# ASN.1(Abstract Syntax Notation 1)

## 基本概念

ASN.1: Abstract Syntax Notation One, as defined in X.208.

### 约定

ASN.1区分大小写

标识符由大、小写字母,数字和横线组成,如sysName  
内部类型标识符全部大写,如BOOLEAN,INTEGER。  
用户定义的类型名和模块名以大写字母开头,School

### 符号说明

|  |  |
| --- | --- |
| 符号 | 含义 |
| ::= | 定义为 |
| | | 或 |
| -- | 后面是注释(行) |
| {} | 清单的开始和结束 |
| [] | 标签的开始和结束 |
| () | 子类型的开始和结束 |
| .. | 范围 |

### 语法

ASN.1语法遵循传统的巴科斯范式BNF风格．最基本的表达式如：

Name ::= type . 表示为定义某个名称为Name的元素，它的类型为type．

例如：　MyName ::= IA5String . 表示为定义了一个名为MyName的元素或变量，其类型为ASN.1类型IA5String (类似于ASCII字符串).

#### ASN.1显式值(Explict Value)．

有些时候，我们需要定义一种ASN.1类型，它的子集元素包含预定义值．

Name ::= type(Explict Value)

MyName ::=IA5String (Tom) 表示MyName是字符串Tom的IA5String编码．

又例如：MyName ::= IA5String(Tom|Joe) 表示字符串的值既可以是Tom, 也可以是Joe.

这种语法的使用是为了扩展确定的解码器．例：

        PublicKey::= SEQUENCE {

           KeyType       BOOLEAN(0),

           Modulus        INTEGER,

           PubExponent INTEGER

        }

       PrivateKey ::= SEQUENCE {

           KeyType       BOOLEAN(1)

           Modulus         INTEGER,

           PubExponent  INTEGER,

           PrivateExponent INTEGER

         }

#### ASN.1修改器

* 可选(OPTIONAL)。表示改变一个元素以便在编码时它的类型是可选择的．即编码器可以忽略这个元素，解码器不能假设它将出现．但当邻接的两个元素具有相同的类型时，会给解码器带来一些问题．

        定义： Name ::= Type OPTIONAL

* 默认(DEFAULT)．默认修改器允许容器包含默认值．如果待编码的数据值等同于它的默认值，那么它将在发送的数据流中被忽略．例如：

　　　　　 Command::= SEQUENCE {

                                        Token         IA5String(NOP) DEFAULT,

                                        Parameter   INTEGER

                      }

* 选择(CHOICE).选择修改器允许一个元素在给定的实例中可以有多个可能值．实质上说，解码器将尝试所有期望的解码算法，直到有一个类型符合为止．当一个复杂的容器中包含其他容器时，时候选择器就十分有用了．例如：

　　　　　 UserKey::= SEQUENCE {

         Name            IA5String,                       StartDate       UTCTIME,

                                      Expire           UTCTIME,

                                      KeyData        CHOICE {

                                            ECCKey        ECCKeyType,

                                             RSAKey        RSAKeyType

                                      }

                       }

### TAG

通信时，要确保对方知道每个数据的类型，我们需给各个类型进行系统性的编号。asn.1中，这种编号称为tag

#### 四种tag类

universal, application, context-specific和private

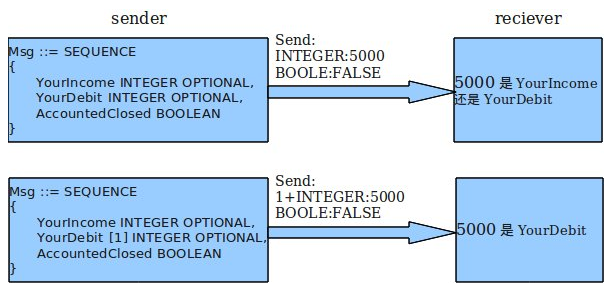
1. UNIVERSAL的Tag是ASN.1标准定义的，给每一种内建类型定义一个固定tag值。

2.APPLICATION的Tag，唯一标志应用内的一个类型。如：

Order-number ::= [APPLICATION 0] NumericString

但是因为使用IMPORTS等方式下，很难保证唯一性，所以这种Tag类已经不推荐使用了。

3.**context-specific**的Tag只能出现在SEQUENCE、SET和CHOICE类型的组件中，如果这些结构类型本身也是另一个结构类型的组件，也可以使用。只要不产生歧义，相同的tag数值可以在不同结构中反复使用。



4.private tag，在一家公司或一个国家内唯一标志一个类型，扩展传输层PDU：

RejectTPDU ::= SET

{

destRef [0] Reference,

yr-tu-nr [1] TPDUnumber,

credit [2] Credit,

extended [PRIVATE 0] BOOLEAN DEFAULT FALSE

}

PRIVATE类的Tag现在也不推荐使用

#### tag模式IMPLICIT,EXPLICIT

1.一个类型被声明了IMPLICIT tag，编码时会用新的tag(即IMPLICIT前面[]中的)值**替换**旧的值(IMPLICIT后面[]中的tag值或universal tag)：

Number1 ::= [0] IMPLICIT INTEGER

Number2 ::= [0] IMPLICIT [1] INTEGER

number1,Number2 的tag值都是0。

复合类型sequence, set被声明了IMPLICIT tag，那么它的成员和成员的成员都默认为IMPLICIT，除非手工为这些成员设定了另外的tag模式：

Msg ::= [0] IMPLICIT SEQUENCE

{

YourIncome [0] INTEGER OPTIONAL,

YourDebit [1] INTEGER OPTIONAL,

AccountedClosed [2] EXPLICIT BOOLEAN

}

Msg 的yourIncome, YourDebit成员的tag是 IMPLICIT模式, AccountedClosed成员tag是 EXPLICIT模式。

2.一个类型被声明了 EXPLICIT tag，编码时会在旧的tlv基础上**嵌套一层**TLV,T是 EXPLICIT tag值，L是tlv的长度，V即原tlv。

当没有其它更多描述信息，并且使用BER编码时，会默认为EXPLICIT模式，将UNIVERSAL类和context-specific类的Tag同时编码出来:

Afters ::= SEQUENCE

{

cheese [0] IA5String,

dessert [1] IA5String

}

3.tag模式用于指定tag在ber编码中是使用嵌套方式还是直接方式，因此

Number ::= IMPLICIT INTEGER，Number ::= EXPLICIT INTEGER和Number ::= [UNIVERSAL 2] IMPLICIT INTEGER都等效于Number ::= INTEGER

#### 全局tag模式

在模块定义时声明tag模式，会对整个模块内的类型产生影响

1.全局IMPLICIT模式：

M DEFINITIONS IMPLICIT TAGS ::=

BEGIN

    T ::= SEQUENCE

    {

        a    [0] INTEGER,

        b    [1] SEQUENCE

        {

            i      [0] INTEGER,

            n     [1] NULL

        },

        c    [2] REAL

    }

END

模块M中的成员a,b,c和b的成员tag都是 IMPLICIT模式。

为了避免冲突，一定要将该形式模块内的CHOICE类型改写成EXPLICIT，例如在下面的模块中

M DEFINITIONS IMPLICIT TAGS ::=

BEGIN

    T ::= SEQUENCE

    {

        a    [0] INTEGER,

        b    [1] CHOICE

        {

            i      [0] INTEGER,

            n     [1] NULL,

            m    [2] BOOLEAN

        },

        c    [2] REAL

    }

END

如果CHOICE取i则会与a冲突，取m就会与c冲突。

2.全局EXPLICIT 模式：模块中的成员tag都是 EXPLICIT模式。

3.全局AUTOMATIC 模式： 模块内所有SEQUENCE、SET和CHOICE类型ASN.1编译器会自动从0开始，步长为1进行自动编码。而其中的成员使用都是IMPLICIT模式，同样，碰到CHOICE类型，要将CHOICE类型tag改为EXPLICIT模式。

|  |  |
| --- | --- |
| MDEFINITIONS AUTOMATIC TAGS ::=  BEGIN       T::= SEQUENCE      {          a INTEGER,          b CHOICE          {              i INTEGER,              n NULL          },          cREAL      }  END | BEGIN      T::= SEQUENCE      {          a[0] IMPLICIT INTEGER,          b[1] EXPLICIT CHOICE          {              i[0] IMPLICIT INTEGER,              n[1] IMPLICIT NULL          },          c[2] IMPLICIT REAL  }  END  注意b的tag模式是EXPLICIT，而他的成员的tag模式是IMPLICIT。 |

### 抽象数据类型

#### 1、简单类型

* INTEGER

ASN.1中没有限制整型的位数,可以任意大小。  
e.g. PageNumber::=INTEGER  
ColorType::=INTEGER{  
red (0),  
blue (1),  
green (2)}  
colorA ColorType::=1

编码：

八位位组{Xk,Xk-1,...., X0}将以递减的顺序从Xk到X0进行存储．**编码规定正整数的第一个字节的最高位必须是0,即Xk的最高为必须是0，为1的话则为负数**．例如：　x = 49468= 193 \* 256 + 60 = 0xC1 \* 0x FF + 0x3C; 即X1=0xC1, X0= 0x3C. 按正常规定，编码应该是 0x02 02 C1 3C, 但是X1的最高位是1, 应该被看成负数．最简单的方法是用前端零字节进行填充．编码变为 0x02 03 00 C1 3C

负整数的编码有些复杂．要先找到一个最小的256的幂，使它比要编码的负数的绝对值还要大．例如：x = - 1555; 被1555大的256的最小的幂是256^2 = 65536; 然后将这个数跟负数相加以得到2的补码． 65536 + (-1555) = 63981 =  0xF9 \* 0xFF + 0x ED. 则编码为 0x02 02 F9 ED.



* BOOLEAN  
  布尔值。取值为TRUE或FALSE。  
  e.g. Employed::=BOOLEAN  
  Married::=BOOLEAN  
  lincoln Married ::= TRUE , lincoln Married ::= FALSE

编码：

布尔编码的负载或者是全0或者是全1的八位位组。头字节以0x01开始，长度编码字节为0x01,负载内容取决于布尔值的取值

|  |  |
| --- | --- |
| 布尔值 | 编码 |
| False | 0x01 01 00 |
| True | 0x01 01 FF |

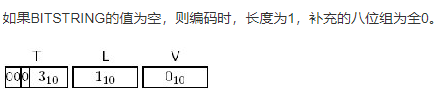
* ENUMERATED  
  e.g. Month::= ENUMERATED {  
  January (1),  
  February (2),  
  March (3),  
  •  
  •  
  •  
  December (12) }  
  hottestMonth Month::=July  
  --or 7

按照整数值的规则编码

* REAL  
  实数。ASN.1对实数的精度没有限制。每个实数可以用M×BE表示,即三元组。{M,B,E}  
  e.g. AngleInRadians::=REAL  
  angle1 AngleInRadians::={31415926,10,-7}
* BIT STRING  
  位串类型,由零个或多个比特组成的有序位串。可用二进制和十六进制表示。如: 10010001B,91H  
  e.g. Occupation::=BIT STRING{  
  clerk (0),  
  editor (1),  
  artist (2),  
  publisher (3) }  
  peter Occupation::={editor, artist}  
  --or "0110B"

编码规则：位串的第一位放到第一个负载字节的第8位；位串的第二位放到第一个负载字节的第7位; 依此类推．填充满第一个负载字节，就继续填充第二个负载字节．如果最后一个负载字节未被填充满，空的位用0来填充, 0的个数存放到头部用来表示填充数据的那个字节里

有一个位串{1,0,0,0,1,1,1,0,1,0,0,1}，开始填充负载字节．第一个字节填充后为10001110= 0x 8E; 第二个字节填充后为10010000 = 0x90, 低位4个0为填充的空位．则，负载为2个字节加上表示填充0个数的一个字节0x04总共3个字节．则完整的编码为：0x03 03 04 8E 90.



* OCTET STRING  
  八位位组串,由0个或多个8位位组组成的有序串。可用十进制(0-255)。二进制和十六进制表示。如:1101000100011010B或D11AH  
  e.g. MacAddress::=OCTET STRING(SIZE(6))  
  pc1 MacAddress::= 11A1D001E001H

例如：对{FE, ED, 6A, B4}编码；首先存储类型0x04, 接着是长度0x04,然后是字节本身0xFE ED 6A B4; 完整的编码为 0x04 04 FE ED 6A B4

* OBJECT IDENTIFIER  
  对象标识符。从对象树派生出的一系列点分数字串的形式,用来表示对象。

例如：MD5的OID 是1.2.840.113549.2.5  表示为"iso(1) member-body (2) US (840) rsadsi(113549) digestAlgorithm(2) md5 (5)", 所以当解码程序看到这个OID时,就知道是MD5散列.

编码规则：

前两部分如果定义为x.y, 那么它们将合成一个字40\*x + y, 其余部分单独作为一个字节进行编码。

每个字首先被分割为最少数量的没有头零数字的7位数字.这些数字以big-endian格式进行组织,并且一个接一个地组合成字节.

如果一个数大于127,就采用多个8位表示，最高位用1表示后续还有octet,用0表示后续没有。

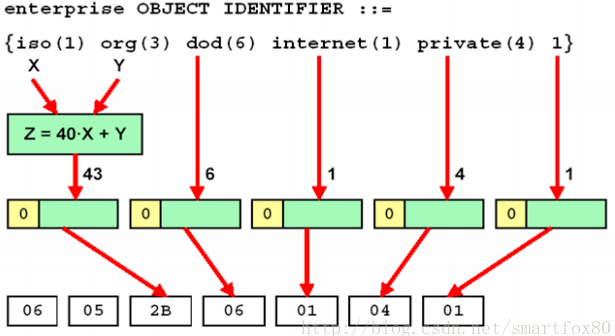
MD5 OID的编码:

       1. 将1.2.840.113549.2.5转换成字数组 {42, 840, 113549, 2, 5}.

       2. 然后将每个字分割为带有最高位的7位数字，{{0x2A},{0x86,0x48},{0x86,0xF7,0x0D},{0x02},{0x05}}.

113549 = 6\*128\*128 + 119 \* 128 + 13 ={0x06,0x77, 0x0D}，前面2个字节的高位设置为1

       3. 最后完整的编码为 0x06 08 2A 86 48 86 F7 0D 02 05.



* NULL  
   空(NULL)类型实际上是"占位符", 它是含有空白选项的选择修改器所特有。当某时刻无法知道数据的标准值,可将值为NULL。  
  e.g. PatientIdentifier::=SEQUENCE{  
  name Visiblestring --取自IA5的图形字符组成,不含控制字符  
  roomNumber CHOICE { INTEGER NULL }

}

patient1 PatientIdentifier::={name “peter”, roomNumber 301}  
patient2 PatientIdentifier::={name “john”, roomNumber NULL}

空类型的编码是 0x05 00.

* CHARACTER STRING  
  字符串类型。包括: NumericString PrintableString IA5String VisibleString GeneralString

可打印字符串对象是ASCII集合的一个有限子集，这个子集包括32,39,40~41,43~58,61,63以及65~122.

 IA5String类型的编码对象是ASCII集合中的大多数．包括NULL,BEL,TAB,NL,LF,CR以及32~126.

e.g. NumString::= NnmericString  
str1 NumString ::= “123456790”  
Surname ::= PrintableString  
person1 Surname ::= “John”

编码：

 可打印字符串和IA5String的编码和八位位组串相似．可打印字符串的头字节是0x13, IA5String的是0x16.

例如："Hello World"的编码为0x13 0B 48 65 6D6D 6F 20 57 6F 72 6D 64

* UTCTIME

世界协调时(UTCTIME)定义了一种相对GMT时间的标准时间(以日期)编码.它使用"YYMMDDHHMMSSZ"的格式分别表示年,月,日,时,分,秒. 其中"Z"是遗留自初始的UTCTIME.如果没有"Z",就允许两种附加组"[+/-]hh 'mm'",其中"hh"和"mm"分别为与GMT的时差和分差. 如果有"Z",则时间是以Zulu或GMT时间表示.

        字符串的编码按照IA5String编码规则进行转换(ASCII字符集),其头字节为0x17. 例如:

       July 4,2003 at 11:33 and 28 seconds编码为"030704113328Z",再编码0x17 0D 30 33 30 37 30 34 31 31 33 33 32 38 5A.

#### 2、构造类型

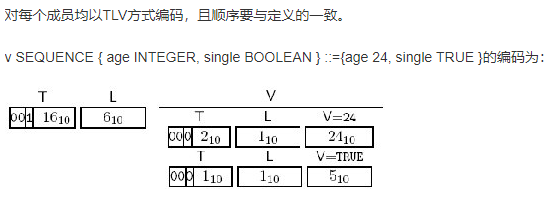
* SEQUENCE  
  序列类型。是包含零个或多个组成元素的有序列表,列表的不同元素可以分属性不同的数据类型。  
  e.g. AirlineFlight::=SEQUENCE{  
  airline IA5string, --取自IA5的字符组成,它与ASCII码基本相同  
  flight Numericstring, --包含数据0到9以及空格,不包含控制字符集  
  seats SEQUENCE{ maximum INTEGER, occupied INTEGER, vacantINTEGER},  
  airport SEQUENCE { origin IA5string, stop1 [0] IA5string OPTIONAL, stop2 [1] IA5string OPTIONAL, destination IA5string },  
  crewsize ENUMERATED { six (6), eight (8), ten (10) },  
  cancle BOOLEAN DEFAULT FALSE

}  
airplane1 AirlineFlight::={airline "china",  
flight "1106",  
seats {320,280,40},  
airport {origin "Beijing", destination "Shanghai"},  
crewsize 10 }  
或 ::= {"china", "1106", {320,107,213}, {"Beijing", "Shanghai"},10}

编码

 User ::== SEQUENCE{         ID        INTEGER,          Active BOOLEAN      }

     当取值为{32,TRUE}时，编码为 0x 3006 02 01 20 01 01 FF



* SEQUENCE OF  
  单纯序列(数组)类型。即序列中的各项都属于是同一类型的ASN.1类型。  
  e.g. Seats::=SEQUENCE OF INTEGER
* SET  
  集合类型。是包含零个或多个组成元素的无序集合,元素顺序无意义,类型可以不相同。  
  e.g. Person::=SET {name IA5string, age INTEGER, female BOOLEAN}  
  personA Person::={"maggie",4,ture} 、{true, "maggie",4} 、{4,true, " maggie "}

集合类型的元素是无序的,有多种编码。  
e.g. SET{breadth INTEGER, bent BOOLEAN}, 值{breadth 7, bent  
FALSE}  
编码:31 06  
02 01 07  
01 01 00  
也可以是:31 06  
01 01 00  
02 01 07

* SET OF  
  单纯集合类型,是包含零个或多个组成元素的无序集合。  
  e.g. Vipseats::=SET OF INTEGER  
  vipseatset Vipseats::={340,342,345}

#### 3、标签类型

Exuniv::=[UNIVERSAL 2] INTEGER  
valA Exuniv ::=9

Exappl::=[APPLICATION 0] INTEGER  
valB Exappl ::=10

Expriv::=[PRIVATE 1] INTEGER  
valC Expriv::=11

Excont::= SET{ type1 [0] INTEGER OPTIONAL, type2 [1] INTEGER OPTIONAL }

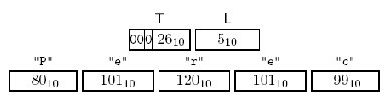
#### 4、其他类型

* CHOICE  
  选择类型。包含一个可供选择的数据类型列表。对于类型能够在事先都知道,可用此类型。  
  e.g. Prize::=CHOICE { car IA5string, cash INTEGER, nothing BOOLEAN }  
  peter Prize::=TRUE 或者  
  John Prize::= “ Lincoln ” 或者  
  Sam Prize::= 25000

严格说CHOICE类型在编码中并不存在，只是在描述中体现一种关系。编码时，是按照具体被选择的成员编码规则编码的。

例如amousCHOICE { name VisibleString, nobody NULL } ::= name:"Perec"

的编码为：



Tag为26,是选定成员的类型的tag。如果CHOICE类型显式（EXPLICIT）指定了Tag，那么该Tag应当以constructed方式编码。

* ANY  
  如果在定义数据时还不能确定数据类型,可以使用ANY型。ANY型可以被任何ASN.1类型置换。  
  e.g. TextBook::=SEQUENCE { author IA5string, reference ANY}  
  实例:book1 TextBook ::= { author "shakespeare " , reference IA5string " ISBN0669123757 "}  
  book2 TextBook ::= { author "shakespeare ", reference INTEGER 1988}

### 子类型

通过对某些类型加以限制,可以定义它们的子类型(subtype)。子类型的值集合是其父类型值集合的子集。

子类型约束是在BER编码规则之后被引入ASN.1的，在编码规则中不能体现约束。

(1) 单个值  
在定义中列举出所有可能的取值,e.g.  
TestResult ::=INTEGER(1|2|3|4)  
sp1 TestResult ::= 2  
(2) 大小限制  
通过限制父类型中元素的个数定义新类型或规模。  
e.g. WorkstationNumber::=OCTET STRING(SIZE(6))  
BitField ::= BIT STRING(SIZE(12))  
map1 BitField ::= „100110100100‟B  
map2 BitField ::= „9A4‟H  
(3) 取值范围(值区间)  
只适用于整数和实数类型,  
e.g. NoID::=INTEGER(1..100)  
PositiveInteger ::=INTEGER(0<..MAX)  
PositiveInteger ::=INTEGER(1..MAX)  
(4) 可用字符  
只用于字符串类型,限制字符集的取值范围。  
DigitString ::=IA5String(FROM(″0″)|(″1″)|(″2″)|(″3″)|(″4″)|(″5″)|(″6″)|(″7″)|(″8″)|(″9″))  
str2 DigitString ::= “46732”  
(5) Inner Subtyping (内部类型)  
适用于SEQUENCE,SEQUENCE OF,SET,SET OF和CHOICE类,主要用于对这些结构类型的元素项进行限制。  
e.g. PDU::=set{ alpha [0] INTEGER, beta [1] IA5striong OPTIONAL,  
gamma [2] SEQUENCE OF parameter, delta [3] BOOLEAN }  
TestPDU::=PDU(WITH COMPONENTS{alpha(min..<0) ,..., delta(FALSE)})  
(6) 包含子类型  
要用到关键字INCLUDES,说明被定义的新子类型包含原子类型的全部可能值。  
e.g. Months::=ENUMERATED{jaunary(1),february(2),...,december(12)}  
First-quarter::=Months(january, February ,march);  
Second-quarter::=Months(april, may, june)  
First-half::=Months (INCLUDES First-quarter | INCLUDE Second-quarter)  
jan First-quarter ::= 1

### 应用类型

ASN.1中的应用类型与特定的应用有关,根据网络管理的实际特点, RFC1155定义了6种SNMP应用类型。  
(1) NetworkAddress::=CHOICE{internet IpAddress} --可从各种网络地址中选择一种,目前只有Internet地址,即IP地址。  
(2) IpAddress::=[APPLICATION 0] IMPLICIT OCTET STRING(SIZE(4))  
(3) Counter::= [APPLICATION 1] IMPLICIT INTEGER(0..4294967295)  
(4) Gauge::=[APPLICATION 2] IMPLICIT INTEGER(0..4294967295)  
(5) TimeTicks::=[APPLICATION 3] IMPLICIT INTEGER(0..4294967295)  
(6) Opaque::= [APPLICATION 4] OCTET STRING --arbitrary ASN.1 value 不透明类型

## 基本编码规则

用ASN.1语言书写的变量必须转换为串行的字节流才能在网络中传输。为此, ASN.1又提供了基本编码规则(BER)来描述传输过程中内容的表示

### BER

Basic Encoding Rules, as defined in X.209.

BER（BasicEncoding Rules）是ASN.1中最早定义的编码规则，其他编码规则是在BER的基础上添加新的规则构成。

BER传输语法的格式一直是TLV三元组<Type,Length, Value>.

T是Tag，L是整个类型的长度，V是类型的Value，它还可以是TLV或TLV组合

BER传输语法是基于八位组大端编码的，高八位在左

#### Tag是一个或若干个八位组

* UniversalTag类型（值是0-30）



第七、六位指明Tag的类型，UniversalTag类型用00表示;

第五位指明该类型以primitive方式编码还是constructed方式编码。

Tagvalue值是基本类型的Tag的值

| **Tag** | **类型** |
| --- | --- |
| 0 | BER保留 |
| 1 | BOOLEAN |
| 2 | INTEGER |
| 3 | BIT STRING |
| 4 | OCTET STRING |
| 5 | NULL |
| 6 | OBJECT IDENTIFIER |
| 7 | ObjectDescripion |
| 8 | EXTERNAL,INSTANCE OF |
| 9 | REAL |
| 10 | ENUMERATED |
| 11 | EMBEDDED PDV |
| 12 | UFT8String |
| 13 | RELATIVE-OID |
| 14 | 保留 |
| 15 | 保留 |
| 16 | SEQUENCE,SEQUENCE OF |
| 17 | SET,SET OF |
| 18 | NumericString |
| 19 | PrintableString |
| 20 | TeletexString,T61String |
| 21 | VideotexString |
| 22 | IA5String |
| 23 | UTCTime |
| 24 | GeneralizedTime |
| 25 | GraphicString |
| 26 | VisibleString,ISO646String |
| 27 | GeneralString |
| 28 | UniversalString |
| 29 | CHARACTER STRING |
| 30 | BMPString |
| 31 | 保留 |

* 当Tag大于30时，多个八位组中编码，第一个八位组后五位全部为1，其余的八位组最高位为1表示后续还有，为0表示Tag结束



第一个八位组高二位的取值：00表示Universal，01表示APPLICATION类型，10表示context-specific,11表示PRIVATE类型

#### BER编码中Length表示Value部分所占八位组的个数，有两大类：定长方式（DefiniteForm）和不定长方式（IndefiniteForm）

(1).定长方式

定长方式中，按长度是否超过一个八位，又分为短、长两种形式：

短：类型长度大于等于0个八位，小于等于127



长：类型长度大于等于127个八位，小于等于256^126-1



length=169=>81 A9（169超过127，长度域为 1000 0001 1010 1001，169是后8位的值，前8位的第一个1表示这是长格式的表示方法，前8位的后7位表示后面有多少个字节表示针对的长度，这里，是000 0001，后面有一个字节表示真正的长度，1010 1001是169，

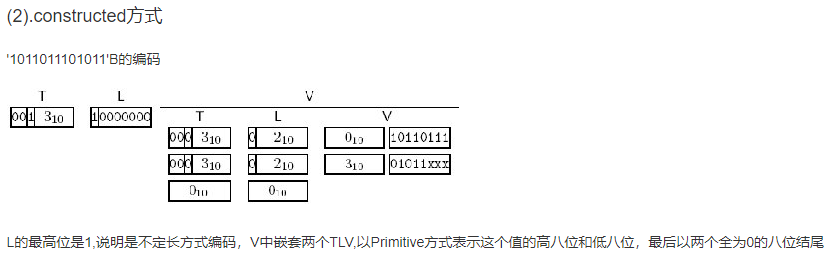
 length=1500=>82 05 DC（1000 0010 0000 0101 1101 1100，先看第一个字节，表示长格式，后面有2个字节表示长度，这两个字节是0000 0101 1101 1100 表示1500）

(2).不定长方式

Length所在八位组固定编码为0x80，但在Value编码结束后以两个0x00结尾。这种方式使得可以在编码没有完全结束的情况下，可以先发送部分消息给对方。



#### 示例



#### TAG对编码的影响

如果Tag是隐式的，或者在模块定义中声明了IMPLICITTAGS或者AUTOMATICTAGS,则只有关键字IMPLICIT左侧的Tag才会被编码

v [1]IMPLICIT INTEGER ::= -38的编码为：



如果一个类型的*Tag*是显式（*EXPLICIT*）的（或者在模块定义中声明了*EXPLICITTAGS*），则要以*constructed*方式编码三元组系列

v[APPLICATION 0] EXPLICIT INTEGER ::= 38的编码为



#### INSTANCEOF

vINSTANCE OF TYPE-IDENTIFIER ::=

{

type-id{iso member-body f(250) type-org(1) ft(16) asn1-book(9)

chapter18(5)integer- type(0)},

valueINTEGER:5

}

的编码应当和如下一个SEQUENCE类型的值类似：

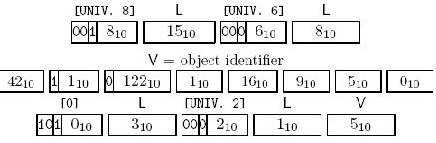
{

direct-reference{iso member-body f(250) type-org(1) ft(16) asn1-book(9)

chapter18(5)integer-type(0)},

encodingsingle-ASN1-type:INTEGER:5

}



### DER

Distinguished Encoding Rules for ASN.1, as defined in X.509, Section 8.7

DER是BER的子集，它为每一个ASN.1类型定义一种唯一的编码方案。

#### DER与BER的区别

DER在BER的基础上增加了如下限制：

长度小于等于127，必须使用短型长度表示法。

长度大于127，必须使用长型长度表示法，并且要尽可能的短。

对于简单的string类型以及在其基础上隐性标签生成的类型使用简单定长表示法。

对于结构化类型以及在其基础上隐性标签生成的类型以及所有的显示类型使用结构化定长表示法。

简单定长表示法和结构化类型表示法的区别是在TLV的T中bit6是0还是1。

#### DER的组成

无论什么类型它的DER编码都是有四部分组成：

类型字段tag  长度字段length   值字段value   结束表示字段

即T L V

* 类型字段：

由一个字节（即八个二进制位）组成。

bit8 bit7用来表示tag类型，universal是00  application是01  context-specific是10   private是11

bit6表示结构化位，0代表简单类型，1代表结构类型

bit5-bit1表示tag值，tag值可以查表获得。

* 长度字段（以字节为计算单位）：

长度字段分为两种:小于等于127和大于127

小于等于127：用一个字节表示。bit8=0,bit7-bit1表示长度值。

例子：38表示为00100110

大于127：由多个字节组成

第一个字节：bit8=1，bit7-bit1表示存放长度所占字节数（通俗的讲就是后面还有几个字节）

其余字节：存放长度值

例如：201表示为10000001 11001001

长度值不定：用0x80表示。

值字段：

不同的类型其值不同，但归根结底类型都是二进制，或十进制，或十六进制 ，或者是ASCLL字符，将他们转换为16进制表示出来即可。

结束标识字段（不在TLV之中）：

两个字节，只有在长度值不定时才会出现。表示为0x000x00

### PEM

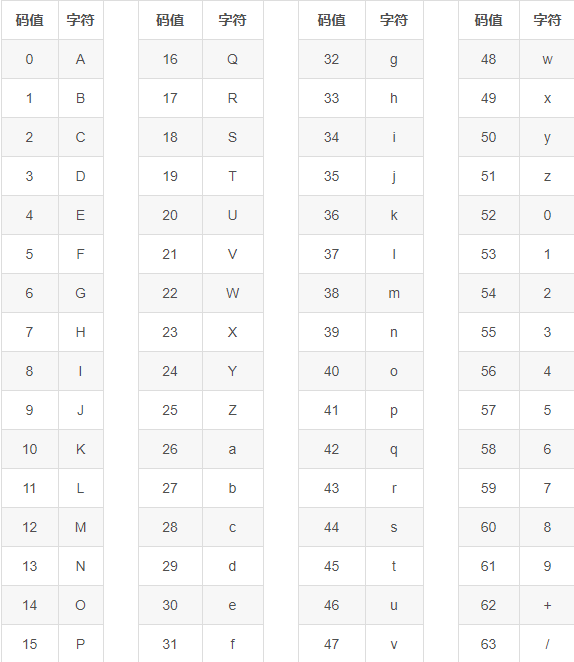
是64进制，这是一种信息编码格式，类似于HEX格式。如果将DER及PEM文件进行对比，DER文件类似于二进制的BIN文件，PEM文件类似于十六进制的HEX格式文件。

举例说明：

你可以用 0x01 来表示数字1， 这就是二进制编码。

你也可以用ASCII码的 '1' 表示数字1，这就是HEX编码。

PEM编码（BASE64）就是指定了一系列符号，使用这些符号来表示特定的数字，参考下表



# BouncyCastle

