录当前数据块文本内容的校验码，元素内容类型为无符号整型 | eb |
| **block\_order** | ID为6，元素内容为数据块解码顺序，记录当前数据块中各存储码流的解码顺序。内容为 ‘0’表示解码先后顺序为元数据-序列长度-碱基序列文本-质量分数文本，‘1’表示解码先后顺序为元数据-序列长度-质量分数文本-碱基序列文本 | eb |
| } |  |  |

* + - * 1. 元数据码流定义

元数据内容的组成，如表 14所示。

表 14元数据码流组成

|  |  |  |
| --- | --- | --- |
| 元数据码流定义 | 说明 | 描述 |
| block\_metadata\_stream () { | - | - |
| **metadata\_encoder** | ID为1，元素内容为元数据的编码器类型，元素内容类型为字符串。当类型为‘0’时，编码器为LZMA；当类型为‘1’时，编码器基于窗口划分的编码；当类型为‘2’时，编码器为基于最高位平面嵌入和流道列的上下文方法 | eb |
| **metadata\_encoder\_ver** | ID为2，元素内容为元数据编码器版本，内容类型为无符号整型 | eb |
| **metadata\_data\_id** | 内容为3，表示接下来内容为元数据编码码流 | vi |
| **metadata\_data\_len** | 元数据编码码流长度 | vi |
| if(ebml\_value(metadata\_encoder)==0) |  |  |
| **metadata\_data\_lzma** | LZMA编码器编码码流 | lz |
| else | - | - |
| metadata\_data\_stream() | - | - |
| } |  |  |

* + - * 1. 序列长度码流定义

序列长度内容的组成，如表 15所示。

表 15序列长度码流组成

|  |  |  |
| --- | --- | --- |
| 序列长度码流定义 | 说明 | 描述 |
| block\_seqlen\_stream () { | - | - |
| **seqlen\_encoder** | ID为1，元素内容为序列长度的编码器类型，元素内容类型为字符串。当类型为‘0’时，编码器为LZMA；当类型为‘1’时，编码器为字节分割编码 | eb |
| **seqlen\_encoder\_ver** | ID为2，元素内容为序列长度的编码器版本，元素内容类型为无符号整型 | eb |
| **seqlen\_data\_id** | 内容为3，表示接下来内容为序列长度编码码流 | vi |
| **seqlen\_data\_len** | 序列长度编码码流长度 | vi |
| seqlen\_data\_stream() | - | - |
| } |  |  |

* + - * 1. 碱基序列码流定义

碱基序列内容的组成，如表 16所示。

表 16碱基序列码流组成

|  |  |  |
| --- | --- | --- |
| 碱基序列码流定义 | 说明 | 描述 |
| block\_seq\_stream () { | - | - |
| **seq\_encoder** | ID为1，元素内容为碱基序列的编码器类型，元素内容类型为字符串。当类型 ‘0’时，编码器为LZMA；当类型为‘1’时，编码器为容错配的序列比对和编码（不含简并碱基编码）；当类型为‘2’时，编码器为容错配的序列比对和编码（含简并碱基编码）；当类型为‘3’时，编码器为经AMGC优化的容错配编码（不含简并碱基编码）；当类型为‘4’时，编码器为经AMGC优化的容错配编码（含简并碱基编码）；当类型为‘5’时，编码器为自组装参考压缩编码 | eb |
| **seq\_encoder\_ver** | ID为2，元素内容为碱基序列的编码器版本，元素内容类型为无符号整型 | eb |
| **seq\_data\_id** | 内容为3，表示接下来内容为碱基序列编码码流 | vi |
| **seq\_data\_len** | 碱基序列编码码流长度 | vi |
| if(ebml\_value(seq\_encoder)==0) | - | - |
| **seq\_data\_lzma** | 使用LZMA编码器编码的序列 | lz |

表16　碱基序列码流组成（续）

|  |  |  |
| --- | --- | --- |
| 碱基序列码流定义 | 说明 | 描述 |
| if(ebml\_value(seq\_encoder)>=1 && ebml\_value(seq\_encoder)<=4) | - | - |
| seq\_data\_stream() | - | - |
| if(ebml\_value(seq\_encoder)==5) | - | - |
| seq\_assem\_stream() | - | - |
| } |  |  |

* + - * 1. 质量分数码流定义

质量分数内容的组成，如表 17所示。

表 17质量分数码流组成

|  |  |  |
| --- | --- | --- |
| 质量分数码流定义 | 说明 | 描述 |
| block\_qual\_stream () { | - | - |
| **qual\_encoder** | ID为1，元素内容为质量分数的编码器类型，元素内容类型为字符串。当类型为‘0’时，编码器为LZMA；当编码器为‘1’时，编码器为ACO。 | eb |
| **qual\_encoder\_ver** | ID为2，元素内容为质量分数的编码器版本，元素内容类型为无符号整型 | eb |
| **qual\_data\_id** | 内容为2，表示接下来内容为质量分数压缩码流 | vi |
| **qual\_data\_len** | 质量分数压缩码流长度 | vi |
| if(ebml\_value(qual\_encoder)==0) |  |  |
| **qual\_data\_lzma** | LZMA编码器编码质量分数码流 | lz |
| else |  |  |
| qual\_data\_stream() | - | - |
| } |  |  |

* + - * 1. FASTA数据码流定义

FASTA数据码流的组成，如表 18所示。

表 18 FASTA数据码流组成

|  |  |  |
| --- | --- | --- |
| FASTA数据码流定义 | 说明 | 描述 |
| fasta\_stream () { | - | - |
| if (next\_ebml\_id()==1) | - | - |
| **fa\_basic** | ID为1，元素内容为基础信息 | eb |
| if (next\_ebml\_id()==2) | - | - |
| **fa\_metadata\_check** | ID为2，元素内容为元数据文本校验码，记录元数据的文本的校验码，元素内容类型为无符号整型，校验算法与前述校验算法字段一致 | eb |
| **fa\_metadata\_id** | 数值3，表示接下来内容为元数据 | vi |
| **fa\_metadata\_len** | 元数据码流的长度 | vi |
| fa\_metadata\_stream() | - | - |
| if (next\_ebml\_id()==4) | - | - |
| **fa\_seq\_check** | ID为4，元素内容为碱基序列文本校验码，记录碱基序列的文本的校验码，元素内容类型为无符号整型，校验算法与前述校验算法字段一致 | eb |
| **fa\_seq\_id** | 数值5，表示接下来内容为碱基序列 | vi |
| **fa\_seq\_len** | 碱基序列码流的长度 | vi |
| fa\_seq\_stream() | - | - |
| if (next\_ebml\_id()==100) | - | - |
| **fa\_encode\_custom** | ID为100，自定义信息，与编码器标识符相关，由编码器控制写入，本标准不定义元素内容 | eb |
| } |  |  |

* + - * 1. 尾部码流定义

尾部码流的组成，如表 19所示。

表 19尾部码流组成

|  |  |  |
| --- | --- | --- |
| 尾部码流定义 | 说明 | 描述 |
| tailer\_stream () { | - | - |
| if(ebml\_value(std\_type)== 'fq'){ |  |  |
| **read\_block\_info\_id** | 内容为1，表示接下来内容为数据块信息 | vi |
| **read\_block\_info\_len** | 数据块信息码流长度 | vi |

表 19尾部码流组成（续）

|  |  |  |
| --- | --- | --- |
| 尾部码流定义 | 说明 | 描述 |
| read\_block\_info\_stream() | - | - |
| } |  |  |
| **encoder\_param\_id** | 内容为2，表示接下来内容为编码器参数 | vi |
| **encoder\_param\_len** | 编码器参数码流长度 | vi |
| encoder\_param\_stream() | - | - |
| if (next\_ebml\_id()==100) | - |  |
| **encoder\_custom** | ID为100，表示数据块自定义信息，与编码器标识符相关，由编码器控制写入，本标准不定义元素内容 | eb |
| } |  |  |

* + - * 1. 数据块信息码流定义

数据块信息的组成，如表 20所示。

表 20数据块信息码流组成

|  |  |  |
| --- | --- | --- |
| 数据块信息码流定义 | 说明 | 描述 |
| read\_block\_info\_stream () { | - | - |
| **block\_num** | ID为1，元素内容为数据块总数，元素内容类型为无符号整型 | eb |
| **size\_bitunit** | ID为2，元素内容为表示单个数据块字节大小的比特数，单位为比特数，元素内容类型为无符号整型 | eb |
| **block\_maxbyte** | ID为3，元素内容为最大数据块文本字节大小，记录最大的块文本大小，元素内容类型为无符号整型 | eb |
| **block\_textbyte** | ID为4，元素内容为各数据块的原始文本字节大小列表，记录各个数据块的原始文本字节大小列表，每个元素占据比特数等于表示单个数据块字节大小的比特数，元素个数等于数据块总数 | eb |
| **block\_compbyte** | ID为5，元素内容为各数据块的压缩后字节大小列表，记录各个数据块的压缩后字节大小列表，每个元素占据比特数等于表示单个数据块字节大小的比特数，元素个数等于数据块总数 | eb |
| **offset\_bitunit** | ID为6，元素内容为表示单个偏移量的比特数，单位为比特数，元素内容类型为无符号整型 | eb |
| **block\_textoffset** | ID为7，元素内容为各数据块在原始文本起始偏移量列表，记录各数据块在原始文本起始偏移量，每个元素占据比特数等于表示单个偏移量的比特数，元素个数等于数据块总数 | eb |
| **block\_compoffset** | ID为8，元素内容为各数据块在压缩码流中起始偏移量列表，记录各数据块在压缩码流中起始偏移量，每个元素占据比特数等于表示单个偏移量的比特数，元素个数等于数据块总数 | eb |
| } |  |  |

* + - * 1. 编码器参数码流定义

编码器参数的组成，如表 21所示。

表 21编码器参数码流组成

|  |  |  |
| --- | --- | --- |
| 压缩方法参数码流定义 | 说明 | 描述 |
| encoder\_param\_stream () { | - | - |
| if (next\_ebml\_id()==1) | **-** | - |
| **param\_metadata** | ID为1，元素内容为元数据编码的全局参数 | eb |
| if (next\_ebml\_id()==2) | **-** | - |
| **param\_seqlen** | ID为2，元素内容为序列长度编码的全局参数 | eb |
| if (next\_ebml\_id()==3) | **-** | - |
| **param\_seq\_id** | 内容为3，表示接下来内容为碱基序列编码的全局参数 | vi |
| **param\_seq\_len** | 碱基序列编码全局参数码流长度 | vi |
| param\_seq\_stream() | - |  |
| if (next\_ebml\_id()==4) | **-** | - |
| **param\_qual** | ID为4，元素内容为质量分数编码的全局参数 | eb |
| } |  |  |

* + - * 1. 碱基序列编码全局参数码流定义

碱基序列编码全局参数的组成，如表 22所示。

表 22碱基序列编码全局参数码流组成

|  |  |  |
| --- | --- | --- |
| 碱基序列编码全局参数码流定义 | 说明 | 描述 |
| param\_seq\_stream () { | - | - |
| **rc\_order** | ID为1，元素内容为整型，记录区间编码的阶数 | eb |
| if(ebml\_value(seq\_encoder)>=1 && ebml\_value(seq\_encoder)<=4) | - |  |
| **ref\_bool** | ID为2，元素内容为bool型，是否使用有参压缩 | eb |
| **ref\_id** | ID为3，元素内容为参考序列的ref-seq编号，类型为字符串，实际存储时可以是参考序列首条染色体的元数据 | eb |
| **ref\_name** | ID为4，元素内容为参考序列文件名，类型为字符串 | eb |
| **ref\_checksum** | ID为5，元素内容为参考序列校验码，类型为字符串 | eb |

表 22碱基序列编码全局参数码流组成（续）

|  |  |  |
| --- | --- | --- |
| 碱基序列编码全局参数码流定义 | 说明 | 描述 |
| if (next\_ebml\_id()==6) | **-** | - |
| **ref\_encoder** | ID为6，元素内容为参考序列的编码器类型。当类型为‘0’时，不保存参考序列数据；当类型为‘1’时，采用LZMA编码；当类型为‘2’时，采用区间编码。 | eb |
| if (next\_ebml\_id()==7) | **-** | - |
| **ref\_encoder\_ver** | ID为7，元素内容为参考序列的编码器版本，元素内容类型为无符号整型 | eb |
| **param\_seq\_id** | 内容8，表示接下来内容为参考序列码流 | vi |
| **param\_seq\_len** | 参考序列码流长度 | vi |
| if(ebml\_value(ref\_encoder)==1) |  |  |
| **ref\_data\_lzma** | 采用LZMA编码的参考序列码流 | lz |
| if(ebml\_value(ref\_encoder)==2 and ebml\_value(ref\_encoder\_ver)==1) | - | - |
| **ref\_data\_rc\_num** | 8字节，记录ref\_data\_rc中的字符个数 | u(64) |
| **ref\_data\_rc** | 参考序列码流，上下文阶数由rc\_order确定 | rc(v,5) |
| ｝ |  |  |

* + - 1. 元数据码流语义

元数据码流语义见表 23。

表 23元数据码流定义

| 元数据码流定义 | 说明 | 描述符 |
| --- | --- | --- |
| metadata\_data\_stream(){ | - | - |
| **metadata\_chunk\_id** | 内容为1，表示接下来内容为元数据块内容 | vi |
| **metadata\_chunk\_len** | 元数据块内容码流长度 | vi |
| metadata\_chunk\_stream() | - | - |
| **vec\_format** | ID为2，元素内容为各元数据块序列数目与首元数据长度，以8个字节为一单位，每个单位前4个字节存储当前元数据块序列数目，后4个字节存储当前元数据块首个元数据的长度 | eb |
| } | - | - |

* + - * 1. 元数据块码流定义

元数据块码流定义见表 24。

表 24元数据块码流定义

| 元数据块码流定义 | 说明 | 描述符 |
| --- | --- | --- |
| metadata\_chunk\_stream(){ | - | - |
| if (ebml\_value(metadata\_encoder\_ver)==1){ |  |  |
| for (i = 0; i<ebml\_length(vec\_format)/8; ++i){ | - | - |

表24　元数据块码流定义（续）

|  |  |  |
| --- | --- | --- |
| 元数据块码流定义 | 说明 | 描述符 |
| **first\_metadata** | 首个元数据码流，首个元数据采用1阶编码，编码范围为所有ASCII字符 | src(128) |
| for (i = 0; i<num\_tokens(first\_metadata); ++i){ | - | - |
| if(is\_number(token(first\_metadata,i))){ | - | - |
| **diff\_type** | 差异类型码流记录在当前元数据的该纯数字组分与上一个元数据的该纯数字组分的差值的类型，有四种取值：0表示二者差值为0，1表示二者差值为1，2表示二者差值不为0或1且为正，3表示二者差值不为0或1且为负。差值类型用编码范围为4的编码器编码 | src(4) |
| **diff\_scale\_n\_len** | 差异值绝对值编码长度，差异值绝对值编码长度是指，将差异值表示为16进制时，该16进制串的长度，然后用范围为16的编码器编码 | src(16) |
| for (i = 0; i<16; ++i) { | - | - |
| **diff\_scale\_n\_num** | 差异值绝对值编码值，差异值绝对值编码值是指，将差异值表示为16进制后，对于该16进制串在某个位上的数字独立用范围为16的编码器编码 | src(16) |
| } | - | - |
| } | - | - |
| else{ | - | - |
| if(ebml\_value(metadata\_encoder)==1){ | 基于窗口划分编码 | - |
| **char\_flag** | 标识符，取值范围为4。0为内容相同，1为内容不同但长度相同，2为当前组分长度大于首个元数据对应组分，3为当前组分长度小于首个元数据对应组分 | src(4) |
| **len\_diff** | 当前组分长度与首个元数据对应组分长度差值的绝对值，在char\_flag为2或3时会编码数值 | src(12) |
| for (j = 0; j < 512; ++j) { | - | - |
| **diff\_char** | 差异标识符码流，差异标识符的取值范围为64。具体来说，开始编码时初始化512个计数器，当char\_flag不为0时，当前元数据在当前组分的每位字符均在此编码，编码方式是组分采用左对齐的方式，每个字符用对应的计数器进行编码，若当前元数据在当前组分的字符数小于512，按实际字符数进行编码 | src(64) |
| } | - | - |
| } | - | - |
| else{ | 基于最高位平面嵌入和流道列上下文方法 | - |

表24　元数据块码流定义（续）

|  |  |  |
| --- | --- | --- |
| 元数据块码流定义 | 说明 | 描述符 |
| **tile** | 流道列，先编码流道列，然后用流道列分别指示X,Y列进行压缩.由于流道列是可枚举的，因此上限为6\*16=96 | ac(0，96) |
| **spilt\_x** | X列的数字，采用16值编码器编码第一个测序序列 | ac(0，16) |
| **spilt\_y** | Y列的数字，采用16值编码器编码第一个测序序列 | ac(0，16) |
| for (j = 0;j<16;++j) { | - |  |
| **next\_msb** | 为下一组的指示最高有效位，帮助解码，以及和正常需要编码的数字一起做联合编码 | ac(0，16) |
| **context\_diff\_char** | 分组过后的差异标识符码流，差异标识符的取值范围为16,因为它是以流道列为上下文后分组的，所以要根据流道的不同分成16组 | ac(0，16) |
| } | - | - |
| } | - | - |
| } | - | - |
| } | - | - |
| } | - | - |
| } |  |  |
| } | - | - |

* + - 1. 序列长度位流语义

序列长度码流定义见表 25。

表 25序列长度定义

| 序列长度定义 | 说明 | 描述符 |
| --- | --- | --- |
| seqlen\_data\_stream() { | - | - |
| if(ebml\_value(seqlen\_encoder)==1 and ebml\_value(seqlen\_encoder\_ver)==1){ |  |  |
| **seqlen\_same** | 序列长度相等，当前序列长度与上一条序列长度相等，则为0，否则为1；seqlen\_same的解析结果中的首位必然是1，因为要记录首个seqlen | src(2) |
| **seqlen\_1byte** | 序列长度首字节信息，序列长度采用小端序表示为2个字节，内存地址从低到高，第一个字节的编码 | src(256) |
| **seqlen\_2byte** | 序列长度次字节信息，序列长度采用小端序表示为2个字节，内存地址从低到高，第二个字节的编码 | src(256) |

表25序列长度定义（续）

| 序列长度定义 | 说明 | 描述符 |
| --- | --- | --- |
| if (ebml\_value(longseq)) { | - | - |
| **seqlen\_3byte** | 序列长度次末字节信息，序列长度采用小端序表示为4个字节，内存地址从低到高，第三个字节的编码 | src(256) |
| **seqlen\_4byte** | 序列长度末字节信息，序列长度采用小端序表示为4个字节，内存地址从低到高，第四个字节的编码 | src(256) |
| } | - | - |
| } |  |  |
| } | - | - |

* + - 1. 碱基序列位流语义
         1. 容错配的序列比对和编码码流语义

容错配的序列比对和编码码流定义见表 26。

表 26容错配的序列比对和编码码流定义

| 容错配的序列比对和编码码流定义 | | 说明 | 描述符 |
| --- | --- | --- | --- |
| seq\_data\_stream() { | | - | - |
| **i**f(next\_ebml\_id()==1){ | | - | - |
| **ambi\_id** | | 内容为1，表示接下来内容为简并碱基码流 | vi |
| **ambi\_len** | | 简并碱基码流长度 | vi |
| ambi\_stream() | | - | - |
| } | | - | - |
| if(ref\_bool){ | | - | - |
| **align\_seq\_id** | 内容为2，表示接下来内容为容错配比对信息码流 | | vi |
| **align\_seq\_len** | 容错配比对信息码流长度 | | vi |
| align\_seq\_stream() | - | | - |
| } | - | | - |
| **unalignseq\_id** | 内容为4，表示接下来内容为比对失败序列内容 | | vi |
| **unalignseq\_len** | 比对失败序列内容码流长度 | | vi |
| if (ebml\_value(seq\_encoder\_ver)==1){ |  | |  |

表26　 容错配的序列比对和编码码流定义（续）

|  |  |  |
| --- | --- | --- |
| 容错配的序列比对和编码码流定义 | 说明 | 描述符 |
| **unalignseq\_stream\_num** | 记录unalignseq\_stream的字符数 | u(32) |
| **unalignseq\_stream** | 无参位流，比对失败序列用范围为4的k阶区间编码器的编码码流,简并碱基统一使用字符‘A’代替，此处阶数由rc\_order确定。 | rc(v，4) |
| **}** |  |  |
| } |  | - |

简并碱基位流语义

简并碱基语义定义见表 27。

表 27简并碱基定义

| 简并碱基定义 | 说明 | 描述符 |
| --- | --- | --- |
| ambi\_stream () { | - | - |
| **ambi\_base\_flag\_id** | 内容为1，表示接下来内容为简并碱基存在标识符 | vi |
| **ambi\_base\_flag\_len** | 简并碱基存在标识符码流长度 | vi |
| if (ebml\_value(seq\_encoder\_ver)==1){ |  |  |
| **ambi\_base\_flag\_num** | ambi\_base\_flag处理的字符数 | u(32) |
| **ambi\_base\_flag** | 简并碱基存在标识符，1位无符号整数数组。值为0表示该碱基序列不包含简并碱基，值为1表示包含简并碱基 | rc(0，2) |
| } |  |  |
| **ambi\_base\_char\_id** | 内容为2，表示接下来内容为简并碱基字符 | vi |
| **ambi\_base\_char\_len** | 简并碱基字符码流长度 | vi |
| if (ebml\_value(seq\_encoder\_ver)==1){ |  |  |
| **ambi\_base\_char\_num** | ambi\_base\_char处理的字符数 |  |
| **ambi\_base\_char** | 简并碱基字符,储存各个简并碱基的字符,编码范围为11，包括：NMRYKSWHBVD | rc(0，11) |
| } |  |  |
| **ambi\_max\_qual\_id** | 内容为3，表示接下来内容为简并碱基最大质量分数 | vi |
| **ambi\_max\_qual\_len** | 简并碱基最大质量分数码流长度 | vi |
| if (ebml\_value(seq\_encoder\_ver)==1){ |  |  |
| **ambi\_max\_qual\_num** | ambi\_max\_qual处理的字符数 |  |
| **ambi\_max\_qual** | 简并碱基最大质量分数，储存各个含简并碱基的碱基序列上，简并碱基对应的最大质量分数，编码范围为95，即质量分数取值范围 | rc(0，95) |
| } |  |  |

表27　简并碱基定义（续）

|  |  |  |
| --- | --- | --- |
| 简并碱基定义 | 说明 | 描述符 |
| **ambi\_lowqual\_normalcount\_id** | 内容为4，表示接下来内容为低质量分数碱基中非简并碱基个数 | vi |
| **ambi\_lowqual\_normalcount\_len** | 低质量分数碱基中非简并碱基个数码流长度 | vi |
| ambi\_lowqual\_normalcount\_stream() | - | - |
| **ambi\_lowqual\_normalintv\_id** | 内容为5，表示接下来内容为非简并碱基间的简并碱基数量 | vi |
| **ambi\_lowqual\_normalintv\_len** | 非简并碱基间的简并碱基数量码流长度 | vi |
| ambi\_lowqual\_normalintv\_stream() | - | - |
| } | - |  |

低质量分数碱基中非简并碱基个数位流语义

低质量分数碱基中非简并碱基个数码流语义定义见表 28。

表 28低质量分数碱基中非简并碱基个数码流定义

| 低质量分数碱基中非简并碱基个数码流定义 | 说明 | 描述符 |
| --- | --- | --- |
| ambi\_lowqual\_normalcount\_stream() { | - | - |
| if (ebml\_value(seq\_encoder\_ver)==1){ |  |  |
| **ambi\_normal\_num** | 低质量分数碱基中非简并碱基个数 | u(32) |
| **ambi\_normal\_num\_n\_len** | 低质量分数碱基中非简并碱基个数编码长度，低质量分数碱基中非简并碱基个数编码长度是指，将低质量分数碱基中非简并碱基个数表示为2进制时，该2进制串的长度，用范围为64的编码器编码。 | src(64) |
| for(i = 0;i< 64;++i) { | - |  |
| **ambi\_normal\_num\_n\_num** | 低质量分数碱基中非简并碱基个数编码值，低质量分数碱基中非简并碱基个数编码值是指，将低质量分数碱基中非简并碱基个数表示为2进制后，对于该2进制串在某个位上的数字独立用范围为2的编码器编码。 | src(2) |
| } |  |  |
| } |  |  |
| } |  |  |

非简并碱基间的简并碱基数量位流语义

非简并碱基间的简并碱基数量码流语义定义见表 29。

表 29非简并碱基间的简并碱基数量码流定义

| 非简并碱基间的简并碱基数量码流定义 | 说明 | 描述符 |
| --- | --- | --- |
| ambi\_lowqual\_normalintv\_stream () { | - | - |
| if (ebml\_value(seq\_encoder\_ver)==1){ |  |  |
| **ambi\_normal\_intv** | 非简并碱基间的简并碱基数量 | u(32) |

表29 非简并碱基间的简并碱基数量码流定义（续）

|  |  |  |
| --- | --- | --- |
| 非简并碱基间的简并碱基数量码流定义 | 说明 | 描述符 |
| **ambi\_normal\_intv\_n\_len** | 非简并碱基间的简并碱基数量编码长度，非简并碱基间的简并碱基数量编码长度是指，将非简并碱基间的简并碱基数量表示为2进制时，该2进制串的长度，然后用范围为64的编码器编码。 | src(64) |
| for (i= 0;i<64;++i) { | - |  |
| **ambi\_normal\_intv\_n\_num** | 非简并碱基间的简并碱基数量编码值，非简并碱基间的简并碱基数量编码值是指，将非简并碱基间的简并碱基数量表示为2进制后，对于该2进制串在某个位上的数字独立用范围为2的编码器编码。 | src(2) |
| } |  |  |
| } |  |  |
| } |  |  |

容错配比对信息位流语义

容错配比对信息定义见表 30。

表 30容错配比对信息定义

| 容错配比对信息定义 | 说明 | 描述符 |
| --- | --- | --- |
| align\_seq\_stream () { | - | - |
| **align\_flag\_id** | 内容为1，表示接下来内容为比对标记符 | vi |
| **align\_flag\_len** | 比对标记符码流长度 | vi |
| if (ebml\_value(seq\_encoder\_ver)==1){ |  |  |
| **align\_flag\_stream\_num** | align\_flag\_stream编码的字符数量 | u(32) |
| **align\_flag\_stream** | 比对标记符，编码范围为2，0为比对失败，1为比对成功 | rc(0，2) |
| } |  |  |
| **align\_pos\_id** | 内容为2，表示接下来内容为比对位置 | vi |
| **align\_pos\_len** | 比对位置码流长度 | vi |
| if(ebml\_value(seq\_encoder)==1 || ebml\_value(seq\_encoder)==2){ | - | - |
| **align\_pos\_plain\_stream\_num** | 比对位置个数 | u(32) |
| **align\_pos\_plain\_stream }** | 比对位置，将比对起始位置数字转为二进制，位数与参考序列长度一致（即二进制位数小于参考序列长度的二进制比特数则填0），对二进制进行范围为2的区间编码 | rc(0，2) |
| if(ebml\_value(seq\_encoder)==3 || ebml\_value(seq\_encoder)==4) | - | - |
| align\_pos\_amgc\_stream() | - | - |

表30　容错配比对信息定义（续）

|  |  |  |
| --- | --- | --- |
| 容错配比对信息定义 | 说明 | 描述符 |
| **align\_strand\_id** | 内容为3，表示接下来内容为比对方向 | vi |
| **align\_strand\_len** | 比对方向码流长度 | vi |
| if (ebml\_value(seq\_encoder\_ver)==1){ |  |  |
| **align\_strand\_stream\_num** | 比对方向个数 | u(32) |
| **align\_strand\_stream** | 比对方向，用范围为2的编码器编码，0为正向比对，1为反向互补比对 | rc(0，2) |
| **mis\_num\_id** | 内容为4，表示接下来内容为错配个数 | vi |
| **mis\_num\_len** | 错配个数码流长度 | vi |
| if (ebml\_value(seq\_encoder\_ver)==1){ |  |  |
| **mis\_num\_stream\_num** | mis\_num\_stream处理字符个数 | u(32) |
| **mis\_num\_stream }** | 错配个数，设错配容纳上限为N，将错配个数转为二进制，位数与N+1一致（即二进制位数小于N+1的二进制比特数则填0），对二进制进行范围为2的区间编码 | rc(0，2) |
| **mis\_pos\_id** | 内容为5，表示接下来内容为错配位置 | vi |
| **mis\_pos\_len** | 错配位置码流长度 | vi |
| if(ebml\_value(seq\_encoder\_ver)==1){ |  |  |
| **mis\_pos\_stream\_num** | 错配位置个数 | u(32) |
| **mis\_pos\_stream** | 错配位置，设当前剩余序列长度为N（若当前错配为当前序列的首个错配，则N为序列长度，否则N为序列长度与上一个错配位置之差），将错配个数转为二进制，位数与N+1一致（即二进制位数小于N+1的二进制比特数则填0），对二进制进行范围为2的区间编码 | rc(0，2) |
| } |  |  |
| **mis\_base\_id** | 内容为6，表示接下来内容为错配碱基字符 | vi |
| **mis\_base\_len** | 错配碱基字符码流长度 | vi |
| if (ebml\_value(seq\_encoder\_ver)==1){ |  |  |
| **mis\_base\_stream\_num** | 错配碱基字符数量 | u(32) |
| **mis\_base\_stream** | 错配碱基字符，对于ACGT任一字符，错配情况均为4种（另三种字符与N），因此用范围为4的区间编码 | rc(0，4) |
| } |  | - |

* + - * 1. 经AMGC优化的容错配比对位置编码码流语义

经AMGC优化的容错配比对位置编码码流定义见表 31。

表 31经AMGC优化的容错配比对位置编码码流定义

| 经AMGC优化的容错配码流定义 | 说明 | 描述符 |
| --- | --- | --- |
| align\_pos\_amgc\_stream () { | - | - |
| **first\_mapped\_pos** | 存储参考序列上第一个比对位置 | u(32) |
| for(i=1;i<cardinality(align\_flag\_stream);++i) { |  |  |
| **is\_equal\_to\_previous** | 表示当前碱基位置是否和前一个碱基相等的信息，1为相同，0为不同 | u(1) |
| **mapped\_pos\_sign** | 存储比对位置差的符号，处理后的差非负 | u(1) |
| **most\_signifi\_bits** | 存储二进制表示的位置差最左边0的个数，等价于最高有效位 | u(8) |
| **bits\_left** | 存储所有位置差去掉最左边的0后，剩余的比特位 | u(1) |
| } |  |  |
| } |  |  |

* + - * 1. 自组装参考碱基序列位流语义

自组装参考碱基序列定义见表 32。

表 32自组装参考碱基序列定义

| 自组装参考压缩的碱基序列定义 | 说明 | 描述符 |
| --- | --- | --- |
| seq\_assem\_stream() { | - | - |
| **reads\_count** | 碱基序列数量，32位无符号整数。表示碱基序列的总数。 | u(32) |
| **order\_preserving\_flag** | 保留顺序压缩标志，1位无符号整数。值为“1”表示保留序列顺序的压缩模式,值为“0”表示重排序列的压缩模式。 | u(1) |
| **reference\_stream\_encode\_method** | 参考序列编码方式保留字，8位无符号整数，保留字，表示自组装参考序列数据流的编码方式，由用户定义，不可为0。 | u(8) |
| **reads\_direction\_stream\_encode\_method** | 碱基序列比对方向编码方式，8位无符号整数，保留字，表示碱基序列比对方向数据流的编码方式，由用户定义，不可为0。 | u(8) |
| **mismatch\_count\_stream\_encode\_method** | 错配碱基数量编码方式， 8位无符号整数，保留字，表示错配碱基数量数据流的编码方式，由用户定义，不可为0。 | u(8) |
| **mismatch\_base\_stream\_encode\_method** | 错配碱基编码方式，8位无符号整数，保留字，表示错配碱基数据流的编码方式，由用户定义，不可为0。 | u(8) |
| **mismatch\_pos\_stream\_encode\_method** | 错配碱基位置编码方式，8位无符号整数，保留字，表示错配碱基位置数据流的编码方式，由用户定义，不可为0。 | u(8) |

表 32自组装参考碱基序列定义（续）

|  |  |  |
| --- | --- | --- |
| 自组装参考压缩的碱基序列定义 | 说明 | 描述符 |
| **reads\_pos\_stream\_encode\_method** | 保留序列顺序模式的碱基序列位置编码方式， 8位无符号整数，保留字，表示保留序列顺序模式下碱基序列位置数组的编码方式，由用户定义，在保留序列顺序模式下不可为0，其他情况可为0。 | u(8) |
| **reads\_offset\_stream\_encode\_method** | 重排序列顺序模式的碱基序列位置偏移编码方式，8位无符号整数，保留字，表示重排序列顺序模式下碱基序列位置偏移数组的编码方式，由用户定义，在重排序列顺序模式下不可为0，其他情况可为0。 | u(8) |
| reference\_stream() | - | - |
| reads\_direction\_stream() | - | - |
| mismatch\_count\_stream() | - | - |
| mismatch\_base\_stream() | - | - |
| mismatch\_pos\_stream() | - | - |
| if (order\_preserving\_flag) { | - | - |
| reads\_pos\_stream() | - | - |
| } | - | - |
| else { | - | - |
| reads\_offset\_stream() | - | - |
| } | - | - |
| } | - | - |

参考序列流定义

参考序列流定义见表 33。

表 33参考序列流定义

| 参考序列流定义 | 说明 | 描述符 |
| --- | --- | --- |
| reference\_stream() { | - | - |
| **reference\_length** | 参考序列长度，64位无符号整数。表示自组装的参考序列长度。 | u(64) |
| for (i = 0; i < reference\_length; ++i) | - | - |
| **reference**[i] | 参考序列，8位无符号字符数组。表示自组装的参考序列字符串。 | uc(8) |
| } | - | - |
| } | - | - |

碱基序列比对方向定义

碱基序列比对方向定义见表 34。

表 34碱基序列比对方向定义

| 碱基序列比对方向定义 | 说明 | 描述符 |
| --- | --- | --- |
| reads\_direction\_stream() { | - | - |
| for (i = 0; i < reads\_count; ++i) { | - | - |
| **reads\_direction**[i] | 碱基序列比对方向，1位无符号整数数组。值为“1”表示碱基序列在参考序列的比对方向为其反向互补，值为“0”表示碱基序列在参考序列的比对方向为其原始序列方向。 | u(1) |
| } | - | - |
| } | - | - |

错配碱基数量流定义

错配碱基数量流定义见表 35。

表 35错配碱基数量流定义

| 错配碱基数量流定义 | 说明 | 描述符 |
| --- | --- | --- |
| mismatch\_count\_stream(){ | - | - |
| for (i = 0; i < reads\_count; ++i) { | - | - |
| **mismatch\_count**[i] | 错配碱基数量，8位无符号整数数组。储存每条碱基序列在参考序列上误匹配的碱基个数。 | u(8) |
| } | - | - |
| } | - | - |

错配碱基数据流定义

错配碱基数据流定义见表 36。

表 36错配碱基数据流定义

| 错配碱基数据流定义 | 说明 | 描述符 |
| --- | --- | --- |
| mismatch\_base\_stream() { | - | - |
| **mismatch\_base\_size** | 错配碱基数组大小，32位无符号整数。表示错配碱基数组的大小。 | u(32) |
| for (i = 0; i < mismatch\_base\_size; ++i) { | - | - |
| **mismatch\_base**[i] | 错配碱基数组，8位无符号字符数组。储存碱基序列在参考序列上匹配错误的碱基。 | uc(8) |
| } | - | - |
| } | - | - |

错配碱基位置流定义

错配碱基位置流定义见表 37。

表 37错配碱基位置流定义

| 错配碱基位置流定义 | 说明 | 描述符 |
| --- | --- | --- |
| mismatch\_pos\_stream() { | - | - |
| **max\_mismatch\_count** | 最大错配碱基数量，8位无符号整数。表示每条碱基序列允许的最大错配碱基个数。 | u(8) |
| for (i = 1; i <= max\_mismatch\_count; ++i) { | - | - |
| **mismatch\_pos\_size**[i] | 错配碱基位置数组大小，32位无符号整数。表示在错配碱基个数为i的情况下，错配碱基位置数组的大小。 | u(32) |
| for (j = 0;j<mismatch\_pos\_size[i]; ++j) { | - | - |
| **mismatch\_pos**[i][j] | 错配碱基位置数组，16位无符号整数二维数组。mismatch\_pos储存在不同错配碱基数量下每条碱基序列的错配位置。 | u(16) |
| } | - | - |
| } | - | - |
| } | - | - |

保留序列顺序模式的碱基序列位置流定义

保留序列顺序模式的碱基序列位置流定义见表 38。

表 38保留序列顺序模式的碱基序列位置流定义

| 保留序列顺序模式的碱基序列位置流定义 | 说明 | 描述符 |
| --- | --- | --- |
| reads\_pos\_stream() { | - | - |
| for (i = 0; i < reads\_count; ++i) { | - | - |
| **order\_preserving\_reads\_pos**[i] | 保留序列顺序模式的碱基序列位置，64位无符号整数数组。按照碱基序列在FASTQ文件的先后顺序储存每条碱基序列在自组装参考序列的起始位置。 | u(64) |
| } | - | - |
| } | - | - |

重排序列顺序模式的碱基序列位置偏移流定义

重排序列顺序模式的碱基序列位置偏移流定义见表 39。

表 39碱基序列位置偏移流定义

| 碱基序列位置偏移流定义 | 说明 | 描述符 |
| --- | --- | --- |
| reads\_offset\_stream() { | - | - |
| **reads\_offset**[0] = 0 | 碱基序列位置，64位无符号整数数组。按照碱基序列在自组装参考序列的先后顺序储存每条碱基序列在参考序列的位置。 | u(16) |
| for (i = 1; i < reads\_count; ++i) { | - | - |
| **reads\_offset**[i] = **reads\_pos**[i]–**reads\_pos**[i–1] | 碱基序列位置偏移，16位无符号整数数组。储存每条碱基序列在自组装参考序列的位置偏移量，通过差分编码得到。 | u(16) |
| } | - | - |
| } | - | - |

* + - 1. 质量分数位流语义

基于ACO编码器的质量分数码流定义见表 40。

表 40基于ACO的质量分数码流定义

| 基于ACO的质量分数序列定义 | 说明 | 描述符 |
| --- | --- | --- |
| qual\_data\_stream() { | - | - |
| **quality\_count** | 质量分数序列数量，32位无符号整数。表示块中质量字符的总数。 | u(32) |
| **traverse\_order** | 块内遍历顺序，1位无符号整数。值为“1”表示采用按行顺序扫描模型；值为“0”表示采用按列顺序扫描模型。 | u(1) |
| **base\_flag** | 碱基标识，1位无符号整数。值为“1”表示采用包含碱基的上下文压缩模型；值为“0”表示采用不包含碱基的上下文压缩模型。 | u(1) |
| **mean\_value\_flag** | 均值标识，1位无符号整数。值为“1”表示采用包含均值的上下文压缩模型；值为“0”表示采用不包含均值的上下文压缩模型。 | u(1) |
| **max\_value** | 质量分数最大值，8位无符号整数。表示当前压缩块中质量分数字符的最大值。 | u(8) |
| **min\_value** | 质量分数最小值，8位无符号整数。表示当前压缩块中质量分数字符的最小值。 | u(8) |
| **reads\_number** | 当前压缩块测序序列总个数，8位无符号整数。表示当前压缩块中测序序列的总个数。 | u(8) |
| for (i = 0; i < quality\_count; ++i) { | - | - |
| **quality\_value** | 当前编码质量分数字符，表示采用最高达7阶的上下文熵编码所得的当前质量分数编码值。 | qs(7) |
| } | - | - |
| if (mean\_value\_flag==1) { | - | - |
| get\_mean\_info() | - | - |
| } | - | - |
| } | - | - |

* + - * 1. 质量分数分块语义

均值标识序列定义见表 41。

表 41均值标识序列定义

| 均值标识序列定义 | 说明 | 描述符 |
| --- | --- | --- |
| get\_mean\_info() { | - | - |
| for (i = 0; i < reads\_count; ++i) { | - | - |
| **mean\_value[i]** | 质量分数均值，8位无符号整数。表示质量分数所在序列的质量分数平均值。 | u(8) |
| } | - | - |
| } | - | - |

* + - 1. FASTA压缩位流语义
         1. 概况

FASTA压缩码流分为元数据与碱基序列码流。

* + - * 1. FASTA元数据码流定义

FASTA元数据码流的组成，如表 42所示。

表 42 FASTA元数据码流定义

| FASTA压缩码流定义 | 说明 | 描述符 |
| --- | --- | --- |
| fa\_metadata\_stream() { | - | - |
| **fa\_metadata\_encoder** | ID为1，元素内容为元数据的编码器类型，元素内容类型为字符串。当类型为‘0’时，编码器为LZMA。 | eb |
| **fa\_metadata\_encoder\_ver** | ID为2，元素内容为元数据的编码器版本，元素内容类型为无符号整型 | eb |
| **fa\_metadata\_data\_id** | 内容为3，表示接下来内容为元数据压缩码流 | vi |
| **fa\_metadata\_data\_len** | 元数据压缩码流长度 | vi |
| **if(**(ebml\_value(fa\_metadata\_encoder)==0) | - | - |
| **fa\_metadata\_data\_lzma** | lzma编码结果 | lz |
| } | - | - |

* + - * 1. FASTA碱基序列码流定义

FASTA碱基序列码流的组成，如表 43所示。

表 43 FASTA碱基序列码流定义

| FASTA碱基压缩码流定义 | 说明 | 描述符 |
| --- | --- | --- |
| fa\_seq\_stream() { | - | - |
| **fa\_seq\_encoder** | ID为1，元素内容为碱基序列的编码器类型，元素内容类型为字符串。当类型为‘0’时，编码器为LZMA；当类型为‘1’时，编码器为区间编码器；当内容为2时，编码器为基于BSC的最大匹配编码器。 | eb |
| **fa\_seq\_encoder\_ver** | ID为2，元素内容为碱基序列的编码器版本，元素内容类型为无符号整型 | eb |
| **fa\_seq\_data\_id** | 内容为3，表示接下来内容为碱基序列压缩码流 | vi |

表 43 FASTA碱基序列码流定义（续）

|  |  |  |
| --- | --- | --- |
| FASTA碱基压缩码流定义 | 说明 | 描述符 |
| **fa\_seq\_data\_len** | 碱基序列压缩码流长度 | vi |
| if(ebml\_value(fa\_seq\_encoder)==0) |  |  |
| **fa\_seq\_data\_lzma** | LZMA编码器编码码流 | lz |
| if(ebml\_value(fa\_seq\_encoder)==1 and ebml\_value(fa\_seq\_encoder\_ver)==1) |  |  |
| **fa\_seq\_data\_rc\_num** | 记录fa\_seq\_data\_rc的字符数 | u(64) |
| **fa\_seq\_data\_rc** | k阶区间编码器编码码流，阶数由rc\_order确定 | rc(v,5) |
| if(ebml\_value(fa\_seq\_encoder)==2) |  |  |
| **fa\_seq\_data\_mm** | 最大匹配编码码流 | bs |
| } | - | - |

* 1. 解码过程
     1. 概况

测序数据解码依次解码头部流码（见8.2）、尾部流码（见8.3）和数据块参数流码（见8.4）。

* + 1. 头部码流解码

头部码流解码过程如下：

1. 按主体码流表 7定义中读取头部魔数header\_magic\_num，判断是否为“avsg”，若否则退出；
2. 按表 6中vi和eb定义依次解析表 8、表 9和表 10中码流获得基础信息和压缩信息。
   * 1. 尾部码流解码

尾部码流解码过程如下：

a) 依次解析表 7主体码流中header\_len和comp\_data\_len获得文件头和压缩数据码流长度，跳转到尾部码流；

b) 按表 6中vi和eb定义依次解析表 20、表 21和表 22获得数据块信息、编码器参数和参考序列信息，供后续其他部分解码使用。

c) 确认主体码流表 7中魔数tailer\_magic\_num内容为“avsg”，即文件完好。

* + 1. FASTQ压缩数据码流解码
       1. 概述

码流基础信息中表 9中若标准类型std\_type为’fq’，则表示编码为FASTQ数据，根据尾部码流中的各数据块偏移量等信息，计算得知各个数据块的起始偏移量，逐个跳转到各个数据块的起始偏移量，基于尾部码流的解析结果按照表 6定义逐个进行解码。单个数据块内部主要包括块基础信息以及数据压缩码流，数据压缩码流包括元数据、碱基序列、质量分数等信息。

* + - 1. 块基础信息数据解码

块基础信息解码过程如下：

a) 按照表 6定义的vi解析表 12中得到数据块长度；

b) 按照表 6定义的eb解码方法依次解析表 13获得总测序序列数、块文本校验码等信息，按照数据长度的数值读取相应的字节数解码出数据。

* + - 1. 序列长度解码

按表 12定义先解析block\_metadata\_check和block\_metadata\_len得到序列长度校验码（可选）和长度压缩码流数据。

对于序列长度压缩码流，按照vi解析block\_seqlen\_id和block\_seqlen\_len得到ID和数据长度，再按照表 15定义依次读取编码器类型、编码器版本，若编码器类型为0则采用LZMA解码。否则继续解析序列长度压缩码流，对于其内容，按照表 6中src定义解码表 25并重复下列步骤至解码完毕：

a) 从码流中解码出seqlen\_1byte和seqlen\_2byte，并根据数据是否为长读长数据，决定是否继续从码流中解码出seqlen\_3byte和seqlen\_4byte各一个元素，由此拼接得到首个序列长度；

b) 从码流中解码出seqlen\_same，若当前seqlen\_same为0，则当前序列长度与前一序列长度相同，若当前seqlen\_same为1，则当前序列长度与前一序列长度不同，则从码流中解码出seqlen\_1byte和seqlen\_2byte，并根据数据是否为长读长数据，决定是否继续从码流中解码出seqlen\_3byte和seqlen\_4byte各一个元素，拼接得到当前序列长度；

c) 重复步骤b）直到码流解码完毕，即得到所有序列长度；

d) 若码流内包含序列长度校验码，则对解压结果计算校验码，并与其比对，若一致则继续解压，否则报错。

* + - 1. 元数据解码
         1. 概述

先按照表 12解析block\_metadata\_check和block\_metadata\_len等到元数据校验码（可选）和元数据长度。根据长度解析元数据压缩码流block\_metadata\_stream()，按照表 14解析读取编码器类型、编码器版本；若编码器类型为0则采用LZMA解码；否则按照表 23解析metadata\_data\_stream()。若编码器类型为1，使用了基于窗口划分的编码方法得到的，则继续解析元数据压缩码流，对于其内容，按照8.4.4.2的步骤解码并重复至解码完毕；若编码器类型为2，说明数据是使用了MSB嵌入和流道上下文方法压缩得到的，则继续解析元数据码流，对于其内容，按照8.4.4.3的步骤解码并重复至解码完毕。

* + - * 1. 基于窗口划分的解码

元数据的头文件解码

从表 23中解析vec\_format得到各个元数据块的序列数目与首元数据长度；

组分解码

概述

从元数据码流表 24解析得到当前元数据块的首个元数据first\_metadata，并进行窗口划分，得到各个组分情况并分别解码。

纯数字组分解码

对于纯数字组分解码过程如下：

1. 初始化范围为4的解码计数器，对结果码流进行解码，得到当前差值类型diff\_type；
2. 初始化范围为16的解码器，对结果码流解码得到当前diff\_scale\_n\_len；
3. 初始化范围为16的解码器，对结果码流解码得到各个diff\_scale\_n\_num，并组合得到原始差异值，再结合diff\_type，得到原始数值;
4. 重复以上过程直到结果码流解码完毕。

字母及数字组分解码

字母及数字组分解码过程如下：

1. 初始化范围为2的解码器，对结果码流解码出当前char\_flag；
2. 若当前char\_flag结果为2或3，对结果码流解析出当前len\_diff；
3. 结合char\_flag和len\_diff，获知当前组分长度，并用范围为64的解码器，对结果码流进行解码，直到元素数量等于当前组分长度，解码结束，得到当前组分的字符串；
4. 重复以上过程直至结果码流解码完毕；

组分汇总

组分汇总时应先对各个组分顺序进行解码，其中组分为纯数字时应按照8.4.4.2.2.2　解码，组分为字母及数字时应按照8.4.4.2.2.3　解码，各个组分都完成各自的解码后，按照下面步骤汇总：

1. 根据首个元数据码流 first\_metadata得到当前同组分元数据码流的首个元数据；
2. 对首个元数据，以符号为分隔符进行拆分，获知组分的数量以及各个组分的属性信息，不同组分码流处理步骤应符合附录B的规定。
3. 各组分的结果拼接后即为各个原始的元数据。
4. 若码流内包含元数据校验码，则对解压结果计算校验码，并与其比对，若一致则继续解压，否则报错。
   * + - 1. 基于最高位平面嵌入和流道列上下文的解码

元数据的头文件解码

从表23中解析vec\_format得到各个元数据块的序列数目与首元数据长度；

组分解码

概述

从元数据码流表24解析得到当前元数据块的首个元数据first\_metadata，并进行窗口划分，得到各个组分情况并分别解码。

非流道列和X，Y列的数字组分解码

对于非流道列和X，Y列的数字组分解码过程与8.4.4.2.2.2　相同。

字母及数字组分解码

字母及数字组分解码过程与8.4.4.2.2.3　相同。

流道列和X，Y列组分解码

流道列和X,Y列组分解码过程如下：

1. 初始化范围为96的解码器，对结果码流解码出当前流道列tile的各位组装成流道；
2. 初始化范围为16的解码器，解码next\_msb出指示数字最高位平面比特；
3. 结合next\_msb和tile，由next\_msb获知当前组分长度，由tile获知当前码流所属的组，并用范围为16的解码器，对结果码流进行解码，直到数字元素数量等于当前组分长度next\_msb，解码结束，得到当前组分的字符串；
4. 重复以上过程直至结果码流解码完毕；

组分汇总

组分汇总时先对各个组分顺序进行解码，其中组分为纯数字时应按照8.4.4.2.2.2　解码，组分为字母及数字时应按照8.4.4.2.2.3　解码，组分为流道列和X，Y列组分时应按照8.4.4.3.2.4　解码；各个组分都完成各自的解码后，按照8.4.4.2.2.4　汇总。

* + - * 1. 校验

若码流包含元数据校验码，则对解压结果计算校验码，并与其比对，若一致则继续解压，否则报错。

* + - 1. 碱基序列解码
         1. 概述

先解析表 12中block\_seq\_check和block\_seq\_stream()获得碱基序列校验码（可选）和碱基序列压缩码流数据。按照vi解析block\_seq\_id和block\_seq\_len得到ID和数据长度，再按照数据长度数值读取表 16中编码器类型seq\_encoder和版本seq\_encoder\_ver。若编码器类型为0，直接采用LZMA对码流seq\_data\_lzma进行解码得到碱基序列。若编码器类型为1或2，说明数据是容错配的序列比对和编码（含简并碱基编码）得到，按照8.4.5.2　解码；若编码器类型为3或4，说明数据是经AMGC优化的容错配编码（不含简并碱基编码）得到，则在8.4.5.2　解码基础上比对位置按照8.4.5.2.5　解码；若编码器类型为5，数据采用自组装参考压缩编码，则采用8.4.5.3　进行解码。

* + - * 1. 容错配的序列比对和编码（含简并碱基编码）的解码

简并碱基解码

简并碱基解码过程如下：

1. 按照表 27定义解码得到存在标识符ambi\_base\_flag、简并碱基字符ambi\_base\_char、简并碱基最大质量分数ambi\_max\_qual、低质量碱基中非简并碱基个数 ambi\_lowqual\_normalcount\_stream()、非简并碱基间的简并碱基数量 ambi\_lowqual\_normalintv\_stream()这五个码流；
2. 分别初始化范围为2、11、95的区间解码器，分别解码前三个码流，得到各自原始数组；
3. 对于ambi\_lowqual\_normalcount\_stream()，先按照表 28定义解码1个无符号整数得到ambi\_normal\_num\_n，然后初始化范围为ambi\_normal\_num\_n的解码器，解码出元素，直到元素个数等于ambi\_base\_flag中为“1”的个数，解码完毕，得到ambi\_normal\_num\_n\_len；
4. 设当前位数为0
5. 初始化范围为2的解码器，对ambi\_normal\_num\_n\_num进行解码，直到元素数量等于ambi\_normal\_num\_n\_len在当前位数上有数字的个数，解码结束，得到当前位数上的ambi\_normal\_num\_n\_num原始数组，然后当前位数+1，重复当前步骤，直到ambi\_normal\_num\_n个数组都解码完毕；
6. 对于ambi\_lowqual\_normalintv\_stream()，先按照表 29解码1个无符号整数得到ambi\_intv\_num\_n，然后初始化范围为ambi\_intv\_num\_n的解码器，解码出元素，直到元素个数等于ambi\_base\_flag中为“1”的个数，解码完毕，得到ambi\_intv\_num\_n\_len；
7. 设当前位数为0；
8. 初始化范围为2的解码器，对ambi\_intv\_num\_n\_num进行解码，直到元素数量等于ambi\_intv\_num\_n\_len在当前位数上有数字的个数，解码结束，得到当前位数上的ambi\_intv\_num\_n\_num原始数组，然后当前位数+1，重复当前步骤，直到ambi\_intv\_num\_n个数组都解码完毕；
9. 结合解码结果，对于含简并碱基的各个碱基序列，在获取其质量分数后，将各个简并碱基替换到对应的原始位置上。

序列有参无参压缩顺序解码

按照表 30定义用二值解码器解码align\_flag\_stream得到一个二进制串，表示各个序列是经有参还是无参压缩的；

有参考碱基序列解码

有参考碱基序列解码过程如下：

1. 根据表 22获取参考序列，首先判断ref\_bool是否使用输入性参考序列，若是，则继续判断ref\_encoder字段是否存在，若存在，则根据ref\_id和ref\_name获取外部参考序列，并使用ref\_checksum校验码验证。若ref\_encoder不存在，根据相应解码器解压参考序列数据码流获得参考序列。
2. 按照表 30解码得到比对位置 align\_pos\_plain\_stream或align\_pos\_amgc\_stream()、比对方向 align\_strand\_stream、错配个数 mis\_num\_stream、错配位置 mis\_pos\_stream、错配字符 mis\_base\_stream五个码流；
3. 如果编码器seq\_encoder的id值为1或者2，则初始化范围为2的区间解码器解析align\_pos\_plain\_stream，得到二进制串，根据参考序列的长度，取能够满足2^n大于参考序列长度的最小n，每n个bit为一个align\_pos\_plain\_stream的初始元素的二进制串，还原得到各个align\_pos；否则，采用8.4.5.2.5　中的位置序列解码方法。
4. 初始化范围为2的解码器解码align\_strand\_stream，得到原始数组；
5. 初始化范围为2的解码器解码mis\_num\_stream，得到二进制串，根据参考序列的长度，取能够满足2^n>max\_mis的最小n，每n个bit为一个比对位置的初始元素的二进制串，还原得到各个比对位置；
6. 初始化范围为2的解码器解码mis\_pos\_stream，得到二进制串，设当前错配所可能出现的子串长度为L（若错配为当前序列第一个错配，则L为当前序列总长，否则，L为当前序列总长与上一个错配所在位置的差值），取能够满足2^n>L的最小n，获取n个bit为当前错配位置的二进制串，还原得到初始值；
7. 初始化范围为4的解码器解码mis\_base\_stream，得到四进制串，根据参考序列上的子串在该位置上的碱基，获知原始碱基（若参考序列上碱基为A，则0123对应GCTN；若参考序列上碱基为C，则0123对应GATN；若参考序列上碱基为G，则0123对应CATN；若参考序列上碱基为T，则0123对应GCAN）；
8. 结合解码结果，对于某条碱基序列来说，会访问参考序列的当前比对位置，截取相应子串，根据当前比对方向，决定是否将子串进行反向互补处理，然后若当前错配个数大于0，根据当前各错配位置，将对应位置的碱基根据错配字符替换成实际字符。重复该过程，直到恢复所有碱基序列。

无参考碱基序列解码

对于不使用参考的碱基序列，初始化范围为4的解码器，对表 26中unalignseq\_stream进行解码，得到无参考压缩的碱基序列。

AMGC参考位置序列解码

AMGC碱基序列比对位置的解码过程如下：

1. 按照表 31定义，根据cardinality(align\_flag\_stream)获取比对位置总数ref\_pos\_count，申请字节数为ref\_pos\_count\*4的内存空间表示为32位整形数组refpos[]；
2. 申请类型为布尔型，长度为ref\_pos\_count的数组is\_equal[]，从码流中解码出is\_equal\_to\_previous的值;
3. 申请类型为布尔型，长度为ref\_pos\_count的数组sign[]，从码流中解码出mapped\_pos\_sign的值;
4. 申请类型为整型，长度为ref\_pos\_count的数组msb[]，从码流中解码出most\_signifi\_bits的值;
5. 申请类型为布尔型，长度为ref\_pos\_count\*32的数组bitleft[]，从码流中解码出bits\_left的值;
6. 遍历msb数组，根据msb[i]的值看还需要多少位，于是从bitleft中取出对应数量的比特位，恢复出位置差diff\_pos，将diff\_pos暂时存储在refpos[]中，注意，bitleft中顺序为从左到右取，且每个位仅被使用一次；diff\_pos从右侧开始存储
7. 根据sign[]的值，给每个diff\_pos的值加上正负号；
8. 解码出first\_mapped\_pos，放到refpos[]最左边，然后用指针p\_d指向diff\_pos的第一个值，跟据is\_equal[]，由diff\_pos得到最终的ref\_pos，这里当is\_equal[i]为1，则直接令refpos[i+1] = refpos[i], 否则refpos[i+1] = diff\_pos[p\_d++]，注意，这里refpos和is\_equal的指针有距离为1的错位，这是由于第一个refpos无法向前比对造成的；
   * + - 1. 基于自组装参考碱基序列解码

匹配信息解码

匹配辅助信息解码过程如下：

1. 按照表 32定义，根据reads\_count统计碱基序列数量；
2. 根据 reads\_count解析下个数据块得到碱基序列链方向（1位无符号整数数组）；
3. 根据 reads\_count解析下个数据块得到错配碱基数量（8位无符号整数数组）
4. 获取保留顺序压缩标志 order\_preserving\_flag；
5. 按照表 33解析64位无符号整数得到参考序列长度 reference\_length；
6. 根据 reference\_length 解码下个数据块得到参考序列（8位无符号字符数组）；
7. 按照表 36解析32位无符号整数得到错配碱基数组大小 mismatch\_base\_size；
8. 根据 mismatch\_base\_size 解析下个数据块得到错配碱基（8位无符号字符数组）

错配碱基位置解码

错配碱基位置解码过程如下：

1. 按照表 37解析8位无符号整数得到最大错配数量 max\_mismatch\_count
2. 依次按照下面的步骤迭代 max\_mismatch\_count 次 得到二维数组 mismatch\_pos ：对于第i次迭代，获取32位无符号整数得到错配碱基位置数组大小 mismatch\_pos\_size[i]；根据 mismatch\_pos\_size[i]获取下个数据块得到错配碱基位置数组 mismatch\_pos[i] (16位无符号整数数组)

碱基序列位置解析

碱基序列位置解码过程如下：

1. 如果 order\_preserving\_flag 为 1，根据 reads\_count 解析下个数据块得到保留序列顺序模式下（按FASTQ文件中的先后顺序）的碱基序列位置数组 order\_preserving\_reads\_pos（64位无符号整数数组）
2. 否则，根据 reads\_count 解析下个数据块得到重排序列模式下的碱基序列位置偏移 reads\_offset（16位无符号整数数组），然后通过8.4.5.3.4　逆向差分编码还原碱基序列在参考序列的位置数组 reads\_position。

逆向差分编码还原

逆向差分编码还原的过程如下：

1. 设第一个序列位置reads\_position[0]为偏移量reads\_offset[0];
2. 设后续序列位置reads\_postion[i]为其前一个序列位置reads\_position[i-1]加上相应的偏移量reads\_offset[i]；
3. 重复b)直至所有序列位置被还原。

碱基序列解码

碱基序列的解码中初始化碱基序列数组seq，数组大小为 reads\_count，初始化错配碱基数组下标 mismatch\_base\_idx 为 0；初始化错配碱基位置数组下标 mismatch\_pos\_idx 为全0数组，数组大小为max\_mismatch\_count + 1。

按照下面的计算过程迭代 解析reads\_count次得到每一条碱基序列字符串，第i轮迭代的过程如下：

1. 根据碱基序列在参考序列的位置以及碱基序列的长度在参考序列截取相应的子串作为seq[i]的初始值
2. 碱基序列的错配碱基个数mis\_cnt为mismatch\_count[i]。如果mis\_cnt大于0，需要还原碱基序列上发生错配的碱基，还原过程应符合附录C的规定。
   * + - 1. 校验

若码流内包含碱基序列校验码，则对解压结果计算校验码，并与其比对，若一致则继续解压，否则报错。

* + - 1. 质量分数解码
         1. 概述

按照表 12定义解析block\_qual\_check和block\_qual\_stream()获得质量分数校验码（可选）和质量分数压缩码流。对于质量分数压缩码流，按照表 17定义解析qual\_encoder获取编码器类型和版本，若编码器类型为0，采用LZMA对质量分数码流进行解码；若编码器类型为1，说明数据是使用了基于ACO编码器，按照下列步骤解码并重复至解码完毕。

* + - * 1. 质量分数码流解码

质量分数分块解码

质量分数分块的解码过程如下：

1. 按照表 40定义，获取质量分数总数quality\_count，申请字节数为quality\_count的内存空间表示为32位整型数组qs[];
2. 从i=0开始解码第i个质量分数：按照8.4.6.2.2　构建的混合上下文模型和8.3　获得的解码器解码获得第i个质量分数；设置qs[i]为当前解码质量分数；
3. i=i+1，重复上述过程直至（i==quality\_count）；
4. 遍历顺序traverse\_order截断qs[]输出质量分数序列：如果块内遍历顺序traverse\_order为1，根据表 12中block\_seqlen\_id字段定义的序列长度按顺序依次截断qs[]并添加换行符；如果块内遍历顺序traverse\_order为0，申请reads\_number个内存空间用于存储相应的质量分数序列，空间大小根据中block\_seqlen\_id字段定义的序列长度确定，遍历qs[]按7.1.6　定义的列扫描顺序依次填入相应质量分数序列空间并加上换行符。

混合上下文模型构建

混合上下文模型构建过程如下：

1. 按照表 40定义获取碱基标识符base\_flag值，如果其值为1获取与当前质量分数字符对应的前两个碱基字符base\_value[i]和base\_value[i-1]；
2. 获取均值标识符mean\_value\_flag的值，如果其值为1，根据7.2.5.1　定义获取当前质量分数字符所在行的质量分数均值mean\_value[i]；
3. 获取当前质量分数分块字符最大值max\_value和最小值min\_value，确定模型范围；
4. 从qs[]中获取质量分数字符的前四个字符q1=qs[i-1]，q2= qs[i-2]，q3= qs[i-3]和q4 =qs[i-4]；
5. 建立熵编码器上下文模型，模型建立规则应符合附录D的规定。
   * + - 1. 校验

若码流内包含质量分数校验码，则对解压结果计算校验码，并与其比对，若一致则继续解压，否则报错。

* + 1. FASTA压缩数据码流解码

8.5.1 FASTQ压缩数据码流解码过程

FASTQ压缩数据码流解码过程如下：

a) 根据头部码流的标准类型为fa，跳转到FASTA压缩数据码流进行解码；

b) FASTA压缩数据码流主要包括元数据以及碱基序列压缩码流，在FASTA压缩数据码流中逐个解析EBML元素即可获得各个部分。

c) 对于元数据码流，解析其中的metadata\_encoder，若结果为0，则用lzma解码元数据压缩码流；

d) 对于碱基序列码流，解析seq\_encoder的内容，若为0，采用LZMA对fa\_data\_lzma进行解码获得碱基数据，若为1，采用区间编码解码器对fa\_data\_rc进行解码，若为2，使用BSC编码解码器对fa\_data\_mm进行解码获得基于最大匹配的有参考编码字符串，根据8.4.4.2.3　中参考序列解码方法得到参考序列。利用参考序列和7.1.5.3.2　中编码定义解析获得FASTA碱基序列。

* + 1. 校验

若码流内包含元数据和/或碱基序列校验码，则对解压结果计算校验码，并与其比对，若一致则继续解压，否则报错。

2. （规范性）  
   简并碱基规则

简并碱基按照国际纯化学和应用化学联合会IUPAC（International Union of Pure and Applied Chemistry）规则见表A.1。

* 1. 简并碱基对照表

| 简并碱基 | 对应碱基 |
| --- | --- |
| A | A |
| C | C |
| G | G |
| T | T |
| R | A或G |
| Y | C或T |
| S | G或T |
| W | A或T |
| K | G或T |
| M | A或C |
| B | C或G或T |
| D | A或G或T |
| H | A或C或T |
| V | A或C或G |
| N | 任何碱基 |
| .或- | 空格 |

1. （规范性）  
   不同组分元数据码流处理步骤
   1. 纯数字组分处理步骤：

获取首个元数据在当前组分的数字，记录为last\_num；

获取diff\_type在当前下标的数字。如果为0，则当前数字等于last\_num；如果为1，则当前数字等于last\_num+1；这两种情况则进入步骤B.1.4。如果为2，则当前数字大于last\_num且差值大于1；如果为3，则当前数字小于last\_num；这两种情况进入步骤B.1.3。

获取diff\_scale\_n\_len在当前下标的数字，然后从diff\_scale\_n\_num中当前下标往后获取相应位数的数字，即为当前的差值绝对值。如果diff\_type的当前下标数字为2，则当前数字等于last\_num与差值绝对值的数字之和，否则，当前数字等于last\_num与差值绝对值之差。

如果首个元数据在该组分的数字存在前缀0的情况，则初始化范围为10的解码器解码得知当前的前缀0个数，加到当前数字前作为前缀。当前数字作为这一轮的结果输出。

如果该组分码流已处理完毕，则结束；否则，将当前数字赋值给last\_num，回到B.1.2。

* 1. 字母及数字组分处理步骤：

记该组分长度为L，执行L次迭代得到各个字符串结果，对于第i次迭代，依次执行下面的步骤：

1）获取首个元数据在当前组分的第i个字符，记为last\_char；

2）获取diff\_char的当前字符，若为0，则当前结果字符为last\_char，否则获取diff\_char的当前字符，作为当前结果字符，并将当前结果字符赋值给last\_char。当前结果字符为这一轮结果输出，重复本步骤直到处理完diff\_char和diff\_char码流。

1. （规范性）  
   错配碱基还原过程

将seq[i]上位置为 mismatch\_pos[mis\_cnt][mismatch\_pos\_idx[mis\_cnt] + j] 的碱基替换为 mismatch\_base[mismatch\_base\_idx + j]，其中 j 代表第 j 轮碱基替换，总共需要迭代 mis\_cnt 次

更新下标 mismatch\_pos\_idx[mis\_cnt] += mis\_cnt，mis\_base\_idx += mis\_cnt

如果碱基序列的比对方向 reads\_direction[i] 等于1，计算碱基序列seq[i]的反向互补序列

（规范性）  
熵编码器上下文模型建立规则

如果base\_flag的值为1且mean\_value\_flag为1，则根据q1、q2、q3、q4、base\_value[i]、base\_value[i-1]和mean\_value，对熵编码上下文模型统一量化建模。

如果base\_flag的值为1且mean\_value\_flag为0，则根据q1、q2、q3、q4、base\_value[i]、和base\_value[i-1]，对熵编码上下文模型统一量化建模。

如果base\_flag的值为0且mean\_value\_flag为1，则根据q1、q2、q3、q4和mean\_value，对熵编码上下文模型统一量化建模。

如果base\_flag的值为0且mean\_value\_flag为0，则根据q1、q2、q3和q4对熵编码上下文模型统一量化建模。