ICS 35.110
L 79

中华人民共和国国家标准

GB/T 16263.2—xxxx/ISO/IEC 8825-2:2021

信息技术　ASN.1 编码规则 第 2 部分:紧缩编码规则 (PER) 规范

Information techbology—ASN.1 encoding rules—
Part 2: Specification of Packed Encoding Rules(PER)

（ISO/IEC 8825-2:2021,IDT）

（征求意见稿）

（在提交反馈意见时，请将您知道的相关专利连同支持性文件一并附上）

XXXX - XX- XX 发布　　XXXX - XX - XX 实施

国家市场监督管理总局
国家标准化管理委员会　发布

目　　次

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GB/T 16263《信息技术　ASN.1 编码规则》的第2部分，GB/T 16263已经发布了以下部分：

——第1部分：基本编码规则（BER）、正则编码规则（CER）和非典型编码规则（DER）规范；

——第2部分：紧缩编码规则(PER)规范；

——第4部分：XML编码规则（XER）；

——第5部分：W3C XML模式定义到ASN.1的映射。

本文件代替GB/T 16263.2-2006《信息技术　ASN.1 编码规则》，与GB/T 16263.2-2006相比，除了结构调整和编辑性改动外，主要技术变化如下：

——增加了紧缩编码规则的应用范围（见 1）；

——更改了规范性引用文件的内容（见 2，2006 年版的 2）；

——增加了 PER 编码指令的定义（见 3.6）；

——修改了非负 2 进制整数编码的内容（见 3.7.20，2006 年版的 3.6.20）；

——增加了 PER 编码指令（见 9）；

——修改了编码过程中完整编码的产生方式（见 11.1，2006 年版的 10.1）；

——修改了位串类型的编码规则（见 16.6，2006 年版的 15.6）；

——修改了八位位串类型的编码规则（见 17.3，2006 年版的 16.3）；

——增加了国际化资源引用类型的编码（见 26）；

——增加了相关国际化资源引用类型的编码（见 27）；

——修改了受限字符串类型的编码规则（见 30.4、30.6.1、30.6.3，2006 年版的 27.4、27.6.1、27.6.3）；

——增加了时间类型、有效时间类型、已定义时间类型和附加时间类型的编码(见 32)；

——修改了传送语法的客体标识符的编码规则的表述（见 33.2，2006 年版的 29）；

——修改了附录 F 的标题和正文的表述（见附录，2006 版附录 F）。

本文件等同采用国际标准ISO/IEC 8825-2：2021《信息技术　ASN.1 编码规则 第2部分：紧缩编码规则(PER)规范》。

本文件由全国信息技术标准化技术委员会(SAC/TC 28)提出并归口。

本文件起草单位：

本文件主要起草人：

本文件及其所代替文件的历次版本发布情况为：

——1996 年首次发布为 GB/T 16263-1996，2006 年第一次修订；

——本次为第二次修订。

引　言

GB/T 16262.1、GB/T 16262.2、GB/T 16262.3和GB/T 16262.4共同描述了抽象语法记法一(ASN.1)，这种记法就对等应用之间交换的报文进行了定义。

本文件定义的编码规则适用于使用GB/ T 16262.1规定的记法所确定的类型值。应用这些编码规则产生对这些值的传送语法。这些编码规则规范也隐含适用于解码。

ASN.1的类型值应用的编码规则可以有多种集合，本文件定义了紧缩编码规则(PER)。之所以称为紧缩编码规则集合，是因为这种编码规则获得了比GB/T 16263.1中描述的基本编码规则(BER)及它派生的编码规则更紧凑的表示，GB/T 16263.1是紧缩编码规则规范的某些部分的参考。

GB/T 16263拟由四个部分构成。

——第1部分：基本编码规则（BER）、正则编码规则（CER）和非典型编码规则（DER）规范。

——第2部分：紧缩编码规则（PER）规范。

——第4部分：XML编码规则（XER）。

——第5部分：W3C XML模式定义到ASN.1的映射。

信息技术　ASN.1 编码规则 第 2 部分：紧缩编码规则(PER)规范

1　范围

本文件规定了紧缩编码规则集合，它们可以用来为ISO/IEC 8824-1中定义的类型值派生传送语法。这些紧缩编码规则也适用于解码这样的传送语法，以便标识出被传送的数据值。

本文件规定的编码规则：

——在通信时使用；

——在选择编码规则时主要关心值的最小化表示规模的场合中使用；

——对在 GB/T 16263.1 中描述的所有扩展形式，在保留现有值的编码时，允许通过增加额外值进行抽象语法的扩展；

——可以根据 ISO/IEC 8825-6 进行修改。

2　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。

注：本文件基于GB/T 13000-2010。不能在该标准的更高版本上使用。上述引用应解释为对已确定的标准及其所有已发布的修订和技术勘误的引用。

GB/T 2311—2000　信息技术　字符代码结构和扩充技术(ISO/IEC 2022:1994,IDT)

GB 13000—2010　信息技术 通用多八位编码字符集(UCS)(ISO/IEC 10646:2003,IDT)

GB/T 16262.1—AAAA　信息技术 抽象语法记法一(ASN.1) 第1部分:基本记法规范（ISO/IEC 8824-1:2021,IDT）

GB/T 16262.2—BBBB　信息技术 抽象语法记法一（ASN.1）第2部分：信息客体规范（ISO/IEC 8824-2:2021,IDT）

GB/T 16262.3—CCCC　信息技术 抽象语法记法一(ASN.1) 第3部分:约束规范（ISO/IEC 8824-3:2021,IDT）

GB/T 16262.4—DDDD　信息技术 抽象语法记法一(ASN.1) 第4部分:ASN.1规范的参数化（ISO/IEC 8824-4:2021,IDT）

GB/T 16263.1—EEEE　信息技术　ASN.1编码规则　第1部分：基本编码规则(BER)、正则编码规则(CER)和非典型编码规则(DER)的规范（ISO/IEC 8825-1:2021,IDT）

SJ/Z 9047—1987　信息处理信息交换用字符串形式表示数值的方法(ISO 6093:1985,IDT)

ISO/IEC 646:1991　信息技术 信息交换用ISO七位编码字符集(Information technology - ISO 7-bit coded character set for information interchange)

ISO/IEC 2375:2003　信息技术 转义序列和编码字符集的登记规程(Information technology - Procedure for registration of escape sequences and coded character sets)

ISO/IEC 8825-6:2021　信息技术 ASN.1编码规则 第6部分：PER编码指令的注册和应用(Information technology — ASN.1 encoding rules:Registration and application of PER encoding instructions)

XX
ISO 结合转义序列使用的编码字符集的国际登记簿

3 术语和定义

下列术语和定义适用于本文件。

3.1 基本记法规范

GB/T 16262.1界定的所有术语和定义适用于本文件。

3.2 信息客体规范

GB/T 16262.2界定的所有术语和定义适用于本文件。

3.3 约束规范

GB/T 16262.3界定的下列术语和定义适用于本文件：
a) 组件关系约束(component relation constraint)；
b) 表约束(table constraint)。

3.4 ASN.1 规范的参数化

GB/T 16262.4界定的下列术语和定义适用于本文件：
可变约束(variable constraint)

3.5 基本编码规则

GB/T 16263.1界定的下列术语和定义适用于本文件：
a) 数据值 (data value)；
b) 动态符合性 (dynamic conformance)；
c) (数据值的）编码(encoding(of a data value))；
d) 接收器 (receiver)；
e) 发送器 (sender)；
f) 静态符合性 (static conformance)。

3.6 PER 编码指令

本文件使用ISO/IEC 8825-6 中定义的下列术语和定义：
标识关键字(identifying keyword)。

3.7 附加定义

下列术语和定义适用于本文件。

3.7.1

2的补码的二进制整数编码 2's-complement-binary-integer encoding

把整个数编码到指定长度的位字段（在ALIGNED变体中八位位组对齐）中，或者编码到可接纳该整个数编码为2的补码的整数的最小数目的八位位组中，该编码按照11.4所规定的等于、大于或小于0的整个数提供表示。

注1：2的补码的二进制数的值通过计数内容八位位组中的位得出，计数从最后1个八位位组的位1开始作为位0,到

第1个八位位组的位8结束。每位赋予一个2^N的数值，这里N是该位在上面计数顺序中的位置。2的补码的二进制数的值利用下列方法得出，即累加计算那些置1的位的数值，但不包括第1个八位位组的位8，然后，若第1个八位位组的位8置为1，则这个累加值减去赋给位8的数值。

注2：整个数(whole number)是数学术语整数(integer)的同义词。这里使用它是为了避免与ASN.1的类型整数(integer)混淆。

3.7.2

抽象语法值 abstract syntax value

要由PER编码或由PER解码生成的抽象语法值（定义为单个ASN.1类型值的集合）。

注：与抽象语法相关联的单个ASN.1类型由类“ABSTRACT-SYNTAX”的客体正式的标识。

3.7.3

位字段 bit-field

编码机制某一部分的产物，由有序的位集合构成，这些位不必是8的倍数。

注：若该术语的使用后面紧跟“ALIGNED变体中八位位组对齐”，这意味着对PER的对齐变体来说，该位字段要求完整的编码在八位位组边界上开始。

3.7.4

正则编码 canonical encoding

通过应用无实现相关选项的编码规则所得到的抽象语法值的完整编码，这种规则导致在传输语法中无歧义且唯一的位串和抽象语法中的值之间一对一映射。

3.7.5

复合类型 composite type

集合、序列、单一集合、单一序列、选择、嵌入式pdv、外部或者不受限的字符串类型。

3.7.6

复合值 composite value

复合类型的值。

3.7.7

受约束的整个数 constrained whole number

受PER可视约束所约束的整个数处于“lb”到“ub”的范围内，“lb”的值小于或等于“ub”的值，且“lb”和“ub”的值是允许的值。

注：受约束的整个数出现在编码中，它标识以下内容：选定的选择类型的替换项；长度已经被PER可视约束限制到最大长度的字符、八位位组和位串类型的长度；单一序列或单一集合类型中的组件数计数已经被PER可视约束限制为最大组件数的计数；已经被PER可视约束所约束的处于有限最小值和最大值范围内的整数类型值；以及表示枚举类型中的枚举值。

3.7.8

有效长度约束（受约束字符串类型的） effective size constraint(for a constrained string type)

单个有限长度约束，它可适用于内置串类型，并且其效果是允许且只允许全部能被受约束的串类型表示的那些长度。

注1：例如，下列串类型有一个有效长度约束：

A ::= IA5String(SIZE(1..4) | SIZE(10..15))

因为它可以用适用于所有值的单个长度约束改写：

A ::= IA5String(SIZE(1..4) | 10..15))

而下面的串类型没有有效长度约束，因为如果该串不包含除“a”、“b”和“c”外的任何字符，则它可以任意长：

B ::= IA5String(SIZE(1..4) | FROM("abc"))

注2：有效长度约束只能用来确定长度的编码。

3.7.9

有效允许字母约束（受约束的受限字符串类型的） effective permitted-alphabet constraint (for a constrained restricted character string type)

单个允许字母表约束，它可以适用于内置已知倍数字符串类型，并且其效果是允许且只允许那些能够出现在某个受约束的受限字符串类型的值中至少一个字符位置上的字符。

注1：例如，在：

Ax ::= IA5String(FROM("AB") | FROM("CD"))

Bx ::= IA5String(SIZE(1..4) | FROM("abc"))

Ax有一个有效允许字母表约束为“ABCD”，Bx有一个由整个IA5String字母表构成的有效允许字母表约束，因为没有更小的允许字母表约束能够适用于Bx的所有值。

注2：有效允许字母表约束只用来确定字符的编码。

3.7.10

枚举索引 enumeration index

与枚举类型中的“EnumerationItem”相关联的非负整个数。枚举索引通过把“EnumerationItem”按照枚举值升序的方式进行排序来确定，枚举索引从0开始赋值，0赋值给第1个“EnumerationItem”，1赋值给第2个，依此类推，直到排序表中的最后一个“EnumerationItem”。

注：“RootEnumeration”中的“EnumerationItem”与“AdditionalEnumeration”中的“EnumerationItem”分开来排序。

3.7.11

PER编码的可扩展 extensible for PER encoding

一种类型特性，需要PER把值的编码标识为根值的编码或扩展附加部分的编码。

注：根值编码一般比扩展附加部分编码的效率更高。

3.7.12

字段列表 field-list

位字段的有序集合，它是把这些编码规则应用于抽象值产生的结果。

3.7.13

不定长度 indefinite-length

其长度大于64K-1或者其最大长度不能由ASN.1记法确定的一种编码。

3.7.14

固定长度类型 fixed-length type

一种类型，该类型编码的最外层长度决定因子的值可以由类型记法（仅在应用PER可视约束之后）来确定（使用本文件规定的机制），并且对于该类型的所有可能值而言，该值相同。

3.7.15

固定值 fixed value

一个值，它可以确定（用本文件规定的机制）支配它的类型的一个唯一允许值（仅在应用PER可视约束之后）。

3.7.16

已知倍数字符串类型 known-multiplier character string type

一种受限字符串类型，其中，对所有允许字符串值，编码的八位位组数是字符串中的字符数的已知固定倍数。已知倍数字符串类型是IA5String、PrintableString、VisibleString、NumericString、UniversalString和BMPString。

3.7.17

长度决定因子 length determinant

确定全部或者部分PER编码长度的（位、八位位组、字符或组件的）计数。

3.7.18

正常小非负整个数 normally small non-negative whole number

编码的一部分，它表示一个无边界非负整数的若干值，但是，小值比大值出现的可能性大。

3.7.19

正常小长度 normally small length

长度编码，它表示一个无边界长度的若干值，但是，小长度比大长度出现的可能性大。

3.7.20

非负二进制整数编码 non-negative-binary-integer-encoding

受约束或半受约束的整个数的编码，该整个数可编码到指定长度的位字段中、或编码到指定长度的位字段(ALIGNED变体中的八位位组对齐）中、或者编码到最小数目的八位位组中，该八位位组将容纳非负二进制整数的整个数，而非负二进制整数为11.3规定的大于或等于0的整个数提供了表示。

注：非负二进制数的值通过计数内容八位位组中的位得出，计数从最后一个八位位组的位1开始作为位0，到第1个八位位组的位8结束。每位赋予一个2^N的数值，这里N是该位在上面计数顺序中的位置。非负二进制数的值为置1的位所赋予数值的和。

3.7.21

最外层类型 outermost type

一种ASN.1类型，其编码被包含在一个非ASN.1载体中或作为其他ASN.1结构（见11.1.1)的值。

注：最外层类型的PER编码总是八位的整数倍。

XX
3.7.22

PER可视约束 PER-visible constraint

影响值的PER编码的ASN.1约束记法的使用实例。

3.7.23

中继安全编码 relay-safe encoding

抽象语法值的一种完整编码，该编码无需知道编码执行环境就可以被解码（包括任何嵌入式编码）。

3.7.24

半受约束的整个数 semi-constrained whole number

一种受PER可视约束所约束的整个数，以值“1b”作为允许值，约束超过或等于某个值“1b”，它是不受约束的整个数。

注：半受约束的整个数出现在不受约束的（和某些受约束的情况下的）字符、八位位组和位串类型的长度编码中，还出现在不受约束的（和某些受约束的情况下的）单一序列和单一集合类型的组件计数的编码中，以及已经约束为超过某个最小值的整数类型值的编码中。

3.7.25

简单类型 simple type

不是复合类型的类型。

3.7.26

文本依赖 textually dependent

用来标识下列情况的一个术语，如果在计算一个元素集合时使用了某个引用名，则该元素集合的值被认为依赖于该引用名，而无论执行的实际集合算术运算结果是元素集合的最终值与赋值给该引用名的实际元素集合值是否有关。

注：例如，下面的Foo的定义文本依赖于Bar，即使Bar对Foo的值集没有影响（因此，根据10.3.6，在Foo上的约束不是PER可视的，因为Bar被一个表约束来约束的，而Foo在文本上依赖于Bar)。

```
MY-CLASS :: = CLASS {&name PrintableString, &age INTEGER} WITH SYNTAX {&name, &age}
MyObjectSet MY-CLASS ::= {{"Jack",7} | {"Jill",5}}
Bar ::=MY-CLASS.&age ({MyObjectSet})
Foo ::=INTEGER (Bar | 1..100)
```

3.7.27

不受约束的整个数 unconstrained whole number

一种不受PER可视约束所约束的整个数。

注：不受约束的整个数仅出现在整数类型值的编码中。

4 缩略语

下列缩略语适用于本文件：

ASN.1 抽象语法记法一

BER	ASN.1的基本编码规则
CER	ASN.1的正则编码规则
DER	ASN.1的非典型编码规则
PER	ASN.1的紧缩编码规则
16K	16384
32K	32768
48K	49152
64K	65536

5 记法

本文件引用GB/T 16262.1定义的记法。

6 约定

6.1 本文件使用术语“最高有效位”和“最低有效位”来定义编码中的每个八位位组的值。

注：低层规范使用相同的记法来定义串行线上位传输的次序，或者将若干位分配给并行信道。

6.2 对于本文件，八位位组的位从8～1编号，其中位8是“最高有效位”，位1是“最低有效位”。

6.3 本文件中频繁使用术语“八位位组”来代表“八个位”。使用该术语代替“八个位”不带有任何对齐的含义，若打算对齐，应在本文件中明确说明。

7 本文件定义的编码规则

7.1 本文件规定了4种编码规则（以及相关联的客体标识符），它们可以用来编码和解码定义为单个（已知）ASN.1类型值的抽象语法值。本章描述其适用性和特性。

7.2 在不知道被编码值的类型的情况下，就不可能确定编码（在任何PER编码规则算法下）的结构。特别是，在不知道被编码类型的情况下，就不能从编码本身确定编码的结束。

7.3 如果类型EXTERNAL、EMBEDDED PDV和CHARACTER STRING的抽象值被约束防止携带OSI表示上下文的标识符，则PER编码总是中继安全的。

7.4 本文件规定的最一般的编码规则算法是BASIC-PER，它通常不产生正则编码。

7.5 本文件规定的第2个编码规则算法是CANONICAL-PER，它产生正则编码，这被定义为对BASIC-PER编码中依赖于实现选择的限制。

注1：当鉴别器需要应用于抽象值时，则CANONICAL-PER产生具有若干应用的正则编码。

注2：任何与CANONICAL-PER编码一致的实现都与BASIC-PER编码一致。任何与BASIC-PER解码一致的实现都与CANONICAL-PER解码一致。因此，按照CANONICAL-PER形成的编码也是BASIC-PER所允许的编码。

7.6 如果用BASIC-PER或CANONICAL-PER所编码的类型含有EMBEDDED PDV、CHARACTER STRING或者EXTERNAL类型，则外层编码不再是中继安全的，除非所有EMBEDDED PDV、CHARACTER STRING和EXTERNAL类型所使用的传送语法都是中继安全的。如果用CANONICAL-PER所编码的类型含有EMBEDDED PDV、CHARACTER STRING或EXTERNAL类型，则外层编码不再是正则的，除非所有EMBEDDED PDV、CHARACTER STRING和EXTERNAL类型所使用的传送语法都是正则的。

注：支持{GB 13000的1级(1)...}形式的所有字符抽象语法的字符传送语法是正则的，支持{GB13000的2级(2)...}

和{GB 13000的3级(3)...}形式的字符传送语法不总是正则的，所有上述字符传送语法都是中继安全的。

7.7 BASIC-PER 和 CANONICAL-PER 各有两个变体：ALIGNED 变体和 UNALIGNED 变体。在 ALIGNED 变体中，常常插入填充位来恢复八位位组对齐，在 UNALIGNED 变体中，则从不插入填充位。

7.8 在 ALIGNED 变体和 UNALIGNED 变体之间没有互工作的可能性。

7.9 只有知道编码值的类型时，PER 编码才是自定界的，编码总是八位的整数倍。当在 **EXTERNAL** 类型中携带这些位时，应在 **OCTET STRING** 选择项中携带它们，除非 **EXTERNAL** 类型本身用 PER 进行编码，在这种情况下，该值可以编码为单个 ASN.1 类型（即开放类型）。当在 OSI 表示协议中携带它们时，应使用带有 **OCTET STRING** 选择项的“全编码”（见 GB/T 15696.1 定义）。

7.10 除非另有说明，本文件的规则适用于两种算法和两种变体(有例外，见 9.2 和 9.3)。

7.11 附录 C 是资料性附录，它就实现 PER 的组合给出了建议，以使互工作的机会最大化。

8 符合性

8.1 动态符合性在第 9 章中规定。

8.2 静态符合性由规定这些紧缩编码规则应用的那些标准来规定。

注：附录C提供了与支持两种编码规则算法的两种变体有关的静态符合性的指南，该指南被设计成能保证互工作，同时认可既不是中继安全、也不是正则的某些编码应用的好处。

8.3 本文件中的规则以编码规程的形式详细说明。如果生成的抽象语法值完整编码的位串与本文件为适用传送语法规定的那些位串之一相同，则不要求此类实现与规定的规程一一对应。

8.4 执行解码的实现要产生与收到的位串相一致的抽象语法值，收到的位串由发送器产生，这些发送器符合与被解码资料相关联的传送语法中所标识的编码规则。

注1：通常，本文件明确说明的BASIC-PER，没有定义替换的编码。通过规定中继安全操作和限制引用其他标准的某些编码选项，使BASIC-PER成为正则的。CANONICAL-PER为要求正则和中继安全编码的非典型编码规则和正则编码规则（见GB/T 16263.1)提供了一种替换的编码规则。

注2：当使用CANONICAL-PER来提供正则编码时，建议将从其派生出的任意一个加密散列值与一个算法标识符相关联，该算法标识符将CANONICAL-PER标识为从抽象语法值到初始位串（然后被散列）的转换。

9 PER 编码指令

9.1 PER 编码指令可以与符合 GB/T 16262.1 和 ISO/IEC 8825-6 规定的类型相关联。

注1：某些PER编码指令的应用程序可能会使其不能对该类型的所有抽象值进行编码。当出现这种情况时，特定的PER编码指令会识别问题，这是设计者的决定，基于可能需要使用多个编码规则，是否对类型添加显式约束，以便将抽象值的范围限制为那些可以通过使用PER编码指令进行编码处理的范围，这可能会使规范的可读性降低，但可以确保所有编码规则都可以对所有允许的抽象值编码，使中继可能没有错误。

注2：每个PER编码指令都以一个标识关键字开始，它可以明确地标识该编码指令。

9.2 如果使用 BASIC-PER 或 CANONICAL-PER 的 ALIGNED 版本，那么所有的 PER 编码指令都应该默认忽略，且对编码没有影响。

9.3 如果使用 BASIC-PER 或 CANONICAL-PER 的 UNALIGNED 版本，那么如果一个类型具有相关的编码指令，则应适用以下子条。

9.3.1 如果标识关键字未知，则应发出“不支持”的错误消息。

9.3.2 如果已知标识关键字，本文件的程序应通过对 PER 编码指令所指定的程序的修订进行修改(参见 ISO/IEC 8825-6)。

注1：如果多个PER编码指令与一种类型相关联，则所有这些指令指定的修订都应适用。

注2：如果由两个或两个以上单独的编码指令产生的修改发生冲突，并且没有说明它们是相互排斥的，则PER编码指令的寄存器存在错误。

10 PER使用的编码方法

10.1 类型记法的使用

10.1.1 这些编码规则专门使用GB/T 16262.1中规定的ASN.1类型记法，并且只能适用于将使用该记法规定的单个ASN.1类型的值编码。

10.1.2 特别是，但不仅仅是，它们依赖于ASN.1类型和值模型中保留的以下信息，这些信息是使用记法的基础：

a) 选择类型中的选择类型的嵌套；
b) 放置在集合类型中的组件上的标签、放置在选择类型中的选项上的标签及枚举项的给定值；
c) 集合或者序列类型的组件是否可选；
d) 集合或者序列类型的组件是否有**DEFAULT**值；
e) （只）通过PER可视约束的应用出现的类型值的受限范围；
f) 组件是否是开放类型；
g) 类型对PER编码是否可扩展。

10.2 使用标签提供正则次序

本文件要求被正则排序的集合类型和选择类型的组件独立于组件的正文排序，如GB/T 16262.1—AAAA的8.6规定的那样，正则排序通过排序每个组件的最外层标签来确定。

10.3 PER可视约束

注：对于解码和编码，某些ASN.1约束可能不是PER可视的，但不会以任何方式影响使用这些约束处理解码期间检测到的差错，也不意味着允许违反这些约束的值被遵守规则的发送器所发送。然而，本文件并不在编码规范中使用这样的约束。

10.3.1 用人类可读文本表达的或在ASN.1注释中所表示的约束不是PER可视的。

10.3.2 可变的约束不是PER可视的（见GB/T 16262.4—DDDD的10.3和10.4）。

10.3.3 用户定义的约束不是PER可视的（见GB/T 16262.3—CCCC的9.1）。

10.3.4 表约束不是PER可视的（见GB/T 16262.3）。

10.3.5 组件关系约束不是PER可视的（见GB/T 16262.3—CCCC中的10.7）。

10.3.6 计算过程文本依赖于表约束或组件关系约束的约束不是PER可视的（见GB/T 16262.3）。

10.3.7 对不是已知倍数字符串类型的（见GB/T 16262.1—AAAA的第41章）受限字符串类型的约束不是PER可视的（见3.7.16）。

10.3.8 模式约束不是PER可视的。

10.3.9 除受上述各条制约外的所有长度约束都是PER可视的。

10.3.10 约束类型的有效长度约束是单个长度约束，因此，当且仅当存在该（允许的）长度的某个约束类型值时，该长度才是允许的。

10.3.11 在应用GB/T 16262.1—AAAA中的52.3到52.5之后，对不可扩展的已知倍数字符串类型的允许字母表约束是PER可视的。可扩展的允许字母表约束不是PER可视的。

10.3.12 当且仅当包含某个字符的受约束类型的某个值，该受约束类型的有效允许字母表约束是允许该字符的单个允许字母表约束。如果受约束类型的所有字符都可以出现在该受约束类型的某个值中，则该有效允许字母表约束是为不受约束的类型而定义的字符集。

10.3.13 在应用 GB/T 16262.1—AAAA 中的 52.3 到 52.5 后，对不可扩展的时间类型（或有用的和定义的时间类型）的属性设置约束不是 PER 可视的。可扩展的属性设置约束不是 PER 可视的。

10.3.14 应用于实数类型的约束不是 PER 可视的

10.3.15 仅当用来限制 **syntaxes** 组件的值为单个值时，或者用来限制 **identification** 为替换的 **fixed** 标识（见第 28 章和第 31 章）时，应用于不受限字符串或者嵌入式 pdv 类型的内部类型约束才是 PER 可视的。

10.3.16 对有用类型上的约束不是 PER 可视的。

10.3.17 应用于字符串类型的单个值子类型约束不是 PER 可视的。

10.3.18 当且仅当所有其他约束适用于整数类型或已知倍数字符串类型时，受上述各条制约的这些约束是 PER 可视的。

10.3.19 通常对类型的约束可由各个约束组合而成，组合时使用了某些或全部包含子类型约束和一系列应用约束的集合算法。

注：有关组合分别为PER可视或PER不可视约束的效果的进一步讨论参见附录B。

10.3.20 如果约束由约束的一系列应用组成，那么，即使不是 PER 可视的约束，也不影响 PER 编码，但能引起可扩展性（和扩展附加部分）出现在任何按 GB/T—AAAA 16262.1 中 50.11 规定的要被取消的早期约束中。

注1：如果在串行应用程序中的最后一个约束不是PER可视的，那么该类型不能针对PER编码进行扩展，并且无扩展位进行编码。

注2：例如，因为可扩展的允许字母表约束不是PER可视的，那么，

```
A ::= IA5String(SIZE(1..4))(FROM("ABCD", ...))
```

就有一个由整个**IA5String**字母组成的有效允许字母表约束，不过，它的有效大小约束为“**SIZE(1..4)**”。

类似地：

```
B ::= IA5String(A)
```

有同样的有效大小约束和同样的有效允许字母表约束。

10.3.21 如果一个 PER 可视约束是 **INTERSECTION** 结构的一部分，那么，得到的约束是 PER 可视的，并由所有 PER 可视部分（非 PER 可视部分被忽略不计）的 **INTERSECTION** 组成。如果一个不是 PER 可视的约束是 **UNION** 结构的一部分，那么得出的约束也不是可视的。如果一个约束有 **EXCEPT** 条款，那么，**EXCEPT** 及随后的值集全部忽略不计，而不管跟在 EXCEPT 之后的值集是否是 PER 可视的。

注：例如：

```
A ::= IA5String (SIZE(1..4) INTERSECTION FROM ("ABCD", ...))
```

有1..4的有效长度约束，但字母表约束不是可视的，因为它是可扩展的。

10.3.22 如果下列情况之一出现，则对于 PER 编码，类型也是可扩展的：

a) 它是从一个**ENUMERATED**类型派生的（通过划分子类型、指出类型引用或者置标签），而且在“Enumerations”产生式中有扩展标记；

b) 它是从一个**SEQUENCE**类型派生的（通过划分子类型、指出类型引用或者置标签），而且在“ComponentTypeLists”或“SequenceType”产生式中有扩展标记；

c) 它是从一个**SET**类型派生的（通过划分子类型、指出类型引用或者置标签），而且在“ComponentTypeLists”或“SetType”产生式中有扩展标记；

d) 它是从一个**CHOICE**类型派生的（通过划分子类型、指出类型引用或者置标签），而且在“AlternativeTypeLists”产生式中有扩展标记。

10.4 编码使用的类型和值模型

10.4.1 ASN.1 类型要么是简单类型，要么是使用其他类型构造的类型。记法允许使用类型引用和类型置标签。对于这些编码规则，使用类型引用和类型置标签对编码没有影响，而且在模型中是不可视的，除非在 10.2 中所述。记法也允许应用约束和差错规范。PER 可视约束在模型中作为类型值的限制出现。其他约束和差错规范不影响编码，并且在 PER 类型和值模型中是不可视的。

10.4.2 被编码的值可以被视为一个简单值或者一个用结构化机制从简单或复合值的组件构建的复合值，与 ASN.1 类型定义的结构并行。

10.4.3 当约束包括一个在根中出现的扩展附加部分的值时，该值总是编码为根的值，而不是扩展附加部分的值。

示例：

```
INTEGER (0..10,...,5)
--值5被编码为根值，而不是一个扩展附加部分的值。
```

10.5 编码的结构

10.5.1 这些编码规则的规定

a) 简单值到字段列表的编码；
b) 复合值到字段列表的编码，使用了由这些编码规则应用到复合值的组件而产生的字段列表；
c) 最外层值的字段列表到抽象语法值的完整编码的变换（见 11.1）。

10.5.2 数据值组件的编码：

a) 由 3 部分构成，如图 1 所示，它们以下列次序出现：
   1) 前导码（见第 19、21 和 23 章）；
   2) 长度决定因子（见 11.9）；
   3) 内容。

注：前导码、长度及内容都是“字段”，它们被拼接在一起形成“字段列表”，除选择类型外，复合类型的字段列表可以由几个值拼接在一起的字段构成。任一前导码、长度和/或任意值的内容都可能没有。

图 1 复合值到字段列表的编码

b) （若内容很长）由任意个部分构成，如图 2 所示，其中第 1 部分是前导码（见第 19、21 和 23 章），随后的各部分是多对位字段（在 ALIGNED 变体中八位位组对齐），首先是内容段的长度决定因子，其次是各内容段；各字段的最后一对由长度决定因子部分来标识，如 11.9 规定。

图 2 长数据值的编码

10.5.3 在 10.5.2 中提到的每一部分生成：

a)一个空字段（无内容）；
b)一个位字段（未对齐）；
c)一个位字段(ALIGNED 变体中八位位组对齐）；
d)一个字段列表，它可以包含位字段（未对齐）、位字段(ALIGNED 变体中八位位组对齐）、或者两者都有。

10.6 被编码的类型

10.6.1 下列各章规定将以下类型编码到字段列表中：布尔、整数、枚举、实数、位串、八位位组串、空、序列、单一序列、集合、单一集合、选择、开放、客体标识符、相关客体标识符、嵌入式pdv、外部、受限字符串和不受限字符串的类型。

10.6.2 选择类型应编码为所选类型的编码。

10.6.3 除 10.2 说明外，有标签类型的编码不包括在本文件中，因为在这些编码规则所使用的类型和值模型中置标签是不可视的。因此，有标签的类型根据已有标签的类型的编码来编码。

10.6.4 带前缀类型的编码根据带前缀的类型进行编码。

10.6.5 下列“有用的类型”应进行的编码，就好像它们已被 GB/T 16263.1—AAAA 的第 45 章中给出的定义所取代：

——通用时间；
——世界协调时；
——客体描述符。

对有用类型的约束不是PER可视的，GB/T 16263.1—EEEE的11.7和11.8对通用时间和世界协调时的编码的限制适用于此。

10.6.6 使用值集赋值定义的类型应像使用 GB/T 16262.1—AAAA 的 16.8 中规定的产生式定义的类型一样进行编码。

11 编码过程

11.1 完整编码的产生式

11.1.1 如果 ASN.1 类型使用在 33.2 中（或通过本文件的直接文本引用）列出的客体标识符所标识的任何编码规则进行编码，并且编码被包括在：

a) ASN.1 八位位组串中；
b) ASN.1 位串中；
c) ASN.1 开放类型中；
d) ASN.1 外部或嵌入式 pdv 类型的任何部分中；
e) 任何不使用 ASN.1 定义的载体协议中；

则该ASN.1类型被定义为这种应用的最外层类型，应遵循11.1.2中适用于其值的所有编码。

注1：这意味着使用此方法的所有完整的PER编码（对所有变体）总是八位的整数倍，除非使用UNALIGNED变体，并且在ASN.1的位串中（例如b）进行编码。

注2：可以使用编码控制记法（见GB/T 16263.3）来规定PER编码的变体，该编码不会被填充到11.1.2中规定的八位位组边界，许多工具都支持这一选项。

注3：普遍认为，在不使用ASN.1定义的载体协议中，不需要显式携带额外的0位用于填充（在11.1.2中规定），但可以隐含0位的存在。

11.1.2 对于最外层类型抽象值应用本文件的结果所产生的字段列表，应按照以下步骤生成该抽象语法值的完整编码：应依序取出字段列表中的每个字段，拼接到位串的末端，形成的位串即为抽象语法值的完整编码，该抽象语法值的完整编码可按如下规定添加填充的附加 0 位。

11.1.3 这些编码规则的 UNALIGNED 变体中，所有字段应进行拼接，而无需填充。在 11.1.1 中，除了情况 b 以外的其他情况适用 11.1.3.1，情况 b 适用 11.1.3.2。

11.1.3.1 (对于编码的结果不包括在 ASN.1 位串中的情况)如果最外层值编码的结果是一个空位字符串，则该位串应该用所有位置为 0 的单个八位位组来替换。如果是非空位字符串，并且它不是八位的倍数，则应在空位串后面附加(0 到 7 个）0 位以产生 8 位的倍数。

11.1.3.2 (对于编码的结果包括在一个 ASN.1 位串中的情况)如果最外层值编码的结果是空位字符串，则该位串应该用 0 位来代替，不得附加任何填充位。

11.1.4 如果最外层值编码的结果是空位串，则该位串应该用所有位置为 0 的单个八位位组来代替。如果是非空位串，且不是八位的倍数，则应在空位串后面附加(0 到 7 个）0 位以产生 8 位的倍数。

11.1.5 在编码规则的 ALIGNED 变体中，字段列表中的任意位字段应进行拼接，而无需填充。在已拼接(0 到 7 个）0 位使目前产生的编码长度是 8 位的整数倍之后，任意八位位组对齐位字段应被拼接。如果最外层值编码的结果是空位串，则用一个所有位置为 0 的八位位组代替该位串。如果是个非空位串，且不是 8 位的整数倍，则应在其后面附加(0 到 7 个）0 位产生 8 位的整数倍。

注 1：如果抽象语法值是空类型值或者被约束为单个值的整数型，则最外层值的编码是个空位串。

注 2：0 长度的八位位组对齐位字段永远不会出现在字段列表中（见 11.9.3.3）。

11.1.6 所产生的位串是最外层类型的抽象语法值的完整编码。

11.2 开放类型字段

11.2.1 为了编码开放类型字段，占用该字段的实际类型的值应编码到一个字段列表中，然后，该字段列表应被转换成如 11.1 规定的抽象语法值的完整编码，从而产生长度为 n 的八位位组串。

11.2.2 要嵌入的开放类型值的字段列表应被添加上（如 11.9 规定）不受约束的长度 n(以八位位组为单位）和含有 11.2.1 中产生位的关联位字段(对于 ALIGNED 变体，采用八位位组对齐的位字段）。

注：当开放类型编码中八位位组的数目很大时，应使用11.9中的分片过程，开放类型编码将是断开的，而与占用开放类型字段的类型编码中分片边界的位置无关。

11.3 非负二进制整数的编码

注：（说明性）本条给出术语“非负二进制整数编码”的精确描述，以便把该整数放到固定位数目的字段中，或放到固定数目的八位位组的字段中，或者放到容纳该整数所需的最少数目八位位组的字段中。

11.3.1 以下条涉及进入某一字段的非负整个数的非负二进制整数编码的生成，而该字段是规定长度为单个八位位组、两个八位位组的位字段或者是表示该值的最少数目八位位组的位字段，本条(11.3)规定进行引用时所使用的精确编码。

11.3.2 该字段的引导位定义为位字段的引导位，或者是该字段第 1 个八位位组的最高有效位，该字段的末尾位定义为位字段的末尾位，或者是该字段最后 1 个八位位组的最低有效位。

11.3.3 只对以下定义，这些位应加以编号，该字段的末尾位应被编号为 0，下一位编号为 1，依此类推，直到该字段的引导位。

11.3.4 在非负二进制整数编码中，编码所表示的整数的值应是每位所指定值之和。置为“0”的位的指定值为 0，编号为“n”的位，若置为“1”则该位的指定值为2^n。

11.3.5 对被编码值的求和（按上述定义）的编码即是对该值的编码。

注：如果被编码字段长度为固定（指定长度的单个八位位组或两个八位位组的位字段）的，则存在唯一的编码，它是被编码的值的和。

11.3.6 整个数（它不能预先确定用于编码的八位位组的数目）的最少数目八位位组的非负二进制整数编码具有八位的整数倍的某一字段，并且该字段应满足如下条件，除非该字段是准确的八位长，否则该字段的引导段八位应不全是 0。

注：这是产生唯一编码的充分必要条件。

11.4 2 的补码的二进制整数的编码

注：（说明性）本条给出术语“2的补码的二进制整数编码”的精确描述，以便把有符号的整数放到为容纳该有符号整数所需的最少数目的八位位组中，这些过程在后面的编码规范中引用。

11.4.1 后续条规定如何将整数（可能是负数、0 或正数）编码到表示其值的最少数目八位位组的 2 的补码二进制整数的编码中。本条（11.4）规定进行这种引用时所使用的精确编码。

11.4.2 该字段的引导位定义为第 1 个八位位组的最高有效位，末尾位定义为最后 1 个八位位组的最低有效位。

11.4.3 仅对以下定义，这些位应加以编号，该字段的末尾位应被编号为 0，下一位编号为 1，依此类推，直到该字段的引导位。

11.4.4 在 2 的补码的二进制整数编码中，编码所表示的整个数的值应是每位所指定值之和。置为“0”的位的指定值为 0。除引导位外，编号为“n”的位，若置为“1”则该位的指定值为2^n，如果引导位为 1，则该位的指定值为（负数）-2^n。

11.4.5 对被编码值的求和（按上述定义）的任何编码均为对该值的编码。

11.4.6 整个数的最少数目八位位组 2 的补码的二进制整数编码具有八位的整数倍的字段宽度，并且应满足如下条件：该字段的引导的 9 个位应不全是 0 也应不全是 1。

注：这是产生唯一编码的充分必要条件。

11.5 受约束的整个数的编码

注：（说明性）本条被其他条引用，并引用前面条，以便产生非负二进制整数编码或2的补码的二进制整数编码。对于UNALIGNED变体，该值总是编码在表示该范围（在11.5.3中定义）所需的最少数目位中，本注的其余部分针对ALIGNED变体，当范围小于或等于255时，该值被编码到表示该范围的最小长度的位字段中。当范围正好是256时，该值被编码到单个八位位组对齐的位字段中，当范围在257到64K之间时，该值被编码到两个八位位组对齐位字段中，当范围大于64K时，则忽略范围，并且该值被编码到八位位组对齐位字段中，该位字段是表示该值的最少数目八位位组，在最后一种情况下，后面的过程（见11.9）将编码一个长度字段（通常是单个八位位组），以指示编码的长度，对其他情况，编码的长度与被编码的值无关，不被显式编码。

11.5.1 本条（11.5）规定了如何将受约束的整个数映射到位字段（不对齐的）或者位字段（在 ALIGNED 变体中的八位位组对齐），并在本文件中的后续条引用。

11.5.2 仅当受约束的被编码的整个数是可用的，并且下边界“lb”和上边界“ub”的值已从类型记法中确定下来（在应用 PER 可视约束之后），本条的过程才能被引用。

注：如果对一个无限数求MIN值，则不能确定下边界，如果对一个无限数求MAX值，则不能确定上边界。例如，对INTEGER(MIN..MAX)，则上边界和下边界都不能确定。

11.5.3 设“范围”定义为整数值（“ub”-“lb”+1），且设被编码的值为“n”。

11.5.4 如果“范围”值为 1，则编码结果应是一个空的位字段（没有任何位）。

11.5.5 有 5 种其他情况（导致不同的编码）要考虑，一种适用于 UNALIGNED 变体，其他 4 种适用于 ALIGNED 变体。

11.5.6 在UNALIGNED变体的情况下，值(“n”-“1b”)应编码为按照11.3规定的位字段中的非负二进制整数，并具有为表示该范围所需的最少数目位数。

注：如果“范围”满足不等式2^m < “范围” ≤ 2^{m+1}，则位数为m+1。

11.5.7 在ALIGNED变体的情况下，编码取决于是否：

a) “范围”小于或等于255（位字段的情况）；

b) “范围”刚好等于256（一个八位位组的情况）；

c) “范围”大于256且小于或等于64K（两个八位位组的情况）；

d) “范围”大于64K（不定长度的情况）。

11.5.7.1 (位字段的情况）如果“范围”值小于或等于255，则本条的引用要求生成下表所规定的位数的位字段，并包含按11.3规定的位字段中作为非负二进制整数编码的值(“n”-“1b”)。

“范围”	位字段长度（按位计）
2	1
3、4	2
5、6、7、8	3
9~16	4
17~32	5
33~64	6
65~128	7
129~255	8

11.5.7.2 (一个八位位组的情况）如果“范围”值为256，则值(“n”-“1b”)应被编码为11.3规定的非负二进制整数，放在一个八位位组位字段中(ALIGNED变体中的八位位组对齐）。

11.5.7.3 (两个八位位组的情况）如果“范围”值大于或等于257并且小于或等于64K，则值编码为11.3规定的非负二进制整数(“n”-“1b”)放在两个八位位组位字段（ALIGNED变体中的八位位组对齐）中。

11.5.7.4 （不定长度的情况）否则，值(“n”-“1b”)应按11.3的规定编码为非负二进制整数，放在表示该值的最少数目八位位组的位字段（ALIGNED变体中的八位位组对齐）中，并且该编码中使用的八位位组的数目“1en”被引用本条的其他条用来规定长度的编码。

11.6 正常小非负整数的编码

注：（说明性）当编码期望为小非负整个数，但其长度由于存在扩展标记而引起潜在地不受限制时，可使用本过程，例如，在处理选择索引时。

11.6.1 如果非负整个数“n”小于或等于63，则一个单个位的位字段应被附加到字段列表后，该位置为0，并且“n”应编码为非负二进制整数，放入一个6位的位字段中。

11.6.2 如果“n”大于或等于64，则一个置为1的单个位的位字段应被附加到字段列表后。然后，值“n”应编码为半受约束的且“1b”等于0的整个数，并应引用11.9的过程将它附加到字段列表中，该字段列表前面是长度决定因子。

11.7 半受约束的整个数的编码

注：（说明性）当在能够标识出下边界但不能标识上边界时，可适用本编码过程。编码过程将与下边界的偏移值作为非负二进制整数放入最少数目的八位位组，并且要求一个按后面过程规定的显式长度编码（典型的是单个八位位组）。

11.7.1 本条规定了从半受约束的整个数到位字段（ALIGNED 变体中的八位位组对齐）的映射，并且它将被本文件的后续条所引用。

11.7.2 仅当被编码的半受约束的整个数可用，并且下边界“lb”的值已经从类型记法中确定（在应用 PER 可视约束之后）时，本条（11.7）的过程才能被引用。

注：如果对一个无限数求MIN值，则不能确定下边界。例如，对INTEGER(MIN..MAX)不能确定任何下边界。

11.7.3 本条（11.7）的过程总是产生不定长度的情况。

11.7.4 （不定长度的情况）值(“n”-“lb”)应按 11.3 的规定编码为非负二进制整数，放在表示该值的最少数目八位位组的位字段（ALIGNED 变体中的八位位组对齐）中，并且该编码中使用的八位位组的数目“len”被引用本条的其他条用来规定长度的编码。

11.8 不受约束的整个数的编码

注：（说明性）这种情况只在没有下边界的整数型值的编码中出现。本过程将该值编码为2的补码二进制整数，并放入容纳该编码所需的最少数目八位位组中，并且要求有一个按后面过程规定的显式长度编码（典型的是单个八位位组）。

11.8.1 本条（11.8）规定了从不受约束的整数（如“n”)到位字段（ALIGNED 变体中的八位位组对齐）的映射，它被本文件的后续条引用。

11.8.2 本条（11.8）的过程总是产生不定长度的情况。

11.8.3 （不定长度的情况）值“n”应按 11.4 的规定编码为 2 的补码二进制整数，放在表示该值的最少数目八位位组的位字段（ALIGNED 变体中的八位位组对齐）中，并且该编码中使用的八位位组的数目“len”被引用本条的其他条用来规定长度的编码。

11.9 长度决定因子编码的一般规则

注1：（说明性）本条规定的过程只在编码的某些部分需要一个显式的长度字段时被引用，无论长度计数是否有上边界（通过PER可视约束）。使用长度的编码部分可能是位串（带有以位表示的长度计数）、八位位组串（带有以八位位组表示的长度计数）、已知倍数字符串（带有以字符表示的长度计数）、或者字段列表（带有以单一序列或单一集合的组件表示的长度计数）。

注2：（说明性）在ALIGNED变体的情况下，如果长度计数上限小于64K，则长度使用受约束的整数编码。对于足够小的范围，其结果是一个位字段，否则，不受约束的长度（如“n”），将按以下列三种方式之一编码到八位位组对齐字段中（依长度递增的次序）：

a) （“n”小于128）含有“n”的位8置为0的单个八位位组；

b) （“n”小于16K）含有“n”的两个八位位组，第1个八位位组的位8置为1，位7置为0；

c) （大“n”）含有计数“m”的单个八位位组，位8置为1，位7也置为1。计数“m”为1~4，该长度表示其后面有一个资料分片（16K项的“m”倍）。对于所有“m”值，该分片后面跟有另一个表示资料余下部分的长度编码。

注3：(说明性）在UNALIGNED变体中，如果长度计数上限小于64K，则受约束的整数编码用来编码表示该范围所需的最少数目位的长度，否则不受约束的长度（如“n”）按上述注2的方式编码到位字段中。

11.9.1 按照后面各条的规范，如果长度决定因子“n”的值被类型定义（被 PER 可视约束所约束）限定为小于 64K 的值，则不能引用本条。

11.9.2 本条被引用来向字段列表附加字段或者字段列表，其前面是一个长度决定因子“n”，长度决定因子用于决定：

a) 以关联字段的八位位组表示的长度（单位为八位位组）；

b) 以关联字段的位表示的长度（单位为位）；

c) 以关联的字段列表表示的组件编码的数目（单位为单一集合或单一序列的组件）；

d) 以关联的已知倍数字符串类型的值表示的字符数（单位为字符）。

11.9.3 （ALIGNED 变体）ALIGNED 变体用的过程在 11.9.3.1 到 11.9.3.8.4 中规定（UNALIGNED 变体用的过程在 11.9.4 中规定）。

11.9.3.1 作为类型定义（在后面的条中规定）的分析结果，长度决定因子（一个整个数“n”）将被确定为：

a) 一个正常小长度，其下边界“lb”等于 1；或者

b) 一个受约束的整数，具有（大于或等于 0）的下边界“lb”，上边界“ub”小于 64K；

c) 一个半受约束的整个数，具有（大于或等于 0）的下边界“lb”，或者一个受约束的整个数，具有（大于或等于 0）的下边界“lb”，大于或等于 64K 的上边界“ub”。

11.9.3.2 引用本条(11.9）过程的各条将确定长度的下边界“lb”的值（如果长度是不受约束的，则它为 0）和长度的上边界“ub”的值。如果从 PER 可视约束不能确定上边界，则“ub”为不设置。

11.9.3.3 当长度决定因子是一个“ub”小于 64K 的受约束的整数时，则按 11.5 中规定的方法，长度决定因子的受约束整个数的编码应被附加到字段列表上。如果“n”为非 0，则其后应有关联字段或字段列表，过程结束。如果“n”为 0，则对字段列表不应进一步附加任何内容，过程结束。

注1：例如：

```
A ::= IA5String (SIZE(3..6))            --长度以2位的位字段编码
B ::= IA5String (SIZE(40000..40254))    --长度以8位的位字段编码
C ::= IA5String (SIZE(0..32000))        --长度以两个八位位组(ALIGNED变体中的八位位组对齐)的位
                                          字段编码
D ::= IA5String (SIZE(64000))           --长度不予编码
```

注2：当这些过程被引用来添加0长度的八位位组对齐位字段时，在“n”等于0的情况下，不进行添加的影响是指对八位位组边界的填充并不出现，除非11.5条要求。

11.9.3.4 当长度决定因子是一个正常小长度，并且“n”小于或等于 64，则一个单个位的位字段将被附加到字段列表上，且该位被置为 0，值“n-1”应被编码为非负二进制整数放入一个 6 位的位字段中。其后是关联的字段，此时这些过程结束。按照 11.9.3.5 到 11.9.3.8.4 中的过程，如果“n”大于 64，则一个单个位的位字段应被附加到字段列表上，且该位置为 1，其后是不受约束长度决定因子 n 的编码，再后是关联的字段。

注：正常小长度只用来指示位图的长度，该位图表示集合或序列类型的扩展附加部分值的前缀。

11.9.3.5 否则（不受约束的长度，或大“ub”），“n”被编码并添加到字段列表上，其后是以下规定的关联字段。

注：下边界“lb”不影响11.9.3.6到11.9.3.8.4中规定的长度编码。

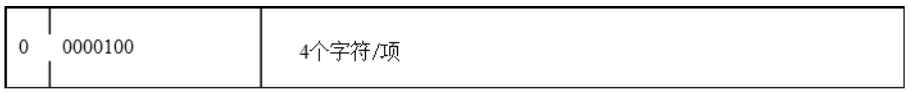
11.9.3.6 如果“n”小于或等于 127，则“n”应作为非负二进制整数（使用 11.3 的过程）被编码到一个单个八位位组位 7（最高有效位）至位 1（最低有效位）中，且位 8 应置为 0。它应作为位字段（在 ALIGNED 变体中的八位位组对齐）附加到字段列表后，其后是关联的字段或字段列表，过程结束。

注：例如，如果下列A的值为4个字符长，并且B的值为4个项长：

```
A ::= IA5String
B ::= SEQUENCE (SIZE (4..123456) ) OF INTEGER
```

则这两个值都在占用一个八位位组的长度八位位组中编码，且最高有效位置为0，以表明长度小于或等于127：

XX



11.9.3.7 如果“n”大于 127 且小于 16K，则“n”应（使用 11.3 的过程）编码为非负二进制整数放入两个八位位组位字段（在 ALIGNED 变体中的八位位组对齐）中的第 1 个八位位组（最高有效）的位 6 到第 2 个八位位组（最低有效）的位 1 中，同时第 1 个八位位组的位 8 置为 1，第 1 个八位位组的位 7 置为 0。它应被附加到字段列表后，其后是关联字段或字段列表，过程结束。

注：如果在11.9.3.6示例中A的值为130个字符，B的值为130项，则这两个值在占用2个八位位组的长度组件和该八位位组的2个最高有效位（位8和位7）均置为10的情况下进行编码，表示长度大于127小于16K。

11.9.3.8 如果“n”大于或者等于 16K，则在位字段（在 ALIGNED 变体中的八位位组对齐）中的单个八位位组应被附加到字段列表上，该单个八位位组的位 8 置为 1，位 7 置为 1，并且位 6 到位 1 是值 1、2、3 或 4，作为非负二进制整数（使用 11.8 的过程）编码。该单个八位位组后应是下面规定的部分关联字段或字段列表。

注：位6到位1的值限制为1～4（而不是理论值0～63），是为了把实现必须了解的项数限制到更易于管理的数目(64K而不是1024K)上。

11.9.3.8.1 位 6 到位 1 的值（1 到 4）应乘以 16K 来给出计数（记“m”）。位 6 到位 1 中的整数的选择应是最大可允许的值，这样，相关联的字段或字段列表含有多于或正好等于合适的“m”个八位位组、位、组件或字符。

注1：不分片形式处理的长度达16K，因此分片可以提供高达64K的长度，粒度为16K。

注2：如果在11.9.3.6的示例中，B的值长度为144K+1（即64K+64K+16K+1）项长，则该值被分片，前三个分片的两个最高有效位（位8和位7）置为11，表明后面的1到4个块，每块16K项，另一个长度组件跟在每个分片的最后一块之后。

11.9.3.8.2 “m”指定的内容部分应以以下方式添加到字段列表中：

a) “m”个八位位组的单个位字段（在 ALIGNED 变体中的八位位组对齐）含有关联字段的前“m”个八位位组，单位为八位位组；

b) “m”位的单个位字段（在 ALIGNED 变体中的八位位组对齐）含有关联字段的前“m”位，单位为位；

c) 字段列表编码关联字段列表中的前“m”个组件，单位为单一集合类型或单一序列类型的组件；

a) “m”个字符的单个位字段（在 ALIGNED 变体中的八位位组对齐）含有关联字段的前“m”个字符，单位为字符。

11.9.3.8.3 应重新应用 11.9 的过程将关联字段或字段列表的其余部分附加到带有长度的字段列表上，该长度是等于(“n”-“m”)的半受约束整个数，下边界为 0。

注：如果含有部分编码值的最后一个分片的长度正好是16K的整数倍，则其后的最末一个分片只由单个八位位组的长度组件构成，该长度组件置为0。

11.9.3.8.4 重复应用上述过程，只将关联字段的一部分添加到字段列表上称为分片过程。

11.9.4 (UNALIGNED 变体)UNALIGNED 变体用的过程在 11.9.4.1 到 11.9.4.2 中规定（ALIGNED 变体用的过程在 11.9.3 中规定）。

11.9.4.1 如果要编码的长度决定因子“n”是“ub”小于 64K 的受约束的整个数，则应使用编码“范围”(“ub”-“lb”+1）所需的最少数目位数将(“n”-“lb”)编码为非负二进制整数（在 11.3 中规定），除非范围是 1，在这种情况下，不应有长度编码。如果“n”为非 0，则其后是关联字段或字段列表，过程结束。如果“n”是 0，则不再向该字段列表添加字段表，过程结束。

注：如果“范围”满足不等式2^m < “范围” ≤ 2^{m+1}，则长度决定因子中的位数为m+1。

11.9.4.2 如果被编码的长度决定因子“n”是一个正常小长度、或是一个大于或等于 64K 的受约束整个数、或是一个半受约束整个数，则“n”应按照 11.9.3.4 到 11.9.3.8.4 中的规定进行编码。

注：因此，如果“ub”大于或等于64K，长度决定因子编码与长度不受约束时相同。

12 布尔类型的编码

12.1 布尔类型的值应被编码为由单个位构成的位字段。

12.2 该位应置 1 为 TRUE，置 0 为 FALSE。

12.3 位字段应被添加到不带长度决定因子的字段列表上。

13 整数类型的编码

注1：(ALIGNED变体说明性）所允许的全部值编码到一个或更少的八位位组的范围，放入一个没有长度计数的最小长度的位字段，所允许的全部值编码到两个八位位组的范围，放入一个没有长度计数的八位位组对齐的位字段中的两个八位位组。否则，该值被编码到最少数目的八位位组（当合适时使用非负二进制整数或2的补码二进制整数编码）中并添加上长度决定因子。在这种情况下，如果整数值可以被编码在小于127个八位位组（作为可能确定的任何下边界的偏移）中，并且没有有限上边界和下边界，则存在一个八位位组的长度决定因子，否则，该长度被编码在所需的最少位数中。其他情况没有实际意义，但为了完整性也进行了规定。

注2：（UNALIGNED变体说明性）受约束的整数被编码为表示其范围所需的最小位数，而不管其大小。不受约束的整数编码如注1所示。

13.1 如果在整数类型的约束规范中出现扩展标记，则单个位应作为长度为 1 的位字段添加到字段列表上。如果被编码的值不在扩展根的范围内，则该位应置为 1，否则置为 0。对前一种情况，该值应按 13.2.4 到 13.2.6 的规定，作为不受约束的整数值添加到字段列表中，结束此过程。对后一种情况，该值应按照不存在扩展标记的情况进行编码。

13.2 如果整数类型的约束规范中不存在扩展标记，则使用下列条。

13.2.1 如果 PER 可视约束将整数值限制为单个值，则不应向字段列表添加任何内容，过程结束。

13.2.2 如果 PER 可视约束将整数值限制为受约束的整个数，则应按照 11.5（受约束的整个数的编码）的过程将其转换成某个字段，然后应使用 13.2.5 到 13.2.6 的过程。

13.2.3 如果 PER 可视约束将整数值限制为半受约束的整个数，则应按照 11.7（半受约束的整个数的编码）的过程将其转换成某个字段，然后应使用 13.2.6 的过程。

13.2.4 如果 PER 可视约束不将整数值限制为受约束的整个数或者半受约束的整个数，则按照 11.8（不受约束的整个数的编码）的过程将其转换成某个字段，然后应使用 13.2.6 的过程。

13.2.5 如果引用将整数值编码到某个字段的过程不产生不定长度的情况（见 11.5.7.4 和 11.8.2），则该字段应被添加到字段列表上，过程结束。

13.2.6 否则（不定长度情况），不应引用 11.9 的过程把该字段附加到其前面是下列内容之一的字段列表上：

a) 受约束的长度决定因子等于“len”（按 11.5.7.4 所确定的），如果 PER 可视约束限制了带有有限上下边界的类型，并且如果该类型是可扩展的，则该值位于扩展根范围内。长度决定因子使用的下边界“lb”应是 1，上边界“ub”应是容纳该整数值范围所需的八位位组数的计数。

注：因此，值“foo INTEGER (256..1234567)::= 256”的编码在 ALIGNED 变体中被编码为 00xxxxxx00000000，其中‘x’表示一个零填充位，它可能存在也可能不存在，这取决于长度在八位位组中出现的位置（例如，如果长度开始于一个八位位组的边界，则编码是00xxxxxx00000000，如果长度开始于一个八位位组的两个最低有效位（位2和位1），则编码是00 00000000）。

b) 不受约束的长度决定因子等于“len”（按 11.7 和 11.8 所确定的），如果 PER 可视约束不限制具有有限上下边界的类型，或者如果该类型是可扩展的，并且该值不在扩展根的范围内。

14 枚举类型的编码

注：（说明性）没有扩展标记的枚举类型被编码为一个受约束整数，其子类型约束不包含扩展标记。这意味着枚举类型在实际中，枚举类型差不多总是被编码为表示每个枚举所需的最少位数中的某个位字段。存在扩展标记时，如果该值不在扩展根中，则将其编码为正常小非负整个数。

14.1 枚举根中的若干枚举应根据其枚举值升序排列，然后应赋予第一个枚举以 0 开始的枚举索引，1 赋予第 2 个枚举，以此类推，直到排序表中的最后一个枚举。扩展附加枚举（始终按升序定义）应赋予从 0 开始的枚举索引，1 赋予给第 2 个枚举，以此类推，直到扩展附加枚举中的最后一个枚举。

注：GB/T 16262.1—AAAA要求每个连续的扩展附加枚举值应大于前一个。

14.2 如果在枚举类型定义中不存在扩展标记，则应编码枚举索引。其编码应被视为不存在扩展标记的受约束整型值，其中下边界为 0，而上边界为与该类型相关联的最大枚举索引，过程结束。

14.3 如果存在扩展标记，则单个位应添加到长度为 1 的位字段内的字段列表中。如果被编码值不在扩展根中，则应将该位置为 1，否则置为 0。对前一种情况，应按照 14.1 将附加枚举排序，该值应作为正常小非负整个数添加到字段列表中，该整数的值是附加枚举的枚举索引，并且其“lb”置为 0，过程结束。对后一种情况，就好像扩展标记不存在那样，该值应按照 14.2 的规定进行编码。

注：没有可适用于枚举类型的PER可视约束对这些编码规则来说是可视的。

15 实数类型的编码

注：（说明性）实数使用以长度决定因子开头的CER/DER内容八位位组，在实际中，该长度决定因子是单个八位位组。

15.1 如果抽象值的基数是 10，则编码值的基数应是 10，如果抽象值的基数是 2，则编码值的基数应该是 2。

15.2 应使用 GB/T 16263.1—EEEE 的 11.3 中的 CER 和 DER 所规定的 **REAL** 编码来给出某个位字段（在 ALIGNED 变体中的八位位组对齐），该位字段是 CER/DER 编码的内容八位位组。该编码的内容八位位组由“n”个八位位组构成，并且被放在“n”个八位位组的位字段（在 ALIGNED 变体中的八位位组对

齐）中。应引用 11.9 的过程把这一“n”个八位位组的位字段（在 ALIGNED 变体中的八位位组对齐）添加到字段列表上，前面是等于“n”的不受约束的长度决定因子。

16 位串类型的编码

注：（说明性）受约束于固定长度小于或等于16位的位串不会引起八位位组对齐,较大位串是在ALIGNED变体中的八位位组对齐的。如果长度被若干约束所固定，且上边界小于64K，则没有显式的长度编码，否则，所包含的长度编码可以采用任何一种前面规定的长度编码的形式，包括大型位串的分片。

16.1 PER 可视约束只能约束位串的长度。

16.2 在不存在 PER 可视约束，并且 GB/T 16262.1—AAAA 的 22.7 适用时，该值应编码为不带尾 0 位（注，这意味着不带 1 位的值总被编码为空位串）。

16.3 在存在 PER 可视约束，并且 GB/T 16262.1—AAAA 的 22.7（即位串类型是用“NamedBitList”定义）适用时，当有必要保证发送值的长度是能携带该值的最小长度，并且满足有效长度约束时，该值应是使用增加或去掉尾 0 位的编码。

16.4 设位串（在长度上由 PER 可视约束所确定）中的最大位数为“ub”，最少位数为“lb”。如果没有有限最大长度，则认为“ub”未设置。如果在最小长度上没有约束，则“lb”为 0，设要编码的实际位串值的长度为“n”位。

16.5 当位串值按 16.6 到 16.11 的规定放到某个位字段中时，位串值的引导位应放在该位字段的引导位上，并且位串值的末尾位应放在该位字段的末尾位上。

16.6 如果位串类型对 PER 编码具有可扩展（见 10.3.9），则应将一个由单个位组成的位字段添加到字段列表中。如果该编码的长度不在扩展根的范围内，则该位应被置 1，否则置 0。对前一种情况，应按照 16.11 将长度作为半受约束的整数加到字段列表中，其后是位串值。对后一种情况，长度和值应按扩展不存在的情况进行编码。

16.7 如果位串类型的长度约束规范中不存在扩展标记，则适用 16.8 到 16.11。

16.8 如果位串被约束为 0 长度(“ub”等于 0)，则不应对其进行编码（字段列表不添加位串），本条过程结束。

16.9 如果位串的所有值都被约束为同一长度(“ub”等于“lb”)，并且长度小于或等于 16 位，则位串应被放入到约束长度为“ub”的位字段中，该“ub”应被添加到无长度决定因子的字段列表上，本条过程结束。

16.10 如果位串的所有值都被约束为同一长度(“ub”等于“lb”)，并且该长度大于 16 位，但小于 64K 位，则位串应被放入到长度为“ub”(不必是 8 位的整数倍）的位字段（在 ALIGNED 变体中的八位位组对齐）中，且该位串应被添加到不带长度决定因子的字段列表上，本条过程结束。

16.11 如果 16.8 到 16.10 不适用，则位串应被放入到长度为“n”位的位字段（在 ALIGNED 变体中的八位位组对齐）中，且应根据 11.9 的过程把这个“n”位的位字段（在 ALIGNED 变体中的八位位组对齐）添加到字段列表中，如果设置了“ub”且小于 64K，则该字段列表前面是作为受约束的整数的等于“n”位的长度决定因子，或者，如果未设置“ub”，则该字段列表前面是作为半受约束的整数的等于“n”位的长度决定因子，“lb”如上所述。

注：对于16K、32K、48K或者64K位之后不受约束的或大型的“ub”，分片过程适用。

17 八位位组串类型的编码

注：（说明性）固定长度小于或等于两个八位位组的八位位组串不是八位位组对齐的。所有其他八位位组串都是在ALIGNED变体中八位位组对齐的。如果固定长度八位位组小于64K，则使用无长度八位位组进行编码。对于

XX

不受约束的八位位组串，该长度则被显式编码（必要时使用分片）。

17.1 PER 可视约束只能约束八位位组串的长度。

17.2 设八位位组串（在长度上由 PER 可视约束所确定）中八位位组的最大数目为“ub”，最小数目为“lb”。如果没有有限最大值，则认为“ub”未设置。如果在最小值上没有约束，则“lb”为 0。设要编码的实际八位位组串的值的长度为“n”个八位位组。

17.3 如果八位位组串类型对 PER 编码具有可扩展（参见 10.3.9），则应将由单个字段组成的位字段添加到字段列表中。如果该编码的长度不在扩展根的范围内，则该位应被置 1，否则置 0。对前一种情况，应按照 17.8 条将长度作为半受约束的整个数加到字段列表中，其后是八位位组串值。对后一种情况，长度和值应按扩展不存在的情况进行编码。

17.4 如果在八位位组串类型的约束规范中不存在扩展标记，则适用 17.5 至 17.8。

17.5 如果八位位组串被约束为 0 长度（“ub”等于 0）,则不应对其进行编码（对字段列表不添加八位位组），本条过程结束。

17.6 如果八位位组串的所有值都被约束为同一长度（“ub”等于“lb”），并且该长度小于或等于两个八位位组，则八位位组串应放到位数等于受约束长度为八倍“ub”的位字段中，该长度应被添加到不带长度决定因子的字段列表上，本条过程结束。

17.7 如果八位位组串的所有值都被约束为同一长度（“ub”等于“lb”），并且该长度大于两个八位位组，但小于 64K，则八位位组串应放到受约束长度为“ub”的位字段（在 ALIGNED 变体中的八位位组对齐）中，且该长度应被添加到不带长度决定因子的字段列表上，本条过程结束。

17.8 如果 17.5 到 17.7 不适用，则八位位组串应放到长度为“n”个八位位组的位字段（在 ALIGNED 变体中的八位位组对齐）中，且应按照 11.9 的过程把该“n”个八位位组的位字段（在 ALIGNED 变体中的八位位组对齐）添加到字段列表上，如果设置了“ub”，则该字段列表前面是作为受约束的整数的等于“n”个八位位组的长度决定因子，如果未设置“ub”，则该字段列表前面是作为半受约束的整个数的等于“n”个八位位组的长度决定因子。“lb”如上确定。

注：在16K、32K、48K或64K个八位位组之后，分片过程可能适用。

18 空值类型的编码

注：（说明性）空值类型基本上是一个位置占位符，仅在选择、可选集合或序列组件的情况下才有实际意义。在这些编码规则中，执行选择中空值的标识或其作为可选元素存在，而无需用八位位组表示空值。因此，空值从不构成编码的八位位组。

对于空值的字段列表不应有任何添加。

19 序列类型的编码

注：（说明性）序列类型以位图前导码开始。如果该序列类型没有扩展标记，则该位图只记录该类型中默认或可选组件是否存在，编码为固定长度的位字段。如果该序列类型有扩展标记，则该位图前有单个位，说明编码中扩展附加部分的值是否真的存在。只要前导码的长度小于64K位，其编码就不带任何长度决定因子，否则对长度决定因子进行编码以获得分片。该前导码后面是对每个组件依次编码的字段，如果存在扩展附加部分，则第1个被编码的扩展附加部分前面是被编码类型中的扩展附加数目计数的编码（作为通常很小的长度），后面是长度等于该计数的位图，记录每个扩展附加部分存在与否，其后是各扩展附加部分的编码，各扩展附加部分就像是开放类型字段的值。

19.1 在“ComponentTypeLists”或“SequenceType”中，如果序列类型具有扩展标记，则应首先将单个位添加到长度为 1 的位字段内的字段列表中。如果编码中存在扩展附加值，则该位应被置 1，否

则置0（该位在下文中称为“扩展位”）。如果在“ComponentTypeLists”或“SequenceType”中没有扩展标记，则不应添加扩展位。

19.2 如果序列类型在扩展根内有“n”个组件，标记为**OPTIONAL**或者**DEFAULT**，则应产生带有“n”位的单个位字段作为字段列表的附加部分。该位字段的位应依次对序列类型中每个可选或者默认组件的编码存在与否进行编码，位值为1应编码为组件的编码存在，位值为0编码为组件的编码不存在。前导码中的引导位应对第1个可选或者默认组件的存在与否进行编码，末尾位应对最后1个可选或者默认组件的存在与否进行编码。

19.3 如果“n”小于64K，则该位字段应添加到字段列表上。如果“n”大于或等于64K，则应按照11.9的过程把这个“n”位的位字段添加到字段列表上，其前面是作为受约束的整数的等于“n”位的长度决定因子，它的“ub”和“lb”都是“n”。

注：在这种情况下，“ub”和“lb”都将被长度过程忽略。这里引用这些过程是为了提供大前导码的分片。这种情况应该很少出现。

19.4 前导码后面应依次列出存在的序列值组件的字段列表。

19.5 对于CANONICAL-PER，如果要被编码的值是个默认值，则标记为DEFAULT的组件的编码应始终不存在。对于BASIC-PER，如果要被编码的值是个简单类型（见3.7.25)的默认值，则标记为DEFAULT的组件的编码应始终不存在，否则由发送器选择是否对其进行编码。

19.6 如果扩展位不存在或者扩展位为0，则编码完成。如果扩展位存在并且置为1，则下列过程适用。

19.7 设要被编码的类型中扩展附加部分的数目为“n”，则应产生“n”位的位字段，以便加到字段列表。该位字段的位应依次对要被编码的类型中每个附加部分编码的存在与否进行编码。位值为1是对扩展附加部分编码的存在进行编码，位值为0是对扩展附加部分编码的不存在进行编码。位字段中引导位应对第1个扩展附加部分的存在与否进行编码，末尾位应对最后一个扩展附加部分的存在与否进行编码。

注：如果声称与规范的特定版本一致，则值“n”总是等于该版本的扩展附加的数目。

19.8 应根据11.9的过程把该“n”位的位字段添加到字段列表上，其前面是作为正常小长度的等于“n”的长度决定因子。

注：“n”不能是0，因为该过程只在至少存在一个被编码的扩展附加部分时调用。

19.9 其后应依次列出包含每个存在的扩展附加部分的编码的字段列表。每个“ComponentType”(即，不是“ExtensionAdditionGroup”)的扩展附加部分应按11.2.1规定的那样进行编码，好像它是一个开放类型字段的值。每个“ExtensionAdditionGroup”的扩展附加部分应按19.2到19.6的规定编码为序列类型，然后再如11.2.1规定的那样进行编码，好像它是一个开放类型字段的值。如果“ExtensionAdditionGroup”的所有组件值都缺少，则“ExtensionAdditionGroup”应编码为缺少扩展附加部分（即在18.7中描述的位字段的相应位应置为0)

注1：如果“ExtensionAdditionGroup”含有标记为**OPTIONAL**或**DEFAULT**的组件，则“ExtensionAdditionGroup”用位图做前缀，表明每个标记为**OPTIONAL**或**DEFAULT**的组件值是否存在。

注2：在“扩展标记对“之后定义的“RootComponentTypeList”组件按其在扩展标记对前面直接定义的那样进行编码。

20 单一序列类型的编码

20.1 PER可视约束能约束单一序列类型组件的数目。

20.2 设单一序列中的最大组件数目（由 PER 可视约束确定）为“ub”，最小组件数目为“lb”。如果没有有限最大数目或者“ub”大于或等于 64K，我们说“ub”未设置。如果在最小值上没有约束，则“lb”的值为 0，设要被编码的实际单一序列值组件的数目为“n”个组件。

20.3 单一序列的每个组件的编码将生成许多字段，以便添加到单一序列类型的字段列表。

20.4 如果其中具有 PER 可视约束且扩展标记存在，则应将单个位添加到长度为 1 的位字段内的字段列表中，如果该编码的组件数目不在扩展根的范围内，则该位应置 1，否则置 0。对前一种情况，应引用 11.9 将长度决定因子作为半受约束的整个数添加到字段列表中，其后是组件值，对后一种情况，长度和值应按扩展标记不存在的情况进行编码。

20.5 如果组件数固定（“ub”等于“lb”），且“ub”小于 64K，则对于单一序列应没有长度决定因子，每个组件的字段应依次添加到该单一序列的字段列表上。

20.6 否则，应引用 11.9 将“n”个组件生成的字段列表添加到字段列表中，如果设置了“ub”，则在前面加一个等于“n”个组件的长度决定因子作为受约束的整个数，如果不设置“ub”，则作为半受约束的整个数，“lb”如上所述。

注1：在16K、32K、48K或64K个组件之后，分片过程可能适用。

注2：分片的断点在字段之间，在断点之前的位数不必是八的整数倍。

21 集合类型的编码

集合类型应将“RootComponentTypeList”中的元素按照GB/T 16262.1—AAAA中8.6规定的正则次序进行排序，另外，当一个或多个组件是无标签的选择类型时，为了决定组件编码的次序，每个无标签的选择类型按照它具有标签等于该选择类型或者任何被嵌套的无标签的选择类型的“RootComponentTypeList”中最小的标签进行排序。出现在“RootComponentTypeList”中的集合元素应被编码，就像它声明为序列类型一样，出现在“ExtensionAdditionList”中的集合元素应按照19.9条中制定的序列类型的组件进行编码（即它们按照定义的顺序进行编码）。

示例：IMPLICIT TAGS 的置标签环境假设如下：

```
A::=SET
{
    a [3]  INTEGER,
    b [1]  CHOICE,
    {
        c [2]  INTEGER,
        d [4]  INTEGER
    },
    e CHOICE
    {
        f CHOICE
        {
            g [5]  INTEGER,
            h [6]  INTEGER
        },
        i CHOICE
        {
            j [0]  INTEGER
```

```
        }
    }
}
```

被编码的集合组件的次序总是e、b、a，因为标签[0]排序最低，然后是[1]、[3]。

22 单一集合类型的编码

22.1 对于 CANONICAL-PER，单一集合类型组件值的编码应以升序出现，作为与位串比较的组件编码，其尾端用最多 7 个 0 位被填充到八位位组的边界，如果需要，则向比较短的位串增加 0 个八位位组以使其长度等于较长的位串。

注：为排序而增加的任何填充位或填充八位位组都不出现在实际的编码中。

22.2 对于 BASIC-PER，单一集合应像它已经被声明为单一序列类型那样进行编码。

23 选择类型的编码

注：（说明性）选择类型是通过编码指定的选定项的索引来进行编码，它被编码为一个受约束的整数（除非在选择类型中存在扩展标记，在这种情况下，它是一个正常小非负整个数），因此通常会占用编码该索引所需的最小位数的固定长度位字段（尽管在理论上它可以任意大），接下来是所选替代的编码，在扩展附加部分的选定项像它们是一个开放类型字段的值那样进行编码，只有唯一选定项时不编码索引。

23.1 选择类型的编码不受 PER 可视的影响。

23.2 选择的每个组件有一个相关的索引，在选择的根中的第 1 个选定项的索引值为 0（按 GB/T 16262.1—AAAA 中 8.4 规定的正则次序取选择项），第 2 个为 1，直到选择的扩展根中的最后一个组件。索引值以类似方式赋给“ExtensionAdditionAlternativesList”中的每个“NamedType”，从 0 开始，就像以扩展根的组件开始一样，设“n”为根中的最大索引值。

注：GB/T 16262.1 — AAAA 的 29.7 要求每个连续添加的扩展应有比最后添加到“ExtensionAdditionAlternativesList”中的标签值更大的标签值。

23.3 对于含有无标签选择的选择项的正则排序，每个无标签的选择类型应像它有一个标签等于该选择类型或者任意被嵌套的无标签选择类型的最小标签那样排序。

23.4 如果选择在扩展根中只有一个选择项，那么选择了该选择项，则不应对索引进行编码。

23.5 如果在“AlternativeTypeLists”中选择类型有一个扩展标记，则首先应将单个位添加到长度为 1 的位字段内的字段列表中，如果在编码中存在扩展附加值，则该位应置 1，否则置 0（该位在下列文本中称为“扩展位”），如果在“AlternativeTypeLists”中没有扩展标记，则应不增加扩展位。

23.6 如果不存在扩展位，选定项的选择索引应按照第 13 章的规定当作一个约束到范围 0 至“n”的整数值（在其子类型约束中没有扩展标记）编码到一个字段中，且应添加该字段到字段列表上，其后应是选定项的字段，本条过程结束。

23.7 如果存在扩展位，并且选定项位于扩展根内，则根据第 13 章，应编码该选定项的选择索引，本条过程结束。

23.8 如果扩展位存在，并且选定项不在扩展根内，则该选定项的选择索引应编码为一个正常小非负整个数，其“1b”置为 0，该字段应添加到字段列表上，其后应是含有选定项编码的字段，该选定项是 11.2 规定的开放类型字段的值，本条过程结束。

注：在选择扩展附加的版本括号不影响如何对“ExtensionAdditionAlternatives”进行编码。

24 客体标识符类型的编码

注：（说明性）客体标识符类型的编码使用前面有长度决定因子的BER的内容八位位组，在实际中长度是单个八位位组。

应使用为BER规定的编码给出一个位字段（在ALIGNED变体中的八位位组对齐），即BER编码的内容八位位组，该BER编码的内容八位位组由“n”个八位位组构成，被放入一个“n”个八位位组的位字段（在ALIGNED变体中的八位位组对齐）中，应按照11.9将这个位字段（在ALIGNED变体中的八位位组对齐）添加到字段列表上，其前面是等于“n”的长度决定因子，作为半受约束整数的八位位组计数。

25 相关客体标识符类型的编码

注：（说明性）相关客体标识符类型的编码使用前面有长度决定因子的BER内容八位位组，在实际中长度是单个八位位组，下列正文与24章中的正文相同。

应使用为BER规定的编码给出一个位字段（在ALIGNED变体中的八位位组对齐），即BER编码的内容八位位组，该BER编码的内容八位位组由“n”个八位位组构成，被放入一个“n”个八位位组的位字段（在ALIGNED变体中的八位位组对齐）中，应按照11.9将这个位字段（在ALIGNED变体中的八位位组对齐）添加到字段列表上，其前面是等于“n”的长度决定因子，作为半受约束整数的八位位组计数。

26 国际化资源引用类型的编码

注：（说明性）国际化资源引用类型的编码使用前面有长度决定因子的BER内容八位位组，下面的正文与24章的正文相同。

应使用为BER规定的编码给出一个位字段（在ALIGNED变体中的八位位组对齐），即BER编码的内容八位位组，该BER编码的内容八位位组由“n”个八位位组构成，被放入一个“n”个八位位组的位字段（在ALIGNED变体中的八位位组对齐）中，应按照11.9将这个位字段（在ALIGNED变体中的八位位组对齐）添加到字段列表上，其前面是等于“n”的长度决定因子，作为半受约束整数的八位位组计数。

27 相关国际化资源引用类型的编码

注：（说明性）相关国际化资源引用类型的编码使用前面有长度决定因子的BER内容八位位组。下面的正文与24章的正文相同。

应使用为BER规定的编码给出一个位字段（在ALIGNED变体中的八位位组对齐），即BER编码的内容八位位组，该BER编码的内容八位位组由“n”个八位位组构成，被放入一个“n”个八位位组的位字段（在ALIGNED变体中的八位位组对齐）中，应按照11.9将这个位字段（在ALIGNED变体中的八位位组对齐）添加到字段列表上，其前面是等于“n”的长度决定因子，作为半受约束整数的八位位组计数。

28 嵌入式 pdv 类型的编码

28.1 嵌入式 pdv 类型可以用下列方式编码：

a) 嵌入式 pdv 类型的选项 syntaxes 使用 PER 可视内部类型约束为单个值，或者选项 identification 使用 PER 可视内部类型约束为 fixed 选择项，在这种情况下，应只有 data-value 被编码，这称为“预定义”情况。

b) 内部类型约束既不用来约束 syntaxes 选项为单个值，也不约束 identification 为 fixed 选择项，在这种情况下，identification 和 data-value 都应被编码，这称为“通常”情况。

28.2 在“预定义”情况下，嵌入式 pdv 类型值的编码应是 OCTET STRING 类型值的 PER 编码，该 OCTET STRING 值应是按照 GB/T 16262.1—AAAA 的 36.3 a)中引用的单个数据值完整编码的八位位组。

28.3 在“通常”情况下，嵌入式 pdv 类型值的编码应按照 GB/T 16262.1—AAAA 的 36.5 中定义的类型的 PER 编码，同时去掉 data-value-descriptor 元素（即在 SEQUENCE 编码的头部应没有 OPTIONAL 位图），类型 OCTER STRING 的 data-value 组件的值应按照 GB/T 16262.1—AAAA 的 36.3 a)中的单个数据值完整编码所组成的八位位组。

29 外部类型值的编码

29.1 外部类型值的编码应是下列序列类型的 PER 编码，假设在 EXPLICIT TAGS 环境中定义的，其值在下面子条中规定：

```
[UNIVERSAL 8] IMPLICIT SEQUENCE {
        direct-reference            OBJECT IDENTIFIER OPTIONAL,
        indirect-reference          INTEGER OPTIONAL,
        date-value-descriptor       ObjectDescriptor OPTIONAL,
        encoding                    CHOINCE{
            single-ASN1-type        [0] ABSTRACT-SYNTAX.&Type,
            octet-aligned           [1] IMPLICIT OCTET STRING,
            arbitrary               [2] IMPLICIT BIT STRING}}
```

注：此序列类型不同于GB/T 16262.1—AAAA中的规定。

29.2 组件值依赖于被发送的抽象值，它是 GB/T 16262.1—AAAA 的 36.5 中定义的类型值。

29.3 当且仅当 data-value-descriptor 存在于抽象值中，且它们应有相同的值，上述 data-value-descriptor 才应存在。

29.4 上述 direct-reference 和 indirect-reference 的存在与否与表 1 一致，表 1 将 GB/T 16262.1—AAAA 的 36.5 中定义的 identification 的外部类型选项映射到 29.1 中定义的外部类型组件 direct-reference 和 indirect-reference 上。

表 1 identification 的替换编码

identification	direct-reference	indirect-reference
syntaxes	***不能出现***	***不能出现***
syntax	syntax	不存在
presentation-context-id	不存在	presentation-context-id
context-negotiation	传送语法	presentation-context-id
transfer-syntax	***不能出现***	***不能出现***
fixed	***不能出现***	***不能出现***

29.5 数据值应按照编码所标识的传送语法进行编码，并应放在下文规定的 encoding 选择的备选项中。

29.6 如果数据值是单个 ASN.1 数据类型值（见 29.7 的注），并且该数据值的编码规则是在本文件中规定的，则发送实现应使用 single-ASN1-type 选项。

29.7 否则，如果数据值使用同意或协商的编码进行编码，其编码是一个八位位组的整数，则发送实现应编码为 octet-aligned。

注：一个数据值，它是一系列ASN.1类型的值，并且传送语法为其规定了八位位组串的简单拼接，这些八位位组串是在每个ASN.1类型上应用ASN.1基本编码规则产生的，属于本类别，而不是29.6。

29.8 否则，如果数据值使用同意或协商的编码进行编码，其编码不是八位位组的整数倍，则 encoding 选择应是 arbitrary。

29.9 如果 encoding 选择选定为 single-ASN1-type，则 ASN.1 类型应按 11.2 的规定进行编码，其值等于要被编码的数据值。

注：可能出现在开放类型中的值的范围由与direct-reference相关联的客体标识符值和/或与indirect-reference相关联的整数值的注册来确定。

29.10 如果 encoding 选择是 octet-aligned，则该数据值应按照同意或协商的传送语法进行编码，其结果八位位组应形成八位位组串的值。

29.11 如果 encoding 选择是 arbitrary，则该数据值应按照同意或协商的传送语法进行编码，其结果形成位串的值。

30 受限字符串类型的编码

注1：(指导ALIGNED变体）小于或等于两个八位位组的固定长度字符串不是八位位组对齐的，被约束为最大长度小于两个八位位组的可变长度的字符串不是八位位组对齐的，所有其他的字符串在ALIGNED变体中都是八位位组对齐的。如果固定长度字符串的长度小于64K字符，则其编码在没有长度八位位组下进行，对不受约束的字符串或长于64K—1的受约束字符串，其长度被显式编码（必要时分片），每个NumericString、PrintableString、VisibleString(ISO646String)、IA5String、BMPSring、UniversalString字符被编码为能接纳有效允许字母表约束所允许的所有字符的2的最小幕的位数。

注2：(指导UNALIGNED变体）字符串不是八位位组对齐的。如果只有一种可能长度值，如果长度小于64K，则不存在长度编码。对不受约束的字符串或长于64K—1的受约束字符串，长度被显式编码（必要时分片）。每个NumericString、PrintableString、VisibleString(ISO646String)、IA5String、BMPSring、UniversalString字符被编码为能接纳有效允许字母表约束所允许的所有字符的最小位数。

注3：(指导每个被编码字符的长度）每个字符的编码依赖于有效允许字母表约束（见10.3.12)，该约束定义类型使用的字母表。假设该字母表由字符集ALPHA构成，对于每个已知倍数字符串类型（见3.7.16)，有一个与每个字符相关联的整数值，其通过引用该受限字符串类型相关联的某个编码表获得，与字符集ALPHA对应的值集BETA用来确定所使用的编码，过程如下：每个字符编码的位数仅由值集BETA（或字符集ALPHA）中元素的个数N来确定，对于UNALIGNED变体，它是能把值N—1编码为非负二进制整数的最小位数，对于ALIGNED变体，它是2的幕且能把值N—1编码的最小位数。假设选择的位数是B，如果在值集BETA中的每个值都可以编码（无变换时）在B位中，则用值集BETA中的值来表示字符集ALPHA中相应的字符。否则，将值集BETA中的值按照升序取出，用值0、1、2，依此类推，直到N-1所代替，用这些值表示对应的字符。总之，总是使用最小位数（对于ALIGNED变体取下一个2的幕），给出使用与这些字符正常关联的值的优先权，但是如果这些值不能按最小位数编码，则使用紧缩编码。

30.1 下列受限字符串类型是已知倍数字符串类型：NumericString、PrintableString、VisibleString(ISO646String)、IA5String、BMPSring、UniversalString，有效允许字母表约束只对这些类型是PER可视的。

30.2 有效长度约束记法可以确定抽象字符串长度的上边界“aub”，否则“aub”不设置。

30.3 有效长度约束记法可以确定抽象字符串长度的非0下边界“a1b”，否则“a1b”为0。

注：PER可视约束只适用于已知倍数字符串类型，对于其他受限字符串类型，将不设置“aub”，且“a1b”为0。

30.4 如果类型对PER编码可扩展(见10.3.18)，则应将由单个位构成的位字段添加到字段列表上。如果该值处在扩展根范围内，则该单个位应置0，否则置1。如果该值处在扩展根的范围外，则下列的编码应是好像不存在有效长度约束一样，且应具有10.3.12中规定的有效允许字母表约束。

注1：只有已知倍数字符串类型对PER编码可扩展，其他字符串类型上的可扩展标记对PER编码没有影响。

注2：有效允许字母表约束不可以扩展，因为可扩展的允许字母表约束是不可见的（见10.3.11）。

30.5 本条适用于已知倍数字符串，其他受限字符串类型的编码在30.6规定。

30.5.1 有效允许字母表定义为被允许字母表约束所允许的字母表，或者如果没有允许的字母表约束，则定义为内置类型的完整字母表。

30.5.2 设N为有效允许字母表中的字符数，设B为2的B次幂大于或等于N的最小整数，设B2为大于等于B的2的最小幂，那么，在ALIGNED变体中，每个字符应编码为B2个位，在UNALIGNED变体中为B位，设此规则所标识出的位数为“b”。

30.5.3 通过如下引用GB/T 16262.1—AAAA中的第43章，使一个数值“v”同每个字符相关联。对于UniversalString，该值用来确定GB/T 16262.1—AAAA的43.3中的正则顺序（该值在0～2^{32}-1之间）的值；对于BMPString，该值用来确定43.3中的正则顺序（该值在0～2^{16}-1之间）的值；对于NumericString和PrintableString、VisibleString(ISO646String)、IA5String，该值是为ISO/IEC 646相应字符的编码定义的值(对于IA5String，该值的范围在0～127之间；对于VisibleString，该值的范围在32～126之间；对于NumericString，该值的范围在32～57之间；对PrintableString，该值的范围在32～122之间；对于IA5String和VisibleString，范围内的所有值都存在。但是，对于PrintableString和NumericString不是范围内的所有值都被使用)。

30.5.4 设允许字母表中字符集范围内的最小值为“1b”，最大值为“ub”，则进入“b”位中的字符编码是如下标识出的值“v”的非负二进制整数编码：

a) 如果“ub”小于或等于2b-1，则“v”是上述规定的值;

b) 字符按GB/T 16262.1—AAAA的第43章规定的正则顺序加以放置，第1个字符赋予的值为0，正则顺序的下一个字符赋予的值比上一个字符赋予的值大1，这些赋予的值是“v”。

注：上面a)项不适用于受约束或不受约束的NumericString字符，它总是使用b)编码为4位或者更少位数。

30.5.5 应通过将每个字符（使用合适的“v”值）编码为非负二进制整数进入“b”位中，将其拼接形成b″位的整数倍的位字段，来获得完整字符串的编码。

30.5.6 如果“aub”等于“a1b”并且小于64K，如果“aub”乘以“b”大于16，则该位字段应作为位字段（在ALIGNED变体中的八位位组对齐）添加到位字段列表上，否则，应作为非八位位组对齐的位字段添加到位字段列表上，本条过程结束。

30.5.7 如果“aub”不等于“a1b”，或者大于或等于64K，则应引用11.9的过程在位字段之前添加“n”长度决定因子作为字符串中的字符计数，同时长度决定因子的下边界为“a1b”，上边界为

“aub”。如果“aub”乘以“b”大于或等于16，则该位字段作为一个字段（在ALIGNED变体中的八位位组对齐）添加到位字段列表上，否则应作为非八位位组对齐的位字段添加到位字段列表上，本条过程结束程。

注：30.5.6和30.5.7都规定若“aub”乘以“b”小于16，则不对齐，若大于16，则对齐。对正好等于16的值，30.5.6规定不对齐，30.5.7规定对齐。

30.6 本条适用于未知倍数字符串的字符串。在这种情况下，约束从不是PER可视的，该类型对PER编码也从不可扩展。

30.6.1 对于BASIC-PER，下面对于“基本编码”的引用意思是在GB T 16263.1的8.23.5中规定的BER编码的八位位组。对于CANONICAL-PER，它的意思是在GB/T 16263.1—EEEE的11.4中为CER和DER编码规定的八位位组。

30.6.2 “基本编码”应适用于给出“n”个八位位组字段的字符串。

30.6.3 应引用11.9，在前面添加一个作为八位位组计数的不受限的长度决定因子“n”并且“n”个八位位组的字段应作为一个位字段（在ALIGNED变体中的八位位组对齐）添加，本条过程结束。

31 不受限字符串类型的编码

31.1 不受限字符串类型可以用下列两种方式编码：

a) 不受限字符串类型的选项 **syntaxes** 使用PER可视内部类型约束约束为单个值，或者 **identification** 使用PER可视内部类型约束约束为 **fixed** 选项，在这种情况下，应只编码 **string-value**，这称为“预定义”情况。

b) 内部类型约束既不用来约束 **syntaxes** 选项为单个值，也不约束 **identification** 为 **fixed** 选项，在这种情况下，**identification** 和 **string-value** 都应被编码，这称为“一般”情况。

31.2 对于“预定义”情况，**CHARACTER STRING** 类型值的编码应是 **OCTET STRING** 类型值的PER编码，**OCTET STRING** 类型值是构成GB/T 16262.1—AAAA的44.3 a)中引用的字符串值完整编码的八位位组。

31.3 对于“一般”情况，不受限字符串类型值的编码是GB/T 16262.1—AAAA的44.5中定义的类型的PER编码同时去掉 **data-value-descriptor** 组件（即在 **SEQUENCE** 编码的头部应没有 **OPTIONAL** 位图）。类型 **OCTER STRING** 的 **data-value** 组件的值应是构成GB/T 16262.1—AAAA的44.3 a)中的字符串值完整编码的八位位组。

32 时间类型、有效时间类型、已定义时间类型和附加时间类型的编码

32.1 常规情况

32.1.1 有效时间类型、已定义时间类型和附加时间类型的编码应由这些类型的抽象值的属性设置来决定。有用的和已定义的时间类型的抽象值的属性设置分别在GB/T 16262.1—AAAA的38.4和附录B中规定，附加时间类型的抽象值的属性设置由父类的属性设置决定，适用于每个可见约束限制（见10.3.13）。

32.1.2 如果要编码的类型的所有抽象值都具有表2的第2列行中列出的属性设置之一，则该类型应被编码为具有约束的类型（如果有），即已被表2的第3列对应行中指定的类型所取代。否则，应按照32.11中的规定进行编码。

注：如果表2中没有为特定行列出时间属性（例如**Midnight**），则对其设置没有限制。

32.1.3 该类型的所有抽象值都必须在Fn中具有相同的n值，才能适用表2中第24行至第32行。

32.1.4 表 2 的第 3 列中指定的类型是在 32.2 到 32.10 定义的(使用 ASN.1 符号)，并被假设是在 AUTOMATIC TAGS 的环境中定义。

注 1：在 PER 编码规范中使用这些类型的引用名称并不能使它们可以在 ASN.1 中的应用程序设计器中使用，在这样的规范中也没有保留的词语。但是，在删除-ENCODING 后，它们对应于在 GB/T 16262.1—AAAA 的 38.4 和附录 B 中指定的有用的时间类型或已定义的时间类型的名称。

注 2：所有有效的和已定义时间类型都满足表 2 中其中一行的条件，因此已经优化了编码，附加时间类型可能满足其中一行的条件，但将按照 32.11 中的规定进行编码，无约束的 TIME 类型按照 32.11 中的规定进行编码。

表 2　具有指定属性设置的所有抽象值的时间子类型的编码

行数	属性设置	ASN.1 类型的编码
1	“Basic=Date Date=C Year=Basic” or “Basic=Date Date=CYear=Proleptic”	CENTURY-ENCODING (见 32.2.1)
2	“Basic=Date Date=C Year=Negative” or “Basic=Date Date=C Year=L*n*” (for any *n*)	ANY-CENTURY-ENCODING (见 32.2.2)
3	“Basic=Date Date=Y Year=Basic” or “Basic=Date Date=Y Year=Proleptic”	YEAR-ENCODING (见 32.2.3)
4	“Basic=Date Date=Y Year=Negative” or “Basic=Date Date=Y Year=Ln” (for any n)	ANY-YEAR-ENCODING (见 32.2.4)
5	"Basic=Date Date=YM Year=Basic” or “Basic=Date Date=YM Year=Proleptic”	YEAR-MONTH-ENCODING (见 32.2.5)
6	“Basic=Date Date=YM Year=Negative” or “Basic=Date Date=YM Year=L*n*” (for any *n*)	ANY-YEAR-MONTH-ENCODING (见 32.2.6)
7	“Basic=Date Date=YMD Year=Basic” or “Basic=Date Date=YMD Year=Proleptic”	DATE-ENCODING (见 32.2.7)
8	“Basic=Date Date=YMD Year=Negative” or “Basic=Date Date=YMD Year=L*n*” (for any *n*)	ANY-DATE-ENCODING (见 32.2.8)
9	"Basic=Date Date=YD Year=Basic” or “Basic=Date Date=YD Year=Proleptic”	YEAR-DAY-ENCODING (见 32.2.9)
10	“Basic=Date Date=YD Year=Negative” or “Basic=Date Date=YD Year=Ln” (for any n)	ANY-YEAR-DAY-ENCODING (见 32.2.10)

表 2　具有指定属性设置的所有抽象值的时间子类型的编码（续表）

行数	属性设置	ASN.1 类型的编码
11	“Basic=Date Date=YW Year=Basic” or “Basic=Date Date=YW Year=Proleptic”	YEAR-WEEK-ENCODING (见 32.2.11)
12	“Basic=Date Date=YW Year=Negative” or “Basic=Date Date=YW Year=Ln” for any n)	ANY-YEAR-WEEK-ENCODING (见 32.2.12)
13	“Basic=Date Date=YWD Year=Basic” or “Basic=Date Date=YWD Year=Proleptic”	YEAR-WEEK-DAY-ENCODING (见 32.2.13)
14	“Basic=Date Date=YWD Year=Negative” or “Basic=Date Date=YWD Year=Ln” (for any n)	ANY-YEAR-WEEK-DAY-ENCODING (见 32.2.14)
15	“Basic=Time Time=H Local-or-UTC=L”	HOURS-ENCODING (见 32.3.1)
16	“Basic=Time Time=H Local-or-UTC=Z”	HOURS-UTC-ENCODING (见 32.3.2)
17	“Basic=Time Time=H Local-or-UTC=LD"	HOURS-AND-DIFF-ENCODING (见 32.3.3)
18	“Basic=Time Time=HM Local-or-UTC=L”	MINUTES-ENCODING (见 32.3.4)
19	“Basic=Time Time=HM Local-or-UTC=Z”	MINUTES-UTC-ENCODING (见 32.3.5)
20	“Basic=Time Time=HM Local-or-UTC=LD”	MINUTES-AND-DIFF-ENCODING (见 32.3.6)
21	“Basic=Time Time=HMS Local-or-UTC=L”	TIME-OF-DAY-ENCODING (见 32.3.7)
22	“Basic=Time Time=HMS Local-or-UTC=Z”	TIME-OF-DAY-UTC-ENCODING (见 32.3.8)
23	“Basic=Time Time=HMS Local-or-UTC=LD”	TIME-OF-DAY-AND-DIFF-ENCODING (见 32.3.9)
24	“Basic=Time Time=HFn Local-or-UTC=L” (但见 32.1.3)	HOURS-AND-FRACTION-ENCODING (见 32.3.10)
25	“Basic=Time Time=HFn Local-or-UTC=Z” (但见 32.1.3)	HOURS-UTC-AND-FRACTION-ENCODING (见 32.3.11)
26	“Basic=Time Time=HFn Local-or-UTC=LD” (但见 32.1.3)	HOURS-AND-DIFF-AND-FRACTION-ENCODING (见 32.3.12)
27	“Basic=Time Time=HMFn Local-or-UTC=L” (但见 32.1.3)	MINUTES-AND-FRACTION-ENCODING (见 32.3.13)

表 2　具有指定属性设置的所有抽象值的时间子类型的编码（续表）

行数	属性设置	ASN.1 类型的编码
28	“Basic=Time Time=HMFn Local-or-UTC=Z" (但见 32.1.3)	MINUTES-UTC-AND-FRACTIO （见 32.3.14）
29	“Basic=Time Time=HMFn Local-or-UTC=LD” (但见 32.1.3)	MINUTES-AND-DIFF-AND-FRACTION-ENCODING (见 32.3.15)
30	“Basic=Time Time=HMSFn Local-or-UTC=L” (但见 32.1.3)	TIME-OF-DAY-AND-FRACTION-ENCODING (见 32.3.16)
31	“Basic=Time Time=HMSFn Local-or-UTC=Z” (但见 32.1.3)	TIME-OF-DAY-UTC-AND-FRACTION-ENCODING (见 32.3.17)
32	“Basic=Time Time=HMSFn Local-or-UTC=LD” (但见 32.1.3)	TIME-OF-DAY-AND-DIFF-AND-FRACTION ENCODING (见 32.3.18)
33	“Basic=Date-Time” 所有抽象值都必须具有“Basic=Date”第 7、8、9、10、13 和 14 行中指定的相同的附加属性设置，以及“Basic=Time”第 15 到 32 行中指定的相同的附加属性设置。	DATE-TIME-ENCODING {Date-Type, Time-Type} (如 32.4.1 中的实例化)
34	"Basic=Interval Interval-type=SE SE-point=Date” 所有抽象值都必须具有在第 1 行到第 14 行中为“Basic=Date”指定的相同的附加属性设置	START-END-DATE-INTERVAL-ENCODING {Date-Type} (见 32.5.1)
35	“Basic=Interval Interval-type=SE SE-point=Time” 所有抽象值都必须具有在“Basic=Time”的第 15 到 32 行中指定相同的附加属性设置。	START-END-TIME-INTERVAL-ENCODING {Time-Type} (见 32.5.2)
36	“Basic=Interval Interval-type=SE SE-point=Date-Time” 所有抽象值都必须具有在“Basic=Date”的第 7、8、9、10、13 和 14 行中指定的相同的附加属性设置，以及在“Basic=Time”的第 15 到 32 行中指定的相同的附加属性设置。	START-END-DATE-TIME-INTERVAL-ENCODING {Date-Type, Time-Type} (见 32.5.3)
37	“Basic=Interval Interval-type=D”	DURATION-INTERVAL-ENCODING (见 32.6.1)
38	“Basic=Interval Interval-type=SD SE-point=Date” 所有抽象值都必须具有在第 1 行到第 14 行中为“Basic=Date”指定的相同的附加属性设置。	START-DATE-DURATION-INTERVAL-ENCODING {Date-Type} (见 32.7.1)
39	“Basic=Interval Interval-type=SD SE-point=Time” 所有抽象值都必须具有在“Basic=Time”的第 15 到 32 行中指定相同的附加属性设置。	START-TIME-DURATION-INTERVAL-ENCODING {Time-Type} (见 32.7.2)

表 2　具有指定属性设置的所有抽象值的时间子类型的编码（续表）

行数	属性设置	ASN.1 类型的编码
40	“Basic=Interval Interval-type=SD SE-point=Date-Time” 所有抽象值都必须具有在“Basic=Date”的第 7、8、9、10、13 和 14 行中指定的相同的附加属性设置，以及在“Basic=Time”的第 15 到 32 行中指定的相同的附加属性设置。	START-DATE-TIME-DURATION INTERVAL-ENCODING {Date-Type, Time-Type} （见 32.7.3）
41	“Basic=Interval Interval-type=DE SE-point=Date” 所有抽象值都必须具有在“Basic=Date”的第 1 到第 14 行中指定的相同的附加属性。	DURATION-END-DATE-INTERVAL-ENCODING {Date-Type}
42	“Basic=Interval Interval-type=DE SE-point=Time” 所有抽象值都必须具有在“Basic=Time”的第 15 到 32 行中指定的相同的附加属性。	DURATION-END-TIME-INTERVAL-ENCODING {Time-Type} （见 32.7.5）
43	“Basic=Interval Interval-type=DE SE-point=Date-Time” 所有抽象值必须具有在“Basic=Date”的第 7、8、9、10、13 和 14 行中指定的相同的附加属性，以及在“Basic=Time”的第 15 到 32 行中指定的相同的附加属性设置。	DURATION-END-DATE-TIME-INTERVAL ENCODING {Date-Type, Time-Type} （见 32.7.6）
44	“Basic=Rec-Interval Interval-type=SE SE-point=Date” 所有抽象值都必须具有在“Basic=Date”的第 1 行到第 14 行中指定的相同的附加属性设置。	REC-START-END-DATE-INTERVAL-ENCODING {Date-Type} （见 32.8.1）
45	“Basic=Rec-Interval Interval-type=SE SE-point=Time” 所有抽象值都必须具有在“Basic=Time”的第 15 到 32 行中指定相同的附加属性设置。	REC-START-END-TIME-INTERVAL-ENCODING {Time-Type} （见 32.8.2）
46	“Basic=Rec-Interval Interval-type=SE SE-point=Date-Time” 所有抽象值都必须具有在“Basic=Date”的第 7、8、9、10、13 和 14 中指定的相同的附加属性设置，以及在“Basic=Time”的第 15 到 32 行中指定的相同的附加属性设置。	REC-START-END-DATE-TIME-INTERVAL ENCODING {Date-Type, Time-Type} （见 32.8.3）
47	“Basic=Rec-Interval Interval-type=D"	REC-DURATION-INTERVAL-ENCODING （见 32.9.1）
48	“Basic=Rec-Interval Interval-type=SD SE-point=Date” 所有抽象值都必须具有在“Basic=Date”的第 1 到 14 行中指定的相同的附加属性设置。	REC-START-DATE-DURATION-INTERVAL ENCODING {Date-Type} （见 32.10.1）

表 2　具有指定属性设置的所有抽象值的时间子类型的编码（续表）

行数	属性设置	ASN.1 类型的编码
49	“Basic=Rec-Interval Interval-type=SD SE-point=Time” 所有抽象值都必须具有在“Basic=Time”的第 15 到 32 行中指定相同的附加属性设置。	REC-START-TIME-DURATION-INTERVAL ENCODING {Time-Type} (见 32.10.2)
50	“Basic=Rec-Interval Interval-type=SD SE-point=Date-Time” 所有抽象值都必须具有在“Basic=Date”的第 7、8、9、10、13 和 14 行中指定的相同的附加属性设置，以及在“Date-Time”的第 15 到 32 行中指定的相同的附加属性设置。	REC-START-DATE-TIME-DURATION-INTERVAL ENCODING {Date-Type, Time-Type} (见 32.10.3)
51	“Basic=Rec-Interval Interval-type=DE SE-point=Date” 所有抽象值都必须具有在“Basic=Date”的第 1 到 14 行中指定的相同的附加属性。	REC-DURATION-END-DATE-INTERVAL-ENCODING {Date-Type}
52	“Basic=Rec-Interval Interval-type=DE SE-point=Time” 所有抽象值都必须具有在“Basic=Time”的第 15 到 32 行中指定的相同的附加属性。	REC-DURATION-END-TIME-INTERVAL-ENCODING {Time-Type} (见 32.10.5)
53	“Basic=Rec-Interval Interval-type=DE SE-point=Date-Time” 所有抽象值都必须具有在“Basic=Date”的第 7、8、9、10、13 和 14 行中指定的相同的附加属性，以及在“Basic=Time”的第 15 到 32 行中指定的相同的附加属性设置。	REC-DURATION-END-DATE-TIME-INTERVAL ENCODING {Date-Type, Time-Type} (见 32.10.6)

32.2　使用“Basic=Date”属性设置对子类型进行编码

本文件规定了表2第3列中引用ASN.1的类型，该类型中所有抽象值都具有“Basic=Date”属性设置。

32.2.1　CENTURY-ENCODING 类型为：

```
CENTURY-ENCODING ::= INTEGER (0..99)  --  7 bits
```

将整数值设置为由抽象值的年份分量的前两个数字所指定的值。

32.2.2　ANY-CENTURY-ENCODING 类型为：

```
ANY-CENTURY-ENCODING ::= INTEGER (MIN..MAX)
```

如果将整数值设置为由抽象值的年份分量指定的值，则忽略最后两个数字。

32.2.3　YEAR-ENCODING 类型为：

```
YEAR-ENCODING ::= CHOICE {  --  2 bits for choice determinant
        immediate       INTEGER (2005..2020),  --  4 bits
        near-future     INTEGER (2021..2276),  --  8 bits
        near-past       INTEGER (1749..2004),  --  8 bits
```

```
                remainder          INTEGER (MIN..1748 | 2227 ..MAX)}
```

将整数值设置为抽象值的年份分量。

注：通常情况下已被优化为提供6位或10位编码。

32.2.4 ANY-YEAR-ENCODING 类型为：

```
ANY-YEAR-ENCODING ::= INTEGER (MIN..MAX)
```

将整数值设置为抽象值的年份分量。

32.2.5 YEAR-MONTH-ENCODING 类型为：

```
YEAR-MONTH-ENCODING ::= SEQUENCE {
          year       YEAR-ENCODING,
          month      INTEGER (1..12) -- 4 bits -- }
```

根据32.2.3设置的YEAR-ENCODING，将month整数值设置为抽象值的月份分量。

注：通常情况下已被优化为提供10位或14位编码。

32.2.6 ANY-YEAR-MONTH-ENCODING 类型为：

```
ANY-YEAR-MONTH-ENCODING ::= SEQUENCE {
          year     ANY-YEAR-ENCODING,
          month    INTEGER (1..12) }
```

根据32.2.4设置的ANY-YEAR-ENCODING，将month整数值设置为抽象值的月份分量。

32.2.7 DATE-ENCODING 类型为：

```
DATE-ENCODING ::= SEQUENCE {
          year     YEAR-ENCODING,
          month    INTEGER (1..12), -- 4 bits
          day      INTEGER (1..31) -- 5 bits -- }
```

根据32.2.3设置的YEAR-ENCODING，将month整数值设置为抽象值的月份分量，day整数值设置为抽象值的日分量。

32.2.8 ANY-DATE-ENCODING 类型为：

```
ANY-DATE-ENCODING ::= SEQUENCE {
          year     ANY-YEAR-ENCODING,
          month    INTEGER (1..12),
          day      INTEGER (1..31)}
```

根据32.2.4设置的ANY-YEAR-ENCODING编码，将month整数值设置为抽象值的月份分量，day整数值设置为抽象值的日分量。

32.2.9 YEAR-DAY-ENCODING 类型为：

```
YEAR-DAY-ENCODING ::= SEQUENCE {
          year     YEAR-ENCODING,
          day      INTEGER (1..366)}
```

根据32.2.3设置YEAR-ENCODING，将day整数值设置为抽象值的日分量。

32.2.10 ANY-YEAR-DAY-ENCODING 类型为：

```
ANY-YEAR-DAY-ENCODING ::= SEQUENCE {
          year     ANY-YEAR-ENCODING,
          day      INTEGER (1..366)}
```

根据32.2.4设置的ANY-YEAR-ENCODING，将day整数值设置为抽象值的日分量。

32.2.11 YEAR-WEEK-ENCODING 类型为：

```
YEAR-WEEK-ENCODING ::= SEQUENCE {
```

```
        year     YEAR-ENCODING,
        week     INTEGER (1..53) -- 6 bits --}
```

根据32.2.3设置的YEAR-ENCODING，将week整数值设置为抽象值的周分量。

注：通常情况下已被优化为提供12位或16位编码。

32.2.12 ANY-YEAR-WEEK-ENCODING 类型为：

```
ANY-YEAR-WEEK-ENCODING ::= SEQUENCE {
        year       ANY-YEAR-ENCODING,
        week       INTEGER (1..53)}
```

根据32.2.4设置ANY-YEAR-ENCODING，将week整数值设置为抽象值的周分量。

32.2.13 YEAR-WEEK-DAY-ENCODING 类型为：

```
YEAR-WEEK-DAY-ENCODING ::= SEQUENCE {
        year       YEAR-ENCODING,
        week       INTEGER (1..53), -- 6 bits
        day        INTEGER (1..7) -- 3 bits -- }
```

根据32.2.3设置YEAR-ENCODING，将week整数值设置为抽象值的周分量，day整数值设置为抽象值的日分量。

32.2.14 ANY-YEAR-WEEK-DAY-ENCODING 类型为：

```
ANY-YEAR-WEEK-DAY-ENCODING ::= SEQUENCE {
        year     ANY-YEAR-ENCODING,
        week     INTEGER (1..53),
        day      INTEGER (1..7)}
```

根据32.2.4设置ANY-YEAR-ENCODING，将week整数值设置为抽象值的周分量，day整数值设置为抽象值的日分量。

32.3 使用“Basic=Time”属性设置对子类型进行编码

本文件规定了在表2第3列中引用ASN.1的类型，该类型中所有抽象值都具有Basic=Time属性设置。

32.3.1 HOURS-ENCODING 类型为：

```
HOURS-ENCODING ::= INTEGER(0..24)  --  5 bits
```

将整数值设置为抽象值的小时分量。

注：此类型已经被优化为提供一个5位编码。

32.3.2 HOURS-UTC-ENCODING 类型为：

```
HOURS-UTC-ENCODING ::= INTEGER(0..24)  --  5 bits
```

将整数值设置为抽象值的小时分量。

注：此类型已经被优化为提供一个5位编码。

32.3.3 HOURS-AND-DIFF-ENCODING 类型为：

```
HOURS-AND-DIFF-ENCODING::= SEQUENCE {
        local-hours      INTEGER (0..24),
        time-difference TIME-DIFFERENCE }
```

在哪里：

```
TIME-DIFFERENCE ::= SEQUENCE {
        sign         ENUMERATED { positive, negative },
        hours        INTEGER (0..15),
        minutes        INTEGER (1..59) OPTIONAL }
```

将local-hours整数值设置为抽象值的局部时间的小时分量，将time-difference设置为抽象值的时间差分量的符号、小时和分钟。如果时间差的分钟分量为零，则应省略TIME-DIFFERENCE minutes。

32.3.4 MINUTES-ENCODING 类型为：

```
MINUTES-ENCODING :: = SEQUENCE {
        Hours       INTEGER (0..24),   -- 5 bits
        Minutes     INTEGER (0..59)   -- 5 bits -- }
```

将hours整数值设置为抽象值的小时分量，将minutes整数值设置为分钟分量。

注：此类型已经被优化为提供一个10位编码。

32.3.5 MINUTES-UTC-ENCODING 类型为：

```
MINUTES-UTC-ENCODING ::= SEQUENCE {
        hours     INTEGER (0..24),   -- 5 bits
        minutes     INTEGER (0..59)   -- 5 bits -- }
```

将hours整数值设置为抽象值的小时分量，将minutes整数值设置为分钟分量。

注：此类型已经被优化为提供一个10位编码。

32.3.6 MINUTES-AND-DIFF-ENCODING 类型为：

```
MINUTES-AND-DIFF-ENCODING ::= SEQUENCE {
        local-time    SEQUENCE {
            hours            INTEGER (0..24),
            minutes            INTEGER (0..59) },
            time-difference    TIME-DIFFERENCE }
```

将local-hours设置为抽象值的本地时间的小时和分钟分量，将time-difference设置为抽象值的时间差分量的符号、小时和分钟，如32.3.3中规定。

32.3.7 TIME-OF-DAY-ENCODING 类型为：

```
TIME-OF-DAY-ENCODING ::= SEQUENCE {
        hours     INTEGER (0..24),   -- 5 bits
        minutes     INTEGER (0..59),   -- 5 bits
        seconds     INTEGER (0..60)   -- 5 bits --}
```

将hours整数值设置为抽象值的小时分量，将minutes整数值设置为分钟分量，将seconds整数值设置为秒分量。

注：此类型已经被优化为提供一个15位编码。

32.3.8 TIME-OF-DAY-UTC-ENCODING 类型为：

```
TIME-OF-DAY-UTC-ENCODING ::= SEQUENCE {
        hours     INTEGER (0..24),   -- 5 bits
        minutes   INTEGER (0..59),   -- 5 bits
        seconds   INTEGER (0..60),   -- 5 bits --}
```

将hours整数值设置为抽象值的小时分量，将minutes整数值设置为分钟分量，将seconds整数值设置为秒分量。

注：此类型已经被优化为提供一个15位编码。

32.3.9 TIME-OF-DAY-AND-DIFF-ENCODING 类型为：

```
TIME-OF-DAY-AND-DIFF-ENCODING ::= SEQUENCE {
        local-time SEQUENCE {
          hours     INTEGER (0..24),
          minutes   INTEGER (0..59),
```

```
        seconds    INTEGER (0..60) },
        time-difference        TIME-DIFFERENCE }
```

将local-hours设置为抽象值的本地时间的小时、分钟和秒分量，将time-difference设置为抽象值的时差分量的符号、小时和分钟，如32.3.3中规定。

32.3.10 HOURS-AND-FRACTION-ENCODING 类型为：

```
HOURS-AND-FRACTION-ENCODING ::= SEQUENCE {
        hours      INTEGER (0..24), -- 5 bits
        fraction     INTEGER (0..999, ..., 1000..MAX)
            -- 11 bits，精度最高可达 3 位数  -- }
```

将hours整数值设置为抽象值的小时分量，将minutes整数值设置为fraction小时乘以10^n，其中N是分数部分中指定的位数。

注：此类型已经被优化为提供一个16位，最高达3位精度的编码。

32.3.11 HOURS-UTC-AND-FRACTION-ENCODING 类型为:

```
HOURS-UTC-AND-FRACTION-ENCODING ::= SEQUENCE {
        hours    INTEGER (0..24), -- 5 bits
        fraction  INTEGER (0..999, ..., 1000..MAX)
        -- 11 bits，精度最高可达 3 位数  -- }
```

将 hours 整数值设置为抽象值的小时分量，将 minutes 整数值设置为 fraction 小时乘以10^n，其中 N 是分数部分中指定的位数。

注：此类型已经被优化为提供一个16位，最高达3位精度的编码。

32.3.12 HOURS-AND-DIFF-AND-FRACTION-ENCODING 类型为:

```
HOURS-AND-DIFF-AND-FRACTION-ENCODING ::= SEQUENCE {
        local-hours    INTEGER (0..24), -- 5 bits
        fraction       INTEGER (0..999, ..., 1000..MAX)
        -- 11 bits，精度最高可达 3 位数  --,
        time-difference      TIME-DIFFERENCE }
```

将local-hours整数值设置为抽象值的本地时间的小时分量，将fraction整数值设置为分数小时乘以10^n（其中N是分数部分中指定的数字数），将time-difference设置为32.3.3中指定的时差分量抽象值的符号、小时和分钟。

32.3.13 MINUTES-AND-FRACTION-ENCODING 类型为:

```
MINUTES-AND-FRACTION-ENCODING ::= SEQUENCE {
        hours    INTEGER (0..24), -- 5 bits
        minutes  INTEGER (0..59), -- 5 bits
        fraction  INTEGER (0..999, ..., 1000..MAX)
        --  11 bits，精度最高可达 3 位数  -- }
```

将hours整数值设置为抽象值的小时分量，将minutes整数值设置为分钟分量，将fraction整数值设置为分数小时乘以10^n，其中N是分数部分中指定的位数。

注：此类型已经被优化为提供一个21位，最高达3位精度的编码。

32.3.14 MINUTES-UTC-AND-FRACTION-ENCODING 类型为:

```
MINUTES-UTC-AND-FRACTION-ENCODING ::= SEQUENCE {
        hours    INTEGER (0..24), -- 5 bits
        minutes  INTEGER (0..59), -- 5 bits
        fraction INTEGER (0..999, ..., 1000..MAX)
```

```
        --  11 bits, 精度最高可达 3 位数  --}
```

将hours整数值设置为抽象值的小时分量，将minutes整数值设置为分钟分量，将fraction整数值设置为分数小时乘以10^n（其中N是分数部分中指定的位数）。

注：此类型已经被优化为提供一个21位，最高达3位精度的编码。

32.3.15 MINUTES-AND-DIFF-AND-FRACTION-ENCODING 类型为:

```
MINUTES-AND-DIFF-AND-FRACTION-ENCODING ::= SEQUENCE {
        local-time  SEQUENCE {
        hours    INTEGER (0..24),
        minutes  INTEGER (0..59),
        fraction   INTEGER (0..999, ..., 1000..MAX)},
        time-difference TIME-DIFFERENCE }
```

将local-hours设置为抽象值的本地时间的小时和分钟分量，将fraction整数值设置为分数分钟乘以10^n（其中N是分数部分中指定的位数），将time-difference设置为32.3.3中指定的抽象值的时差分量的符号、小时和分钟。

32.3.16 TIME-OF-DAY-AND-FRACTION-ENCODING 类型为:

```
TIME-OF-DAY-AND-FRACTION-ENCODING ::= SEQUENCE {
        hours    INTEGER (0..24), -- 5 bits
        minutes  INTEGER (0..59), -- 5 bits
        seconds  INTEGER (0..60), -- 5 bits --
        fraction  INTEGER (0..999, ..., 1000..MAX)
        --  11 bits, 精度最高可达 3 位数  --}
```

将hours整数值设置为抽象值的小时分量，将minutes整数值设置为分钟分量，将seconds整数值设置为秒分量，将fraction整数值设置为分数秒乘以10^n，其中N是分数部分中指定的位数。

注：此类型已经被优化为提供一个26位编码。

32.3.17 TIME-OF-DAY-UTC-AND-FRACTION-ENCODING 类型为:

```
TIME-OF-DAY-UTC-AND-FRACTION-ENCODING ::= SEQUENCE {
        hours    INTEGER (0..24), -- 5 bits
        minutes    INTEGER (0..59), -- 5 bits
        seconds  INTEGER (0..60), -- 5 bits --
        fraction  INTEGER (0..999, ..., 1000..MAX)
        -- 11 bits, 精度最高可达 3 位数  --}
```

将hours整数值设置为抽象值的小时分量，将minutes整数值设置为分钟组件，将seconds整数值设置为秒分量，将fraction整数值设置为分数秒乘以10^n，其中N是分数部分中指定的位数。

注：此类型已经被优化为提供一个26位编码。

32.3.18 TIME-OF-DAY-AND-DIFF-AND-FRACTION-ENCODING 类型为:

```
TIME-OF-DAY-AND-DIFF-AND-FRACTION-ENCODING ::= SEQUENCE {
          local-time  SEQUENCE {
          hours    INTEGER (0..24),
          minutes  INTEGER (0..59),
          seconds  INTEGER (0..60),
          fraction  INTEGER (0..999, ..., 1000..MAX)},
        time-difference     TIME-DIFFERENCE }
```

将local-time设置为抽象值的局部时间的小时、分钟、秒和分数部分分量，将time-difference设置为抽象值的时间差分量的符号、小时和分钟，如32.3.3中规定。

32.4 使用“Basic=Date-Time”属性设置对子类型进行编码

本文件规定了在表2第3列中引用的ASN.1类型，该类型中所有抽象值都具有“Basic=Date-Time”属性设置。

32.4.1 DATE-TIME-ENCODING 类型为：

```
DATE-TIME-ENCODING {Date-Type, Time-Type} ::= SEQUENCE {
        date      Date-Type,
        time      Time-Type}
```

32.4.2 编码应是该类型的实例化编码，Date-Type 和 Time-Type 实际参数设置为表 2 第列 3 中“Basic=Date”和“Basic=Time”行（分别）指定的类型，该行指定该类型的所有抽象值的附加属性设置。

注：通常情况下，此类型已被优化为提供32位编码。

32.5 使用“Basic= Interval Interval-type=SE”属性设置对子类型进行编码

本文件规定了在表2第3列中引用ASN.1的类型，该类型中所有抽象值都具有“Basic= Interval Interval-type=SE”属性设置。

32.5.1 START-END-DATE-INTERVAL-ENCODING 类型为：

```
START-END-DATE-INTERVAL-ENCODING {Date-Type} ::= SEQUENCE {
        start     Date-Type,
        end       Date-Type}
```

编码应是该类型的实例化编码，Date-Type实际参数设置为表2第3列中“Basic=Date”行指定的类型，该行指定该类型的所有抽象值的附加属性设置，start分量应设置为开始日期，end分量应设置为时间间隔的结束日期。

32.5.2 START-END-TIME-INTERVAL-ENCODING 类型为：

```
START-END-TIME-INTERVAL-ENCODING {Time-Type} ::= SEQUENCE {
        start     Time-Type,
        end       Time-Type}
```

编码应是该类型的实例化编码，Time-Type实际参数设置为表2第3列中“Basic=Time”行指定的类型，该行指定该类型的所有抽象值的附加属性设置，start分量应设置为开始时间，end分量应设置为时间间隔的结束时间。

32.5.3 START-END-DATE-TIME-INTERVAL-ENCODING 类型为：

```
START-END-DATE-TIME-INTERVAL-ENCODING {Date-Type, Time-Type} ::=
        SEQUENCE {
          start   DATE-TIME-ENCODING {Date-Type, Time-Type},
          end     DATE-TIME-ENCODING {Date-Type, Time-Type}}
```

编码应是该类型的实例化编码，Date-Type 和 Time-Type 实际参数设置为表 2 第 3 列中“Basic=Date”和“Basic=Time”行（分别）指定的类型，该行指定该类型的所有抽象值的附加属性设置，start分量应设置为开始日期-时间（如32.4中规定），end分量应设置为间隔的结束日期-时间。

32.6 使用“Basic= Interval Interval-type=D”属性设置对子类型进行编码

本文件规定了在表2第3列中引用ASN.1的类型，该类型中所有抽象值都具有“Basic= Interval Interval-type=D”属性设置。

32.6.1 DURATION-INTERVAL-ENCODING 类型为:

```
DURATION-INTERVAL-ENCODING ::= SEQUENCE { -- 8 bits 可选
        years           INTEGER (0..31, ..., 32..MAX) OPTIONAL,
                          -- 5 bits，最长可达 31 年
        months          INTEGER (0..15, ..., 16..MAX) OPTIONAL,
                          -- 4 bits，最长可达 15 个月
        weeks           INTEGER (0..63, ..., 64..MAX) OPTIONAL,
                          -- 6 bits，最长可达 63 年周
        days            INTEGER (0..31, ..., 32..MAX) OPTIONAL,
                          -- 5 bits，最长可达 31 年天
        hours           INTEGER (0..31, ..., 32..MAX) OPTIONAL,
                          -- 5 bits，最长可达 31 小时
        minutes         INTEGER (0..63, ..., 64..MAX) OPTIONAL,
                          -- 6 bits，最长可达 63 分钟
        seconds         INTEGER (0..63, ..., 64..MAX) OPTIONAL,
                          -- 6 bits，最长可达 63 秒
        fractional-part   SEQUENCE {
                  number-of-digits    INTEGER(1..3, ..., 4..MAX),
                    -- 3 bits，精度最高可达 3 位数
                  fractional-value    INTEGER(0..999, ..., 1000..MAX)
                    -- 11 bits，精度最高可达 3 位数
                            } OPTIONAL }
```

32.6.2 当且仅当 years、months、days、hours、minutes 和 seconds 分量全部缺失时，weeks 分量才应存在。

注：这反映了在DURATION抽象值的定义中使用时间元素的限制。

32.6.3 如果抽象值的时间元素分量为零，并且没有分数部分，则没有 DURATION-INTERVAL-ENCODING 的相应分量，除非该时间元素是抽象值中最不显著的时间元素。如果抽象值的时间元素的值为零，并且是抽象值中最不重要的时间元素，或者具有分数部分，则相应的分量应以 DURATION-INTERVAL-ENCODING 的形式出现，值为 0。

注：这就确保了编码是规范的。

32.6.4 如果没有任何时间元素的分数部分，则不存在 DURATION-INTERVAL-ENCODING 的 fractional-part，否则应设置为 32.6.5 中规定的分数部分（最不重要的时间元素）。

32.6.5 分数部分中的位数应以 number-of-digits 表示。如果位数为 N，则分数部分的值乘以10^n，得到的整数值为 fractional-value。

注 1：解码器可以从这些编码中恢复原始的分数部分，包括任何末尾的 0

注 2：对于抽象值中只有少数非零时间元素，且时间元素的值较小的情况，这种编码进行了优化，在简单的情况下会出现小于 16 位的编码。

32.7 使用“Basic= Interval Interval-type=SD”或者“Basic= Interval Interval-type=DE”属性设置对子类型进行编码

本文件规定了在表2第3列中引用ASN.1的类型，该类型中所有抽象值都具有“Basic= Interval Interval-type=SD”或者“Basic= Interval Interval-type=DE”属性设置。

32.7.1 START-DATE-DURATION-INTERVAL-ENCODING 类型为：

```
START-DATE-DURATION-INTERVAL-ENCODING {Date-Type} ::= SEQUENCE {
        start       Date-Type,
        duration      DURATION-INTERVAL-ENCODING}
```

编码应是该类型的实例化编码，Date-Type实际参数设置为表2第3列中“Basic=Date”行指定的类型，该行指定该类型的所有抽象值的附加属性设置，start分量应设置为开始日期，duration分量应设置为时间间隔的持续时间（如32.6中规定）。

32.7.2 START-TIME-DURATION-INTERVAL-ENCODING 类型为：

```
START-TIME-DURATION-INTERVAL-ENCODING {Time-Type} ::= SEQUENCE {
        start       Time-Type,
        duration      DURATION-INTERVAL-ENCODING }
```

编码应是该类型的实例化编码，Time-Type实际参数设置为表2第3列中“Basic=Time”行指定的类型，该行指定该类型的所有抽象值的附加属性设置，start分量应设置为开始时间，duration分量应设置为时间间隔的持续时间（如32.6中规定）。

32.7.3 START-DATE-TIME-DURATION-INTERVAL-ENCODING 类型为：

```
START-DATE-TIME-DURATION-INTERVAL-ENCODING {Date-Type, Time-Type} ::=
        SEQUENCE {
          start     DATE-TIME-ENCODING {Date-Type, Time-Type},
          duration  DURATION-INTERVAL-ENCODING }
```

编码应是该类型的实例化编码，Date-Type和Time-Type实际参数设置为表2第3列中“Basic=Date”和“Basic=Time”行（分别）指定的类型，该行指定该类型的所有抽象值的附加属性设置，start分量应设置为开始日期-时间（如32.4中规定），duration分量应设置为时间间隔的持续时间（如32.6中规定）。

32.7.4 DURATION-END-DATE-INTERVAL-ENCODING 类型为：

```
DURATION-END-DATE-INTERVAL-ENCODING {Date-Type} ::= SEQUENCE {
        duration    DURATION-INTERVAL-ENCODING,
        end       Date-Type }
```

编码应是该类型的实例化编码，Date-Type实际参数设置为表2第列3中“Basic=Date”行指定的类型，该行指定该类型的所有抽象值的附加属性设置，duration分量应设置为时间间隔的持续时间（如32.6中规定），end分量应设置为结束日期。

32.7.5 DURATION-END-TIME-INTERVAL-ENCODING 设置为：

```
DURATION-END-TIME-INTERVAL-ENCODING {Time-Type} ::= SEQUENCE {
        duration    DURATION-INTERVAL-ENCODING,
        end       Time-Type }
```

编码应是该类型的实例化编码，Time-Type实际参数设置为表2第3列中“Basic=Time”行指定的类型，该行指定该类型的所有抽象值的附加属性设置，duration分量应设置为时间间隔的持续时间（如32.6中规定），end分量应设置为结束时间。

32.7.6 DURATION-END-DATE-TIME-INTERVAL-ENCODING 设置为：

```
DURATION-END-DATE-TIME-INTERVAL-ENCODING {Date-Type, Time-Type}::=
    SEQUENCE {
        duration    DURATION-INTERVAL-ENCODING,
```

```
        end         DATE-TIME-ENCODING {Date-Type, Time-Type}}
```

编码应是该类型的实例化编码，Date-Type和Time-Type实际参数设置为表2第3列中“Basic=Date”和“Basic=Time”行（分别）指定的类型，该行指定该类型的所有抽象值的附加属性设置，duration分量应设置为时间间隔的持续时间（如32.6中规定），end分量应设置为结束日期-时间（如32.4中规定）。

32.8 使用“Basic= Interval Interval-type=SE”属性设置对子类型进行编码

本文件规定了在表2第3列中引用ASN.1的类型，该类型中所有抽象值都具有“Basic= Interval Interval-type=SE”属性设置。

32.8.1 REC-START-END-DATE-INTERVAL-ENCODING设置为：

```
REC-START-END-DATE-INTERVAL-ENCODING {Date-Type} ::= SEQUENCE {
        recurrence     INTEGER OPTIONAL,
        start          Date-Type,
        end            Date-Type}
```

编码应是该类型的实例化编码，Date-Type实际参数设置为表2第3列中“Basic=Date”行指定的类型，该行指定该类型的所有抽象值的附加属性设置。在抽象值中出现无限数量的递归时，应缺少recurrence分量，否则应设置为递归次数。start分量应设置为开始日期，end分量应设置为时间间隔的结束日期。

32.8.2 REC-START-END-TIME-INTERVAL-ENCODING设置为：

```
REC-START-END-TIME-INTERVAL-ENCODING {Time-Type} ::=
    SEQUENCE {
        recurrence     INTEGER OPTIONAL,
        start          Time-Type,
        end            Time-Type}
```

编码应为该类型的实例化编码，Time-Type实际参数设置为表2第3列中“Basic=Time”行指定的类型，该行指定该类型的所有抽象值的附加属性设置。在抽象值中出现无限数量的递归时，应缺少recurrence分量，否则应设置为递归次数。start分量应设置为开始时间，end分量应设置为时间间隔的结束时间。

32.8.3 REC-START-END-DATE-TIME-INTERVAL-ENCODING设置为：

```
REC-START-END-DATE-TIME-INTERVAL-ENCODING {Date-Type, Time-Type} ::=
    SEQUENCE {
        recurrence     INTEGER OPTIONAL,
        start          DATE-TIME-ENCODING {Date-Type, Time-Type},
        end            DATE-TIME-ENCODING {Date-Type, Time-Type}}
```

编码应为该类型的实例化编码，Date-Type和Time-Type实际参数设置为表2第3列中“Basic=Date”和“Basic=Time”行（分别）指定的类型，该行指定该类型的所有抽象值的附加属性设置。在抽象值中出现无限数量的递归时，应缺少recurrence分量，否则应设置为递归次数。start分量应设置为开始日期-时间（如32.4中规定），end分量应设置为循环间隔的结束日期-时间。

32.9 使用“Basic= Rec-Interval Interval-type=D”属性设置对子类型进行编码

本文件规定了在表2第3列中引用ASN.1的类型，该类型中所有抽象值都具有“Basic=Rec-Interval Interval-type=D”属性设置。

32.9.1 REC-DURATION-INTERVAL-ENCODING设置为：

```
REC-DURATION-INTERVAL-ENCODING ::= SEQUENCE {
        recurrence     INTEGER OPTIONAL,
        duration     DURATION-INTERVAL-ENCODING}
```

32.9.2 对于抽象值中出现无限数量的递归，应缺少 recurrence 分量，否则应设置为递归次数。duration 分量应设置为循环间隔的持续时间（如 32.6 中规定）。

32.10 使用“Basic= Rec-Interval Interval-type=SD”或者“Basic=Rec-Interval Interval-type=DE”属性设置对子类型进行编码

本文件规定了在表2第3列中引用ASN.1的类型，该类型中所有抽象值都具有“Basic=Rec-Interval Interval-type=SD”或者“Basic=Rec-Interval Interval-type=DE”属性设置。

32.10.1 REC-START-DATE-DURATION-INTERVAL-ENCODING 类型设置为：

```
REC-START-DATE-DURATION-INTERVAL-ENCODING {Date-Type} ::= SEQUENCE {
        recurrence       INTEGER OPTIONAL,
        start          Date-Type,
        duration         DURATION-INTERVAL-ENCODING}
```

编码应是该类型的实例化编码，Date-Type实际参数设置为表2第3列中“Basic=Date”行指定的类型，该行指定该类型的所有抽象值的附加属性设置。在抽象值中出现无限数量的递归时，应缺少recurrence分量，否则应设置为递归次数。start分量应设置为开始日期，duration分量应设置为间隔的持续时间（如32.6中规定）。

32.10.2 REC-START-TIME-DURATION-INTERVAL-ENCODING 类型为：

```
REC-START-TIME-DURATION-INTERVAL-ENCODING {Time-Type} ::= SEQUENCE {
        recurrence       INTEGER OPTIONAL,
        start          Time-Type,
        duration         DURATION-INTERVAL-ENCODING }
```

编码应是该类型的实例化编码，Time-Type实际参数设置为表2第3列中“Basic=Time”行指定的类型，该行指定该类型的所有抽象值的附加属性设置。在抽象值中出现无限数量的递归时，应缺少recurrence递归分量，否则应设置为递归次数。start分量应设置为开始时间，duration分量应设置为时间间隔的持续时间（如32.6中规定）。

32.10.3 REC-START-DATE-TIME-DURATION-INTERVAL-ENCODING 类型为：

```
REC-START-DATE-TIME-DURATION-INTERVAL-ENCODING {Date-Type, Time-Type} ::=
        SEQUENCE {
          recurrence     INTEGER OPTIONAL,
          start          DATE-TIME-ENCODING {Date-Type, Time-Type},
          duration       DURATION-INTERVAL-ENCODING }
```

编码应是该类型的实例化编码，Date-Type和Time-Type实际参数设置为表2第3列中“Basic=Date”和“Basic=Time”行（分别）指定的类型，该行指定该类型的所有抽象值的附加属性设置。在抽象值中出现无限数量的递归时，应缺少recurrence分量，否则应设置为递归次数。start分量应设置为开始日期-时间（如32.4中规定），duration分量应设置为循环间隔的持续时间（如32.6中规定）。

32.10.4 REC-DURATION-END-DATE-INTERVAL-ENCODING 类型为：

```
REC-DURATION-END-DATE-INTERVAL-ENCODING {Date-Type} ::= SEQUENCE {
        recurrence          INTEGER OPTIONAL,
        duration            DURATION-INTERVAL-ENCODING,
```

```
        end             Date-Type }
```

编码应是该类型的实例化编码，Date-Type实际参数设置为表2第列3中“Basic=Date”行指定的类型，该行指定该类型的所有抽象值的附加属性设置。在抽象值中出现无限数量的递归时，应缺少recurrence分量，否则应设置为递归次数。duration分量应设置为时间间隔的持续时间（如32.6中规定），end分量应设置为结束日期。

32.10.5 REC-DURATION-END-TIME-INTERVAL-ENCODING 类型为：

```
REC-DURATION-END-TIME-INTERVAL-ENCODING {Time-Type} ::= SEQUENCE {
        recurrence          INTEGER OPTIONAL,
        duration            DURATION-INTERVAL-ENCODING,
        end               Time-Type }
```

编码应是该类型的实例化编码，Time-Type实际参数设置为表2第3列中“Basic=Time”行指定的类型，该行指定该类型的所有抽象值的附加属性设置。在抽象值中出现无限数量的递归时，应缺少recurrence分量，否则应设置为递归次数。duration分量应设置为间隔的持续时间（如32.6中规定），而end分量应设置为结束时间。

32.10.6 REC-DURATION-END-DATE-TIME-INTERVAL-ENCODING 类型为：

```
REC-DURATION-END-DATE-TIME-INTERVAL-ENCODING {Date-Type, Time-Type} ::=
        SEQUENCE {
        recurrence     INTEGER OPTIONAL,
        duration       DURATION-INTERVAL-ENCODING,
        end          DATE-TIME-ENCODING {Date-Type, Time-Type}}
```

编码应是该类型的实例化编码，Date-Type和Time-Type实际参数设置为表2第3列中“Basic=Date”和“Basic=Time”行（分别）指定的类型，该行指定该类型的所有抽象值的附加属性设置。在抽象值中出现无限数量的递归时，应缺少recurrence分量，否则应设置为递归次数。duration分量应设置为时间间隔的持续时间（如32.6中规定），end分量应设置至结束日期-时间（如32.4中规定）。

32.11 使用 Basic 属性的混合设置对子类型进行编码

本文件规定了TIME类型和该类型子集的编码，这些子集的抽象值并不都具有相同的Basic属性设置，或者在表2中没有适用的行（例如，由于使用了多个准确性，参见32.1.3），它定义并使用类型DATE-TYPE、TIME-TYPE和MIXED-ENCODING（参见32.11.5到32.11.7），这些类型是使用ASN.1在之前的部分规定了的类型。

32.11.1 对于 TIME 类型的所有抽象值，表 2 中只有一行，其中第 2 列中指定的属性设置与第 2 列中列出的所有属性设置的抽象值的属性设置相匹配（抽象值可能具有第 2 列中未列出的其他属性设置），这被称为主确定行。

32.11.2 如果主确定行是第 33、34、36、38、40、41、43、44、46、48、50、51 或 53 行，则要求附加属性与第 1 至 14 行中指定的附加属性匹配，适用的第 1 行至第 14 行被称为日期确定行。

32.11.3 如果主确定行是第 33、35、36、39、40、42、43、45、46、49、50、52 或 53 行，则要求附加属性与第 15 至 32 行中指定的附加属性匹配，适用的第 15 行至第 32 行被称为时间确定行。

32.11.4 在 DATE-TYPE、TIME-TYPE 和 MIXED-ENCODING 类型中，如果日期确定行、时间确定行或主确定行（分别）为第 n 行，则应选择第 n 行替代方案。

32.11.5 抽象值的编码应为 MIXED-ENCODING 的编码：

```
MIXED-ENCODING ::= CHOICE {
        row-1    CENTURY-ENCODING,
```

```
row-2    ANY-CENTURY-ENCODING,
row-3    YEAR-ENCODING,
row-4    ANY-YEAR-ENCODING,
row-5    YEAR-MONTH-ENCODING,
row-6    ANY-YEAR-MONTH-ENCODING,
row-7    DATE-ENCODING,
row-8    ANY-DATE-ENCODING,
row-9    YEAR-DAY-ENCODING,
row-10     ANY-YEAR-DAY-ENCODING,
row-11     YEAR-WEEK-ENCODING,
row-12     ANY-YEAR-WEEK-ENCODING,
row-13     YEAR-WEEK-DAY-ENCODING,
row-14     ANY-YEAR-WEEK-DAY-ENCODING,
row-15     HOURS-ENCODING,
row-16     HOURS-UTC-ENCODING,
row-17     HOURS-AND-DIFF-ENCODING,
row-18     MINUTES-ENCODING,
row-19     MINUTES-UTC-ENCODING,
row-20     MINUTES-AND-DIFF-ENCODING,
row-21     TIME-OF-DAY-ENCODING,
row-22     TIME-OF-DAY-UTC-ENCODING,
row-23     TIME-OF-DAY-AND-DIFF-ENCODING,
row-24     FRACTIONAL-TIME{HOURS-AND-FRACTION-ENCODING},
row-25     FRACTIONAL-TIME{HOURS-UTC-AND-FRACTION-ENCODING},
row-26     FRACTIONAL-TIME{HOURS-AND-DIFF-AND-FRACTION-
               ENCODING},
row-27     FRACTIONAL-TIME{MINUTES-AND-FRACTION-ENCODING},
row-28     FRACTIONAL-TIME{MINUTES-UTC-AND-FRACTION-ENCODING},
row-29     FRACTIONAL-TIME{MINUTES-AND-DIFF-AND-FRACTION-
               ENCODING},
row-30     FRACTIONAL-TIME{TIME-OF-DAY-AND-FRACTION-ENCODING},
row-31     FRACTIONAL-TIME{TIME-OF-DAY-UTC-AND-FRACTION-
               ENCODING},
row-32     FRACTIONAL-TIME{TIME-OF-DAY-AND-DIFF-AND-FRACTION-
               ENCODING},
row-33     DATE-TIME-ENCODING {DATE-TYPE, TIME-TYPE},
row-34     START-END-DATE-INTERVAL-ENCODING {DATE-TYPE},
row-35     START-END-TIME-INTERVAL-ENCODING {TIME-TYPE},
row-36     START-END-DATE-TIME-INTERVAL-ENCODING
               {DATE-TYPE, TIME-TYPE},
row-37     DURATION-INTERVAL-ENCODING,
row-38     START-DATE-DURATION-INTERVAL-ENCODING {DATE-TYPE},
row-39     START-TIME-DURATION-INTERVAL-ENCODING {TIME-TYPE},
```

```
        row-40    START-DATE-TIME-DURATION-INTERVAL-ENCODING
                      {DATE-TYPE, TIME-TYPE},
        row-41    DURATION-END-DATE-INTERVAL-ENCODING {DATE-TYPE},
        row-42    DURATION-END-TIME-INTERVAL-ENCODING {TIME-TYPE},
        row-43    DURATION-END-DATE-TIME-INTERVAL-ENCODING
                      {DATE-TYPE, TIME-TYPE},
        row-44    REC-START-END-DATE-INTERVAL-ENCODING {DATE-TYPE},
        row-45    REC-START-END-TIME-INTERVAL-ENCODING {TIME-TYPE},
        row-46    REC-START-END-DATE-TIME-INTERVAL-ENCODING
                      {DATE-TYPE, TIME-TYPE},
        row-47    REC-DURATION-INTERVAL-ENCODING,
        row-48    REC-START-DATE-DURATION-INTERVAL-ENCODING {DATE-TYPE},
        row-49    REC-START-TIME-DURATION-INTERVAL-ENCODING {TIME-TYPE},
        row-50    REC-START-DATE-TIME-DURATION-INTERVAL-ENCODING
                      {DATE-TYPE, TIME-TYPE},
        row-51    REC-DURATION-END-DATE-INTERVAL-ENCODING {DATE-TYPE},
        row-52    REC-DURATION-END-TIME-INTERVAL-ENCODING {TIME-TYPE},
        row-53    REC-DURATION-END-DATE-TIME-INTERVAL-ENCODING
                      {DATE-TYPE, TIME-TYPE} }
```

其中每个备选方案类型的编码应按照表2第3列中主确定行指定的子条的规定。

32.11.6 FRACTIONAL-TIME 定义如下：

```
FRACTIONAL-TIME{Time-Type} ::= SEQUENCE {
number-of-digits INTEGER (1..MAX),
time-value Time-Type}
```

number-of-digits对抽象值的分数部分的数字进行编码。

32.11.7 DATE-TYPE 为：

```
DATE-TYPE ::= CHOICE {
        row-1    CENTURY-ENCODING,
        row-2    ANY-CENTURY-ENCODING,
        row-3    YEAR-ENCODING,
        row-4    ANY-YEAR-ENCODING,
        row-5    YEAR-MONTH-ENCODING,
        row-6    ANY-YEAR-MONTH-ENCODING,
        row-7    DATE-ENCODING,
        row-8    ANY-DATE-ENCODING,
        row-9    YEAR-DAY-ENCODING,
        row-10     ANY-YEAR-DAY-ENCODING,
        row-11     YEAR-WEEK-ENCODING,
        row-12     ANY-YEAR-WEEK-ENCODING,
        row-13     YEAR-WEEK-DAY-ENCODING,
        row-14     ANY-YEAR-WEEK-DAY-ENCODING }
```

其中每个备选方案类型的编码应按照表2第3列中日期确定行确定的子条的规定。

32.11.8 TIME-TYPE 为：

```
TIME-TYPE ::= SEQUENCE {
        number-of-digits   INTEGER (1..MAX) OPTIONAL,
        time-type CHOICE {
        row-15     HOURS-ENCODING,
        row-16     HOURS-UTC-ENCODING,
        row-17     HOURS-AND-DIFF-ENCODING,
        row-18     MINUTES-ENCODING,
        row-19     MINUTES-UTC-ENCODING,
        row-20     MINUTES-AND-DIFF-ENCODING,
        row-21     TIME-OF-DAY-ENCODING,
        row-22     TIME-OF-DAY-UTC-ENCODING,
        row-23     TIME-OF-DAY-AND-DIFF-ENCODING,
        row-24     HOURS-AND-FRACTION-ENCODING,
        row-25     HOURS-UTC-AND-FRACTION-ENCODING,
        row-26     HOURS-AND-DIFF-AND-FRACTION-ENCODING,
        row-27     MINUTES-AND-FRACTION-ENCODING,
        row-28     MINUTES-UTC-AND-FRACTION-ENCODING,
        row-29     MINUTES-AND-DIFF-AND-FRACTION-ENCODING,
        row-30     TIME-OF-DAY-AND-FRACTION-ENCODING,
        row-31     TIME-OF-DAY-UTC-AND-FRACTION-ENCODING,
        row-32     TIME-OF-DAY-AND-DIFF-AND-FRACTION-ENCODING}}
```

其中每个备选方案类型的编码应按照表2第3列中时间确定行确定的子条的规定。

32.11.9　当且仅当 time-type 替代方案是第 24 行到第 32 行之一时，Time-Type 中应显示 number-of-digits，将对抽象值小数部分的位数进行编码。

33　传送语法的客体标识符

33.1　本文件规定的编码规则可以在需要为单个 ASN.1 类型所有值规定无歧义位串表示时被引用和应用。

33.2　下列客体标识符，OID（分配了 Unicode 标签）和客体描述值被赋予用来标识和描述本文件规定的编码规则：

对BASIC-PER,ALIGNED变体：

{joint-iso-itu-t asn1(1) packed-encoding(3) basic(0) aligned(0)}

"/ASN.1/Packed-Encoding/Basic/Aligned"

"Packed encoding of a single ASN.1 type(basic aligned)"

对BASIC-PER,UNALIGNED变体：

{joint-iso-itu-t asn1(1) packed-encoding(3) basic(0) unaligned(1)}

"/ASN.1/Packed-Encoding/Basic/Unaligned"

"Packed encoding of a single ASN.1 type(basic unaligned)"

对CANONICAL-PER,ALIGNED变体：

{joint-iso-itu-t asn1(1) packed-encoding(3) canonical(1) aligned(0)}"

"/ASN.1/Packed-Encoding/Canonical/Aligned"

"Packed encoding of a single ASN.1 type(canonical aligned) "

XX

对CANONICAL-PER,UNALIGNED变体：

{joint-iso-itu-t asn1(1) packed-encoding(3) canonical(1) unaligned(1)}

"/ASN.1/Packed-Encoding/Canonical/Unaligned"

"Packed encoding of a single ASN.1 type(canonical unaligned)"

33.3 如果应用标准把抽象语法定义为抽象值集合，其值是使用 ASN.1 记法定义的某个特定命名的 ASN.1 类型的值，则 33.2 中指定的客体标识符值可以与抽象语法名一起使用，以识别由于把本文件中规定的编码规则应用到定义抽象记法时使用的特定命名的 ASN.1 类型而产生的传输语法。

33.4 如果 33.3 中抽象语法定义的条件不满足，则 33.2 中规定的名字不应与抽象语法名一起用来标识传输语法。

附 录 A
（资料性）
编码的示例

本附录通过展示使用ASN.1定义的（假设的）个人记录的八位位组表示，来说明本文件规定的紧缩编码规则的应用。

A.1 不使用子类型约束的记录

A.1.1 记录结构的ASN.1描述

下面使用GB/T 16262.1—AAAA规定的用于定义类型的ASN.1正式描述假定的人事记录的结构，它与GB/T 16263.1附录A中定义的示例相同。

```
PersonnelRecord ::= [APPLICATION 0 ] IMPLICIT SET {
    name            Name,
    title             [0] VisbleString,
    number          EmployeeNumbe,
    dateOfHire        [1] Date,
    nameOfSpouse    [2] Name,
    children          [3] IMPLICIT
        SEQUENCE OF ChildInformation    DEFAULT{}}

ChildInformation ::= SET
     {name            Name,
    dateOfBirth       [0] Date}

Name ::= [APPLICATION 1 ] IMPLICIT SEQUENCE
     {givenName       VisibleString,
    initial             VisibleString,
    familyName       VisibleString}

EmployeeNumber ::= [APPLICATION 2 ] IMPLICIT INTEGER

Date ::= [APPLICATION 3 ]IMPLICIT VisibleString --YYYYMMDD
```

A.1.2 记录值的ASN.1描述

下面使用ASN.1描述了John Smith个人记录的值：

```
  { name { givenName "John", initial "P", familyName "Smith"},
       title         "Director",
       number        51,
       dateOfHire    "19710917",
 nameOfSpouse    { givenName "Mary", initial "T", familyName "Smith"},
```

XX

children

{{name { givenName "Ralph", initial "T", familyName "Smith"},

dateOfBirth "19571111"},

{{name { givenName "Susan", initial “B”, familyName "Jones"},

dateOfBirth ""19590717"}}}

A.1.3 本记录值的ALIGNED PER表示

上面给出的记录值的表示如下（在应用本文件定义的紧缩编码规则的ALIGNED变体之后），编码以十六进制显示，后面是对编码的注释描述，以二进制显示。

该编码的长度是94个八位位组，而相同的使用PER的UNALIGNED变体进行编码的个人记录值则为84个八位位组，限定长度形式的BER至少为136个八位位组，不定长度形式的BER至少为161个八位位组。

A.1.3.1 十六进制视图

```
80044A6F  686E0150  05536D69  74680133  08446972  6563746F  72083139  37313039
3137044D  61727901  5405536D  69746802  0552616C  70680154  05536D69  74680831
39353731  31313105  53757361  6E014205  4A6F6E65  73083139  35393037  3137
```

A.1.3.2 二进制视图

为了更容易读取数据的二进制视图，用空行将逻辑上属于一起的字段进行分组（通常为长度/值对），用换行分割字段，用空格分割字符串内的字符，‘x’表示一个0填充位，它有时用来对齐八位位组边界上的字段。

```
1xxxxxxx                                          位图位=1表示存在"children"

00000100                                          name.givenName的长度=4
01001010  01101111  01101000  01101110            name.givenName="John"

00000001                                          name.initial的长度=1
01010000                                          name.initial="P"

00000101                                          name.familyName的长度=5
01010011  01101101  01101001  01110100  01101000  name.familyName="Smith"

00000001                                          (employee) number的长度=1
00110011                                          (employee) number=51

00001000                                          title的长度=8
01000100  01101001  01110010  01100101  01100011
01110100  01101111  01110010                      title="Director"

00001000                                          dateOfHire 的长度= 8
00110001  00111001  00110111  00110001  00110000
```

```
00111001  00110001  00110111                        dateOfHire="19710917"

00000100                                            nameOfSpouse.givenName的长度=4
01001101  01100001  01110010  01111001              nameOfSpouse.givenName="Mary"

00000001                                            nameOfSpouse.initial的长度=1
01010100                                            nameOfSpouse.initial="T"

00000101                                            nameOfSpouse.familyName的长度=5
01010011  01101101  01101001  01110100  01101000    nameOfSpouse.familyName="Smith"

00000010                                            children的个数

00000101                                            children[0].givenName的长度=5
01010010  01100001  01101100  01110000  01101000    children[0].givenName="Ralph"

00000001                                            children[0].initial的长度=1
01010100                                            children[0].initial="T"

00000101                                            children[0].familyName的长度=5
01010011  01101101  01101001  01110100  01101000    children[0].familyName="Smith"

00001000                                            children[0].dateOfBirth的长度=8
00110001  00111001 00110101  00110111  00110001
00110001  00110001  00110001                        children[0].dataOfBirth="19571111"


00000101                                            children[1].givenName的长度=5
01010011  01110101  0111001  10110000  101101110    children[1].givenName="Susan"

00000001                                            children[1].initial的长度=1
01000010                                            children[1].initial="B"

00000101                                            children[1].familyName的长度=5
01001010  01101111  01101110  01100101  01110011    children[1].familyName="Jones"

00001000                                            children[1].dateOfBirth的长度=8
00110001  00111001  00110101  00111001  00110000
00110111  00110001  00110111                        children[1].dateOfBirth="19590717"
```

A.1.4 本记录值的UNALIGNED PER表示

XX

上面给出的记录值的表示如下（在应用本文件定义的紧缩编码规则的UNALIGNED变体之后），编码以十六进制显示，后面是对编码的注释描述，以二进制显示。注意填充位没有出现在UNALIGNED变体中，而字符也尽可能用最少的位数编码。

该编码的长度是84个八位位组，而相同的使用PER的ALIGNED变体编码的个人记录值为94个八位位组，限定长度形式的BER至少为136个八位位组，不定长度形式的BER至少为161个八位位组。

A.1.4.1 十六进制视图

824ADFA3 700D005A 7B74F4D0 02661113 4F2CB8FA 6FE410C5 CB762C1C B16E0937
0F2F2035 0169EDD3 D340102D 2C3B3868 01A80B4F 6E9E9A02 18B96ADD 8B162C41
69F5E787 700C2059 5BF765E6 10C5CB57 2C1BB16E

A.1.4.2 二进制视图

为了更容易读取数据的二进制视图，用空行将逻辑上属于一起的字段进行分组（通常为长度/值对），用换行分割字段，用空格分割字符串内的字符，‘x’表示一个0填充位，它有时用来对齐八位位组边界上的字段。

```
1                                                   位图位表示存在"children"

0000010.0                                           name.givenName的长度=4
1001010.11011111.10100011.01110                     name.givenName="John"

000.00001                                           name.initial的长度=1
101.0000                                            name.initial="P"

0000.0101                                           name.familyName的长度=5
1010.01111011.01110100.11110100.1101000             name.familyName="Smith"

0.0000001                                           (employee) number的长度=1
0.0110011                                           (employee) number=51

0.0001000                                           title的长度=8
1.000100  11.01001  111.0010  1100.101  11000.11
111010.0  1101111  .1110010                         title="Director"

0.0001000                                           dateOfHire的长度=8
0.110001  01.11001  011.0111  0110.001  01100.00
011100.1  0110001  .0110111                         dateOfHire="19710917"

0.0000100                                           nameOfSpouse.givenName的长度=4
1.001101  11.00001  111.0010  1111.001              nameOfSpouse.givenName="Mary"

00000.001                                           nameOfSpouse.initial的长度=1
10101.00                                            nameOfSpouse.initial="T"
```

```
000001.01                                          nameOfSpouse.familyName的长度=5
101001.1  1101101  .1101001  1.110100  11.01000    nameOfSpouse.familyName="Smith"

000.00010                                          children的个数

000.00101                                          children[0].givenName的长度=5
101.0010  1100.001  11011.00  111000.0  1101000      children[0].givenName="Ralph"

.00000001                                          children[0].initial的长度=1

.1010100                                           children[0].initial="T"

0.0000101                                          children[0].familyName的长度=5
1.010011  11.01101  110.1001  1110.100  11010.00   children[0].familyName="Smith"

000010.00                                          children[0].dateOfBirth的长度=8
011000.1  0111001  .0110101  0.110111  01.10001
011.0001  0110.001  01100.01                       children[0].dateOfBirth="19571111"

000001.01                                          children[1].givenName的长度=5
101001.1  1110101  .1110011  1.100001  11.01110    children[1].givenName="Susan"


000.00001                                          children[1].initial的长度=1
100.0010                                           children[1].initial="B"

0000.0101                                          children[1].familyName的长度=5
1001.010  11011.11110111.0  1100101  .1110011      children[1].familyName="Jones"

0.0001000                                          children[1].dateOfBirth的长度=8
0.110001  01.11001  011.0101  0111.001  01100.00
011011.1  0110001  .0110111x                       children[1].dateOfBirth="19590717"
```

A.2 使用子类型约束的记录

除了利用子类型记法在某些项上施加一些约束之外，本实例与A.1所示的相同。

A.2.1 记录结构的ASN.1描述

使用GB/T 16262.1—AAAA规定的用于定义类型的ASN.1正式描述假定的人事记录的结构如下：

```
PersonnelRecord :: = [APPLICATION 0 ] IMPLICIT SET{
    name                    Name,
```

```
        title                    [0] VisibleString,
        number                EmployeeNumber,
        dateOfHire              [1] Date,
        nameOfSpouse          [2] Name,
        children                [3] IMPLICIT,
            SEQUENCE OF ChildInformation DEFAULT{}}
    ChildInformation ::= SET
        {name                   Name,
        dateOfBirth ,           [0] Date}
    Name ::= [APPLICATION1] IMPLICIT SEQUENCE
       {givenName             NameString,
        initial                   NameString(SIZE(1)),
        familyName            NameString}
    EmployeeNumber :: = [ APPLICATION 2 ] IMPLICIT INTEGER
    Date ::= [APPLICATION 3 ] IMPLICIT VisibleString (FROM("0".."9") ^ SIZE(8))
                                        --YYYYMMDD
    NameString :: = VisibleString (FROM("a".."z" | "a".."Z" | "-.") ^ SIZE(1..64))
```

A.2.2 本记录值的ASN.1描述

下面使用ASN.1描述了John Smith个人记录的值：

```
{ name { givenName "John", initial "P", familyName "Smith"},
  title            "Director",
  number         51,
  dateOfHire         "19710917",
  nameOfSpouse     {givenName "Mary", initial"T", familyName "Smith"},
  children
  {{name {givenName "Ralph", initial"T", familyName "Smith"},
        dateOfBirth "19571111"},
  {name { givenName "Susan", initial "B", familyName "Jones"},
  dateOfBirth "19590717"}}}
```

A.2.3 本记录值的ALIGNED PER表示

上面给出的记录值的表示如下（在应用本文件定义的紧缩编码规则的ALIGNED变体之后），编码以十六进制显示，后面是对编码注释描述，以二进制显示，在二进制视图中，‘x’用来表示编码为0位的填充位，它们有时用来对齐字段。

该编码的长度是74个八位位组，而相同的使用PER的UNALIGNED变体编码的个人记录值为61个八位位组，限定长度形式的BER至少为136个八位位组，不定长度形式的BER至少为161个八位位组。

A.2.3.1 十六进制视图

```
864A6F68  6E501053  6D697468  01330844  69726563  746F7219  7109170C  4D617279
5410536D  69746802  1052616C  70685410  536D6974  68195711  11105375  73616E42
104A6F6E  65731959  0717
```

A.2.3.2 二进制视图

为了更容易读取数据的二进制视图，用空行将逻辑上属于一起的字段进行分组（通常为长度/值对），用换行分割字段，用空格分割字符串内的字符，‘x’表示一个0填充位，它有时用来对齐八位位组边界上的字段。

```
1                                                   位图位=1表示存在"children"

0000011x                                            name.givenName的长度=4
01001010  01101111  01101000  01101110              name.givenName="John"

01010000                                            name.initial="P"

000100xx                                            name.familyName的长度=501010011
01101101  01101001  01110100  01101000              name.familyName="Smith"

00000001                                            (employee) number的长度=1
00110011                                            (employee) number=51

00001000                                            title的长度=8
01000100  01101001  01110010  01100101
01100011  01110100  01101111  01110010              title="Director"

0001  1001  0111  0001  0000  1001  0001  0111      dateOfHire="19710917"

000011xx                                            nameOfSpouse.givenName的长度=4
01001101  01100001  01110010  01110010              nameOfSpouse.givenName="Mary"

01010100                                            nameOfSpouse.initial="T"
000100xx                                            nameOfSpouse.familyName的长度5
01010011  01101101  01101001  01110100  01101000  =nameOfSpouse.familyName="Smith"

00000010                                            children的个数

000100xx                                            children[0].givenName的长度=5
01010010  01100001  01101100  01110000  01101000  children[0].givenName="Ralph"

01010100                                            children[0].initial="T"

000100xx                                            children[0].familyName的长度=5
01010011  01101101  01101001  01110100  01101000  children[0].familyName="Smith"

0001  1001  0101  0111  0001  0001  0001  0001      children[0].dateOfBirth="19571111"
```

```
XX
000100xx                                          children[1].givenName的长度=5
01010011  01110101  01110011  01100001  01101110  children[1].givenName="Susan"

01000010                                          children[1].initial="B"

000100xx                                          children[1].familyName的长度=5
01001010  01101111  01101110  01100101  01110011  children[1].familyName="Jones"


0001  1001  0101  1001  0000  0111  0001  0111    children[1].dateOfBirth="19590717"
```

A.2.4 本记录值的UNALIGNED PER表示

上面给出的记录值的表示如下（在应用本文件定义的紧缩编码规则的UNALIGNED变体之后），编码以十六进制显示，后面是对编码的注释描述，以二进制示显示，注意填充位没有出现在UNALIGNED变体中，而字符也尽可能用最少的位数编码。

编码的长度是61个八位位组，而相同的使用PER的ALIGNED变体编码的个人记录值为74个八位位组，限定长度形式的BER至少为136个八位位组，不定长度形式的BER至少为161个八位位组。

A.2.4.1 十六进制视图

```
865D51D2  888A5125  F1809984  44D3CB2E  3E9BF90C  B8848B86  7396E8A8  8A5125F1
81089B93  D71AA229  4497C632  AE222222  985CE521  885D54C1  70CAC838  B8
```

A.2.4.2 二进制视图

为了更容易读取数据的二进制视图，用空行将逻辑上属于一起的字段进行分组（通常为长度/值对），用换行分割字段，用空格分割字符串内的字符，‘x’表示一个0填充位，它有时用来对齐八位位组边界上的字段。

```
1                                                 位图位=1表示存在"children"

000011                                            name.givenName的长度=4
0.01011  101.010  10001.1  101001                 name.glvenName="John"

0.10001                                           name.initial="P"

000.100                                           name.familyName的长度=5
01010.0  101000  1.00100  101.111  10001.1        name.familyName="Smith"

0000000.1                                         (employee) number的长度=1
0011001.1                                         (employee) number=51

0000100.0                                         title的长度=8
1000100  .1101001  1.110010  11.00101  110.0011
1110.100  11011.11  111001.0                      title="Director"
```

```
0001  100.1  0111  000.1  0000  100.1  0001  011.1  dateOfHire="19710917"

000011                                          nameOfSpouse.givenName的长度=4
0.01110  011.100  10110.1  110100               nameOfSpouse.givenName="Mary"

0.10101                                         nameOfSpouse.initial="T"

000.100                                         nameOfSpouse.familyName的长度=5
01010.0  101000  1.00100  101.111  10001.1       nameOfSpouse.familyName="Smith"

0000001.0                                       children的个数

000100                                          children[0].givenName的长度=5
0.10011  011.100  10011.1  101011  1.00011      children[0].givenName="Ralph"

010.101                                         children[0].initial="T"

00010.0                                         children[0].familyName的长度=5
010100 1.01000 100.100 10111.1 100011           children[0].familyName="Smith"

0.001  1001  0.101  0111  0.001  0001  0.001  0001  children[0].dateOfBirth="19571111"

0.00100                                         children[1].givenName的长度=5
010.100  11000.0  101110  0.11100  101.001      children[1].givenName="Susan"

00001.1                                         children[1].initial="B"


000100                                          children[1].familyName的长度=5
0.01011  101.010  10100.1  100000  1.01110      children[1].familyName="Jones"

000.1  1001  010.1 1 001  000.0  0111  000.1  0111xxx children[1].dateOfBirth="19590717"
```

A.3 使用扩展标记的记录

A.3.1 记录结构的ASN.1描述

使用GB/T 16262.1—AAAA规定的用于定义类型的ASN.1正式描述假定的人事记录的结构如下：

```
PersonnelRecord :: = [APPLICATION 0] IMPLICIT SET{
    name            Name,
    title               [0] VisibleString,
    number          EmployeeNumber,
```

```
        dateOfHire          [1] Date,
        nameOfSpouse        [2] Name,
        children            [3] IMPLICIT,
            SEQUENCE (SIZE(2,...)) OF ChildInformation OPTIONAL,
        ...
    }
    ChildInformation :: = SET
        {name               Name,
        dateOfBirth         [0] Date
        ...,
        sex                 [1] IMPLICIT ENUMERATED { male(1), female(2),
                                    unknown(3)} OPTIONAL
        }

    Name :: = [APPLICATION 1] IMPLICIT SEQUENCE
        {givenName          NameString,
        initial                 NameString(SIZE(1)),
        familyName          NameString,
        ...
        }
    EmployeeNumber :: = [APPLICATION 2] IMPLICIT INTEGER( 0..9999,...)
    Date :: = [APPLICATION 3 ]IMPLICI TVisibleString
                            (FROM ("0".."9") ^ SIZE(8,...,9..20))--YYYYMMDD

        NameString :: = VisibleString ( FROM( "a".."z" | "A".."Z" | "-.") ^ SIZE
      (1..64,...))
```

A.3.2 记录值的ASN.1描述

下面使用ASN.1描述了John Smith个人记录的值：

```
{name {givenName "John", initial "P", familyName "Smith"},
    title           Director,
    number        51,
    dateOfHire        "19710917",
    nameOfSpouse    {givenName "Mary", initial "T", familyName "Smith"},
    children
        {{name {givenName "Ralph", initial "T", familyName "Smith"},
        dateOfBirth "19571111"},
        {name{ givenName "Susan", initial "B", familyName "Jones"},
        dateOfBirth "19590717",sex female}}}
```

A.3.3 本记录值的ALIGNED PER表示

上面给出的记录值的表示如下（在应用本文件定义的紧缩编码规则的ALIGNED变体之后，编码以十六进制显示，后面是对编码的注释描述，以二进制示显示，在二进制显示中，‘x’用来表示编码为0位的填充位，它们用来对齐字段。

该编码的长度是83个八位位组，而相同的用PER的UNALIGNED变体编码的个人记录值为65个八位位组，限定长度形式的BER至少为139个八位位组，不定长度形式的BER至少为164个八位位组。

A.3.3.1 十六进制视图

```
40C04A6F  686E5008  536D6974  68000033  08446972  6563746F  72001971  0917034D
61727954  08536D69  74680100  52616C70  68540853  6D697468  00195711  11820053
7573616E  42084A6F  6E657300  19590717  010140
```

A.3.3.2 二进制视图

为了更容易读取数据的二进制视图，用空行将逻辑上属于一起的字段进行分组（通常为长度/值对），用换行分割字段，用空格分割字符串内的字符，‘x’表示一个0填充位，它有时用来对齐八位位组边界上的字段。

```
0                                                   在个人记录中不存在扩展值

1                                                   位图位=1表示存在"children"

0                                                   在"name"中不存在扩展值

0                                                   长度在扩展根范围内
0000  11xxxxxx                                     name.givenName的长度=4
01001010  01101111  01101000  01101110              name.givenName="John"

01010000                                            name.initial="P"

0                                                   长度在扩展根范围内
000100xx                                            name.familyName的长度=5name
01010011  01101101  01101001  01110100  01101000  name.familyName="Smith"

0xxxxxxx                                            值在扩展根范围内
00000000  00110011                                  (employee) number=51

00001000                                            title的长度=8
01000100  01101001  01110010  01100101  01100011
01110100  01101111  01110010                       title="Director"

0xxxxxxx                                            长度在扩展根范围内
0001  1001  0111  0001  0000  1001  0001  0111     dateOfHire="19710917"

0                                                   在nameOfSpouse中不存在扩展值
```

```
0                                                   长度在扩展根范围内

000011                                              nameOfSpouse.givenName的长度=4
01001101  01100001  01110010  01111001              nameOfSpouse.givenName="Mary"

01010100                                            nameOfSpouse.initial="T"

0                                                   长度在扩展根范围内
000100xx                                            nameOfSpouse.familyName的长度=5
01010011  01101101  01101001  01110100  01101000  nameOfSpouse.familyName="Smith"

0                                                   children的个数在扩展根范围内

0                                                   在children[0]中不存在扩展值

0                                                   在children[0].name中不存在扩展值

0                                                   长度在扩展根范围内

000100xxxxxx                                        children[0].givenName的长度=5
01010010  01100001  01101100  01110000  01101000  children[0].givenName="Ralph"

01010100                                            children[0].initial="T"

0                                                   长度在扩展根范围内
000100xx                                            children[0].familyName的长度=5
01010011  01101101  01101001  01110100  01101000  children[0].familyName="Smith"

0xxxxxxx                                            长度在扩展根范围内
0001  1001  0101  0111  0001  0001  0001  0001     children[0].dataOfBirth="19571111"
1                                                   在children[1]中存在扩展值
0                                                   在children[1].name中不存在扩展值

0                                                   长度在扩展根范围内
000100xx                                            children[1].familyName的长度=5
01010011  01110101  0111001  10110000  101101110  children[1].givenName="Susan"

01000010                                            children[1].initial="B"

0                                                   长度在扩展根范围内
000100xx                                            children[1].familyName的长度=5
01001010  0110111  10110111  00110010  101110011  children[1].familyName="Jones"
```

```
0xxxxxxx                                          长度在扩展根范围内
0001  1001 0101  1001  0000  0111  0001  0111     children[1].dateOfBirth="19590717"

0000000                                           children[1]扩展附加位图的长度=1
1                                                 表示存在sex的扩展值

00000001                                          sex完整编码的长度
01xxxxxx                                          sex=female的完整编码
```

A.3.4 本记录值的UNALIGNED PER表示

上面给出的记录值的表示如下（在应用本文件定义的紧缩编码规则的UNALIGNED变体之后），编码以十六进制显示，后面是对编码的注释描述，以二进制示显示，注意填充位没有出现在UNALIGNED变体中，字符尽可能用最少位数编码。

编码的长度是65个八位位组，而相同的使用PER的ALIGNED变体编码的个人记录值为83个八位位组，限定长度形式的BER至少为139个八位位组，不定长度形式的BER至少为164个八位位组。

A.3.4.1 十六进制视图

```
40CBAA3A  5108A512  5F180330  889A7965  C7D37F20  CB8848B8  19CE5BA2  A114A24B
E3011372  7AE35422  94497C61  95711118  22985CE5  21842EAA  60B832B2  0E2E0202
80
```

A.3.4.2 二进制视图

为了更容易读取数据的二进制视图，用空行将逻辑上属于一起的字段进行分组（通常为长度/值对），用换行分割字段，用空格分割字符串内的字符，‘x’表示一个0填充位，它有时用来对齐八位位组边界上的字段。

```
0                                                 在个人记录中不存在扩展值
1                                                 位图位=1表示存在"children"

0                                                 在"name "中不存在扩展值
0                                                 长度在扩展根范围内
0000.11                                           name.givenName的长度=4
001011  .101010  10.0011  1010.01                 name.givenName="John"

010001                                            name.initial="P"

.0                                                长度在扩展根范围内
000100                                            name.familyName的长度=5
0.10100  101.000  10010.0  101111  1.00011        name.familyName="Smith"

0                                                 长度在扩展根范围内
00.00000011.0011                                  (employee) number=51
```

```
XX
0000.1000                                          title的长度=8
1000.100  11010.01  111001.0  1100101  1100011
1.110100  11.01111  111.0010                       title="Director"

0                                                  长度在扩展根范围内
000.1  1001  011.1  0001  000.0  1001  000.1  0111  dateOfHire=“19710917”

0                                                  在nameOfSpouse中不存在扩展值

0                                                  长度在扩展根范围内
0.00011                                            nameOfSpouse.givenName 的长度=4
001.110  001110.0  101101  1.10100                 nameOfSpouse.givenName 的长度="Mary"

010.101                                            nameOfSpouse.initial="T"

0                                                  长度在扩展根范围内
0001.00                                            nameOfSpouse.givenName 的长度=5
010100  .101000  10.0100  1011.11  100011         nameOfSpouse.givenName 的长度="Smith"

.0                                                 children的个数在扩展根范围内

0                                                  在 children[0]中不存在扩展值

0                                                  在children[0].name中不存在扩展值

0                                                  长度在扩展根范围内
0001.00                                            children[0].givenName的长度=5
010011  .011100  10.0111  1010.11  100011         children[0].givenName="Ralph"

.010101                                            children[0].initial="T"

0                                                  长度在扩展根范围内
0.00100                                            children[0].familyName的长度=5
010.100  10100.0  100100  1.01111  100.011        children[0].familyName="Smith"

0                                                  长度在扩展根范围内
0001  .1001  0101  .0111  0001  .0001  0001  .0001  children[0].dateOfBirth="19571111"

1                                                  在children[1]中存在扩展值

0                                                  在children[1].name中不存在扩展值长度

0                                                  长度在扩展根范围内
```

```
0.00100                                          children[1].givenName的长度=5
010.100  11000.0 101110  0.11100  101.001        children[1].givenName="Susan"

00001.1                                          children[1].initial="B"

0                                                长度在扩展根范围内
000100                                           children[1].familyName的长度=5
.001011 10.1010 1010.01 100000 .101110               children[1].familyName="Jones"

0                                                长度在扩展根范围内
0.001  1001  0.101  1001  0.000  0111  0.001 0111 children[1].dateOfBirth="19590717"

0.000000                                         children[1]扩展附加位图的长度=1
1                                                表示存在sex的扩展值

0.0000001                                        sex完整编码的长度
0.lxxxxxx                                        sex=female的完整编码
x                                                填充位创建完整的个人记录编码
```

A.4 使用扩展附加组的记录

A.4.1 记录结构的ASN.1描述

下面使用GB/T 16262.1—AAAA规定的用于定义类型的ASN.1正式描述假定的人事记录的结构，假设AUTOMATIC TAGS:

```
Ax ::= SEQUENCE{
  a   INTEGER (250..253),
  b   BOOLEAN,
  c   CHOICE {
      d   INTEGER,
      ...,
      [[
          e   BOOLEAN,
          f   IA5String
      ]],
      ...,
  },
  ...,
  [[
      g   NumbericString (SIZE(3)),
      h   BOOLEAN OPTIONAL
  ]],
  ...,
```

```
XX
        i    BMPString OPTIONAL,
        j    PrintableString OPTIONAL,
        }
```

A.4.2 记录值的ASN.1描述

下面使用ASN.1形式地描述了Ax的值：

```
{a 253, b TRUE, c e : TRUE, g "123", h TRUE}
```

A.4.3 本记录值的ALIGNED PER表示

上面给出的记录值的表示如下（在应用本文件定义的紧缩编码规则的ALIGNED变体之后），编码以十六进制显示，后面是对编码的注释描述，以二进制示显示，在二进制显示中，‘x’用来表示编码为0的填充位，它们用来对齐字段。

该编码的长度是8个八位位组，而相同的使用PER的UNALIGNED变体编码的个人记录值为8个八位位组，限定长度形式的BER至少为22个八位位组，不定长度形式的BER至少为26个八位位组。

A.4.3.1 十六进制视图

```
9E000180  010291A4
```

A.4.3.2 二进制视图

为了更容易读取数据的二进制视图，用空行将逻辑上属于一起的字段进行分组（通常为长度/值对），用换行分割字段，用空格分割字符串内的字符，‘x’表示一个0填充位，它有时用来对齐八位位组边界上的字段。

```
1                                       在Ax中存在扩展附加值
00                                      位图位=0表示不存在可选字段(i和j)

11                                      a=253


1                                       b=TRUE

1                                       c的选择项的值为一个扩展附加值
0000000xx                               选择索引选择c.e

00000001                                c.e的长度
1xxxxxxx                                c.e=TRUE

0000000                                 在Ax=1中定义的扩展附加部分的数目

1                                       存在第1个扩展附加部分

00000010                                扩展附加部分编码的长度=2
1                                       位图=1表示h存在
```

0010 0011 0100	g="123"
1xx	h=TRUE

A.4.4 本记录值的UNALIGNED PER表示

上面给出的记录值的表示如下（在应用本文件定义的紧缩编码规则的UNALIGNED变体之后）如下所示。编码以十六进制显示，后面是对编码的注释描述，以二进制示显示。注意填充位不会出现在UNALIGNED变体中，除非可能出现在最外层值的编码末尾，因此它隐含在开放类型所携带值的末尾。

该编码的长度是8个八位位组，而相同的用PER的ALIGNED变体编码的个人记录值为8个八位位组，限定长度形式的BER至少为22个八位位组，不定长度形式的BER至少为26个八位位组。

A.4.4.1 十六进制视图

9E000600 040A4690

A.4.4.2 二进制视图

为了更容易读取数据的二进制视图，用空行将逻辑上属于一起的字段进行分组（通常为长度/值对），用换行分割字段，用空格分割字符串内的字符，‘x’表示一个0填充位，它有时用来对齐八位位组边界上的字段。

1	在Ax中存在扩展附加值
00	位图位=0表示不存在可选字段(i和j)
11	a=253
1	b=TRUE
1	c的选择项的值为一个扩展附加值
0.000000	选择索引选择c.e
00.000001	c.e的长度
1x.xxxxxx	c.e=TRUE
00.0000	在Ax=1中定义的扩展附加部分的数目
1	存在第1个扩展附加部分
00.000010	扩展附加部分编码的长度=2
1	位图=1表示h存在
0.010 0011 0.100	g="123"
1xxxx	h=TRUE

附 录 B

（资料性）
组合 PER 可视约束和 PER 非可视约束

B.1 概述

B.1.1 PER可扩展性的正确决定因素是实现互工作的关键，不同的实现形成值的相同决定因素也很重要，这些值被PER编码为根值，这些根值又为扩展类型而编码为扩展附加部分。

B.1.2 用户写出的东西通常是简单的，PER编码是直观的，但对复杂的结构来说，需要进一步讨论PER可视、PER可扩展性和集合运算之间的交互作用，这即是本章的内容。

B.1.3 由于有些约束被定义为PER不可视的（见10.3），那么，类型可能被GB/T 16262.1—AAAA中的规则定义为可扩展的，但对PER编码来说，被认为是不可扩展的（带有涵盖所有可能的扩展的放松约束）。

B.1.4 在两种情况下类型被认为是可扩展的，PER编码的根值集合不总是与认为用GB/T 16262.1—AAAA定义的根值的值集合相同。

B.1.5 对实际规范中出现的大多数情况，两种决定因素都是简单、易懂的。

B.1.6 然而，ASN.1在复杂约束的应用中提供了值得考虑的能力和普遍性，这些复杂约束是由于集合运算和/或简单或复杂约束的一系列应用而产生的。

B.1.7 用户的规范不大可能定义涉及本附录讨论的复杂性的ASN.1结构，但如果实际上应用了这样的约束，工具的实现者需要知道产生何种编码。

B.1.8 用于非常复杂的约束（可能涉及类型引用名）的规则不总是直观的，但是设计这些规则是为了简化工具实现和ASN.1规范的复杂性。

B.1.9 对SEQUENCE、SET、CHOICE和ENUMERATED来说，如果含有扩展标记(省略号“...”)，那么即使被约束，其类型也是可扩展的(见10.3.22)。当且仅当一个值在省略号后不包括任何元素（或CHOICE中的选项及ENUMERATED中的枚举），该值即为根值。一个不可扩展的SEQUENCE、SET、CHOICE或ENUMERATED可以是应用了可扩展约束的父类型，由此产生一个可扩展的序列、集合、值选择或枚举类型。然而，对这些类型的约束从不是PER可视的，且导致按PER的类型编码时没有可扩展位。这些类型在本附录中不作进一步讨论，本附件只关注在整数和受限的已知倍数字符串类型上使用可扩展约束而产生的可扩展性(在其他类型上的约束不影响PER编码，除非在八位位组串和位串类型上的长度约束，它们类似于在字符串类型上的长度约束，这里不做进一步考虑)。

B.1.10 规范文本中规定了精度规则，但本附录旨在帮助工具厂商理解这些规则。

B.1.11 为简化说明，下面将不可扩展类型或约束中的值集合描述为根值，尽管该术语只严格地适用于扩展类型或约束。

B.1.12 GB/T 16262.1—AAAA的I.4提供了所有约束都是PER可视的约束组的指导信息，应与本附录结合起来阅读。当涉及的约束不是PER可视的，或约束适用于字符串类型时，则该规则需要进一步补充，这些补充的规则在B.2中。

B.2 PER中约束的可扩展性和可视性

B.2.1 概述

B.2.1.1 在BER中，对于根值和扩展附加部分，值的编码是一样的，所以可扩展性对该编码没有什么影响。在PER中，若抽象值处于（通常但不是必须，是有限的）根值集合中，那么，一般用有效方式编码这些抽象值，若抽象值是扩展附加的，则该编码的效率较低。

B.2.1.2 然而，对于许多PER编码，在类型的扩展添加中（被GB/T 16262.1确定）有一些值被PER编码，就好像这些值是根值一样，而不是作为扩展附加部分的值那样进行编码。这些值的精确标识是通过注释某些约束都是“非PER可视的”来进行的。

B.2.1.3 在本文件中引入PER可视的概念是为了使编码者在试图确定要被编码的值是否处于可扩展类型的根中时，减轻其任务。对编码者来说，难于以有效的方式处理的约束被定义为对PER编码（对其没有影响）是“不可视”的。

B.2.1.4 有一个例外，简单约束的可视性仅取决于被约束的类型和/或与可扩展性无关的约束方面。例如，约束在文本上依赖于表约束，还是变量约束（在文本上依赖抽象语法参数的约束）？

B.2.1.5 若一个约束是可变的，或是在正文上依赖表约束，则不管它适用什么类型，它决不是PER可视的。

B.2.1.6 此外，除非约束适用于整数型或已知倍数的受限字符串类型（或对位串、八位位组串上长度约束），否则约束决不是PER可视的。

B.2.1.7 例外是已知倍数受限字符串类型上的允许字母表约束，当且仅当它不是可扩展的，该约束是PER可视的。

B.2.1.8 同样重要的是，要注意在字符串类型上的单个值子类型约束不是PER可视的。

B.2.1.9 在PER中，对字符串类型的约束有两个独立方面：对串长度的约束和对允许字母表的约束。前一种约束影响编码中长度字段的存在和形式，第二种约束影响用于编码每个字符的位数。在简单应用中，很明显约束规定了其中的一个或者另一个。因此：

```
A1 :: = VisibleString (SIZE (20) )
        --长度约束
A2 :: = VisibleString (FROM("A.."F") )
        --允许的字母表约束
A3 :: = VisibleString (SIZE (2) ) (FROM("A.."F") )
        --长度约束和允许字母表约束
```

B.2.1.10 但应考虑：

```
B :: = VisibleString (SIZE (20 ) INTERSECTION FROM("A.."F")
          UNION
          SIZE (3) INTERSECTION FROM("F.."K") )
```

B.2.1.11 为规定带有这种复杂约束的类型的编码，PER引入了一个有效长度约束的概念，和一个有效允许字母表约束的概念。这些约束加在一起，是允许实际约束根中的所有抽象值，但通常是一些附加的抽象值。在上面的例子中，有效长度约束为3..20，有效允许字母表约束是FROM("A".."K")。

B.2.1.12 为处理可扩展性，本文件引入了一个进一步的概念，即有效长度约束和有效允许字母表约束中的一个或两个都可以是可扩展的(后者是PER不可视的，并且在确定编码时被忽略)，并且有必要考虑可扩展允许字母表约束的（非）PER可视对类型的有效约束的影响。

B.2.1.13 下列各条主要说明的是：PER可视的影响，以及对约束的系列应用和集合运算的有效性约束的计算。

B.2.2 约束的PER可视性

B.2.2.1 B.2.2.10描述了一个完整的（复杂的）PER约束何时是PER可视的，何时不是。然而首先我们简单地考虑约束的系列应用，哪一个（或作为一个整体）是PER可视的，哪一个不是。

B.2.2.2 规则很简单：如果在约束的系列应用中的一个完整约束不是PER可视的，那么，对于PER编码，简单地全部忽略该约束。

注：当为定义PER编码去掉非可视的约束时，不意味着这些应用能合法地发送附加抽象值，原有的约束仍适用于能发送的值，尽管编码器通常使用PER可视约束来执行检验和发布诊断。

B.2.2.3 重要的是要意识到在复杂情况中去掉非可视约束会有相当显著的效果，在去掉系列应用的不可视的PER约束之后考虑可扩展性（以及什么是根值）也总是重要的(若没有任何系列应用的约束是PER可视的，那么该类型对PER编码是不受约束的，并是不可扩展的)。

B.2.2.4 按照GB/T 16262.1的可扩展类型可能无法对PER进行扩展。

B.2.2.5 即使效果不是那么明显，按照GB/T 16262.1的扩展附加部分的值也可能是某些约束被去掉的根值的一部分，因此在PER中这些值可编码为根值，而不是编码为扩展附加部分。

注：这意味着PER编码可能比理论上的更冗长，但仍然是要编码的该类型中的所有抽象值使用的唯一编码。

B.2.2.6 影响系列应用复杂约束可视性的因素主要有三种。

B.2.2.7 第1种要考虑的因素是该约束是可变约束（在文本上依赖于抽象语法的参数），还是在文本上依赖的表约束，在这种情况下，系列应用的整个约束不是 PER 可视的，并且整个约束要被丢弃。

B.2.2.8 第2种要考虑的因素是只对字符串适用的约束，在这些类型上的单个值子类型约束不是PER可视的，但如果在该约束中存在集合运算，那么，单个值子类型约束的存在不必形成系列应用不可视的完整约束。

B.2.2.9 确定这种情况的PER可视的规则在10.3.21中规定，总结如下，其中设“V”表示PER可视的，"I"表示非可视的（不可视）。

B.2.2.10 因为**UNION**和**INTERSECTION**都是可交换的，那么只对第1种情况V给出结果的规则。若所有组件都是V，则适用GB/T 16262.1的正常规则，这里不做进一步讨论，所有组件都是I的情况，也不再列出。规则是：

```
V UNION I =>  I
V INTERSECTION I => V
    --导出的I恰好是交集部分的V
V EXCEPT I => V
    --导出的I恰好是不带差集的V
I EXCEPT V => I
V,...,I => I
I,...,V => I
```

B.2.2.11 用这种方式去掉单个值子类型约束（和**EXCEPT**子句）有一个重要结果，这意味着适用于字符串类型的所有“原子”约束既是单纯的长度约束，也是单纯的允许字母表约束。整个约束（仅）由使用这种“原子”单元的（任意复杂的）交集、并集和扩展附加部分构成。

B.2.2.12 这明显简化了PER引用对字符串类型的“有效约束＂的计算。

B.2.2.13 第3种主要因素是允许字母表约束是否可扩展，这些约束不是PER可视的，不过，当它们的存在不影响任何可能存在的长度约束时，对其的处理不同于上述所列的那些，这方面将在B.2.3中讨论。

B.2.3 有效的约束

B.2.3.1 对已知倍数字符串类型的每个约束计算为一对有效的约束：一个有效的允许字母表约束和一个有效的长度约束，其中的一个或两个可能是可扩展的，也可能是空的（无有效约束）。

B.2.3.2 在系列应用中，由于GB/T 16262.1中的规则，只有最后一个约束可以具有可扩展的对。

B.2.3.3 有效的长度约束和有效的允许字母表约束的定义在3.7.8和3.7.9中给出，这里不再重复，但实际上要按B.2.2.9和B.2.2.10中的规定，将定义应用于带有被去掉的“不可视“约束的类型。

B.2.3.4 随着去掉非PER可视的约束，用系列应用有效长度约束和有效允许字母表约束替换实际约束将增加用于PER编码的新抽象值（现在包括的带有有效长度约束长度的任何值，并且该值仅使用有效的允许字母表约束）。然而，这些值将不再通过一致的应用来发送，并且其效果是再一次简单地使PER编码比理论上所能达到的效率还要低。

B.2.3.5 示例

```
A :: = VisibleString ( SIZE (10) INTERSECTION FROM("A")
        UNION
        SIZE (20)    INTERSECTION FROM("B"))
```

只有2个值，所以1位的编码只是理论上可能，但PER编码使用有效的约束并能在下列内容中编码总数为一百万（大约）个值：

```
B ::= VisibleString ( SIZE( 10 UNION 20)
                                INTERSECTION
                                FROM ("AB") )
```

B.2.3.6 对2个值并集的有效约束总是对每个值有效约束的并集，但通常情况下（若所有约束都是PER可视的），这一简单规则可能不适用于交集。

B.2.3.7 然而在这里去掉了单个值子类型约束和EXCEPT条款是很重要的，当所有“原子”约束是单纯的长度约束或单纯的允许字母表约束（可能扩展）时，有效的约束可以用简单的方式对任意集合运算（无EXCEPT条款）进行计算。

B.2.3.8 设{S，A}表示被与允许字母表约束A一起串行使用的长度约束S所允许的所有值的集合(还要注意，交集和并集是可交换的)，则有：

```
{S1,A1}    INTERSECTION  {S2,A2 }=> {S1 INTERSECTION S2,
                                      A1 INTERSECTION A2}
{S1,A1}    UNION{S2,A2} => {S1 UNION S2,
                            Al UNION A2}
{S1,Al},... => {S1,...}
```

B.2.3.9 最后一种情况需要做些解释。一个可扩展的允许字母表约束对编码没有影响，因为PER不支持根值所需的不同位数的字符，也不支持扩展附加值所需的不同位数的字符。因此，如果一个（有效的）允许字母表约束是可扩展的，它就不再是一个约束—所有字符都要能够表示。在最后一种情况中的“...”的效果是使允许的字母表和长度可扩展，但只有可扩展的长度仍然保持为一个约束。在规范正文中，表示可扩展的有效允许字母表约束不是PER可视的。

B.3 示例

本条提供了一些例子进一步进行说明。

```
A ::= INTEGER (MIN..MAX,...,1..10)
  --A是可扩展的，但根不受约束，并且可扩展位总是置为0。
Al ::= INTEGE R (1..32,...,33..128)
  --A1是可扩展的，并且在根的1~32中包含值1~128，值33~128作为扩展附加部分。
A2::= INTEGER (1..32,...,33..128)(1..128)
  --这是非法的，因为128不在双亲的根中（见GB/T 16262.1—AAAA的第50章）。
A3 ::= INTEGER ((1..32,...,33..128)^(1..128) )
```

```
    --这是合法的，A3是可扩展的，包含根中的1~32，并且33~128作为扩展部分。
A4 ::= INTEGER (1..32)^(MIN..63)
    --MIN为1，并且63是不合法的。
A6 ::= INTEGER ((1..64,...,B)^((1..32) )
    --A6总是（仅）包含值1..32，而不管B值包含的是什么，但它不过是形式上可扩展，并且
    --PER将编码所有5位中的值，其扩展位（总是）置为0。
A7 ::= INTEGER (1..32,...,B)(1..256)
    --A7是非法的，作为（1..256）的双亲，它不再包含比1~32多，而不管B包含什么。
A8 ::= IA5String (SIZE(3..4) | SIZE(9..10) )
    --A8具有有效长度约束SIZE(3..4 | 9..10)，
    --PER将使用3位来编码长度字段，就好像它有SIZE(3..10)那样。
A9 ::= IA5String ( FROM ("AB" )^SIZE (1..2) |
                   FROM ("DE" )^SIZE (3) |
                   FROM ("AXE")^(SIZE (1..5) )
    --A9具有有效长度约束SIZE(1..5)，PER将用3位来编码长度。
    --A9还具有有效字母表约束FROM ("ABDEX")，PER将用3位编码每一个字母。
A10 :: = IA5String (SIZE(1..4) | SIZE (5..10)^
                   FROM ("ABCD" ) | SIZE (6..10))
    --A10具有有效长度约束SIZE(1..10)，但允许的字母由整个IA5String字母组成。
A11 :: = IA5String ( SIZE (1..10) | FROM ("A.."D"))
    --没有长度约束，字母表是整个IA5String字母表。
A12 :: = IA5String ( SIZE (1..10)^FROM("A.."D"),...)
    --A12具有可扩展的有效长度约束SIZE(1..10,...)，字母表是整个IA5String字母表。
A13:: =IA5String ( SIZE (1..10,...)^FROM("A.."D"))
    --A13具有可扩展的有效长度约束SIZE(1..10,...)，有效字母表约束是FROM ("A.."D")。
A14 :: = IA5String ( SIZE(1..10,...,29) ) ( FROM("A.."D"))
    --有效长度约束SIZE(1..10)，由于FROM的系列应用,不可扩展。
    --有效字母表约束是FROM("A.."D")。
A15 :: = IA5String (SIZE (1..10,...) | FROM("A.."D"),...)
    --从MIN至MAX的可扩展的有效长度约束，带有根中的所有值,
    --可扩展位的编码总是置为0，字母表是整个IA5String字母表。
A16 :: = IA5String (FROM ("A.."D") ^ SIZE (1..10),...)
    --有效长度约束SIZE(1..10)，可以扩展。
    --字母表是整个IA5String字母表。
A17 :: = IA5String (FROM ("A.."D"),...)^(SIZE(1..10))
    --有效字母表约束FROM("A.."D")，由于SIZE的系列应用,不可扩展。
    --有效长度约束是SIZE(1..10)。
```

附 录 C

（资料性）

对 PER 算法的支持

应用标准或者国际标准化配置文件，可能规定支持哪一种紧缩编码，以及在协商中提供或接受的相应的传送语法。

当要求使用EMBEDDED PDVs（或EXTERNALs）或CHARACTER STRING中的中继安全和/或正则编码时，宜明确声明。

下列文本提供了可用于编制规范性文本的的指南。

C.1 正则编码旨在在当安全特性适用于编码时使用。当被编码的值包括单一集合类型时，使用CANONICAL-PER涉及重大额外的CPU利用成本，因此，一般不推荐使用，除非要求安全特性。

C.2 当抽象语法值含有嵌入内容，并且该嵌入内容使用不同于该抽象语法值相关联的传送或抽象语法进行编码时，强烈推荐嵌入内容用中继安全的方式进行编码。如果安全特性很重要，则要求用正则编码规则。在这种情况下，应特别注意用于类型BMPString或UniversalString的GB 1300 的级别，只有GB 13000 实现级别 1 才能保证是正则的。

C.3 强烈建议支持任何PER ALIGNED变体传送语法的所有实现支持BASIC-PER ALIGNED变体解码（因此也支持CANONICAL-PER ALIGNED变体），UNALIGNED变体情况类似。

C.4 从互工作利益的角度出发，建议PER的所有实现支持ALIGNED变体和UNALIGNED变体（增加的实现复杂性很小），在通信实例中提供哪一个（1 个或者 2 个）是本地管理事项，如果两者都提供则接受哪一个也是本地管理事项，如果只提供一个，应该接受。

C.5 接受这些建议对一般工具厂商尤其重要，当某个实现是特定于某些特定应用时，支持单个PER传送语法（可能被该应用的设计者指定）也完全可以接受。

附 录 D

（资料性）

对可扩展 ASN.1 规则的支持

D.1 这些紧缩编码规则依赖于其所使用类型的总体定义。通常，如果对类型定义做了不同于那些纯语义特性的任何改变，那么，使用本规范部分的所有值的编码都将受到影响。特别是对于序列可选组件的增加、将组件转换成该组件的CHOICE或者其他类型、或者对某个组件约束的放松或加紧都可能改变该类型值的编码。

D.2 但是这些编码规则已设计成能保证满足类型扩展ASN.1 模型（见GB/T 16262.1 中规定的对编码规则的要求。

D.3 当类型不是扩展序列的一部分（不存在扩展标记）时，本附录前面的文本适用：PER不为该类型的扩展性提供支持。当序列或集合类型有扩展标记，但没有扩展附加部分时，与类型相同但不带扩展标记的类型相比，会多出 1 位的开销（由于在ALIGNED变体中的填充，可变成一个八位位组）。当类型中存在扩展附加部分，而且在通信实例中被实际发送时，与去掉扩展标记的相同类型相比，多出大约一个八位位组的进一步开销，加上为发送的每个扩展附加部分用的一个附加长度字段。

D.4 重要的是注释扩展标记的增加和去掉都将改变线上的位，而且通常要求改变协议的版本号。

D.5 在一个信息客体集中包含扩展标记、或者增加和去掉异常规范都不改变编码，但是，这些当然可能表示在要求实现行为方面的一些变化，并且仍然要求改变协议的版本号。

附 录 E

（资料性）

关于 PER 编码拼接的指导附录

E.1 在已知编码规则和编码类型的情况下，PER编码是自定界的。ALIGNED和UNALIGNED变体的完整编码总是 8 位的整数倍。

E.2 为了在OSI表示层协议中载有PER编码，可以在八位位组串选项中拼接ALIGNED和UNALIGNED变体的编码。

附 录 F
（资料性）
编码规则的标识

在本文件中，以下客体标识符值、OID国际化资源标识符和客体描述值被赋值：

对于BASIC-PER，ALIGNED变体：

{joint-iso-itu-t asn1 (1) packed-encoding (3) basic (0) aligned (0)}

"/ASN.1/Packed-encoding/Basic/Aligned"

" Packed encoding of a single ASN.1 type (basic aligned)"

对于BASIC-PER，UNALIGNED变体：

{joint-iso-itu-t asn1 (1) packed-encoding (3) basic (0) unaligned (1)}

"/ASN.1/Packed-encoding/Basic/Unaligned"

"Packed encoding of a single ASN.1 type (basic unaligned)"

对于CANONICAL-PER，ALIGNED变体：

{joint-iso-itu-t asn1 (1) packed-encoding (3) canonical (1) aligned (0)}

"/ASN.1/Packed-encoding/Canonical/Aligned"

"Packed encoding of a single ASN.1 type (canonical aligned)"

对于CANONICAL-PER，UNALIGNED变体：

{joint-iso-itu-t asn1 (1) packed-encoding (3) canonical (1) unaligned (1)}

"/ASN.1/Packed-encoding/Canonical/Unaligned"

"Packed encoding of a single ASN.1 type (canonical unaligned)"