ICS 27.100
F 21
备案号：29043-2010

DL

中华人民共和国电力行业标准

DL / T 790.6 — 2010 / IEC 61334 - 6: 2000

采用配电线载波的配电自动化
第 6 部分：A-XDR 编码规则

Distribution automation using distribution line carrier system -
Part 6: A-XDR encoding rule

(IEC 61334-6: 2000, IDT)

2010-05-24 发布　　2010-10-01 实施

国家能源局　发布

目　次

前　言

随着我国电网技术的发展，对配电自动化的要求已日益迫切。与传输配电自动化信息的其他通信方式相比，配电线载波可以降低建设投资和运行费用，便于管理，是一种经济实用的通信方式。配电电压不高，但电网结构复杂，信号传输衰减大。针对配电网信号传输特点，自 1995 年起，国际电工委员会陆续发布了 IEC 61334 系列的国际标准或技术报告。采用这些文件使之成为我国的标准文件对于我国这方面工作的开展有很好的指导意义，便于与国际接轨。

系列标准 DL 790《采用配电线载波的配电自动化》采用国际系列标准 IEC 61334《采用配电线载波的配电自动化》，包括标准和标准化指导性技术文件，共有以下 20 部分。

DL/Z 790.11　采用配电线载波的配电自动化　第 1 部分：总则　第 1 篇：配电自动化系统的体系结构

DL/Z 790.12　采用配电线载波的配电自动化　第 1 部分：总则　第 2 篇：制定规范的导则

DL/Z 790.14　采用配电线载波的配电自动化　第 1-4 部分：总则　中低压配电线载波传输参数

DL/T 790.31　采用配电线载波的配电自动化　第 3 部分：配电线载波信号传输要求　第 1 篇：频带和输出电平

DL/T 790.321　采用配电线载波的配电自动化　第 3-21 部分：配电线载波信号传输要求　中压绝缘电容型相相结合设备

DL/T 790.322　采用配电线载波的配电自动化　第 3-22 部分：配电线载波信号传输要求　中压相地和注入式屏蔽地结合设备

DL/T 790.41　采用配电线载波的配电自动化　第 4 部分：数据通信协议　第 1 篇：通信系统参考模型

DL/T 790.432　采用配电线载波的配电自动化　第 4-32 部分：数据通信协议　数据链路层—逻辑链路控制

DL/T 790.433　采用配电线载波的配电自动化　第 4-33 部分：数据通信协议　数据链路层　面向连接的协议

DL/T 790.441　采用配电线载波的配电自动化　第 4-41 部分：数据通信协议　应用层协议—配电线报文规范

DL/T 790.442　采用配电线载波的配电自动化　第 4-42 部分：数据通信协议　应用协议　应用层

DL/T 790.4511　采用配电线载波的配电自动化　第 4-511 部分：数据通信协议　系统管理　CIASE 协议

DL/T 790.4512　采用配电线载波的配电自动化　第 4-512 部分：数据通信协议　系统管理　采用 DL/T 790.51 协议集的系统管理信息库

DL/T 790.461　采用配电线载波的配电自动化　第 4-61 部分：数据通信协议　网络层无连接协议

DL/T 790.51　采用配电线载波的配电自动化　第 5 部分：低层协议集　第 1 篇：扩频型移频键控（S-FSK）协议

DL/Z 790.52　采用配电线载波的配电自动化　第 5-2 部分：低层协议集　移频键控（FSK）协议

DL/Z 790.53　采用配电线载波的配电自动化　第 5-3 部分：低层协议集　自适应宽带扩频（SS-AW）协议

DL/Z 790.54　采用配电线载波的配电自动化　第 5-4 部分：低层协议集　多载波调制（MCM）协议

DL/Z 790.55　采用配电线载波的配电自动化　第 5-5 部分：低层协议集　快速跳频扩频通信

（SS-FFH）协议

DL/T 790.6　采用配电线载波的配电自动化　第 6 部分：A-XDR 编码规则

DL 790 的本部分等同采用 IEC 61334-6：2000《采用配电线载波的配电自动化　第 6 部分：A-XDR 编码规则》（英文版）。

本部分的附录 A、附录 B、附录 C 为资料性附录。

本部分由中国电力企业联合会提出。

本部分由全国电力系统管理及其信息交换标准化技术委员会归口。

本部分起草单位：国网电力科学研究院、北京四方继保自动化股份有限公司、中国电力科学研究院。

本部分主要起草人：于跃海、任雁铭、但富中、陈道元、沐连顺。

本部分在执行过程中的意见或建议反馈至中国电力企业联合会标准化中心（北京市白广路二条 1 号，100761）。

引　言

本部分根据《(国家发展改革委办公厅关于印发2007年行业标准修订、制定计划的通知)》(发改办工业［2007］1415号）的安排制定。

国际电信联盟（ITU-T）的X.208建议定义了一种形式语言——ASN.1（Abstract Syntax Notation One，抽象语法记法1），使应用层规范能定义需要交换的信息类型[1)]。将一组编码规则应用于ASN.1定义的类型的数值可以形成这种信息的一种表示方法。这些编码规则的应用产生了这些数值的转化语法。

虽然人们已经设想了多种编码规则，但是长期以来只有一种BER（Basic Encoding Rule，基本编码规则）用作标准（见ITU-T的X.209）。这主要是因为BER能够满足绝大多数应用的需要。但在某些情况下，它显得冗余。如何采用其他编码规则以避免冗余，是最近开发的一些转化语法标准（DER、CER、PER）的研究目的。这些转化语法标准和BER不同，比BER更适合于某些场合。它们不是通用的，而是专用的。

与通用编码规则不同，DL 790的本部分定义了一种新的、专用的编码规则——A-XDR，它最适用于DLMS（Distrubion Line Message Specification，配电线报文规范，见DL/T 790.441）环境。主要目的是将DLMS的PDU（Protocol Data Unit，协议数据单元）的编码优化[2)]，包括PDU的字节数量及编码解码的复杂性、所需的编码长度、处理性能和时间等。这个目的通过以下两个基本原则实现：

a）　A-XDR只为ASN.1类型应用于DLMS特殊环境的子集规定编码规则(因此，A-XDR是专用的)；

b）　A-XDR规定的是面向字节的编码规则。

1）ASN.1为已定义类型值的规范规定了记法。

2）如只考虑PDU的大小，PER比A-XDR性能好。但是，PER的主要目标是压缩容量，这是通过大量使用位域而不是字节域对不同数值进行编码而获得的。为了进一步减少PDU的大小，它使用了更复杂的PER变量（不对齐的PER），也对类型及数值加以限制。因此，它的压缩是以增加计算负担为代价而获得的。此外，PER有对齐和不对齐两种情况，实现时都必须支持。PER的复杂性使它在DLMS环境下不是优选的。"轻型"的A-XDR编码规则更适合于DLMS环境，这种环境的资源量有时很少。

采用配电线载波的配电自动化
第 6 部分：A-XDR 编码规则

1 范围

DL 790 的本部分定义了 A-XDR [1] 编码规则，它可用来导出在 DLMS 核心标准中用 ASN.1（见 DL/T 790.441—2004）定义的数值类型的转化语法规范。这些 A-XDR 编码规则也可用于对转化语法解码以识别转化的数据值。

A-XDR 编码规则：

a） 在通信中使用；

b） 是 DLMS PDU 的优化 [2] 编码方法。

注：如 A-XDR 确是 DLMS PDU 的优化编码方法，它将成为基于 DLMS 的通信协议的默认编码规则。然而，不管是默认的还是可选择的编码规则，作为应用层环境的一部分，它都会在已有协议（如 DL/T 790.442）的应用层文件中定义。

2 规范性引用文件

下列文件中的条款通过本部分的引用而成为本部分的条款。凡是注日期的引用文件，其随后所有的修改单（不包括勘误的内容）或修订版均不适用于本部分，然而，鼓励根据本部分达成协议的各方研究是否可使用这些文件的最新版本。凡是不注日期的引用文件，其最新版本适用于本部分。

DL/T 790.441—2004 采用配电线载波的配电自动化 第 4-41 部分：数据通信协议 应用层协议—配电线报文规范（idt IEC 61334-4-41：1996）

DL/T 790.442—2004 采用配电线载波的配电自动化 第 4-42 部分：数据通信协议 应用协议 应用层（idt IEC 61334-4-42：1996）

ISO/IEC 8825-2：1997 信息技术 ASN.1 编码规则 包式编码规则（PER）规范

ITU X.208：1988 抽象语法记法 1（ASN.1）规范

ITU X.209：1988 抽象语法记法 1（ASN.1）基本编码规则

3 A-XDR 的一般特征

A-XDR 规定了一种编码规则，可用于对定义为单一 ASN.1 类型（最外层的类型）的抽象语法数值进行编码或解码。这种单一 ASN.1 类型可以是简单类型也可以是复合类型。复合类型的数据项可以是简单类型或复合类型。

A-XDR 编码规则利用了 DLMS PDU 发送端和接收端都用相同的抽象语法规范运行这一事实。用 BER 时，抽象语法的任何值的编码结构都是 TLV（type-length-value，类型—长度—数值）形式。A-XDR 只在必要时才对数据的类型和长度编码，如不知道编码值的类型，就无法确定编码的结构。

注：这种编码方法使 A-XDR 编码规则不可扩展（见附录 A）。

为使 A-XDR 尽量简单，进行抽象语法编码时，应用以下限制条件：

1）A-XDR 是一种改进的 XDR。这种编码规则由 Unix 环境下事实上的标准 XDR（eXternal Data Representation，外部数据表示，RFC1014）发展而来。

2）见引言的脚注 2）。

a） 不支持 DLMS 以外的 ASN.1 类型数据的编码 [3]；

b） ASN.1 选择（CHOICE）类型数据只用于有显式标记 [4] 的数据项。

A-XDR 是面向字节的编码规则。这意味着每一部分乃至全部的编码的字节数是整数。

4 编码结构

BER 编码（见 ITU-T 的 X.209）的基本结构：类型、长度和内容，这三部分在 BER 中可表示为标识符 I（Identifier）、长度 L（Length）和内容 C（Contents），如图 1 所示。标识符部分用于标识类型，长度部分用于查找内容区 [5] 的末端，内容部分用于传送这种类型的可能值。

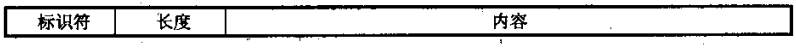


图 1 BER 编码的基本结构

内容域可简化为一系列字节 [6]（原始编码）或一系列嵌套编码（构造编码），如图 2 所示。

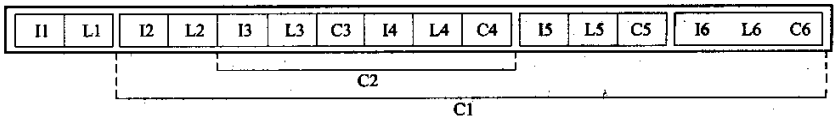


图 2 构造 BER 的编码结构

嵌套的深度按需要决定，并以原始编码或空内容的构造编码结束。

A-XDR 的编码结构与 BER 基本相同，但为了利用 DLMS PDU 发送端和接收端按同样的抽象语法规范运行这一事实，当标识符和长度域传送冗余的信息时，A-XDR 就不对它们编码（不对其中一个域或两个域编码不会导致编码不可解释或含糊不清）。因此，A-XDR 编码的结构如图 3 所示。

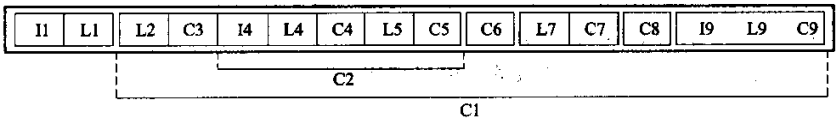


图 3 构造 A-XDR 编码的结构

A-XDR 编码规则规定：

——用于内容域的编码规则；

——如有长度域，对长度域编码；

——如有标识域，对标识域编码。

以下面 ASN.1 复合类型为例：

```
Value ::= SEQUENCE {
        A           Integer16,
        B           Unsigned16
    }
```

3）附录 B 枚举了 DLMS 规范中使用的 ASN.1 类型和关键字。

4）A-XDR 中的术语“显式标记（explicit tagging）”和“隐式标记（implicit tagging）”在 ASN.1 和 BER 中的含义有些不同。在 6.7 里将解释这些名词，并介绍一个新术语“ASN.1 显式标记”。

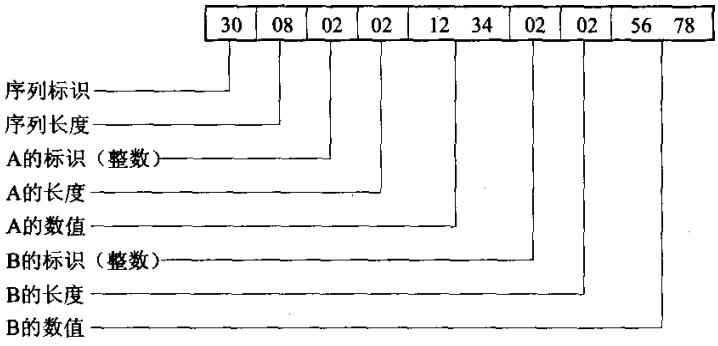
5）实际上，在 BER 中，长度区并不总表示内容区的长度。BER 规定有 2 种形式的长度域：确定型和不确定型。用确定型时，长度域表示内容域的字节数，而用不确定型则表示内容域以内容结束字节结束。

6）零个或多个。

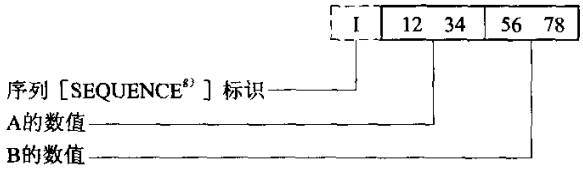
Integer16 ::= INTEGER (–32768～32767)

Unsigned16 ::= INTEGER (0～65535)

假设对 A 和 B 编码，其值分别为 0x1234 和 0x5678 [7]。上述序列的 BER 编码如下：



同一序列的 A-XDR 编码如下：



5 编码规则

对任何 ASN.1 类型的数据进行 A-XDR 编码，其字节数都是整数，每个字节有 8 位。这些字节从最外层的 ASN.1 类型的标识符域编码的第一个字节开始，可认为这个字节为最高位。DL/T 790 的本部分做以下规定：

——不系统地对 A-XDR 编码的字节编号，但有时，为便于理解可加说明（例如，值的第 1 字节等）。

——每个字节的位的编号为 1～8，其中第 8 位是最高位。

5.1 标识符域

标识符域的作用是确定编码值的类型。假如信息的发送端和接收端都以相同抽象句法规范运行，只在以下情况下标识符域才传输信息：

a） 应从不同 CHOICE 类型中选择一种数据类型；

b） 应指出序列（SEQUENCE）中有可选（OPTIONAL）项存在；

c） 应指出序列（SEQUENCE）中有默认（DEFAULT）项存在。

A-XDR 只在上述情况时有标识符域。此外，当 ASN.1 规范（ASN.1 的显式标记，见 6.7）要求对标识符域进行编码时，A-XDR 对标识符编码。

在 a）情况下，A-XDR 要求选择（CHOICE）的所有备选在 ASN.1 中定义为显式标记类型。这时，编码标识形成了标识符域。

在 b）和 c）情况下，可选（OPTIONAL）和默认（DEFAULT）项是否存在可由布尔（BOOLEAN）类型的存在标记表示。这些可选项值的标识符域就是存在标记值的 A-XDR 编码（见 6.9）。

另一方面，当 ASN.1 定义中包含 ASN.1 的显式标记（见 6.7）时，A-XDR 就必须对标识符域编码。这些类型的 A-XDR 编码定义和它们的 BER 编码相同。这样做的目的是使对长度编码，便于省略一些结构。

7）0x...表示后面的数是十六进制的。

8）A-XDR 只在特殊情况下需要编码标识（例如：当本例的 SEQUENCE 是 CHOICE 类型的选择中的一个时）。

这些类型的标识符域是 ASN.1 标记的编码值，所占字节数是整数，至少为 1，如 ITU-T 的 X.209 规定。

5.2 长度域

在 A-XDR 中长度域（如存在）位于内容域的前面。它明确地表示内容区的长度，所占字节数为整数。如信息的发送端和接收端都以相同的抽象句法规范运行，只在对可变长度的 ASN.1 类型编码时长度域才传输信息。可能的情况如下：

a） 长度可变的整数（INTEGER）；

b） 长度可变的位串（BIT STRING）；

c） 长度可变的字节串（BYTE STRING）；

d） 长度可变的类型序列（SEQUENCE OF）。

只在以上情况以及 ASN.1 规范（ASN.1 显式标记，见 6.7）要求时，A-XDR 才对长度域编码。

在 a）、b）、c）和 d）四种情况下，长度域编码为一个可变长度的整数。其他情况，除只有确定形式可以［见脚注 7］使用的限制外，A-XDR 采用与 BER 定义相同的方法（见 ITU-T 的 X.209）。不允许 A-XDR 的长度域用不确定型。

5.3 内容域

内容域是编码的本体。它传输实际值，由零或多个字节组成。以下条款规定了数值编码的方法。

6 编码过程

6.1 整型（INTEGER）值的编码

A-XDR 为 ASN.1 整型提供了两种编码方式。具体用哪种方式取决于整型的 ASN.1 定义的值是否受约束。如一个整型的数值范围确定时（例如数值范围为–128～127），按固定长度的整数编码；如数值范围未确定，按可变长度的整数编码。

6.1.1 固定长度的整数值编码

A-XDR 为固定长度的整数提供了两种不同的编码。对于值不是负数的整数，可以将它表示并编码为无符号二进制数；对于值可以为负数的整数，可以将它表示并编码为 2 的补码表示的二进制数。在这两种情况下，只有整数的值被编码，并形成编码的内容域。这样做的目的是使编码的长度最小。

6.1.1.1 固定长度的无符号整型值编码

当一个整型的数值范围为非负数时，它的编码是一个无符号的二进制数。编码的字节数由规定的取值范围决定，与表示规定范围内任一取值必需的最少字节数相等。固定长度的无符号整数的范围总是边界对齐的。

例：

INTEGER（0～255）的编码为 1 个字节；

INTEGER（0～256）的编码为 2 个字节；

INTEGER（237～256）的编码为 2 个字节。

一个固定长度的无符号整数以无符号二进制符号表示。其编码是一个等于该整数值的无符号二进制数，由第 1 字节的第 8 至第 1 位，接着第 2 字节的第 8 至第 1 位，以下依次为其他各字节，直到编码的最后字节组成。

例：

一个值为 61478 的整型的 A-XDR 编码如下：

第1字节								第2字节							
8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	0	0	1	0	0	1	1	0

6.1.1.2 固定长度的有符号整数值编码

一个整型的数值范围内包含负数时，可按二进制的补码进行编码。编码的字节数由其取值范围决定，与表示规定范围内任一取值必需的最少字节数相等。一个固定长度的整数的范围总是边界对齐的。

例：

INTEGER（–32768～32767）的编码为 2 个字节；

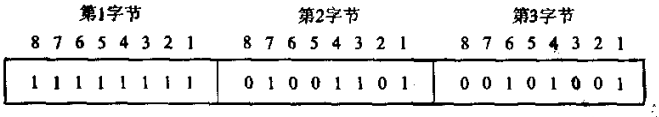
INTEGER（–14300～8700）的编码为 2 个字节；

INTEGER（–32768～32768）的编码为 3 个字节。

编码是一个等于该整数值的以 2 的补码表示的二进制数，由第 1 字节的第 8 至第 1 位，接着第 2 字节的第 8 至第 1 位，以下依次为其他各字节，直到编码的最后字节组成。

例：

一个值为–45783 的整型（–50000～1）的 A-XDR 编码如下：



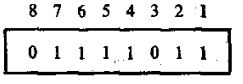
6.1.2 可变长度的整数值编码

如一个整型的取值范围未规定，其编码是可变长度的整型编码。可变长度整数的编码有两种形式，取决于被编码的值。

如一个未受约束的 INTEGER 值位于 0～127（0≤值＜128）之间，值的编码只有一个字节。该字节的第 8 位显然为零。

例：

数值为 123（＝0x7B）的 INTEGER 值的 A-XDR 编码如下：



如一个未约束的整型值的取值范围超出上述 0≤值＜128，其编码有两个域：先是固定长度域，即长度域 Length，它表示后面的可变长度域的长度（字节数），其次是可变长度域，即内容域，它传输编码值，有整数个字节。

长度区在一个字节里编码[9]，第 8 位设置为 1，表示长度区是存在的。其他 7 位编码为一个固定长度的无符号整数，其值表示内容区的字节数。

编码的字节数取决于编码的值，等于表示该值的必需的最少字节数。

编码是一个等于该整数值的以 2 的补码表示的二进制数，由第 1 字节的第 8 至第 1 位，接着第 2 字节的第 8 至第 1 位，以下依次为其他各字节，直到编码的最后字节组成，与固定长度的有符号整数值的编码相同（见 6.1.1.2）。可变长度的整数值的编码结构如图 4 所示。

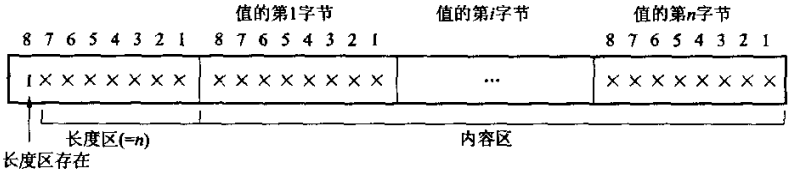
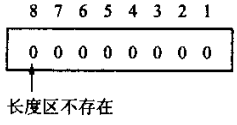


图 4 可变长度的整数值的编码结构

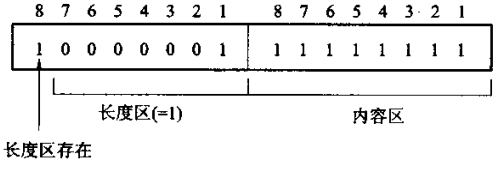
9）长度域的值是 0～127 之间的任意值，最多可表示 1016 位的整型编码。

可变长度的整数值的 A-XDR 编码示例：

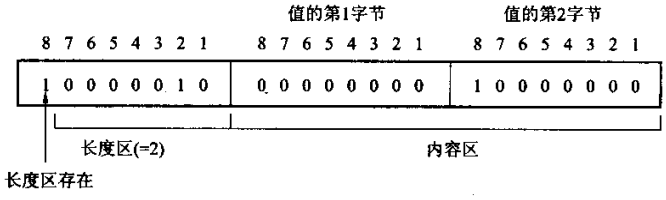
a） 0 的编码结构如下：



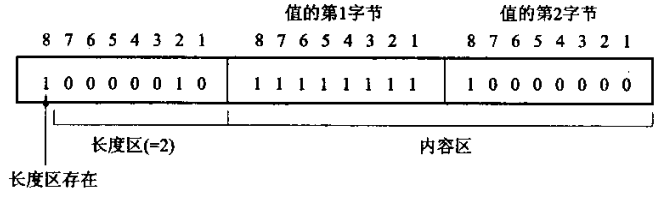
b） –1 的编码结构如下：



c） 128 的编码结构如下：



d） –128 的编码结构如下：

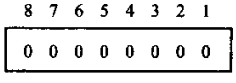


6.2 布尔（BOOLEAN）型值的编码

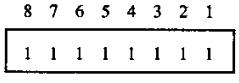
布尔型只能取两种值：TURE 或 FALSE。布尔值的 A-XDR 编码只有内容域，由一个字节组成。如编码值为 FALSE，这个字节为零（所有位都是零）；如值为 TRUE，这个字节可以是任意非零值，由发送者选择。

例：

FALSE 布尔值的 A-XDR 的编码如下：



TRUE 布尔值的 A-XDR 的有效编码如下：



6.3 枚举（ENUMERATED）型值的编码

枚举型 A-XDR 编码的取值范围为 0～255，一个枚举型的 A-XDR 编码及取值范围与 0～255 的整型相同，可作为一个固定长度、受约束的无符号型整数 INTEGER（0～255）编码，由一个字节组成。

6.4 位串（BIT STRING）型值的编码

根据位串在 ASN.1 定义中是否规定了其大小，A-XDR 提供了两种编码方法：位串的大小规定时，为固定长度编码；位串的大小未规定时，为可变长度编码。在这两种情况中，位串型编码的字节的边界都是对齐的，通过增加 0 值的追踪位实现。

6.4.1 规定大小的位串值的固定长度编码

如位串的大小在 ASN.1 中已经规定，其 A-XDR 编码只有内容域。内容域的字节数由规定的大小确定，其值等于传输位串中位的个数必需的最少字节数。

例：

BIT STRING（SIZE（3））的编码为 1 个字节；

BIT STRING（SIZE（8））的编码为 1 个字节；

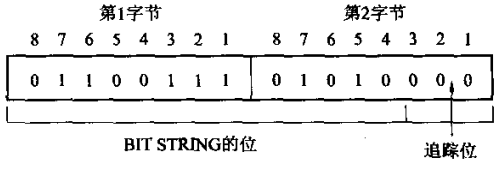
BIT STRING（SIZE（14））的编码为 2 个字节。

位串中的位从第 1 位开始到追踪位结束。先排第 1 个字节的第 8 至第 1 位，接着第 2 个字节的第 8 至第 1 位，以下依次为各字节的第 8 至第 1 位，直到需要的最后一个字节，都由第 8 位开始。

除最后一个字节外，其他每个编码的字节都包括位串的 8 位。最后一个字节包括位串余下的位以及值为零的追踪位。

例：

值为 0110011101010 的 BIT STRING（SIZE（13））的 A-XDR 编码如下：



6.4.2 未规定大小的位串值的可变长度编码

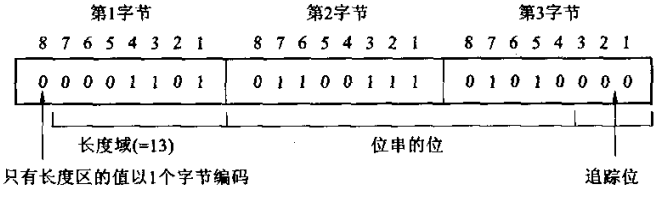
如一个位串在 ASN.1 定义中没有规定其大小，其 A-XDR 编码有两个域：长度域和内容域。

长度域表示的值等于位串编码值的位的个数。长度域本身的编码规则和可变长度的整数的编码相似。但因负值对于长度域没有意义，因此整数的编码为二进制编码，而不是 2 的补码表示的二进制数。（与可变长度的无符号整数的编码相似）

内容域传输位串的编码值有整数个字节。该域的编码规则与 6.4.1 相同。

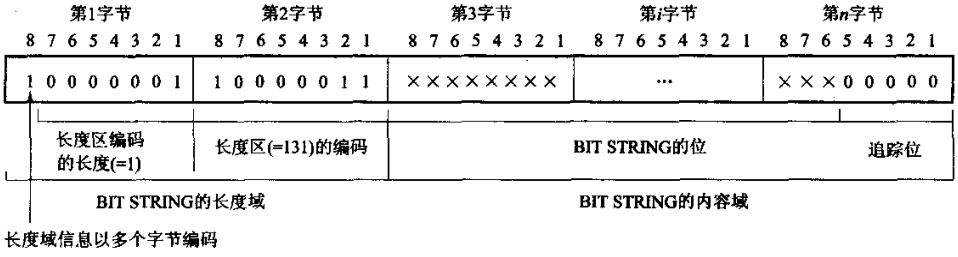
例：

一个值为 0110011101010 的 BIT STRING 的 A-XDR 编码如下：



例：

一个由 131 位组成的 BIT STRING 值的 A-XDR 编码如下：



6.5 字节串（BYTE STRING）型值的编码

根据在 ASN.1 定义中是否规定了字节串的大小，A-XDR 提供了两种编码方法。ASN.1 定义中规定了字节串的大小时，为固定长度编码；未规定字节串的大小时，为可变长度编码。

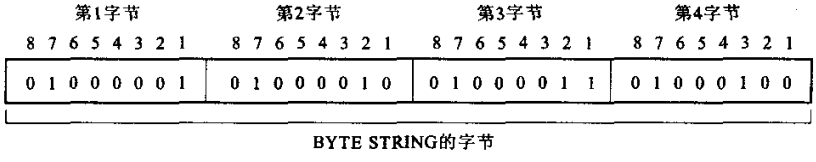
6.5.1 规定大小的字节串的固定长度编码

如字节串的大小在 ASN.1 定义中已经规定，则该字节串的 A-XDR 编码只有内容域。内容域中的字节数等于规定的大小。

字节串的字节从第一个起至最后一个，只需排在内容域的各字节中。

例：

一个值为“ABCD”的 BYTE STRING（SIZE（4））的 A-XDR 编码如下：



6.5.2 未规定大小的字节串的可变长度编码

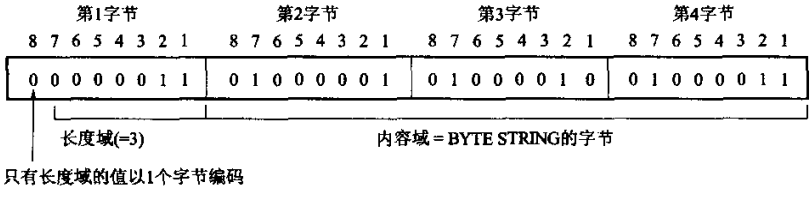
如在 ASN.1 定义中没有规定一个字节串的大小，其 A-XDR 编码包括两个域：长度域和内容域。

长度域表示的值等于内容域的字节数。长度域本身的编码规则与可变长度的字节串的编码相同（见 6.4.2）。

内容域传输字节串的编码值。其编码规则和 6.5.1 中规定的字节串相同。

例：

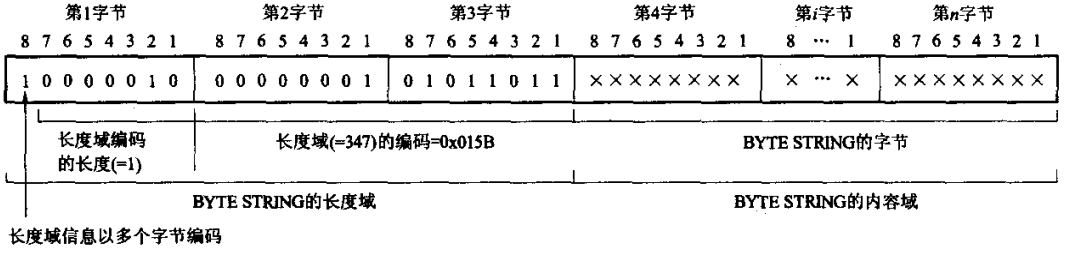
一个值为“ABC”的 BYTE STRING 的编码如下：



例：

一个 BYTE STRING 的 A-XDR 编码由 347 个字节组成[10]：

10）编码第 n 字节中的 n = 长度域字节数 + 内容域字节数 = 3 + 347 = 350。



6.6 选择（CHOICE）型值的编码

A-XDR 的一个主要概念是信息的发送端和接收端按相同的抽象句法运行时，BER 编码的标识符域在大多情况下传输的是冗余的信息。如不对标识符编码不会导致编码含糊不清。因此，A-XDR 不对 ASN.1 型的标识区进行系统的编码。

选择 ASN.1 类型以各数据项类型的集合定义，它的可选择类型必须十分明确。每一个选择类型的值是它的一个可选项[11)]，编码规则应保证任一可选项的编码值不会被含糊地标识。因此，选择类型的值的编码应有标识符域。

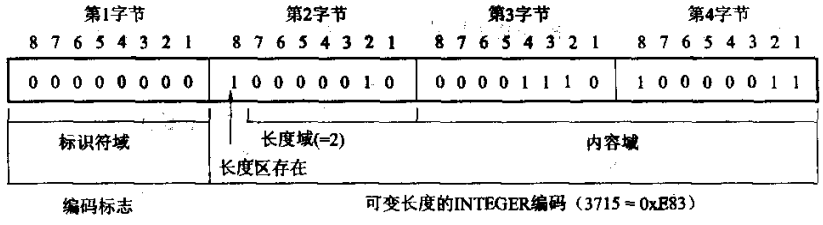
为了表达清楚，一个选择类型的 ASN.1 的所有数据项的类型都应是显式标记[12)]。没有显式标记数据项的选择类型不能用 A-XDR 编码。

一个选择类型值的 A-XDR 编码应是选择的可选类型的 A-XDR 编码，由标识符（标记）的一个字节的编码开始，像固定长度的无符号整数一样。

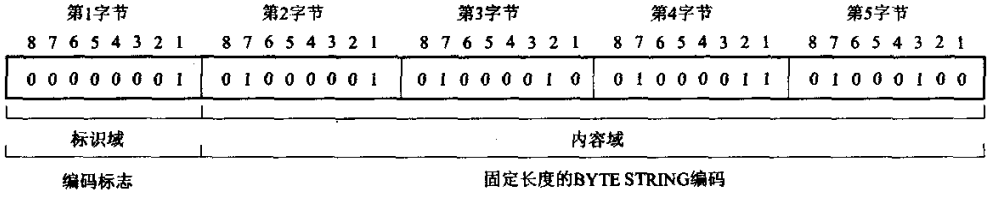
以下面的 ASN.1 选择类型的结构为例：

```
Dummy_PDU::= CHOICE{
    a       [0] INTEGER,
    b       [1] BYTE STRING(SIZE(4))
}
```

当“a”被选择且 a＝3715 时，其 A-XDR 编码如下：



当“b”被选择且 b＝ “ABCD”时，其 A-XDR 的编码如下：



6.7 标记类型（隐式标记、显式标记和 ASN.1 显式标记）

回顾一下 6.6 中的 Dummy_PDU（虚拟协议数据单元）：

11）ASN.1 选择类型与 C 语言的联合（UNION）类型类似。

12）术语“标记”在 6.7 中定义。

```
Dummy_PDU::= CHOICE {
        a       [0] INTEGER,
        b       [1] BYTE STRING(SIZE(4))
}
```

符号［0］整型和［1］字节串是 ASN.1 的标记的符号（由在原来类型的符号前加带方括号的数字得来）。标记用以表示类型之间的区别。

事实上，每种 ASN.1 类型（甚至像整型、位串类型等这些内部 ASN.1 型）都有其标记[13)]。标记分为四类，任何一类中的每种标记以正整数编号。这四类是：通用类、上下文相关类、应用范围类和专用类。内部 ASN.1 型在通用类中有标记，称为基本型。通用类以外的所有类型的标记构成各种标记型。

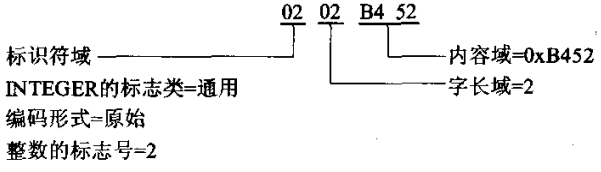
每种标记不是隐式的就是显式的，在标记型中可通过在“]”和基本型之间加关键字“IMPLICIT”或“EXPLICIT”区别。如什么关键字也没有，则为缺省标记。应注意，关键字“IMPLICIT”和“EXPLICIT”形容的是标记而不是标记型，并只在编码中起作用。

在 BER 中，标记类型及其编号在标识符域传递。当其标记为隐式时，其值的 BER 编码有两个标记：一个是新的，在方括号内规定；一个是原来的，是基本型的。编码常由嵌套编码，即由基本型的编码构成。

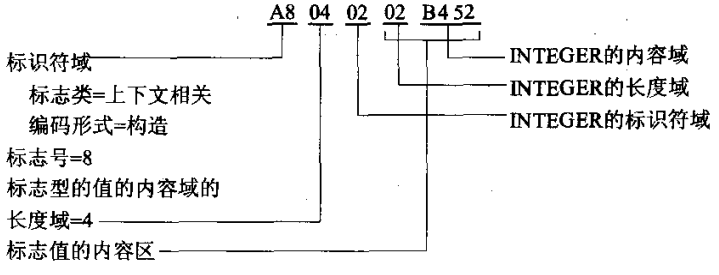
当标记为 IMPLICIT 时，标记值的 BER 编码只有新的标记，它代替原来的标记（包括类和编号）。

例：

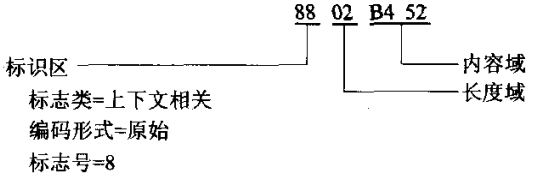
——INTEGER 型的值–19374（0xB452）的 BER 编码如下：



——数值相同，但标记为[8]INTERGER（显式标记，这是 BER 中的默认标记）的 BER 编码如下：



——数值相同，但标记为[8]IMPLICIT INTEGER（IMPLICIT 标记）的 BER 编码如下：



13）除选择类型的每种可选项有标记外，其他任何类型都有各种可能的标记。

在 A-XDR 中，显式标记和隐式标记的差别很小。如 A-XDR 完全不对非标记型的标识符域编码，新的标记替代了旧的就没有意义。因此，隐式标记在 BER 中的意义并不适用于 A-XDR。此外，有两种显式标记适用于 A-XDR，即 A-XDR 显式标记和 ASN.1 显式标记。

在 A-XDR 中，为区分 ASN.1 的选择类型和序列（SEQUENCE）类型，当标记号明确地存在于 ASN.1 规范中时，称为显式标记。这些被标记的数据项的编码规则：选择类型的数据项的标记在 A-XDR 中编码为可变长度的整数，表示标记的值；序列类型的数据项的标记完全不编码。如不论“IMPLICIT”关键字是否存在于 ASN.1 规范中，这些显式标记型的 A-XDR 编码都是相同的，因此可以认为在 A-XDR 中可以忽略“IMPLICIT”关键字。

当标记的类型和编号在 ASN.1 规范中已经明确规定时，例如［APPLICATION 30］，其类型符号是指 A-XDR 中的 ASN.1 的显式标记。这种标记类型只适用于序列类型的数据项。

除长度域不允许用不确定类型外，ASN.1 显示标记类型的值的 A-XDR 编码和它的 BER 编码相同。这种标记的目的只是使长度必须编码，从而便于在结构上有所省略。

6.8 可选（OPTIONAL）和默认（DEFAULT）数据项（COMPONET）

任一种 ASN.1 复合类型都含有带 ASN.1 关键字“OPTIONAL”和“DEFAULT”的数据项。在 ASN.1 规范中，这些关键字可以放在数据项类型的后面，用以十分直观地表达它的意思。带有可选“OPTIONAL”标记的数据项可能被省略，它的值在编码中不一定存在（省略这些数据项的实际条件和意义应由设计者规定）。

一个数据项是可选的原因之一是假设它为一个特定值时（一般来说，该值经常出现）可以将它省略。用这种方法可以避免显式地传送这个值。ASN.1 的关键字“DEFAULT”用来表示数据项的默认值。

一般来说，一个数据项是可选值还是默认值很不一样（在 ASN.1 中它们相互排斥）。可选数据项可以完全省去，而 DEFAULT 数据项实际上总是存在的，因为即使将它省去它也还代表一个特定值。

可选或默认数据项的 A-XDR 编码由使用标记（usage flag）的附加元素（这是 ASN.1 语法的补充，ASN.1 语法没有规定此元素）开始。这个使用标记为布尔型，其值表示可选或默认数据项的值是否在编码中存在，如下所示：

——OPTIONAL 数据项：

usage flag = TRUE	该数据项存在于编码中
usage flag = FALSE	该数据项不存在于编码中

——DEFAULT 数据项：

usage flag = TRUE	该数据项存在于编码中（其值和 DEFAULT 值不同）
usage flag = FALSE	该数据项不存在于编码中（传送的是 DEFAULT 值）

使用标记表明所述的数据项存在于编码中时，使用标记位于可选或默认数据项值的 A-XDR 编码之后。如使用标记表明该数据项不存在于编码中，该数据项以使用标记的编码结束 A-XDR 编码，后面不再有数据项的值的编码。

在 6.9 和附录 C 中给出了可选和默认数据项的编码示例。

6.9 序列（SEQUENCE）型值的编码

与选择类型类似，ASN.1 序列类型也是根据数据项类型的集合定义的，他们都非常明确。但是与选择类型不同的是，序列类型中的每一数值都有每种数据项类型的一个值[14]。数据项值在编码中出现顺序是固定的，与定义中各数据项类型出现的次序相同。

序列值的 A-XDR 编码应是从序列型的 ASN.1 定义表所列每一类型中的数据值的 A-XDR 编码，按它们在定义中的顺序出现，除非该类型与关键字可选“OPTIONAL”或默认“DEFAULT”有关。

序列值的有显式标记数据项的标记是冗余的信息，因此不对该标记编码，即有显式标记的数据项值

14）数据项类型标为 OPTIONAL 或 DEFAULT 的除外（见 6.7 和 6.8）。

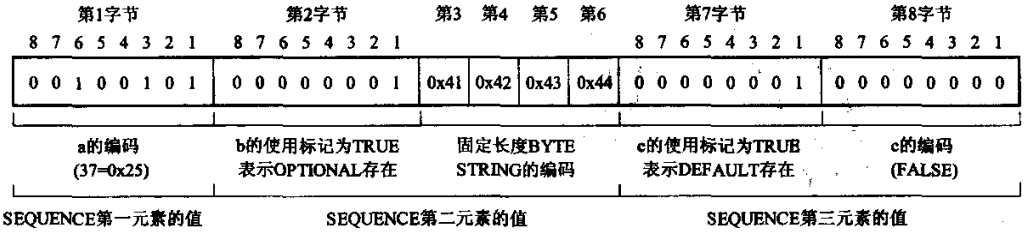
的 A-XDR 编码是该数据项值的 A-XDR 编码。

如可选或默认数据值的编码存在，它在编码（按 6.8 的规定）中出现的位置与在 ASN.1 定义中出现的相对应。

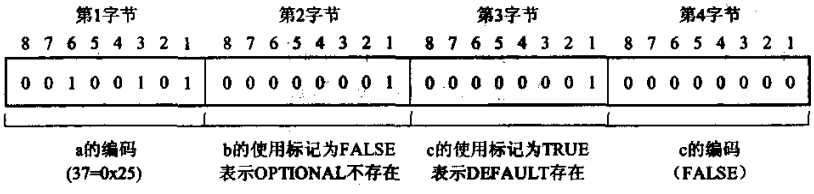
以下面的 ASN.1 定义为例：

```
Dummy_PDU ::= SEQUENCE {
    a       INTEGER(0...127),
    b       OCTET  STRING(SIZE(4)) OPTIONAL,
    c       [1] BOOLEAN DEFAULT TRUE,
}
```

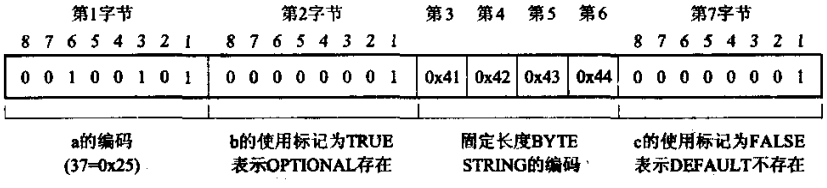
——当 a = 37，b = “ABCD”，c = FALSE 时，该 SEQUENCE 型值的 A-XDR 编码如下：



——当 a = 37，b 不存在，c = FALSE 时，该 SEQUENCE 型值的 A-XDR 编码如下：



——当 a = 37，b = “ABCD”，c = TRUE（传送 DEFAULT 值）时，该 SEQUENCE 型值的 A-XDR 编码如下：



6.10 SEQUENCE OF 型值的编码

ASN.1 的 SEQUENCE OF 型按单个数据项类型定义，它的值是数据项类型值的有序集合。

根据 SEQUENCE OF 型在 ASN.1 定义中是否规定了其大小，A-XDR 可提供两种编码方式。如在 ASN.1 定义中已规定 SEQUENCE OF 的大小，用固定长度编码；如没有规定大小则用可变长度编码。

6.10.1 规定大小的 SEQUENCE OF 型值的固定长度编码

如在 ASN.1 中已规定 SEQUENCE OF 的大小，其 A-XDR 编码只有内容区。内容区的字节由 ASN.1 定义所列类型的 *N* 个数据值的 A-XDR 编码组成，这里的 *N* 即为规定的大小。这些数据值的编码顺序和被编码的 *SEQUENCE OF* 值的数据值相同。

以下面的 ASN.1 类型定义为例：

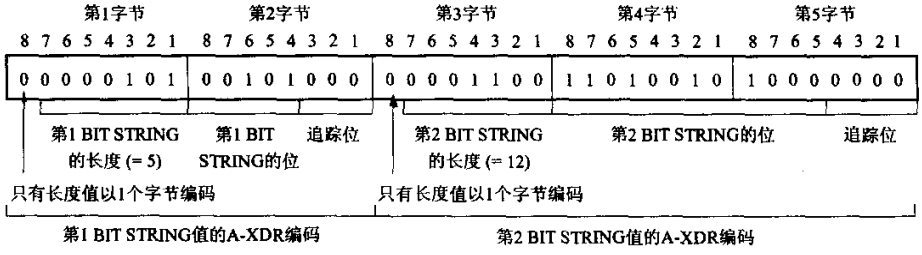
```
Dummy_List ::= SEQUENCE (SIZE(2)) OF BIT STRING
```

BYTE STRING 数据项的编码如下：

第 1 BYTE STRING: 00101

第 2 BYTE STRING: 110100101000

该 SEQUENCE OF 型的值的 A-XDR 编码如下：



6.10.2 未规定大小的 SEQUENCE OF 型值的可变长度编码

ASN.1 定义中未规定 SEQUENCE OF 的大小时，其 A-XDR 编码有两个区：长度域和内容域。

长度区表示的值为 SEQUENCE OF 型的编码值的数据项的个数。长度区的编码方式和可变长度的 BIT STRING 的长度区相同（见 6.4.2）。

内容区应由 ASN.1 的定义所列类型的 N 个数据值的 A-XDR 编码组成。这里的 N 为长度区表示的值。这些数据值的编码顺序应与被编码的 SEQUECE OF 的值的顺序相同。

以下面的 ASN.1 类型定义为例：

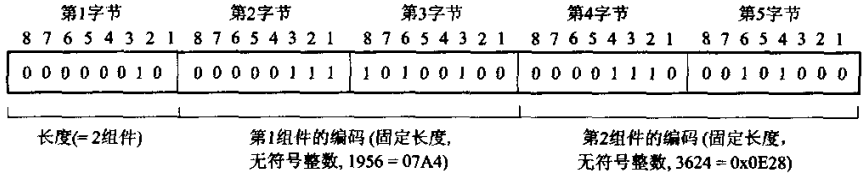
Dummy_List ::= SEQUENCE OF INTEGER(0...4000)

要编码的两个数据项为：

INTEGER 1 = 1956

INTEGER 2 = 3624

该 SEQUENCE OF 型的上述值的 A-XDR 编码如下：



6.10.3 特殊选择序列（SEQUENCE OF CHOICE）型的编码

选择序列（SEQUENCE OF CHOICE）型是特殊情况。虽然 SEQUENCE OF 类型是按单个数据项类型定义的，但选择序列结构的元素的类型不同，这就是选择类型的各种可选项。

除 SEQUENCE OF 型的每个数据项的编码以所选的选择数据项的标记的编码开始外，选择序列的编码方法与按 6.10.1 和 6.10.2 所述方法相同。

附录 C 的［例 C.5］为选择序列类型编码示例。

6.11 可视串（VisibleString）型的编码

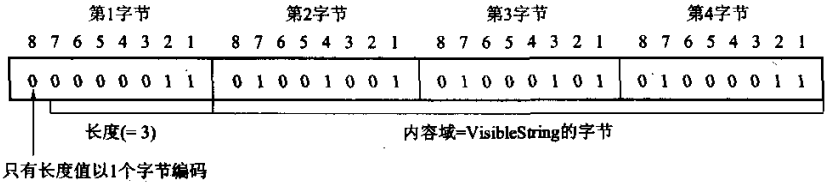
可视串（VisibleString）ASN.1 是一种限定的字符串类型。限定的字符串是一种特殊的字节串，可以包含取自限定的字符集中的字符。虽然 ASN.1 规定了 8 种类型，但只有可视串类型在 DLMS 规范中使用（见 DL/T 790.441—2004）。因此 A-XDR 只支持这种 ASN.1 类型 [15]。

ASN.1 为每种特殊字符串类型都定义了通用类的标记。像其他 ASN.1 的内部类型的标记一样，在 A-XDR 中这些标记不编码。因此，可视串的编码方法与字节串相同。如 DL/T 790.441—2004 中只规定

15）虽然其他 ASN.1 限定的字符串类型也可能用这种编码方法。

可变长度的可视串，则只可以使用可变长度的字节串（见 6.5.2）。

以可视串的“IEC”值的 A-XDR 编码为例：



6.12 通用时间（GeneralizedTime）型的编码

通用时间（GeneralizedTime）以可视串（VisibleString）为基础，是一种 ASN.1 的有用型（Useful Type），定义如下：

GeneralizedTime ::= [UNIVERSAL 24] IMPLICIT VisibleString

如 A-XDR 不对通用（UNIVERSAL）类 ASN.1 的标记编码，则与可视串编码相似，通用时间型的 A-XDR 编码和字节串相同。也和可视串的原因一样，可变长度的字节串的编码适用于通用时间型的编码（见 6.5.2）。

6.13 ASN.1 空值（NULL）的编码

ASN.1 的空值（NULL）是一种很特殊的类型：只有一个值。因此，容纳信息的能力很有限。人们可能认为空值没有用，然而，在上下文内容很少的情况下，例如，必须送出一个数据类型，但又不需要传送任何信息时，它还是很有用的。可以认为空值丰富了一些没有值的类型的域。

例如，要发送一个 BOOLEAN 值，这个值有时已知，有时又不存在，就可用以下类型：

```
OutputValue ::= CHOICE {
        Known       [0] BOOLEAN
        Unknown     [1] NULL
}
```

为了能用 A-XDR 编码，空值应是标记型。

用 6.7 规定的标记值 A-XDR 编码的方法进行空值的 A-XDR 编码。

附 录 A
（资料性附录）
可 扩 展 性

第 3 章曾提到 A-XDR 编码规则与 BER 不同，是不可扩展的。这是什么含义呢？

下面以 SEQUENCES 为例：

```
dummy_sequence_1   SEQUENCE
  {
        a       INTEGER,
        b       INTEGER
  }
dummy_sequence_2   SEQUENCE
  {
        a   INTEGER,
        c   BOOLEAN OPTIONAL,
        b   INTEGER
  }
```

SEQUENCE 值的 BER 编码在 OPTIONAL 数据项“c”不存在相同。这意味着，如采用 BER，在应用协议将来的版本中，可以将更多的可选数据项加到序列中。如这些数据项被省略，例如，用一个按老规范开发的过程进行通信时，这个过程表现相同。BER 编码规则的这一性质称为可扩展性。

从这种意义上说，A-XDR 编码是不可扩展的。实际上，由以上讨论可知，即使“c”不存在，A-XDR 编码还与可选（OPTIONAL）数据项有关。

另一方面，A-XDR 也不是封闭的。它不但可以为现行 DLMS 规范中的 ASN.1 类型编码，如必要，还可以为其他 ASN.1 类型编码，例如实数（REAL）、集合（SET OF）等。

注：IEC TC57 的第 9 工作组负责将 A-XDR 不断更新。

附 录 B
（资料性附录）
DLMS 中用的 ASN.1 类型和关键字

以下内部 ASN.1 类型用于 DLMS 规范（见 DL/T 790.441—2004）：

INTEGER	（整数）
BOOLEAN	（布尔）
ENUMERATED	（枚举）
BYTE STRING	（字节串）
BIT STRING	（位串）
VisibleString	（可视串）
CHOICE	（选择）
SEQUENCE	（序列）
SEQUENCE OF NULL	（空序列）

DL/T 790.441—2004 也用一种 ASN.1 Useful Type（有用型），即

GeneralizedTime	（通用时间）

在 DLMS 中定义了以下一些有用类型：

Integer8	（8 位整数）
Integer16	（16 位整数）
Integer32	（32 位整数）
Unsigned8	（8 位无符号）
Unsigned16	（16 位无符号）
Unsigned32	（32 位无符号）

在 DLMS 规范中还用了以下一些 ASN.1 关键字：

IMPLICIT	（显式）
OPTIONAL	（可选）
DEFAULT	（默认）

附 录 C
（资料性附录）
DLMS PDU 的 A-XDR 编码示例

DLMS PDU 的最外层的 ASN.1 类型是选择（CHOICE）类型。这里只提出一些未加密的，按以下定义的协议数据单元作为示例：

```
DLMSpdu ::= CHOICE {
    confirmServiceRequest          [0]      ConfirmedServiceRequest,
    initiateRequest                [1]  IMPLICIT  InitiateRequest,
    getStatusRequest               [2]  IMPLICIT  GetStatusRequest,
    getNameListRequest             [3]  IMPLICIT  GetNameListRequest,
    getVariableAttributeRequest    [4]  IMPLICIT  GetVariableAttributeRequest
    readRequest                    [5]  IMPLICIT  ReadRequest
    writeRequest                   [6]  IMPLICIT  WriteRequest
    confirmedServiceResponse       [7]            ConfirmedServiceResponse,
    initiateResponse               [8]  IMPLICIT  InitiateResponse
    getStatusResponse              [9]  IMPLICIT  GetStatusResponse,
    getNameListResponse            [10] IMPLICIT  GetNameListResponse
    getVariableAttributeResponse   [11]           GetVariableAttributeResponse
    readResponse                   [12] IMPLICIT  ReadResponse
    writeResponse                  [13] IMPLICIT  WriteResponse
    confirmedServiceRequest        [14]           ConfirmedServiceRequest
    unconfirmedServiceRequest      [15]           UnconfirmedServiceRequest
    abortRequest                   [16] IMPLICIT  AbortRequest
    unsolicitedWriteRequest        [17] IMPLICIT  UnsolicitedWriteRequest
    unsolicitedServiceRequest      [18]           UnsolicitedServiceRequest
    informationReportRequest       [19] IMPLICIT  InformationReportRequest
```

相应于加密的协议数据单元的其他选择：

```
    ...
    ded-informationReportRequest   [88] IMPLICIT BYTE STRING
}
```

[例 C.1] InitiateRequest DLMS PDU 的 A-XDR 编码。

InitiateRequest 型的定义如下：

```
InitiateRequest::=SEQUENCE{
    dedicated-key                  BYTE STRING OPTIONAL,
    respond-allowed                BOOLEAN DEFAULT TRUE,
    proposed-quality-of-service    [0] IMPLICIT Integer8 OPTIONAL,
    proposed-dlms-version-number   Unsigned8,
    proposed-conformance           Comformance,
    proposed-max-pdu-size          Unsigned16
}
```

其中，Conformance 可能如下：

Conformance::=[APPLICATION 30] IMPLICIT BIT STRING (SIZE(16))

{

get-data-set-attribute	(0),
get-ti-attribute	(1),
get-variable-attribute	(2),
read	(3),
write	(4),
unconfirmedWrite	(5),
change-scope	(6),
stop-resume	(7),
make-usable	(8),
data-set-load	(9),
selection-in-get-name-list	(11),
detailed-access-low-bit	(12),
detailed-access-high-bit	(13),
multiple-variable-list	(14),
data-set-upload	(15)

}

本例优先采用以下值：

dedicated-key	= 不存在
response-allowed	= TRUE (默认值)
proposed-quality-of-service	= 存在，值为 4
proposed-dlms-version-number	= 1
proposed-max-pdu-size	= 134 (= 0x86)
proposed-conformance	= 0x1C00，如下所示：

B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	
0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	=0x1C00[16]

具有上述值的 initiateRequest PDU 的 A-XDR 编码如下：

01	DLMS PDU CHOICE （InitiateRequest）的（显式）标记	
00	dedicated-key 数据项的使用标记（FALSE，不存在）	
00	response-allowed 数据项的使用标记（FALSE，传送 DEFAULT 值）	
01	proposed-quality-of-service 数据项的使用标记（TRUE，存在）	
04	proposed-quality-of-service 数据项的编码（4）	
01	proposed-dlms-version-number 数据项的编码（1）	
5E	[APPLICATION 30]标记（ASN.1 显式标记）的编码	Comformance 数据项的 BER 编码
03	内容区字节长度（3）	
00	编码最后一个字节中未用的位的编号（0）	
10		
3C	Proposed-conformancd 数据项的编码	
00		
86	proposed-max-pdu-size 数据项值的编码（0x86）	

16）0x1C00 值表示除支持必备服务外建议也支持以下服务：读（第 3 位）、写（第 4 位）、非确认写（第 5 位）。

[例 C.2] InitiateResponse DLMS PDU 的 A-XDR 编码。

InitiateResponse 型定义如下：

```
InitiateResponse：：= SEQUENCE
{
  negotiated-quality-of-service      [0]  IMPLICIT Integer8 OPTIONAL,
  negotiated-dlms-version-number     Unsigned8,
  negotiated-conformance             Conformance,
  negotiated-max-pdu-size            Unsigned16,
  vaa-name                           ObjectName
}
```

Conformance 的定义与［例 C.1］相同，ObjectName 定义如下：

```
ObjectName ：：= Integer16
```

此外，vaa-name 是有效的，这意味着 vaa-name 为 111（即 vaa-name Integer16 的最低三位为 111）。

Negotiated-quality-of-service、negotiated-dlms-version-number、negotiated-conformance 和 negotiated-max-pdu-size 的值与［例 C.1］相同；vaa-name 的值选择为 0x0037。

InitiateRequest PDU 为上述值时，A-XDR 编码如下：

08	DLMS PDU CHOICE（InitisteResponse）的（显式）标记	
01	negotiated-quality-of-service 数据项的使用标记（TRUE，存在）	
04	negotiated-quality-of-service 数据项的编码（4）	
01	negotiated-quality-of-service 的编码（1）	
5E	[APPLICATION 30]标记（ASN.1 显式标记）的编码	一致性数据项的 BER 编码
03	内容区字节长度（3）	
00	最后一个字节中未使用位的号（0）	
1C 00	negotiated-conformance 数据项的编码	
00 86	negotiated-max-pdu-size 数据项的编码（0x86）	
00 37	vaa-name 值的编码（0x0037）	

[例 C.3] 确认的服务出错 PDU 的 A-XDR 编码。

本例说明当 InitiateRequest 服务不被 DLMS 服务器接受时的 DLMS PDU 的 A-XDR 编码。服务不被接受的原因是建议的内容不适用于所用服务器的服务表。

Confirmed Service Error（确认的服务出错）定义如下：

```
ConfirmedServiceError ：：= CHOICE
{
  -- 标记 0 保留
  initiateError                  [1]  ServiceError
  getStatus                      [2]  ServiceError
  getNameList                    [3]  ServiceError
  ...
  terminateUpload                [19] ServiceError
}
```

其中 ServiceError 意义如下：

```
ServiceError ： ： = CHOICE
{
    ...
    Initiate                        [6]   IMPLICIT ENUMERATED
    -- 初始化服务出错
    {
        other                       (0),
        dlm-version-too-low         (1), --建议的 DLMS 版本太低
        incompatible-conformance    (2), --建议的服务不够用
        pdu-size-too-short          (3), --建议的 PDU 尺寸太大
        refused-by-the-VDE-Handler  (4) --不能或不允许创建 vaa
    }
    ...
}
```

因此，在上述情况下，ConfirmedServiceError PDU 的 A-XDR 编码如下：

0E　DLMS　PDU　CHOICE 的（显示）标记（ConfirmedServiceError = 14）

01　ConfirmedServiceError CHOICE 的（显式）标记（initiateError = 1）

06　ServiceError CHOICE 的（显式）标记（initiate = 6）

02　ENUMERATED 值的编码（incompatible-conformance = 2）

[例 C.4] 读状态请求/响应 PDU 的 A-XDR 编码。

本例说明在以下情况下读状态请求（Get Status Request）PDU 及其响应的 A-XDR 编码：

——请求不需要条件参数（Indentify = FALSE）。

——用于接收请求的 VDE 实例规定如下：

```
Dummy_VDE: ： =  VDE
{
    VDE-handler                 ,   --VDE-handler 在其他地方定义
    VDE-type            0x0001,     --VDE 类型值（在其他地方定义）
    Serial-Number       "1234",     --在其他地方定义
    Vendor-Name                 ,   --由制造商定义
    Model                       ,   --由制造商定义
    Version-Number              ,   --由制造商定义
    Resource                    ,   --在其他地方定义
    List-of-vaa         (7,15,23),  --本 VDE 中的 VAA 列表
    Status              READY
}
```

GetStatusRequest 定义如下：

```
GetStatusRequest::=Identify
```

其中　Identify ::= BOOLEAN

因此，在上述情况下，GetStatusRequest PDU 的 A-XDR 编码如下：

02　DLMS PDU CHOICE 的（显式）标识（GetStatusRequest = 2）

00　Identify BOOLEAN 的 A-XDR 编码（FALSE 表示响应中只有必备参数）

GetStatusResponse 定义如下：

```
GetStatusResponse ::=   SEQUENCE
```

```
{
    vde-type                    Integer16,
    serial-nmuber               BYTE STRING,
    status                      ENUMERATED
        {
            ready               (0),
            nochange            (1),
            inoperable          (2)
        } DEFAULT ready,
    list-of-vaa                 SEQUENCE OF ObjectName,
    identify                    SEQUENCE
        {
            resource            VisibleString
            vendor-name         VisibleString
            model               VisibleString
            version-number      Unsigned8
        } OPTIONAL
}
```

相应于标识为 FALSE 的 GetStatusRequest 的 GetStatusResponse PDU 的 A-XDR 编码如下：

```
09      DLMS PDU  CHOICE（GetStatusResponse）的（显式）标记
00 ┐
01 ┘   VDE-Type 值（Integer16 = 0x0001）的编码
04      Serial-Number（可变长度）BYTE STRING 的长度（= 4）
31 ┐
32 │
33 │
34 ┘   Serial-Number BYTE STRING（= "1234"）的内容区
00      状态数据项的使用标记，它不存在于编码中 17)
03      List-of-vaa 表的长度（可变长度）SEQUENCE OF（= 3）
00 ┐
07 ┘   表中第 1 个 VAA-Name（0x0007）

00 ┐
0F ┘   表中第 2 个 VAA-Name （15 = 0x000F）

00 ┐
17 ┘   表中第 3 个 VAA-Name（23 = 0x0017）
00      OPTIONAL Identify SEQUENCE 的使用标记（FALSE，表示不存在）
```

[例 C.5] 变量访问服务

DLMS 命名变量（Named Variable）定义如下：

```
Dummy_NamedVariable ::=     NamedVariable
```

17）此值为传送协议数据单元的默认值。

```
{
    Variable-Name                   0x0010,
    Scope-Of-Access                 VDE 特定,
    Scope-May-Change                FALSE,
    Lifetime                        VDE,
    Type-Description                Dummy_Type,
    Read-Write Flag                 READ-WRITE
    Available                       TRUE
}
Dummy_Type ::=    SEQUENCE
{
    Number_Of_Counters Unsigned8,
    Counter_Values                  SEQUENCE OF Unsigned16
}
```

作为 ReadRequest 和 ReadResponse 的例子，参照 Dummy_NamedVariable，可以假设在 VDE 中该变量的值如下：

```
    Number_Of_Counters  = 2
Counters SEQUENCE   OF 有以下两个值：
    Counter_Value_1     = 318 (=0x013E)
    Counter_Value_2     = 715 (=0x02CB)
```

下面的示例要用命名变量。

[例 C5.1] 读没有具体访问的命名变量。

ReadRequest Service 定义如下：

```
    ReadRequest ::=   SEQUENCE OF VariableAccessSpecification
```

其中：

```
    VariableAccessSpecification ::=   CHOICE
{
    variable-name       [2]   IMPLICIT ObjectName,
    detailed-access     [3]   IMPLICIT SEQUENCE
                {
                    Variable-name       ObjectName
                    Detailed-access     DetailedAccess
                }
}
```

这时，应注意没有具体访问（detailed access）的访问。要求 Dummy_NamedVariable 的值的 ReadRequest PDU 的 A-XDR 编码如下：

```
    05      DLMS PDU CHOICE 的（显式）标记（ReadRequest = 15）
    01      可变长度的 SEQUENCE OF 的长度（= 1）
    02      VariableAccessSpecification CHOICE 的（显式）标记（variable-name = 2）
    00 ┐
    10 ┘    可变参照 ObjectName 的编码（variable-name = 0x0010）
```

如请求可接受，该请求的响应为 ReadResponse，定义如下：

```
    ReadResponse ::=    SEQUENCE OF CHOICE{
```

```
        data                    [0] Data
        data-access-error       [1] IMPLICIT    dataAccessError
    }
```

其中：

```
Data ::=    CHOICE{
        array               [1] IMPLICIT    SEQUENCE OF Data
        structure           [2] IMPLICIT    SEQUENCE OF Data
        •                       •               •
        unsigned            [17] IMPLICIT   Unsigned8
        long-unsigned       [18] IMPLICIT   Unsigned16,}[18)
```

包含给定值的 Dummy_NamedVariable 的 ReadResponse PDU 的 A-XDR 编码如下：

```
0C      DLMS PDU CHOICE 的（显式）标记（ReadRespose = 12）
01      ReadRespose 的可变长度 SEQUENCE OF CHOICE 的长度（1）
00      CHOICE 的（显式）标记（选择的数据项是 [0] Data）
02      CHOICE 的（显式）标记（选择的数据项是 [2] structure）
02      可变长度 SEQUENCE OF Data（CHOICE）的长度（结构有 2 个数据项）
11      结构第一数据项的（显式）标记（[17] Unsigned8）
02      Number_Of_Counters 的编码值（= 2）
01      结构第二数据项的（显式）标记（[1]， array）
02      Array（SEQUENCE OF Data （CHLOICE））的长度（= 2，array 中有两个 Data）
12      第一 Data 的（显式）标记（[18] Unsigned16）
01
3E      Counter_Value_1 的编码值（= 0x013E）
12      第二 Data 的（显式）标记（[18] Unsigned16）
02
CB      Counter_Value_2 的编码值（= 0x02CB）
```

18）此时只需要这些 CHOICE 型。