ICS 35.240.40
A 11

JR

中华人民共和国金融行业标准

JR/T 0066.3—2019
代替 JR/T 0066—2011

银行间市场业务数据交换协议
第3部分：适流表示层

Interbank market information exchange protocol—
Part 3: Presentation streaming layer

2019-01-08 发布　　2019-01-08 实施

中国人民银行　发布

目　次

前　言

JR/T 0066《银行间市场业务数据交换协议》分成3部分：

——第 1 部分：语法、结构与会话层；

——第 2 部分：应用层；

——第 3 部分：适流表示层。

本部分为JR/T 0066的第3部分。

本部分依据GB/T 1.1—2009给出的规则起草。

本部分由中国外汇交易中心暨全国银行间同业拆借中心提出。

本部分由全国金融标准化技术委员会（SAC/TC 180）归口。

本部分负责起草单位：中国外汇交易中心暨全国银行间同业拆借中心。

本部分参与起草单位：中国人民银行科技司。

本部分主要起草人：许再越、姚前、杨富玉、朱荣、叶胜国、姜才康、王成勇、胡剑、李正、陈彬、胡卫平、沈峻、崔嵬、郦永达、余波、曲维民、孙小林、沈薇薇、茅廷、杨帆、夏志江、孙英昊、包晓晶、赵俊锋、卢艳民、崔奇、邓钢轶、严璐祎、沈叶。

JR/T 0066于2011年6月2日首次发布，本次为第一次修订。

银行间市场业务数据交换协议
第3部分：适流表示层

1 范围

JR/T 0066的本部分规定了银行间市场参与方基于会话层和应用层的银行间市场成员交互数据进行适流压缩的协议（Interbank Market Information Exchange Adapted for Streaming，简称IMAST）。

本部分适用于外汇市场、货币市场和衍生品市场的前、中、后台以及债券市场的前、中台，不含债券市场的后台清算、结算部分。

2 术语和定义

下列术语和定义适用于本文件。

2.1

IMIX适流 IMIX adapted for streaming（IMAST）

一种压缩IMIX报文数据的二进制流。

2.2

编码 encode

将IMIX报文转换成IMAST流的过程。

2.3

解码 decode

将IMAST流转换成报文的过程。

2.4

应用类型 application type

IMIX会话层或应用层的域、组件或重复组在适流表示层的抽象概念。

2.5

域指令 field instruction

IMIX会话层或应用层的域、组件或重复组在适流表示层的编码规则。

2.6

模板 template

某个场景下若干域指令的集合所实现的编解码控制结构。

2.7

上下文 context

IMAST编码或解码的运行环境。

3 语法和结构

3.1 IMAST 协议框架

适流表示层应用模式见图1。

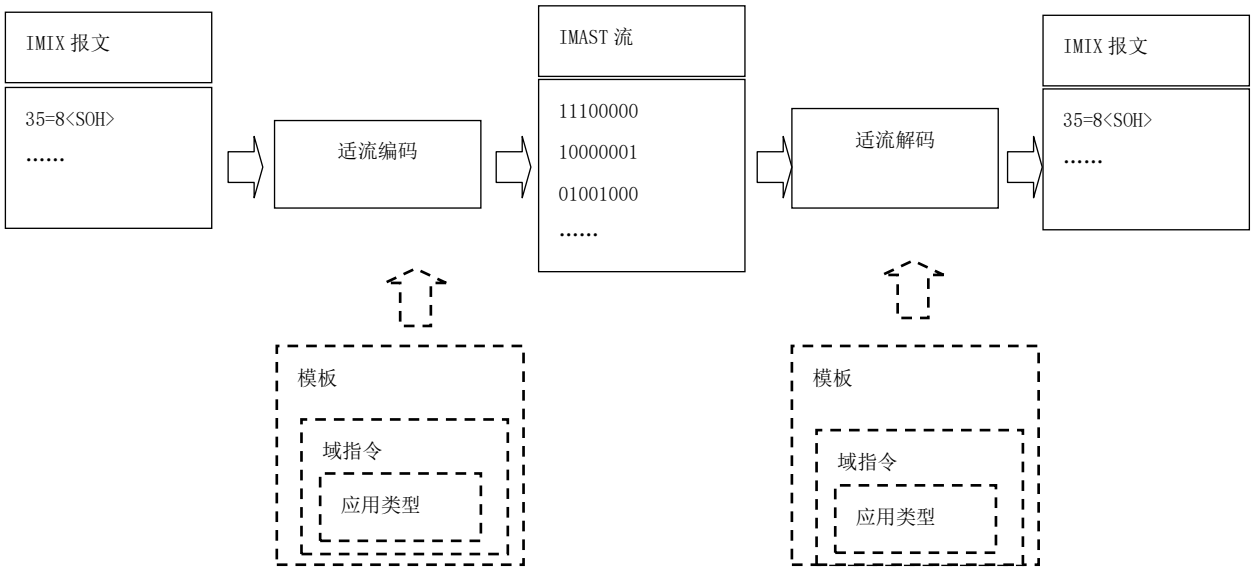


图1 适流表示层应用模式图

IMAST协议框架依赖报文所指向的模板进行编码和解码，形成IMAST流进行网络传输。

模板是针对某一IMIX报文的编解码控制结构，通过规定报文中域、组件、重复组的顺序和结构、运算规则、二进制编码表示方法来控制与应用报文相对应的二进制流的编码和解码。模板是包含若干域指令的序列。

域指令规定报文中域、组件、重复组的编码方式。域指令由名称、应用类型、存在属性和操作符组成。

应用类型是报文中域、组件或重复组在适流表示层的表现形式。

3.2 IMAST 流总体结构

IMAST流的总体结构采用扩展巴科斯范式（EBNF）语法进行定义。

stream::=message*|block*

block::=*BlockSize* message+

message::=segment*

segment::=*PresenceMap TemplateIdentifier*? (field|segment)*

field::=integer|string|Decimal|ByteVector

IMAST总体结构图见图2。

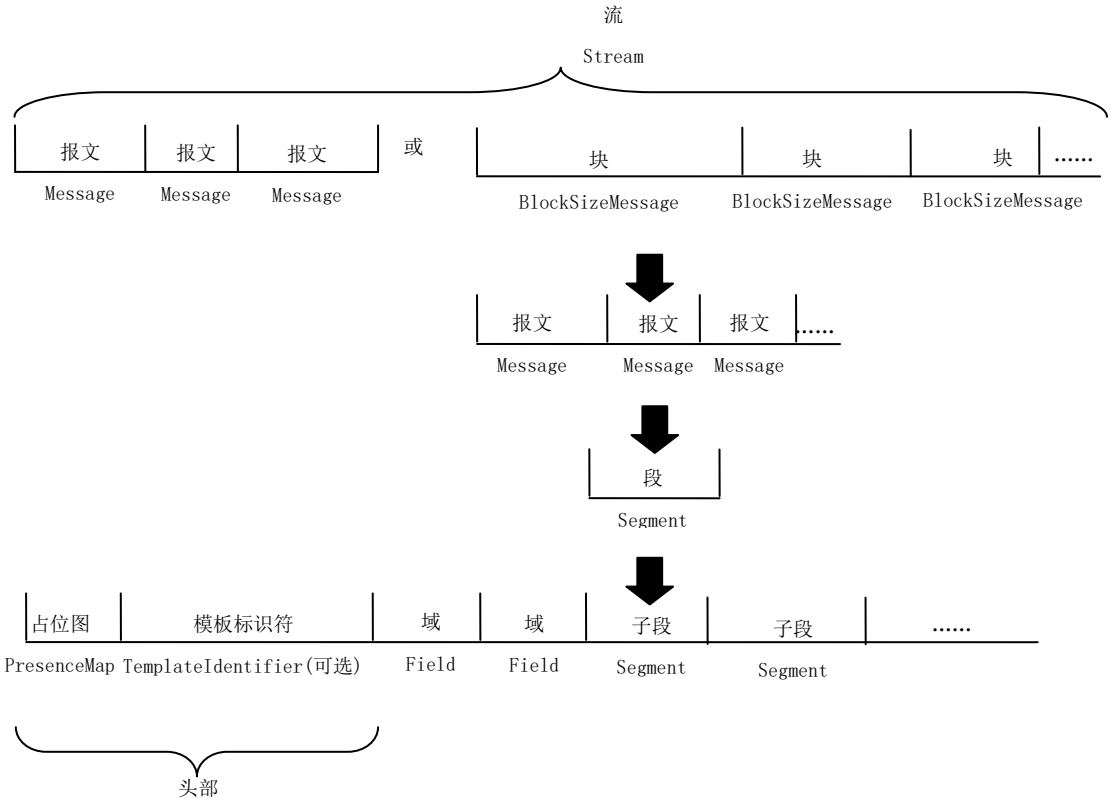


图2　IMAST 流总体结构图

IMAST流总体结构具体说明如下：

a)　流（stream）：一个 IMAST 流（stream）为一个报文的序列（sequence），或为一个块的序列；

b)　块（block）：块（block）是包含单个或多个报文的一个序列，块带有一个前导的、用来表示块包含报文所占字节数量的块大小（blocksize）；

c)　报文（message）：每个报文（message）由一个段（即报文段）来表示；

d)　段（segment）：段（segment）由头部和具体域组成。

IMAST流示例如下：

市场数据请求IMIX报文示例见图3。

8=IMIX.2.0<SOH>9=41<SOH>34=2<SOH>35=V<SOH>49=CFETS-RMB-CSTP<SOH>56=IH<SOH>52=20151208-12:00:00<SOH>263=2<SOH>146=1<SOH>48=CNY<SOH>10=167

图3　市场数据请求 IMIX 报文示例

通过编码后，形成市场数据请求IMAST流，其中PAMP表示占位图，TID表示模板标识符，见图4。

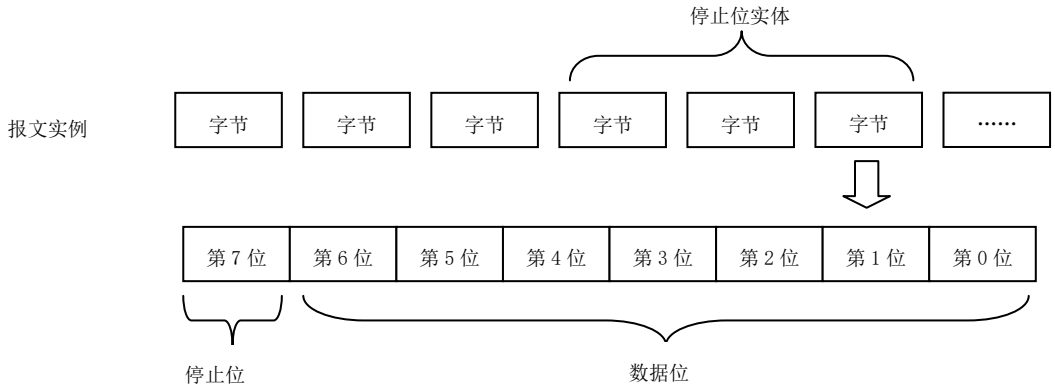
```
01111111 11111000 10000010 10101001 10000010 11010110 01000011 01000110 01000101
--------PMAP-----|--TID---|--9 号域--|---34--|----35--|-------------------------
01010100 01010011 00101101 01010010 01001101 01000010 00101101 01000011 01010011
-----------------------------39-----------------------------------------------
01010100 11010000 01001001 11001000 10000000 00110010 00110000 00110001 00110101
-----------------|-------56--------|--长度--|---------------------------------
00110001 00110010 00110000 00111000 00101101 00110001 00110010 00111010 00110000
-------------------------52-----------------------------------------------------
00110000 00111010 00110000 10110000 10000010 10000001 01000011 01001110 11011001
-----------------------------------|---263--|---146--|---------48--------------|
00000001 10000111
-------10--------
```

图4　市场数据请求 IMAST 流示例

3.3　字节停止位

IMAST流采用停止位编码规则进行编码，即报文中每一个字节都有一个停止位。一个停止位编码实体是一个字节序列，其中每个字节的最高有效位（the most significant bit）指示下一字节是否是实体的一部分。

停止位示例说明见图5。



注：当字节为停止位实体的最后一个字节，停止位设置为1。

图5　停止位示例图

4　编解码语法

4.1　概述

编解码的过程依赖于报文所指向的模板。编码时，编码器通过模板中域的属性对原始消息中的域进行编码操作，并根据规则生成占位图，从而把占位图和编码后的值输出到报文二进制流中。IMAST接收端解码器根据接收的报文二进制流里的模板标识符找到模板后，根据模板中对域的编码规则（主要是由操作符和占位图决定编码）进行解码，还原传输前的报文。

4.2 应用类型

应用类型能被映射为下列模型：

a) 基本类型：包含整数类型、十进制小数类型、字符串类型、字节向量类型；

b) 组件类型：一个包含无序的域指令的集合；

c) 重复组类型：一个包含长度及有序的若干组件类型的域指令的集合。

应用类型模型见图6。

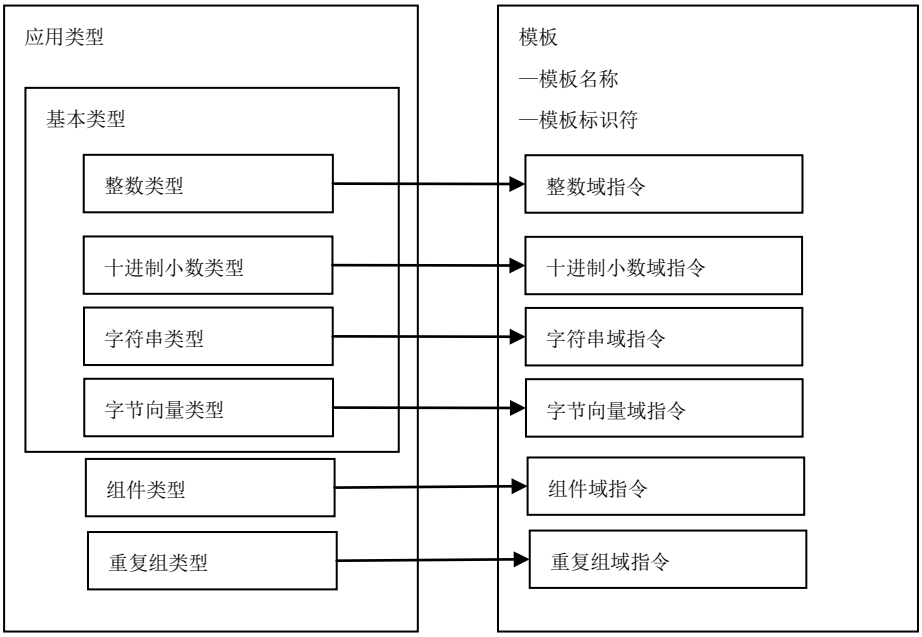


图6 应用类型模型图

4.3 模板

4.3.1 概述

模板的定义应使用XML表达，由“<td:template>”元素定义。一个模板定义的XML文档可包含单个模板或多个模板的集合。模板集合应封装在“<td:templates>”元素中，该元素可包含应用于整个封装的模板集合的命名空间参数。

4.3.2 语法规则

模板语法规则定义如下：

```
start=templates|template
templates=element templates{nsAttr?,templateNsAttr?,dictionaryAttr?,template*}
template=element template{templateNsName,nsAttr?,dictionaryAttr?,instruction*}
instruction=field
```

注：

templates：多个模板；

template：单个模板；

element templates：多个模板元素；

element template：单个模板元素；

nsAttr?：命名空间属性，可选元素；

templateNsAttr?：模板命名空间属性，可选元素；

dictionaryAttr?：字典属性，可选元素；

template*：N个模板；

templateNsName：模板命名空间名；

instruction*：N个域指令。

示例：

下面的XML片段是采用具体语法格式的一个模板定义的例子。

```
<templates xmlns="http:// imix.chinamoney.com.cn "
    templateNs="http:// imix.chinamoney.com.cn /ns/templates/sample" ns="http:// imix.chinamoney.com.cn /ns/IMIX">
    <template name="MarketDataIncrementalRefresh">
        <string name="BeginString" id="8">
            <constant value="IMIX.2.0" />
        </string>
        <string name="MessageType" id="35">
            <constant value="X" />
        </string>
        <string name="SenderCompID" id="49">
            <copy />
        </string>
        <uInt32 name="MsgSeqNum" id="34">
            <increment />
        </uInt32>
        <sequence name="MDEntries">
            <length name="NoMDEntries" id="268" />
            <uInt32 name="MDUpdateAction" id="279">
                <copy />
            </uInt32>
            <string name="MDEntryType" id="269">
                <copy />
            </string>
            <string name="Symbol" id="55">
                <copy />
            </string>
            <string name="SecurityType" id="167">
                <copy />
            </string>
            <decimal name="MDEntryPx" id="270">
                <delta />
            </decimal>
            <decimal name="MDEntrySize" id="271">
```

```
                <delta />
            </decimal>
            <uInt32 name="NumberOfOrders" id="346">
                <delta />
            </uInt32>
            <string name="QuoteCondition" id="276">
                <copy />
            </string>
            <string name="TradeCondition" id="277">
                <copy />
            </string>
        </sequence>
    </template>
</templates>
```

4.3.3 标识符

模板标识符在数据流中报文段头部，表示该报文所使用的模板，以一个无符号整数来表示。如果过长，则产生可报告错误[ERR R6]。

如果一个解码器无法找到与数据流中出现的模板标识符所关联的模板，则产生动态错误[ERR D9]。

模板标识符的编码与拷贝操作符的用法相同。拷贝操作符使用全局字典，且具有所有模板标识符域公用的一个内部关键字（Key）。拷贝操作符定义见4.6.7。

4.3.4 名称

4.3.4.1 定义规则

模板名称的定义由两部分组成：命名空间（namespace）URI和本地名称。

使用应用类型、域和操作符关键字的命名空间URI由“ns”参数指定。

模板的命名空间URI由“templateNs”参数指定。

本地名称由参数“名称（name）”指定。

两个名称相同，当且仅当它们的命名空间标识符和本地名称都相同。

4.3.4.2 语法规范

模板名称语法规范定义如下：

```
nsName=nameAttr,nsAttr?,idAttr?
templateNsName=nameAttr,templateNsAttr?,idAttr?
nameAttr=attribute name{token}
nsAttr=attribute ns{text}
templateNsAttr=attribute templateNs{text}
idAttr=attribute id{token}
```

注：

nsName：命名空间名；

nameAttr：名称属性；

nsAttr?：命名空间属性，可选元素；

idAttr?：标识符属性，可选元素；

templateNsName：模板命名空间名；

templateNsAttr?：模板命名空间属性，可选元素。

4.3.5 指令

模板指令是包含若干指令的序列。指令应与数据在流中的位置相对应。

模板指令主要采用域指令，规定域的实例编码和二进制流的方法。

4.4 上下文

上下文规定编码和解码的运行环境，应包括：

a) 模板的集合；

b) 当前模板；

c) 域指令的集合；

d) 当前的域指令；

e) 字典集合；

f) 可选的初始值。

当前模板指正被处理的模板，在流中处理到一个模板标识符时当前模板被更新。对静态模板的引用也可引起当前模板的改变。

4.5 域指令

4.5.1 定义

域指令由名称、应用类型、存在属性和操作符组成。

域指令通过名称区分，名称在单个组件内具有唯一性。

应用类型规定了域的基本编码方法，具体包括：整数（integerField）、十进制小数（decimalField）、ASCII码字符串（asciiStringField）、Unicode字符串（unicodeStringField）、字节向量（byteVectorField）、重复组（sequence）、组件（group）。

存在属性用于指示域指令是必选域（mandatory）或者是可选域（optional）。如果存在属性未被指定，则该域指令为必选。

域操作符是可选信息，用于规定域指令编码的优化传输方式。除组件域和重复组域之外的域可定义其域操作符。

4.5.2 空值属性

每个域指令的应用类型具有一个空值（nullability）的属性。如果类型为可空（nullable），则NULL值采用一种特殊表示方式。如果类型不可空（non-nullable），则不保留任何NULL的表示方式。所有的可空类型按照以下方式构建，即NULL由所有位均为0的7位实体值来表示。在使用停止位编码时，NULL由0x80来表示。

除非明确指定，否则缺省使用不可空的表示方式。

4.5.3 语法规则

域指令语法规则如下：

field=integerField|decimalField|asciiStringField|unicodeStringField|byteVectorField|sequence|group

fieldInstrContent=nsName,presenceAttr?,fieldOp?

presenceAttr=attribute presence{"mandatory"|"optional"}

注：

integerField：整数域指令；

decimalField：十进制小数域指令；

asciiStringField：ascii码类型字符串域指令；

unicodeStringField：unicode字符串域指令；

byteVectorField；字节向量域指令；

sequence：重复组域指令；

group；组件域指令；

nsName：域指令命名空间名；

presenceAttr：存在属性，可选元素；

fieldOp：域操作符，可选元素。

4.5.4 分类

4.5.4.1 整数域指令

4.5.4.1.1 概述

整数域指令的类型包括32位无符号整数（uInt32），32位整数（int32），64位无符号整数（uInt64），64位整数（int64）。

整数（Integer）在传送编码中的大小不受限。

所有的整数域指令使用大尾端（big endian）方式来表示，其中位元和字节均采用网络字节顺序，即高位在低位前，高字节在低字节前。

4.5.4.1.2 有符号整数域指令

实体的值为一个补码[TWOC]。实体值的最高有效位为符号位。

4.5.4.1.3 无符号整数域指令

实体值即为整数的二进制表示形式。

4.5.4.1.4 语法规则

整数域指令语法规则如下：

```
integerField = element int32{fieldInstrContent}
               |element uInt32{fieldInstrContent}
               |element int64{fieldInstrContent}
               |element uInt64{fieldInstrContent}
```

注：

integerField：整数域指令；

fieldInstrContent：域指令内容；

前缀“int”表示域是带符号的，“uInt”则表示域是无符号的。

4.5.4.1.5 错误发生

整数由停止位编码实体表示。如果一个整数在移除7位或7位以上的最高有效位后，实体值还是表示相同的整数，则该整数过长（overlong）。如果在流中出现一个过长的整型数，除非该整数被用作块大小，否则产生可报告错误[ERR R6]。

如果数据流中的整数大于特定类型的最大值，或小于指定类型的最小值，则产生动态错误[ERR D2]。

各整数类型最大值和最小值见表1。

表1　各整数类型最大值和最小值

类型	最小值	最大值
int32	-2147483648	2147483647
uInt32	0	4294967295
int64	-9223372036854775808	9223372036854775807
uInt64	0	18446744073709551615

4.5.4.1.6　编码示例

可选的正数定义如下：

<int32 id="1" presence="optional" name="Value"/>

可选的正数编码示例见表2。

表2　可选的正数编码示例

输入值	本地十六进制/二进制	IMAST十六进制/二进制
942755	0x0e 0x62 0xa3 00001110 01100010 10100011	0x39 0x45 0xa4 **0**0111001 **0**1000101 **1**0100100
注：停止位由黑体表示。符号位由下划线表示。域可选且值非负，所以自加1。		

必选的正数定义如下：

<int32 id="1" presence="mandatory" name="Value"/>

必选的正数编码示例见表3。

表3　必选的正数编码示例

输入值	本地十六进制/二进制	IMAST十六进制/二进制
942755	0x0e 0x62 0xa3 00001110 01100010 10100011	0x39 0x45 0xa3 **0**0111001 **0**1000101 **1**0100011
注：停止位由黑体表示。符号位由下划线表示。域为必选，因此不自加1。		

可选的负数定义如下：

<int32 id="1" presence="optional" name="Value"/>

可选的负数编码示例见表4。

表4　可选的负数编码示例

输入值	本地十六进制/二进制	IMAST十六进制/二进制
-942755	0xf1 0x9d 0x5d 11110001 10011101 01011101	0x46 0x3a 0xdd **0**1000110 **0**0111010 **1**1011101

输入值	本地十六进制/二进制	IMAST 十六进制/二进制
注：停止位由黑体表示。符号位由下划线表示。域为可选的负数，因此不自加1。		

必选的负数定义如下：

<int32 id="1" presence="mandatory" name="Value" />

必选的负数编码示例见表 5。

表5 必选的负数编码示例

输入值	本地十六进制/二进制	IMAST 十六进制/二进制
-7942755	0xff 0x86 0xcd 0x9d 11111111 10000110 11001101 10011101	0x7c 0x1b 0x1b 0x9d **0**1111100 **0**0011011 **0**0011011 **1**0011101
注1：本地十六进制/二进制中，最左端位的高值被抛掉。 **注2：**IMAST十六进制/二进制中，停止位由黑体表示。符号位由下划线表示。域为必选，因此不自加1。		

带符号位扩展的必选的正数定义如下:

<int32 id="1" presence="mandatory" name="Value" />

带符号位扩展的必选的正数编码示例见表6。

表6 带符号位扩展的必选的正数编码示例

输入值	本地十六进制/二进制	IMAST 十六进制/二进制
8193	0x20 0x01 00100000 00000001	0x00 0x40 0x81 *0*0000000 **0**1000000 **1**0000001
注：停止位由黑体表示。符号位由下划线表示。表示符号所必需的符号位扩展（斜体）。域必选，因此不自加1。		

带符号位扩展的必选的负数定义如下：

<int32 id="1" presence="mandatory" name="Value" />

带符号位扩展的必选的负数编码示例见表7。

表7 带符号位扩展的必选的负数编码示例

输入值	本地十六进制/二进制	IMAST 十六进制/二进制
-8193	0xff 0xdf 0xff 11111111 11011111 11111111	0x7f 0x3f 0xff *0*1111111 **0**0111111 **1**1111111
注：停止位由黑体表示。符号位由下划线表示。表示符号所必需的符号位扩展（斜体）。域为必选，因此不自加1。		

无符号可选的整数定义如下：

<uInt32 id="1" presence="optional" name="Value" />

无符号可选的整数编码示例见表8。

表8　无符号可选的整数编码示例

输入值	本地十六进制/二进制	IMAST 十六进制/二进制
null	N/A	0x80 **1**0000000
0	0x00 0	0x81 **1**0000001
1	0x01 1	0x82 **1**0000010
942755	0x0e 0x62 0xa3 1110 01100010 10100011	0x39 0x45 0xa4 **0**0111001 **0**1000101 **1**0100100
注：停止位由黑体表示。域可选，因此自加1。		

无符号必选的整数定义如下：

<uInt32 id="1" presence="mandatory" name="Value" />

无符号必选的整数编码示例见表9。

表9　无符号必选的整数编码示例

输入值	本地十六进制/二进制	IMAST 十六进制/二进制
0	0x00 0	0x80 **1**0000000
1	0x01 1	0x81 **1**0000001
942755	0x0e 0x62 0xa3 1110 01100010 10100011	0x39 0x45 0xa3 **0**0111001 **0**1000101 **1**0100011
注：停止位由黑体表示。域必选，因此不自加1。		

4.5.4.2　十进制小数域指令

4.5.4.2.1　概述

十进制小数域指令包含两部分：指数（exponent）部分和尾数（mantissa）部分。该域指令两部分整体使用单个域操作符，或者两部分使用各自的操作符。整个十进制小数未指定操作符，或只指定了单个操作符，则运算对象被当成一个十进制小数表示的数值，在编码传输过程中表示为一个带比例数（Scaled Number）。

带比例数（Scaled Number）由一个尾数和一个指数表示。

为了计算效率，浮点数使用以2为底的指数，但为了支持十进制数的确切表示形式，带比例数采用了以10为底数的指数。

$$V=S\times B^{E}$$

式中：

V——数值；

S——尾数；

B——底数；

E——指数。

其中，数值由尾数和10的指数次方相乘来得到。

带比例数由一个带符号的整数指数及其之后的一个带符号的整数尾数来表示。

如果一个带比例数可空，则其指数可空，但尾数不可空。NULL值的带比例数由一个NULL的指数来表示。当且仅当指数不为NULL时，尾数在流中出现。

当对指数和尾数部分分别应用操作符时，指数和尾数部分所生成的名称是该十进制小数域指令的特有名称。

如果十进制小数域是可选存在的，并具有单独的操作符，则尾数存在与否取决于指数是否存在，具体规定见4.7。

4.5.4.2.2 语法规则

十进制小数域指令语法规则如下：

decimalField=element decimal{nsName,presenceAttr?,（fieldOp|decFieldOp）}

decFieldOp=element exponent{fieldOp}?,element mantissa{fieldOp}?

注：

presenceAttr：存在属性，可选元素；

exponent：指数域操作符；

mantissa：尾数域操作符。

4.5.4.2.3 错误发生

虽然指数被看作int32类型，但其值域范围应是[-63,63]。在应用任何操作符后，如果获得的指数超出了该范围，则产生可报告错误[ERR R1]。

当使用单独的操作符时，可对十进制小数科学计数法表示的数的范围和精度进行限制。由于使用某个操作符引入了限制，造成值不应被编码到该域，则产生动态错误[ERR D3]。

4.5.4.2.4 编码示例

必选的正十进制小数定义如下：

<decimal id="1" presence="mandatory" name="Value" />

必选的正十进制小数编码示例见表10。

表10 必选的正十进制小数编码示例

输入值	本地十六进制/二进制		IMAST 十六进制/二进制	
ASCII 码	指数	尾数	指数	尾数
94275500	2	942755	0x82 **1**0000010	0x39 0x45 0xa3 **0**0111001 **0**1000101 **1**0100011
ASCII 码	指数	尾数	指数	尾数
注：停止位由黑体表示。符号位由下划线表示。				

尾数按比例缩放的必选的正十进制小数定义如下：

<decimal id="1" presence="mandatory" name="Value" />

尾数按比例缩放的必选的正十进制小数编码示例见表11。

表11　尾数按比例缩放的必选的正十进制小数编码示例

输入值	本地十六进制/二进制		IMAST 十六进制/二进制	
ASCII 码	指数	尾数	指数	尾数
94275500	1	9427550	0x81 10000001	0x04 0x3f 0x34 0xde 00000100 00111111 00110100 11011110
注：停止位由黑体表示。符号位由下划线表示。指数设为1，以对尾数缩放。				

可选的正十进制小数定义如下：

<decimal id="1" presence="optional" name="Value"/>

可选的正十进制小数编码示例见表12。

表12　可选的正十进制小数编码示例

输入值	本地十六进制/二进制		IMAST 十六进制/二进制	
ASCII 码	指数	尾数	指数	尾数
94275500	2	942755	0x83 10000011	0x39 0x45 0xa3 00111001 01000101 10100011
注：停止位由黑体表示。符号位由下划线表示。域可选且值非负，则指数部分自加1。				

可选的负十进制小数定义如下：

<decimal id="1" presence="optional" name="Value"/>

可选的负十进制小数编码示例见表13。

表13　可选的负十进制小数编码示例

输入值	本地十六进制/二进制		IMAST 十六进制/二进制	
ASCII 码	指数	尾数	指数	尾数
-9427.55	-2	-942755	0xfe 11111110	0x46 0x3a 0xdd 01000110 00111010 11011101
注：停止位由黑体表示。符号位由下划线表示。				

带符号位扩展的可选的负十进制小数定义如下：

<decimal id="1" presence="optional" name="Value"/>

带符号位扩展的可选的负十进制小数编码示例见表14。

表14　带符号位扩展的可选的负十进制小数编码示例

输入值	本地十六进制/二进制		IMAST 十六进制/二进制	
ASCII 码	指数	尾数	指数	尾数
-8.193	-3	-8193	0xfd	0x7f 0x3f 0xff

输入值	本地十六进制/二进制		IMAST 十六进制/二进制	
ASCII 码	指数	尾数	指数	尾数
			1<u>1</u>111101	*0<u>1</u>111111* **0**0111111 **1**1111111
注：停止位由黑体表示。符号位由下划线表示。符号位扩展（斜体）是指定符号所必需的。				

采用单个域操作符的可选的正十进制数定义如下：

<decimal id="1" presence="optional" name="Value"><copy/></decimal>

采用单个域操作符的可选的正十进制数编码示例见表15。

表15　采用单个域操作符的可选的正十进制数编码示例

输入值	本地十六进制/二进制		IMAST 十六进制/二进制	
ASCII 码	指数	尾数	指数	尾数
9427.55	-2	942755	0xfe 1<u>1</u>111110	0x39 0x45 0xa3 **0**<u>0</u>111001 **0**1000101 **1**0100011
注：停止位由黑体表示。符号位由下划线表示。域可选，且在域一级指定了操作符，则PMAP中只占用单个位元。				

采用单独域操作符的可选的正十进制数定义如下：

<decimal id="1" presence="optional" name="Value">
<exponent><copy/></exponent>
<mantissa><delta/></mantissa>
</decimal>

采用单独域操作符的可选的正十进制数编码示例见表16。

表16　采用单独域操作符的可选的正十进制数编码示例

输入值	本地十六进制/二进制		IMAST 十六进制/二进制	
ASCII 码	指数	尾数	指数	尾数
9427.55	-2	942755	0xfe 1<u>1</u>111110	0x39 0x45 0xa3 **0**<u>0</u>111001 **0**1000101 **1**0100011
注：停止位由黑体表示。符号位由下划线表示。指数在PMAP中占1位，但尾数在PMAP中不占位。				

采用单独域操作符的可选的正十进制数的多 PAMP 位元定义如下：

<decimal id="1" presence="optional" name="Value">
<exponent><copy/></exponent>
<mantissa><copy/></mantissa>
</decimal>

采用单独域操作符的可选的正十进制数的多PAMP位元编码示例见表17。

表17　采用单独域操作符的可选的正十进制数的多 PAMP 位元编码示例

输入值	分解的输入值				PMAP 位	IMAST 十六进制/二进制	
ASCII 码	指数		尾数			指数	尾数
	值	编码	值	编码			
9427.55	-2（无前值）	-2	942755（无前值）	942755	11	0xfe **1**1111110	0x39 0x45 0xa3 **0**0111001 **0**1000101 **1**0100011
9427.60	-2	无	942760	942760	01	无	0x39 0x45 0xa8 **0**0111001 **0**1000101 **1**0101000
无	NULL	NULL	无	无	1	0x80 **1**0000000	无
注：停止位由黑体表示。符号位由下划线表示。由于使用单独的域操作符，指数和尾数各自在PMAP占单个位元。							

4.5.4.3　字符串域指令

4.5.4.3.1　概述

字符串域指令具有一个可选的指示字符串所使用字符集的参数“字符集（charset）”。支持的两种字符集为：ASCII和Unicode，分别由参数值“ascii”和“unicode”表示。

字符集默认为ASCII。当传输编码的字符串为Unicode格式，可对可选的“<td:length>”元素进行指定，将基础字节向量的长度前导（length preamble）和某个域指令名称关联。

4.5.4.3.2　ASCII 码字符串

ASCII码字符串由一个停止位编码实体来表示。其实体值被作为7位的ASCII码字符的序列进行解码。

位元序列若以7个0位元作为开始，则被称为具有一个零前导。当一个字符串在移除零前导后有剩余位元，并且剩余位元的前7个位元不全为0，则称该字符串过长（overlong），则产生可报告错误[ERR R9]。

零前导使用说明见表18。

表18　零前导使用说明

实体值	是否可空	说明
0x00	否	空字符串。
0x00 0x00	否	“\0”。
0x00 0x41	否	“A”，过长。
0x00	是	NULL 值。
0x00 0x000	是	空字符串。
0x00 0x41	是	“A”，过长。
0x00 0x00 0x00	是	“\0”。
0x00 0x00 0x41	是	“A”，过长。

4.5.4.3.3 Unicode 字符串

一个Unicode字符串由一个包含UTF-8编码格式的字符串的字节向量来表示。如果一个Unicode字符串可空，则用一个可空的字节向量来表示。

4.5.4.3.4 语法规则

ASCII码和Unicode字符串语法规则如下：

```
asciiStringField=element string{fieldInstrContent,attribute charset{"ascii"}}
unicodeStringField=element string{byteVectorLength?,fieldInstrContent,
attribute charset{"unicode"} }
```

注：

asciiStringField：ascii字符串域指令；

fieldInstrContent：域指令内容；

attribute charset：charset属性；

unicodeStringField：unicode字符串指令；

byteVectorLength：字节向量长度。

4.5.4.4 字节向量域指令

4.5.4.4.1 概述

字节向量域指令由一个无符号整数类型的长度前导及其之后的特定数量的原始字节来表示，并定义长度域。数据部分每一字节的有效数据位为8位。因此，其数据部分不采用停止位编码。

字节向量可空时，其长度前导也可为空。NULL字节向量由一个NULL长度前导来表示。

在具体语法中，可通过指定“<td:length>”元素将字节向量的长度前导和某个域指令名称关联。

4.5.4.4.2 语法规则

字节向量域指令语法规则如下：

```
byteVectorField=element byteVector{byteVectorLength,fieldInstrContent}
byteVectorLength=element length{nsName}
```

注：

byteVectorField：字节向量域指令；

byteVectorLength：字节向量长度；

fieldInstrContent：域指令内容；

nsName：长度命名空间名。

4.5.4.4.3 编码示例

可选的字节向量定义如下：

```
<byteVector id="1" presence="optional" name="Value"/>
```

可选的字节向量编码示例见表19。

表19　可选的字节向量编码示例

输入值	本地十六进制/二进制	IMAST 十六进制/二进制	
		长度	值
Null	N/A	0x80 **1**0000000	N/A
ABC	0x41 0x42 0x43 01000001 01000010 01000011	0x84 **1**0000011	0x41 0x42 0x43 01000001 01000010 01000011
零长度值	N/A	0x81 **1**0000001	N/A
注：域为可选，因此零长度值自加1。停止位由黑体表示。			

必选的字节向量定义如下：

<byteVector id="1" presence="mandatory" name="Value"/>

必选的字节向量编码示例见表20。

表20　必选的字节向量编码示例

输入值	本地十六进制/二进制	IMAST 十六进制/二进制	
		长度	值
ABC	0x41 0x42 0x43 01000001 01000010 01000011	0x83 **1**0000011	0x41 0x42 0x43 01000001 01000010 01000011
零长度值	N/A	0x80 **1**0000000	N/A
注：停止位由黑体表示。			

4.5.4.5　重复组域指令

4.5.4.5.1　概述

重复组域指令指示应用类型的域为重复组类型，可对其包括的域组件重复使用以对分组的每一元素进行编码。如果该分组中的任一域在占位图中占用一个位元，则在传送编码中每一组件由一个段（segment）表示，见图7。

```
<Sequence name="MDEntries">
    <length name="NoMDEntries" id="268"/>
    <uInt32 name="MDUpdateAction" id="279"> <copy/> </uInt32>
    <string name="MDEntryType" id="269"> <copy/> </string>
    <string name="Symbol" id="55"> <copy/> </string>
    <string name="SecurityType" id="167"> <copy/> </string>
    <decimal name="MDEntryPx" id="270"> <delta/> </decimal>
    <decimal name="MDEntrySize" id="271"> <delta/> </decimal>
    <uInt32 name="NumberOfOrders" id="346"> < delta/> </uInt32>
    <string name="QuoteCondition" id="276"> <copy/> </string>
    <string name="TradeCondition" id="277"> <copy/> </string>
</Sequence>
```

图7　重复组域指令示例

重复组域指令具有一个关联的长度域指令，该长度域指令包含一个无符号整数，用以表示包含组件或域指令集合的个数。若数据流中出现长度域，则应在重复组域指令定义后紧接出现。该长度域有名称，类型为uInt32，且可具有域操作符。

长度域指令的命名方式应采用显式方式（explicit）：名称在模板定义中显式的指定。

重复组域指令可为必选（mandatory）或者可选（optional）。若重复组域指令为可选，则其长度域指令也是可选。

在具体语法中，重复组域指令由“<td:sequence>”表示。可在任一指令前指定“<td:length>”子元素，该元素规定了长度域指令的属性。

4.5.4.5.2　语法规则

重复组域指令语法规则如下：

sequence=element sequence{nsName,presenceAttr?,dictionaryAttr?,length,instruction*}

length=element length{nsName?,fieldOp?}

注：

nsName：命名空间名；

dictionaryAttr：字典属性，可选元素；

length：重复组长度；

fieldOp：域操作符。

4.5.4.6　组件域指令

4.5.4.6.1　概述

组件域指令将一个域指令集合形成的组件和某个域指令名称及存在属性相关联，如果组件中的任一域指令在占位图中占用一个位元，则在传送编码中该组件由一个段（segment）表示。

组件域指令的主要用途是用占位图中的单个位元来指示一整组的域指令存在与否，其应用类型不要求与组件对应，即分组解码得到的字段在应用类型中展开为一层。

4.5.4.6.2　语法规则

组件域指令语法规则如下：

group=element group{nsName,presenceAttr?,dictionaryAttr?,instruction*}

注：

nsName：件命名空间名；

presenceAttr：存在属性，可选元素；

dictionaryAttr：字典属性，可选元素；

instruction*：N个域指令。

4.6 域操作符

4.6.1 语法规则

域操作符语法规则如下：

fieldOp=constant|default|copy|increment|delta

注：

fieldOp：域操作符；

constant：常量操作符；

default：默认操作符；

copy：拷贝操作符；

increment：递增操作符；

delta：差值操作符。

4.6.2 错误发生

域操作符不可随意定义。如果为操作符指定了不适用的域类型，则产生静态错误[ERR S2]。

4.6.3 字典

4.6.3.1 语法规则

字典语法规则如下：

opContext=dictionaryAttr?,nsKey?,initialValueAttr?

dictionaryAttr=attribute dictionary{"template"| "type"| "global"| string}

nsKey=keyAttr,nsAttr?

keyAttr=attribute key{token}

注：

dictionaryAttr：字典属性，可选元素；

nsKey：关键字命名空间，可选元素；

initialValueAttr：初始化值属性；

keyAttr：关键字属性；

nsAttr：命名空间属性，可选元素。

4.6.3.2 字典与前值

一些操作符依赖于前值（previous value）实现编码方式的优化，前值在字典中维护。字典是条目集合，其中，每一条目具有名称及特定类型的值。值可为以下三种状态之一：未定义（undefined），空（empty），已指定（assigned）。在协议开始处理时，所有值的初始状态均为未定义。空状态表示前值不存在，已指定状态表示前值存在。空状态只适用于可选的域。设置空状态的具体规则见4.7。

操作符在字典中的前值，是与其“关键字（key）”名称相同的条目的值。操作符的缺省关键字（key）是其域的名称。显式的关键字可用关键字（key）参数来指定。通过指定显式的关键字，不同名称的域的操作符可共享字典中的同一个前值条目。

如果操作符的域的类型与所访问的条目值的类型不相同，则产生动态错误[ERR D4]。

字典的名称在域操作符元素的“字典（dictionary）”参数中指定，或者在XML文件之前模板的某处元素中指定。如果已有多个之前模板的“字典”参数，则使用位置最近的元素的该参数。如果没有指定该参数，则使用全局字典。

下列三种字典是预定义的：

a) 模板（template）字典：字典限于当前模板使用。即当且仅当 T1=T2，模板 T1 中操作符与模板 T2 中操作符共享相同的字典；

b) 类型（type）字典：字典限于当前应用类型使用。即当且仅当 A1=A2，应用类型 A1 的模板 T1 中的操作符与应用类型 A2 的模板 T2 中的操作符共享相同的字典；

c) 全局（global）字典：字典为全局的。无论模板和应用类型如何，所有操作符共享相同的字典。

其余字典均称为用户自定义（user defined）字典。当且仅当指定的字典名称相同时，两个操作符才共享同一用户自定义字典。

字典可被显式地重置（reset）。重置一个字典将其所有条目的状态设置为未定义。在编码器和解码器中，重置出现的顺序应当与流的内容的顺序相对应。

4.6.4 域初始值

初始值（Initial Value）为一个Unicode字符集的字符串，由操作符元素的“值（value）”参数指定。该值可见4.8.4中定义的方法转换为域的类型。在对初始值进行解析的时候，类型转换过程中可能发生的动态错误和可执行错误，均被认为是静态错误[ERR S3]。

如果域为十进制小数类型，值转化称作值的标准化，当对指数和尾数单独应用操作符时，它们的值应是确定的。十进制小数值（decimal）的标准化是对尾数和指数进行调整，使得尾数整除10的余数不为0：即尾数%10≠0。例如100×10^{0} 将被标准化为1×10^{2} 。如果尾数为0，则标准化十进制数的尾数为0，指数也为0。

域初始值语法规则如下：

initialValueAttr=attributevalue {text}

注：

initialValueAttr：初始值属性；

attributevalue：值属性。

4.6.5 常量操作符

4.6.5.1 概述

常量操作符（Constant Operator）规定域的值为固定值不可变，此域的值为初始值，一般用来定义全局值。

如果无初始值则产生静态错误[ERR S4]。

常量域的值不被传送。

常量操作符适用于所有域类型。

4.6.5.2 编码规则

根据该域操作符的存在类型、输入值，编码规则如下：

a) 若存在类型是必选，输入值为初始值，则该域在占位图中不占位，数据流中不编码；
b) 若存在类型是可选，输入值为初始值，则该域在占位图中占一位且置 1，数据流中不编码；
c) 若存在类型是可选，没有输入值，则该域在占位图中占一位且置 0，数据流中不编码。

4.6.5.3 解码规则

根据该域操作符的存在类型、占位图，解码规则如下：
a) 若存在类型是必选，该域在占位图中不占位，则输入值为初始值；
b) 若存在类型是可选，该域在占位图中的位元置 1，则输入值为初始值；
c) 若存在类型是可选，该域在占位图中的位元置 0，则输入值不存在。

4.6.5.4 语法规则

常量操作符语法规则如下：
constant=element constant{initialValueAttr}
注：
initialValueAttr：初始值属性。

4.6.5.5 编码示例

必选的无符号整数常量操作符定义如下：
<uInt32 id="1" presence="mandatory" name="Flag"><constant value="0" /></uInt32>
必选的无符号整数常量操作符编码示例见表21。

表21 必选的无符号整数常量操作符编码示例

输入值	前值	编码值	PMAP 位	IMAST 十六进制/二进制
0	N/A	无	不需要	无
99	N/A	错误	错误	错误
无	N/A	错误	错误	错误

可选的无符号整数常量操作符定义如下：
<uInt32 id="1" presence="optional" name="Flag"><constant value="0" /></uInt32>
可选的无符号整数常量操作符编码示例见表22。

表22 可选的无符号整数常量操作符编码示例

输入值	前值	编码值	PMAP 位	IMAST 十六进制/二进制
0	N/A	无	1*	无
无	N/A	无	0*	无
注：使用常量操作符的可选的域在PMAP中占1位。如果输入值等于初始值，则该位设置为1。如果没有输入值，则该位设置为0。				

4.6.6 默认操作符

4.6.6.1 概述

默认操作符适用于所有域类型。

4.6.6.2 编码规则

根据该域操作符的存在类型、输入值，编码规则如下：

a) 如果存在类型是必选，输入值和初始值相同，则该域在占位图中对应的位元置 0，数据流中不编码；
b) 如果存在类型是必选，输入值和初始值不同，则该域在占位图中对应的位元置 1，数据流中的编码值为输入值；
c) 如果存在类型是可选，输入值和初始值相同，则该域在占位图中对应的位元置 0，数据流中不编码；
d) 如果存在类型是可选，输入值为 NULL，则该域在占位图中对应的位元置 1，数据流中编码为空值，表示该域不存在；
e) 如果存在类型是可选，输入值和初始值不同且不为 NULL，则该域在占位图中对应的位元置 1，数据流中的编码值为输入值。

4.6.6.3 解码规则

当操作符为默认操作符时，如果传输的数据与初始值相同，则不进行传输。解码过程的基本处理流程：

a) 域值在数据流中，则域值为压缩前的输入值；
b) 域值不在数据流中：如果有初始值，则域值为初始值；如果没有初始值，且为可选字段，则值为 NULL。

4.6.6.4 语法规则

默认操作符语法规则如下：

default=element default{initialValueAttr?}

注：

initialValueAttr：初始值属性。

4.6.6.5 编码示例

必选的无符号整数默认操作符定义如下：

<uInt32 id="1" presence="mandatory" name="Flag"><default value="0" /></uInt32>

必选的无符号整数默认操作符编码示例见表23。

表23 必选的无符号整数默认操作符编码示例

输入值	前值	编码值	PMAP 位	IMAST 十六进制/二进制
0	N/A	无	0	无
1	N/A	1	1	0x81 10000001

可选的无符号整数默认操作符定义如下：

<uInt32 id="1" presence="optional" name="Flag"> <default/> </uInt32>

可选的无符号整数默认操作符编码示例见表 24。

表24　可选的无符号整数默认操作符编码示例

输入值	前值	编码值	PMAP 位	IMAST 十六进制/二进制
无	N/A	无	0	无

4.6.7　拷贝操作符

4.6.7.1　概述

拷贝操作符（Copy Operator）规定域值在数据流中是可选存在的。如果域值在数据流中存在，则成为新的前值。

拷贝操作符适用于所有的域类型。

4.6.7.2　编码规则

根据该域操作符的存在类型、输入值，编码规则如下：

a)　如果域存在类型为必选且前值为已指定状态，若输入值和前值相等，则在占位图中与该域相对应的位元置 0，数据流中无编码值，否则位元置 1，数据流中编码值为输入值；

b)　如果域存在类型为可选且前值为未定义状态，若输入值和前值相等，则在占位图中与该域相对应的位元置 0，数据流中无编码值，否则位元置 1，数据流中编码值为输入值；

c)　如果域存在类型为可选且前值为空状态，若输入值不为 NULL，则数据流中编码值为输入值，否则数据流中编码值为 NULL。

4.6.7.3　解码规则

当域操作符为拷贝操作符时，在编码时如果输入值与前值相同，则不传输。其解码过程的基本处理流程为：

a)　域值在数据流中，则域值为该解析值；

b)　如果域值不在数据流中：

1)　如果前值已定义，则为前值；

2)　如果前值未定义，则为初始值；

3)　如果没有初始值，且为可选类型，则值为NULL。

4.6.7.4　语法规则

拷贝操作符语法规则如下：

copy=element copy{opContext}

注：

opContext：结合上下文的拷贝操作。

4.6.7.5　编码示例

必选的字符串拷贝操作符定义如下：

<string id="1" presence="mandatory" name="Flag"><copy/></string>

必选的字符串拷贝操作符编码示例见表25。

表25　必选的字符串拷贝操作符编码示例

输入值	前值	编码值	PMAP 位	IMAST 十六进制/二进制
CME	无	CME	1	0x43 0x4d 0xc5
				01000011　　**0**1001101 **1**1000101
CME	CME	无	0	无
ISE	CME	ISE	1	0x49 0x53 0xc5 **0**1001001　　**0**1010011 **1**1000101

可选的字符串拷贝操作符 NULL 值定义如下：

<string id="1" presence="optional" name="Flag"><copy/></string>

可选的字符串拷贝操作符 NULL 值编码示例见表 26。

表26　可选的字符串拷贝操作符 NULL 值编码示例

输入值	前值	编码值	PMAP 位	IMAST 十六进制/二进制
无	无	Null	1	0x80 **1**0000000
无	Null	无	0	无
CME	Null	CME	1	0x43 0x4d 0xc5 **0**1000011　　**0**1001101 **1**1000101

4.6.8　递增操作符

4.6.8.1　概述

递增操作符（Increment Operator）适用于整数域类型。

整数的递增为自加1。如果其值已经是类型最大值，则递增后的值为类型最小值。

4.6.8.2　编码规则

域操作符为递增操作符，用于比较前后的传输值是否为递增，分为下列三种情况处理：

a)　当前值是否为前值加 1，如果相等，则不进行传输，否则应传输；

b)　如果前值未定义，判断当前值是否与初始值相同，如果相同则不进行传输，否则应传输；

c)　如果域存在类型为可选类型，输入值为 NULL，前值未定义且初始值未定义，编码值为 NULL，不进行传输，否则通过 NULL 表示。

4.6.8.3　解码规则

递增操作符（Increment Operator）规定域的值在数据流中是可选存在的。如果值在数据流中存在，则成为新的前值。

当值在数据流中不存在，则根据前值的状态分为下列三种情况处理：

a)　已指定状态：则域值为前值加 1。计算得到的值成为新的前值；

b) 未定义状态：则域值为初始值，并且成为新的前值。除非域是可选存在的，否则若无初始值，则产生动态错误[ERR D5]。如果域为可选存在的，并且无初始值，则域被视作不存在，同时前值的状态变为空；
c) 空状态：则域值为空。如果域是可选的，则值被视作不存在。如果域是必要的，则产生动态错误[ERR D6]。

4.6.8.4 语法规则

递增操作符语法规则如下：

increment=element increment{opContext}

注：

opContext：结合上下文的递增操作。

4.6.8.5 编码示例

必选的无符号整数递增操作符定义如下：

<uInt32 id=" 1 " presence=" mandatory " name=" Flag " > <increment value=" 1 " /> </uInt32>

必选的无符号整数递增操作符编码示例见表 27。

表27 必选的无符号整数递增操作符编码示例

输入值	前值	编码值	PMAP 位	IMAST 十六进制/二进制
1	1	无	0	无
2	1	无	0	无
4	2	4	1	0x84 10000100
5	4	None		

4.6.9 差值操作符

4.6.9.1 概述

差值操作符定义数据流中出现了一个差值。如果域是可选存在的，则差值可为NULL。如果是这种情况，则该域值被视作不存在。其余情况，域值通过差值 （delta value）和基值（basic value）的结合得到。

根据前值的状态，基值按照下列方式来确定：

a）已指定状态：则基值为前值；

b）未定义状态：如果存在初始值，则基值为初始值。除此之外，使用由类型决定的缺省基值；

c）空状态：如果前值为空则产生动态错误[ERR D6]。

4.6.9.2 语法规则

差值操作符语法规则如下：

delta=element delta{opContext}

注：

opContext：结合上下文的差值操作。

4.6.9.3 具体应用规则

4.6.9.3.1 整数的差值

4.6.9.3.1.1 通则

差值由一个带符号整数来表示。可空的整数差值由一个可空的带符号整数来表示。

结合值是基值和差值的总和。

整数的缺省基值为0。

如果结合值大于特定整数类型的最大值，或小于特定整数类型的最小值，则产生可报告错误[ERR R4]。

注：差值所需的整数大小可能大于域类型的特定大小。例如，如果无符号32位整数的基值为4294967295，新值为17，则应带符号64位整数来表示差值-4294967278。但这并不影响差值在数据流中的表示方式。

4.6.9.3.1.2 编码规则

当域的存在属性为必选类型时，如果初始值和前值均存在，则数据流中的编码值为输入值和前值之差；如果初始值存在但前值未定义，则数据流中的编码值为输入值和初始值之差；如果初始值和前值都是未定义，则数据流中的编码值为输入值和整数类型的基值0之差。

当域的存在属性为可选类型时，除了差值在数据流中以可空的表示方式出现外，其编码规则和必选类型一样；当输入值为空时，表示该域不存在，通过NULL进行表示。

4.6.9.3.1.3 解码规则

整数类型差值的解码相对比较简单，当存在类型为必选类型且初始值存在时，如果前值已指定则解析值为输入值加前值；若前值未定义，则解析值为输入值加初始值。当存在类型为可选类型初始值存在时，如果前值未定义，输入值为NULL，则解析时表示该域缺失；当存在类型可选初始值不存在时，如果前值未定义输入值不为NULL，则解析值为输入值加基值0。

4.6.9.3.2 十进制小数的差值

4.6.9.3.2.1 通则

差值由两个带符号整数来表示。第一个整数为指数的差值，第二个为尾数的差值。如果差值是可空的，则其具有一个可空的指数差值和一个不可空的尾数差值。一个NULL差值由一个NULL的指数差值来表示。当且仅当指数差值不为NULL时，尾数差值在流中出现。

结合值是差值和基值的指数和尾数部分分别对应相加计算得到的值。

如果结合后的指数小于-63或大于63，或者结合后的尾数超过64位整数的表示范围，则产生可报告错误[ERR R1]。

十进制小数的缺省基值为0，指数的缺省基值为0，尾数的缺省基值为0。

4.6.9.3.2.2 编码规则

十进制小数类型差值的编码过程与整数类型相似，其差值为指数差值和尾数差值所表示的十进制小数。

当域的存在属性为必选类型时，如果初始值和前值均存在，则编码值为指数差（输入值的指数和前值的指数之差）和尾数差（输入值的尾数和前值的尾数之差）所表示的十进制小数；如果初始值存在但前值未定义，则编码值为指数差（输入值的指数和初始值的指数之差）和尾数差（输入值的尾数和初始

值的尾数之差）所表示的十进制小数；如果初始值和前值均未定，则编码值为指数差（输入值的指数和基值0的指数之差）和尾数差（输入值的尾数和基值0的尾数之差）所表示的十进制小数。

当域的存在属性为可选类型时，除了差值在数据流中以可空的表示方式出现外，其编码规则和必选类型一样；当输入值为空时，表示该域不存在，通过NULL进行表示。

4.6.9.3.2.3 解码规则

十进制小数按照指数和尾数的方式分别进行差值计算。

当存在类型为必选类型且初始值存在时：如果前值和输入值均存在，则解析值的指数和尾数部分为前值和输入值的指数和尾数部分分别相加之后的结果；如果前值未定义但输入值存在，则解析值的指数和尾数部分为初始值和输入值的指数和尾数部分分别相加之后的结果。

当存在类型为可选类型且初始值不存在时：如果前值未定义输入值存在，则解析值的指数和尾数部分为基值0和输入值的指数和尾数部分分别相加之后的结果。

4.6.9.3.3 ASCII码字符串的差值

4.6.9.3.3.1 通则

差值由一个带符号的整数类型的修剪长度，及其之后的ASCII字符串来表示。如果差值可空，则修剪长度可空。一个NULL差值用一个NULL修剪长度来表示。当且仅当修剪长度不为NULL时，流中存在字符串部分。

差值的修剪长度（subtraction length）规定了要从基值前端或后端移除的字符的数量。当修剪长度为负（negative）时，从前端移除。差值的字符串部分表示按照修剪长度符号所指方向，添加到基值的字符。

修剪长度使用“额外减1（excess-1）”的编解码方式：解码时，如果值为负，则加1后得到减除的字符数量。这使得可将负0编码为-1，以使得可在无需移除任何字符的情况下在前端进行增加的操作。缺省基值为空字符串。

如果减除的长度大于基值中的字符数量，或者超出了32位带符号整数的范围，则产生动态错误[ERRD7]。

4.6.9.3.3.2 编码规则

当域的存在属性为必选类型时，如果初始值和前值均存在，则比较输入值和前值得到修剪长度和字符串，对修剪长度和字符串进行相应编码；如果前值未定义但初始值存在，则比较输入值和初始值得到修剪长度和字符串，对得到的修剪长度和字符串进行相应编码；如果前值和初始值均未定义，则比较输入值和基值字符串得到修剪长度和字符串，并对修剪长度和字符串进行相应编码。

当域的存在属性为可选类型时，除了差值在数据流中以可空的表示方式出现外，其编码规则和必选类型一样；当输入值为空时，表示该域不存在，通过NULL进行表示。

4.6.9.3.3.3 解码规则

当差值在压缩数据流中存在时，通过停止位编码还原可得到差值中的修剪长度和字符串，根据域的存在属性和修剪长度的符号，判断在前值的基础上的操作方向和位数，然后在前值尾部或头部添加差值中的字符串，即得到解析值。

4.6.9.3.4 Unicode字符串的差值

Unicode字符串的差值，在结构上与字节向量相同，但字节向量的内容应为UTF-8编码格式。差值是对编码字节，而不是Unicode字符进行操作，因此差值可能由一个不完整的UTF-8字节序列构成。结合值如果不是一个合法的UTF-8序列的话，则产生可报告错误[ERR R2]。

由于Unicode字符串和字节向量的差值的编解码规则和ASCII字符串的差值的编解码规则相似。

4.6.9.3.5 字节向量的差值

差值由一个带符号整数类型的修剪长度及其之后的字节向量来表示。如果差值可空，则修剪长度可空。当且仅当修剪长度不为NULL时，流中存在字节向量部分。

差值的修剪长度规定了从基值前端或后端移除的字节数量。当修剪长度为负时，字节从前端被移除。差值的字节向量部分表示差值的字符串部分按照修剪长度符号所指方向，添加到基值的字符。

修剪长度使用与ASCII码字符串情况相同的“额外减1”的编码方式：如果解码时值为负，则加1后得到要减除的字符数量。

缺省的基值为空的字节向量。

如果修剪长度大于基值的字符数量，或者超出了int32的表示范围，则产生动态错误[ERR D7]。

4.6.9.4 编码示例

必选的带符号整数差值操作符定义如下：

<int32 id="1" presence="mandatory" name="Price"><delta/></int32>

必选的带符号整数差值操作符编码示例见表28。

表28 必选的带符号整数差值操作符编码示例

输入值	前值	编码值	PMAP位	IMAST十六进制/二进制
942755	0	942755	N/A	0x39 0x45 0xa3 **0**0111001 **0**1000101 **1**0100011
942750	942755	-5	N/A	0xfb **1**1111011
942745	942750	-5	N/A	0xfb **1**1111011
942745	942745	0	N/A	0x80 **1**0000000
注：示例中初始前值为0。缺省值可在模板中进行指定。				

必选的十进制小数差值操作符定义如下：

<decimal id="1" presence="mandatory" name="Price"><delta/></decimal>

必选的十进制小数差值操作符编码示例见表29。

表29 必选的十进制小数差值操作符编码示例

输入值	前值		编码值		PMAP位	IMAST十六进制/二进制	
	指数	尾数	指数	尾数		指数	尾数
9427.55	0	0	-2	942755	N/A	0xfe **1**1111110	0x39 0x45 0xa3 **0**0111001 **0**1000101

输入值	前值		编码值		PMAP 位	IMAST 十六进制/二进制	
	指数	尾数	指数	尾数		指数	尾数
							10100011
9427.51	-2	942755	0	-4	N/A	0x80 10000000	0xfc 11111100
9427.46	-2	942751	0	-5	N/A	0x80 10000000	0xfb 11111011

具有初始值必选的十进制小数差值操作符定义如下：

<decimal id="1" presence="mandatory" name="Price"><delta value="12000" /></decimal>

具有初始值必选的十进制小数差值操作符编码示例见表 30。

表30　具有初始值必选的十进制小数差值操作符编码示例

输入值	前值		编码值		PMAP 位	IMAST 十六进制/二进制	
	指数	尾数	指数	尾数		指数	尾数
12000	0	0	N/A	N/A	N/A	N/A	N/A
12100	3	12	-2	1198	N/A	0xfe 11111110	0x09 0xae 00001001 10101110
12150	1	1210	0	5	N/A	0x80 10000000	0x85 10000101
12200	1	1215	0	5	N/A	0x80 10000000	x85 10000101

必选的字符串差值操作符定义如下：

<decimal id="1" presence="mandatory" name="Price"><delta value="12000" /></decimal>

必选的字符串差值操作符编码示例见表 31。

表31　必选的字符串差值操作符编码示例

输入值	前值	编码值	修剪长度	PMAP 位	IMAST 十六进制/二进制	
					修剪长度	编码字符串
GEH6	空字符串	GEH6	0	N/A	0x80 10000000	0x47 0x45 0x48 0xb6 01000111 01000101 01001000 10110110
GEM6	GEH6	M6	2	N/A	0x82 10000010	0x4d 0xb6 01001101 10110110
ESM6	GEM6	ES	-2	N/A	0xfd 11111101	0x45 0xd3 01000101 11010011
RSESM6	ESM6	RS	-0	N/A	0xff 11111111	0x52 0xd3 01010010 11010011

输入值	前值	编码值	修剪长度	PMAP 位	IMAST 十六进制/二进制	
					修剪长度	编码字符串
注：负的修剪长度用于描述从字符串的前端移除值。修剪长度为负0用于在字符串的前端附加值。如果在模板中没有指定缺省的字符串，则初始的前值为空字符串。						

4.6.10 操作符总结

操作符的总结见图8。

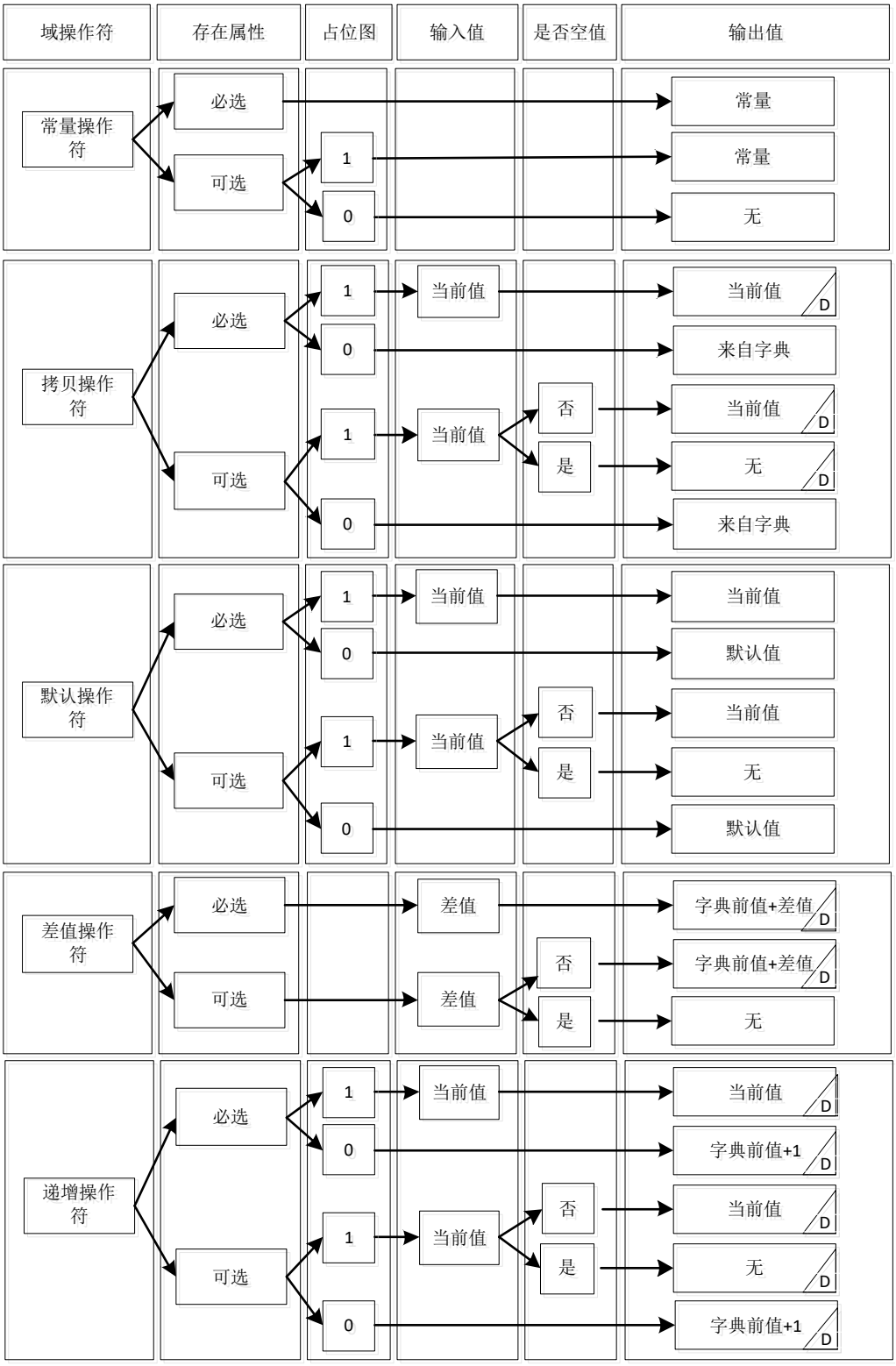


图8　操作符总结图

4.7　占位图使用规则

占位图（Presence Map PMAP）为位元的一个序列。占位图表示段中的域按当前模板的规定来使用位元。

占位图通过停止位编码进行分割。占位图中0表示相应的数据在数据流中不存在，可对以一串0结尾的占位图进行截取，剩余部分长度应为7的倍数。

如果一个超过7位的占位图以7位或7位以上的0结尾，则产生一个可报告错误[ERR R7]。如果占位图包含的有效位元数大于指令所需的位元数，则产生可报告错误[ERR R8]。

不同的域操作符占位图编码规则见表32，域操作符定义见4.6。

表32　占位图编码规则

操作符	是否应占用 PMAP 的位元	
	必选	可选
无	否	否
常量<constant/>	否	是
拷贝<copy/>	是	是
默认<default/>	是	是
差值<delta/>	否	否
递增<increment/>	是	是
注：使用常量操作符的可选的域占用1位。如果输入值等于初始值的话，该位被设置为1。如果输入值不存在，则该位被设置为0。		

占位图按照域条目的顺序分配位元。即在模板中先出现的域比后出现的域所分配的位元的顺序要高。位元在当前段的占位图中进行分配。

域和组件域对占位图的使用规则如下：

a) 如果一个域具有常量操作符且为必选，则其在占位图中不占位。

b) 如果一个域具有常量操作符且为可选，则其在占位图中占 1 位。如果该位在占位图中被设置成 1，则值为初始值。如果该位在占位图中被设置成 0，则值被视作不存在。

c) 如果一个域没有域操作符且为必选，则其在占位图中不占位，但其值应在数据流中存在。

d) 如果一个域没有域操作符且为可选，则其采用可空的表示方法进行编码，并且用 NULL 表示值不存在，其在占位图中不占位。

e) 如果一个组件域为可选，则其在占位图中占 1 位。只有该位在占位图中被设置成 1 时，该组件的内容才可能在数据流中出现。如果该位在占位图中设置成 0，则组件中的域指令将不被处理。

默认、拷贝和递增操作符对占位图和空值的使用规则如下：

a) 必选的整数、十进制小数、字符串和字节向量域：占 1 位。如果在占位图中被设置成 1，则值在数据流中出现。

b) 可选的整数、十进制小数、字符串和字节向量域：占 1 位。如果在占位图中被设置成 1，则值在数据流中以可空的表示方式出现。NULL 表示值不存在，前值的状态也将被设置为空，但是在使用默认操作符时，前值的状态则不作更改。

差值操作符对占位图的使用规则如下：

a) 必选的整数、十进制小数、字符串和字节向量域：不占位。

b) 可选的整数、十进制小数、字符串和字节向量域：不占位。差值在数据流中以可空的表示方式出现。NULL 表示差值不存在。

具有单独操作符的十进制小数域的使用规则如下：

a) 如果十进制小数是必须存在的，则指数域和尾数域将作为两个独立的、必选的整数域，按照前述的相关方式进行处理。
b) 如果十进制小数是可选存在的，则指数被视作可选的整数域，尾数域被视作必选的整数域。尾数域及其在占位图中相关位的存在与否取决于指数是否存在。当且仅当指数值被视作存在的，尾数域才在数据流中出现，尾数的操作符才可在占位图中占 1 位。

4.8 域类型转换

4.8.1 概述

特定情况下，模板的某个域的类型与当前应用类型所对应的域的类型不一致，则在域进行编码或者解码时，应进行值的转换。如果域的应用类型无法转换为模板中定义的域应用类型，或解码时无法由相应的域应用类型转换而得到，则产生动态错误[ERR D1]。

4.8.2 整数转换为其他类型

4.8.2.1 转换为整数

精度未发生丢失的前提下，不同类型的整数可进行相互的转换。如果目标类型无法表示值的话，则产生可报告错误[ERR R4]，如一个负值不应转换为一个无符号类型。

4.8.2.2 转换为十进制小数

当整数可由一个指数范围为[-63，63] 、尾数为int64的带比例数表示时，该整数可被转换为十进制数，否则产生可报告错误[ERR R1]。

4.8.2.3 转换为字符串

数字可使用不以0作起始的“0”-“9”的一个数字序列表示。如果类型是带符号的，且数字为负，则数字序列的前面接减号（“-”）。

4.8.3 十进制小数转换为其他类型

4.8.3.1 转换为整数

只有无小数部分时，一个十进制小数可转换为一个整数。即其值实际上为一个整数。如果其值实际上不为一个整数，则产生可报告错误[ERR R5]。

4.8.3.2 转换为字符串

如果数字是一个整数，则其当作整数类型进行转换。否则数字由以小数点分隔的一个整数部分和一个小数部分表示。每一部分均为“0”-“9”的数字序列。小数点的两边都应至少有1个数字。如果数字为负，则前面接减号（“-”）。整数部分若有2个或2个以上的数字，转换后的字符串不应以0开始。

4.8.4 字符串转换为其他类型

4.8.4.1 基础规则

在字符串转换前，需进行空白切除操作。空白切除（whitespace trimming)是指在字符串被解析前将位于起始和结尾处的空白移除的操作。下列十六进制ASCII码的字符被视作空白：20 （空格)，09（横向制表符)，0D （回车)，0A （换行)。

如果一个字符串与下文中所规定的语法不匹配，则产生动态错误[ERR D11]。

注：虽然带符号整数和十进制数的语法允许形如-0和-0.0的负零值，但此类值将被视为正数形式。负零值在IMAST流中无法表示。

4.8.4.2 字符串转换为整数

字符串被作为“0”-“9”数字的序列解析。如果为带符号类型，则可以起始的减号来表示负数。如果结果数字超出了特定整数的大小，则产生可报告错误[ERR R4]。

4.8.4.3 字符串转换为十进制小数

字符串具有一个整数部分和一个小数部分。可对两者进行指定，或者只对其中之一进行指定。如果对两者都进行了指定，则它们之间应带有小数点。起始的减号表示是负数。转换结果的指数或者尾数超过表示范围，则产生可报告错误[ERR R1]。

4.8.4.4 字符串转换为字节向量

字符串被作为偶数个的[0-9A-Fa-f]的十六进制数字解析，其中可能夹带有空白。文字转换为一个字节向量后，对其中的每对字符按照单字节的十六进制数解析。

4.8.4.5 字符集间的相互转换

ASCII是Unicode的一个子集，将ASCII码字符串转换为Unicode字符串较为简单。如果Unicode字符串只包含ASCII字符的话，可转换为ASCII码字符串，否则产生可报告错误[ERR R3]。

4.8.5 字节向量转换为其他类型

字节向量转换为字符串，字节向量由偶数个[0-9a-f]的十六进制数字的序列表示。其中每一对表示向量中一个字节的十六进制数。

字节向量只能与字符串进行相互转换，字节向量与其余类型的转换均产生动态错误[ERR D10]。

4.9 辅助标识符

可带名称且非引用的构件，可带有辅助的标识符（auxiliary identifier），该标识符由“标识符（id）”参数指定。

例如，当使用IMAST传送IMIX报文时，辅助标识符可用于指定每一个域的IMIX域号。此外，当在双方通信不支持动态字典交换和标识符指定时，辅助标识符可用于指定静态的字典标识符。

附 录 A
（资料性附录）
错误代码汇总

A.1 静态错误

对模板定义进行检验所发现的错误称为静态错误（static error)。编码器和解码器应对静态错误进行捕获，并将发生该类错误的模板丢弃。

当从XML文档中读取模板定义时，若该文档出现下列情况则为一个静态错误[ERR S1]：

a) 不是正确的 XML 定义格式；

b) 不符合 XML 命名空间[XMLNS]中的约束；

c) 不符合附录 A 中规定的格式。

静态错误代码见表 A.1。

表A.1 静态错误代码表

错误代码	说明
ERR S1	如果按照具体XML语法进行编码的模板的实际的格式不正确，或者不符合XML命名空间规则，或者不符合附录A中的模式，则为一个静态的错误。
ERR S2	如果为操作符指定了该操作符不适用的域类型，则为一个静态错误。
ERR S3	如果在具体的语法中由“值（Value)”参数所指定的初始值无法转换为域类型的值，则为一个静态错误。
ERR S4	如果常量操作符没有指定初始值，则为一个静态错误。
ERR S5	如果一个必选的域的默认操作符没有指定初始值，则为一个静态错误。

A.2 动态错误

对IMAST流进行编码或解码时检测到的错误分为动态错误（dynamic error)和可报告错误（reportable error)两种。编码器和解码器应对动态错误进行捕获，宜对可报告错误进行捕获。

动态错误代码见表 A.2。

表A.2 动态错误代码表

错误代码	说明
ERR D1	如果模板中的域的应用类型无法转换为相应的域应用类型，或无法由相应的域应用类型转换而来，则为一个动态错误。
ERR D2	如果数据流中的整数的范围超出了指定类型的范围，则为一个动态错误。
ERR D3	如果由于指数和尾数使用单独的操作符而引入了某些限制，使得一个十进制小数值无法被编码，则为一个动态错误。
ERR D4	如果前值的类型与当前操作符的域类型不相同，则为一个动态错误。
ERR D5	如果一个必选的域在数据流中不存在，且前值为未定义，也无初始值，则为一个动态错误。
ERR D6	如果一个必选的域在数据流中没有存在，且前值为空，则为一个动态错误。
ERR D7	如果修剪长度大于基值的长度，或者超出了int32的表示范围，则为一个动态错误。

错误代码	说明
ERR D8	如果对于编码器或者解码器来说，静态模板引用的名称所指向的模板不可知，则为一个动态错误。
ERR D9	如果解码器无法找到与数据流中出现的模板标识符所关联的模板，则为一个动态错误。
ERR D10	除字符串外，字节向量与其他类型的相互转换均为一个动态错误。
ERR D11	如果字符串的语法不符合类型转换的结果类型的规则，则为一个动态错误。
ERR D12	如果块的大小前导为0，则为一个动态错误。

A.3 可报告错误

可报告错误代码见表A.3。

表A.3 可报告错误代码表

错误代码	说明
ERR R1	如果一个十进制小数无法以[-63,63]范围的指数表示，或者其尾数超过了 int64 的范围，则为一个可报告错误。
ERR R2	如果对一个 Unicode 字符串应用差值操作符，得到的结合值不是一个合法的 UTF-8 序列，则为一个可报告错误。
ERR R3	在将一个 Unicode 字符串转换为 ASCII 码字符串时，如果其包含 ASCII 字符集以外的字符，则为一个可报告错误。
ERR R4	如果在转换时，一个整数类型的值无法由目标整数类型来表示，则为一个可报告错误。
ERR R5	在将一个十进制小数转换为整数时，如果指数为负，或者结果整数不应由目标整数类型来表示，则为一个可报告错误。
ERR R6	如果出现对一个整数的过长编码，则为一个可报告错误。
ERR R7	如果占位图过长，则为一个可报告错误。
ERR R8	如果占位图包含的位元大于所需的位元，则为一个可报告错误。
ERR R9	如果出现对一个字符串的过长编码，则为一个可报告错误。