第5章　二进制XML内容格式规范

5.1　范围

无线应用协议（Wireless Application Protocol，WAP）是WAP论坛经过不断努力得到的成果，它提供了一个业界技术规范，以便开发出适用于各种无线通信网络的应用和业务。 WAP论坛的工作范围就是为各种业务和应用制定一系列的技术规范。无线市场正在快速增长，新的用户不断增多，新的业务不断涌现。为了使运营商和生产者能够从容面对先进业务、多种类业务和快速、灵活的业务生成等诸多的挑战， WAP规定了一系列传输层、会话层和应用层协议。有关 WAP体系结构更多的信息，请参阅“ 无线应用协议体系结构规范 ” (Wireless Application Protocol Architecture Specification) [WAP]。

本规范定义了可扩展标记语言 (Extensible Markup Language，XML) 紧凑的二进制表示方法。设计二进制XML内容格式是为了减少XML文档的传输量，使XML数据能在窄带信道上得到更有效的利用。二进制 XML内容格式的使用示例请参阅 [WML]规范。

定义二进制格式是为了使功能性信息或称为语义信息无损失地紧凑传输，这个格式可以保持XML的元素结构，使一个浏览器可以跳过未知的元素或属性。二进制格式是对 XML文档的实际解析格式进行编码，换句话说，是对文档实体的结构和内容进行编码。当文档被转换成二进制格式时，包含文档类型定义和条件部分的元信息被移走。

5.2　二进制XML内容结构

在XML标记格式规范中使用的数据类型如表 5-1所示。

网络字节顺序采用低字节高置（ big-endian）的存取次序，换句话说，网络首先传输最高字节，然后传输低字节。字节的网络比特顺序也采用低比特高置（ big-endian）的存取次序，即最先描述的比特段存储在这个字节的最高位地址中。

表5-1　标记格式中使用的数据类型

数据类型	含　义
bit	一个数据比特
byte	8个非透明数据比特
u_int8	8比特无符号整数
mb_u_int32	32比特无符号整数，用于多字节整数格式的编码

5.2.1　多字节整数

对于整数值，编码采用多字节表示法，一个多字节整数由一系列的八位组 (octet)构成。一个八位组中，最重要的比特位是连续标志位，其余的 7个比特表示整数数值。连续标志位为 1，表明这个八位组不是多字节序列的尾字节。若一个整数值的编码序列由 N个八位组构成，那么

前面N-1个字节的连续标志位是1，最后一个字节的连续标志位是0。

每个八位组中非连续标记位的7个比特也采用低比特高置的存取次序，即最高位首先被传输。所有的八位组也以低字节高置的存取次序排列，换句话说，最高八位组上的 7个比特首先被传输。对于取值小于7个比特的情况，未被使用的比特位必须置0。

例如，整数值0xA0编成2个字节的序列0x81 0x20，整数0x60被编成1个字节的序列0x60。

5.2.2 字符编码

在XML二进制内容格式中，所有字符串的编码由传输或容器元信息指定，如果二进制XML内容与一个字符集“ charset ”声明一起出现，说明这是字符串的编码。 XML二进制表示法支持任意字符串的编码。为了可靠地检测到一个字符串是否结束，所有的字符串都必须包含一个特殊的编码结束符 (也就是NULL结束符)。若字符编码中含有一个 NULL字符(也就是Unicode、ASCII、ISO-8859-1等字符集中规定的空白符)，这个NULL字符一定被当作结束符。与XML的文本格式一样，所有的标签（ tag）名和属性（attribute）名都可以用目标字符的编码表示。

5.2.3 文档结构的BNF

一个二进制XML文档由一系列的元素组成，每个元素有零个或多个属性，并可能包含嵌入的内容。这种结构非常普通，不需要对 XML元素结构或语义有清楚的了解，这种通用性让用户代理和二进制格式的其他用户可以跳过各种无法理解的元素和数据。

下面是类似于BNF记号结构的描述。除了使用字符“ | ”表示二者选一和稍后给出定义的用大写单词表示单字节记号之外，其余的描述均符合 [RFC822]标准中的习惯用法。简单地说，就是用圆括号来“ (”和“) ”表示组元素，用方括号 “ [”和“] ”表示可选内容，元素前的记号“ <N>* ”，表示紧随其后的元素重复出现N次或多次(没有特别声明时，N的默认值是0)。

```
start       = version publicid strtbl 1*content
strtbl      = length *byte
content     = element | string | extension | entity | pi

element     = stag [ 1*attribute END ] [ *content END ]
stag        = TAG | ( LITERAL index)
attribute   = attrStart *attrValue
attrStart   = ATTRSTART | ( LITERAL index )
attrValue   = ATTRVALUE | string | extension | entity

extension   = ( EXT_I termstr ) | (EXT_T index ) | EXT

string      = inline | tableref
inline      = STR_I termstr
tableref    = STR_T index

entity      = ENTITY entcode
entcode     = mb_u_int32        // UCS字符编码

pi          = PI attrStart *attrValue END

version     = u_int8 （包括一个WBXML 的版本号）
```

```
publicid    = mb_u_int32 | ( zero index )
termstr     = 带有结束符并与字符集有关的字符串。
index       = mb_u_int32          // 字符串表中的整数索引号
length      = mb_u_int32          // 整数长度
zero        = u_int8              // 0值
```

5.2.4 版本号

```
version = u_int8（包括WBXML 版本号）
```

所有的WBXML文档在开始的字节中都包含一个版本号，这个版本号给出了 WBXML的版本。在标明版本的字节中，高 4位表示比主版本号小1的数字，低4位表示次版本号。例如，版本号2.7的编码结果是0x17。在本文件中，WBXML的版本号是1.0。

5.2.5 文档公共标识符

```
publicid    = mb_u_int32 | ( zero index )
zero        = u_int8        //包括零值（0）
```

二进制XML格式包括了 XML文档公共标识符的一种表示方法。 Publicid用于标识在WBXML实体中知名的文档类型。

Publicid的第一种格式是一个大于零的多字节正整数值，这个正值用来表示知名的文档类型(如，-//WAPFORUM//DTD WML 1.0//EN)。

mb_u_int32

公共标识符也可用于字符串的编码，在这种情况下，前面定义的数字标识符无效。

0	index

请参见5.4.2节与公共标识符相关的数字常量。

5.2.6 字符串表

```
strtbl = length *byte
```

在公共标识符之后，紧接着就是二进制 XML文档必须具有的字符串表，字符串表至少由一个mb_u_int32组成，用于对字符串表的字节长度而不是字符串表的长度进行编码（例如，一个字符串表包括 2个字节的字符串，该字符串表的长度编码是 2）。如果长度不等于零，mb_u_int32之后紧跟着1个或多个字符串。根据传输元信息的规定，当前的“ charset ”之后就是字符串的编码。

有许多记号用于对字符串表内容的引用进行编码。对引用进行编码相当于对字节的偏移量进行编码，偏移量是字符串表中本字节相对于第一个字符串的第一个字节的位置。例如，第一个字符串的偏移量是0。

5.2.7 记号结构

记号被划入一组相互重叠的代码空间，一个特殊记号的意义由使用的上下文确定，记号分为两类，以如下方式构成：全局记号（ global token）和应用记号（application token）。

在所有的上下文中，全局记号被分配了一组固定的代码，任何情况下只有唯一的含义。全局代码用于对内联数据(也就是，字符串、实体、非透明数据等)和各式各样的控制函数进行编码。

应用记号的意义依赖于上下文，被分成两个相互重叠的代码空间：标签代码空间（ tag code space）和属性代码空间（attribute code space)。一个给定的记号值(如，0x99)可以有不同的含义，取决于它是标签代码空间的记号还是属性代码空间中的记号。

标签代码空间用于表示特定的标签名称。每个标签记号是一个单字节代码，表示一个特定的标签名(如，CARD)。

属性代码空间又分成两个数字取值范围，分别表示属性前缀和属性值。

一个代码空间又进一步被分成256个代码页，代码页允许知名代码进一步地扩展。一个单独的记号(如SWITCH_PAGE)可以在代码页之间交换。

标签和属性代码的定义与特定的文档类型有关。全局代码有两部分，一部分是通用代码集，为全部文档类型所共有；另一部分是保留代码集，用于特殊文档类型的扩展。

1. 解析状态机

标记格式有两种状态，每一种状态都对应着一个代码空间。根据5.2.3节描述的语法，各种状态可以相互转换。代码空间用下面的方式与解析状态相关连 (见表5-2)：

在一个给定的状态中，当一个代码页转换记号 (SWITCH_PAGE)出现时，就改变给定状态的代码页。这个新的代码页一直保留作为当前代码页，直到同一状态的另一个 SWICH_PAGE 出现，或者是文档结束。每一种解析状态分别保持一个“ 当前代码页 ”，两个解析状态的初始代码页都是0。

表5-2 解析状态

语法解析状态	代码空间
stag	标签
attribute	属性

图5-1给出了状态转换的另一种表示方式，用作参考模型。

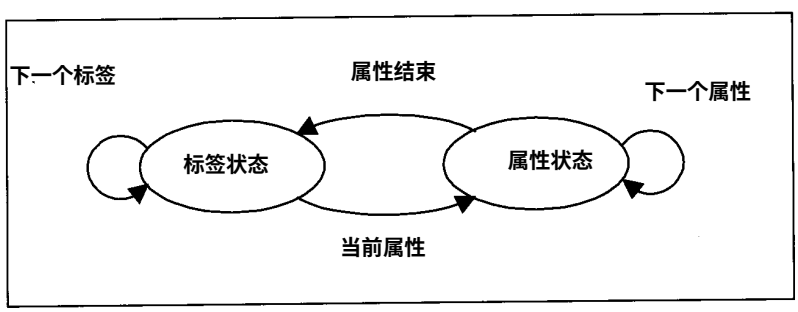

图5-1 状态转移的另一种表示方式

2. 标签代码空间

标签记号是一个u_int8类型，其结构如表5-3所示：

表5-3 标签格式

Bit位	描 述
7(最高位)	指出标签代码后面是否跟有属性。如果这个比特是 0，标签不包含属性；如果是1，说明在标签之后，有一个或多个属性。属性列表以 END记号结束
6	指出这个标签是否以一个包含内容的元素开始。如果这个比特是 0，说明该标签不包含内容；如果是 1，这个标签之后是其所包含的内容，标签以 END记号结束
5-0	标签的标识

例如：

- 标签值0xC6：代表标签六（6），既有属性又有内容，内容跟在标签之后，也就是

```
<Tag arg="1">foo</Tag>
```

- 标签值0x46：代表标签六（6），内容跟在开始标签之后。这个元素不包括属性，如

```
<Tag>test</Tag>
```

- 标签值0x06：代表标签六（6），这个元素即没有属性也没有内容，如

```
</Tag>
```

全局性的特殊代码 LIRERAL(见5.2.7节中“ 文字标签或属性名 ”)表示未知的标签名。在使用更紧凑的格式时，XML标记者应当避免使用LIRERAL或者一个标签的字符串表示。

既包含属性又包含内容的标签总是先对属性编码，然后再对内容编码。

3. 属性代码空间(属性开始和属性值)

属性记号用单个 u_int8类型进行编码。属性代码空间又被划分成两个取值范围 (除了可用于所有代码空间中的取值之外):

属性开始 取值小于128的记号，表示一个属性的开始。属性开始记号可以确切地标识出属性的名(也就是，ULR=)，还可以表示属性值的开始 (如PUBLIC=“ TRUE ”)。未知的属性名使用全局性的特定代码 LITERAL(见2.7.4.4节)进行编码，LITERAL不能用于属性值的编码。

属性值 取值大于或等于128的记号表示一个属性值中出现的知名字符串，这些记号仅能表示属性值。未知的属性值用字符串、实体或者扩展代码进行编码 (见 5.2.7节中“ 全局记号 ”)。

所有被标记的属性都必须以一个属性开始记号开始，其后可以跟着零个或多个属性值、字符串、实体及扩展记号。新的属性开始记号、 LITERAL记号或者 END记号都表示一个属性值已经结束，这样可以使包含知名子串和实体的字符串能够实现紧凑的编码。

例如，如果属性开始记号 TOKEN_URL表示属性名“ URL ”，属性值记号TOKEN_COM代表字符串“ .com ”，而属性值记号 TOKEN_HTTP代表字符串“ http:// ”，那么属性 URL=“ http://foo.com/x ”可以用下面的序列进行编码：

```
TOKEN_URL TOKEN_HTTP STR_I "foo" TOKEN_COM STR_I "/x"
```

另一个例子是，如果属性开始记号 TOKEN_PUBLIC_TRUE表示属性名“ PUBLIC ”和值前缀“ TRUE ”，那么属性PUBLIC=“ TRUE ”可以用下面的序列进行编码：

```
TOKEN_PUBLIC_TRUE
```

当使用更简洁的格式时， XML的标记者应该避免使用 LITERAL和表示属性名的字符串，

同时也要避免使用表示属性值的字符串。

4. 全局记号

在代码空间和所有代码页中，全局记号的意义和结构是一样的。全局记号的类型有：

字符串型（Strings）内联字符串引用和字符串表引用

扩展型（Extension）特定的文档类型扩展记号

非透明型（Opaque）内联的非透明数据

实体型（Entity）字符实体

处理指令（Processing Instruction）XML 的处理指令

文字型（Literal）未知的标签或属性名称

控制代码（Control codes）各种全局控制代码

(1) 字符串型记号

```
string      =inline | tableref
inline      =STR_I termstr
tableref    =STR_T index
```

字符串对内联的字符数据或字符串表中的引用进行编码。字符串表是单个字符串的级连，字符串的结束与对字符文档的编码有关，可能没有 NULL结束符。字符串的引用是在串表中的偏移量，用来指出该字符串被引用的位置。

内联的字符串的引用格式如下：

STR_I	…char data…

字符串表的引用格式如下：

STR_T	mb_u_int32

字符串表的偏移量从字符串表中第一个字符串的第一个字节 (也就是没有偏移量的字节)算起。

(2) 全局扩展型记号

```
extension =( EXT_I termstr ) | ( EXT_T index ) | EXT
```

全局扩展记号用于特定的文档编码。在特殊的文档类型上下文中，只定义了这些记号的语义，但对所有文档中使用的格式作了严格的定义。有三类全局扩展记号：单字节扩展记号、内联字符串扩展记号和内联整数扩展记号。

内联字符串扩展记号(EXT_I*)格式如下：

EXT_I*	…char data …

内联整数扩展记号(EXT_T*)格式如下：

EXT_T*	mb_u_int32

单字节扩展记号(EXT*)格式如下：

EXT*

(3) 字符串实体

```
entity      = ENTITY entcode
entcode     = mb_u_int32        // UCS-4字符代码
```

字符实体记号(ENTITY)用于对数字字符实体进行编码，这里的数字字符实体与 XML的数字字符实体(如&#32)有相同的语义。 mb_u_int32引用UCS-4字符集中的字符。在 XML源文档中，所有的实体必须用单个字符串记号 (即STR_I)或者ENTITY记号来表示。

字符实体的格式如下：

ENTITY	mb_u_int32

(4) 处理指令

```
pi = PI attrStart *attrValue END
```

处理指令(PI)记号用于对一个 XML处理指令进行编码， PI编码与 XMLPI有相同的语义。AttrStart用于PITarget的编码，attrValue用于PI可选值的编码。关于处理指令更详细的信息，参阅[XML]。

PI标签的格式为：

PI	attrStart	attrValue	END

没有取值的PI编码为：

PI	attrStart	END

(5) 文字标签或属性名

文字记号用于对一个标签或未知的记号属性名进行编码，记号的实际意义（如标签与属性名）由记号的解析状态决定，所有的文字记号都用于表示字符串表中的引用，这些引用包含实际的名称。

LITERAL标签格式是：

LITERAL	mb_u_int32

(6) 各种控制代码

- END记号　END记号用于结束属性列表和元素。 END是单字节记号。
- 代码页转换记号　代码页转换记号(SWITCH_PAGE)表示一种转换，以使当前代码页与当前的记号状态相适应。代码页转换使用 2个字节的编码序列。

SWITCH	u_int8

(7) 保留记号

有几种保留的全局记号，标记者不传送这些记号，用户代理将它们看成是一个单字节记号。

代码页255留给特定的实现或实验使用，这个代码页中的记号不用于表示标准XML文档结构。

5.3 编码语义

5.3.1 文档标记

标记XML文档的过程就是将所有的标记和 XML句子结构(如实体、标签和属性等等)转换成对应的标记格式的过程。所有的注释应该被删除，标记者加入的处理提示也可能被删除。

其他的元信息，如文档类型定义和不是必须的附加内容条件也应该被删除。必须把所有的文本和实体转换成字符串 (如，STR_I)或实体(ENTITY)记号，即使目标字符的编码能够表示这个实体，在标记时，原文本表示 (如，&amp)中的各种实体也必须转换成字符串格式。对于文本格式中出现的各种字符，当它们不能用目标字符的编码表示时，可以使用 ENTITY记号进行编码。必须把属性名转换成一个属性开始记号，或者用单个 LITERAL 记号来表示。属性值不能用一个LITERAL 记号表示。

将标记结构编码变成字符串是不合法的，用户代理把所有的文本记号 (如，STR_I 和 ENTITY)看成是CDATA (即没有嵌入标记的文字)。

5.3.2 文档结构的顺从性

经过标记的XML文档必须能够精确地表示原文本文档的结构和语义，这意味着原文本文档必须有很好的格式，如 [XML]中规定的那样。[XML]中规定，标记文档时可以对文档的有效性进行检验，但这不是必须的。如果一个特殊的文档类型（ DOCTYPE）的语义已知，附加的语义验证可以在标记过程中完成。

5.3.3 默认属性值的编码

XML 文档的标记表示可能省略了隐含在 DTD中的所有属性，或者只使用属性的默认值。这意味着对于一个用户代理的具体实现，必须了解给定 DTD版本的属性默认值，这个信息可以从用数据格式标记的版本号中得到。

5.4 数字常量

5.4.1 全局记号

表5-4中给出的各种记号的代码，可以出现在所有的代码空间和代码页中，为全部的文档类型所共有，其中数字都是用十六进制表示的。

表5-4 全局记号

记 号 名	记 号	描 述
SWITCH_PAGE	0	改变当前记号状态的代码页，其后跟着一个 u_int8类型的数据，用于指出新代码页的页号
END	1	表示一个属性列表或一个元素的结束
ENTITY	2	一个字符实体，其后跟着一个 mb_u_int32类型，用于对这个字符实体的编号进行编码
STR_I	3	内联字符串，其后跟着字符串
LITERAL	4	一个未知的标签或属性名，其后跟着一个 mb_u_int32类型，用于对在字符串表中的偏移量进行编码
EXT_I_0	40	内联字符串特殊类型文档扩展记号，其后跟着字符串
EXT_I_1	41	内联字符串特殊类型文档扩展记号，其后跟着字符串
EXT_I_2	42	内联字符串特殊类型文档扩展记号，其后跟着字符串
PI	43	处理指令

（续）

记 号 名	记 号	描 述
LITERAL_C	44	带有内容的未知标签
EXT_T_0	80	内联整数特殊类型文档扩展记号，其后跟着一个 mb_u_int32 类型的数据
EXT_T_1	81	内联整数特殊类型文档扩展记号，其后跟着一个 mb_u_int32 类型的数据
EXT_T_2	82	内联整数特殊类型文档扩展记号，其后跟着一个 mb_u_int32 类型的数据
STR_T	83	字符串表引用，其后跟着一个 mb_u_int32类型，用于对字节偏移量进行编码，这个偏移量从字符串表的起点算起
LITERAL_A	84	带有属性的未知标签
EXT_0	C0	单字节特殊类型文档扩展记号
EXT_1	C1	单字节特殊类型文档扩展记号
EXT_2	C2	单字节特殊类型文档扩展记号
RESERVED_2	C3	为将来使用保留
LITERAL_AC	C4	既有内容又有属性的未知标签

5.4.2 公共标识符

列在表5-5中的值，是知名文档类型的公共标识符。开始的 128个值被保留，以用于将来的WAP规范。所有的数字都是以十六进制格式的表示。

表5-5 公共标识符

取 值	公共标识符
0	后面跟着字符表索引；在字符串表中公共标识用一个文字进行编码
1	未知或者缺少标识符
2	“-//WAPFORUM//DTD WML 1.0//EN”(WML 1.0)
3	“-//WAPFORUM//DTD WTA 1.0//EN”(WTA Event 1.0)
4-7F	保留

5.5 编码示例

5.5.1 一个简单的XML文档

下面是一个简单的经过标记的 XML文档实例，它给出了基本元素、字符串和实体的编码。原文本文档为：

```
<?xml version="1.0"?>
<!DOCTYPE XYZ [
<!ELEMENT XYZ (CARD)+>
<!ELEMENT CARD (#PCDATA | BR)*>
<!ELEMENT BR EMPTY>
<!ENTITY nbsp " ">
]>
<XYZ>
  <CARD>
```

```
      X &amp; Y<BR/>
      X = 1
    </CARD>
</XYZ>
```

下面是标签代码空间中定义的记号：

```
TAG NAME  TOKEN
BR        5
CARD      6
XYZ       7
```

下面这个例子只使用了内联字符串，并假设字符编码使用了 NULL结束字符串格式，同时还假设传输字符编码集是 US-ASCII。这个编码不能支持页面（ deck）中的某些字符(如，)，从而必须使用ENTITY记号。标记格式(数字用十六进制表示)如下：

```
00   01   00   47   46   03   ' '   'X'   ' '   '&'   ' '   'Y'   00   05   03   ' '
'X'  00   02   81   20   03   '='   00   02   81   20   03   '1'   ' '  00   01   01
```

更多的内容和注释，参见表5-6。

表5-6 标记过的页面（Deck）

记 号 串	描 述
00	版本号，即表示WBXML 版本1.0
01	未知公共标识符
00	字符串长度
47	XYZ，带有内容
46	CARD，带有内容
03	后面跟着内联字符串
' ', 'X', ' ', ' & ', ' ', 'Y ', 00	字符串
05	BR
03	后面跟着内联字符串
' ', 'X', 00	字符串
02	ENTITY（实体）
81 20	实体值(0x160)
03	后面跟着内联字符串
'= ',00	字符串
02	ENTITY（实体）
81 20	实体值(0x160)
03	后面跟着内联字符串
'1', ' ',00	字符串
01	END(CARD元素的结束)
01	END(XYZ元素的结束)

5.5.2 一个扩展的示例

下面是另一个经过标记的 XML文档，它示范了属性编码和字符串表的使用方法。原文档如下：

```
<?xml version="1.0"?>
<!DOCTYPE XYZ [
<!ELEMENT XYZ ( CARD )+ >
```

```
<!ELEMENT CARD (#PCDATA | INPUT | DO)*>
<!ATTLIST CARD NAME NMTOKEN #IMPLIED>
<!ATTLIST CARD STYLE (LIST|SET) 'LIST'>
<!ELEMENT DO EMPTY>
<!ATTLIST DO TYPE CDATA #REQUIRED>
<!ATTLIST DO URL CDATA #IMPLIED>
<!ELEMENT INPUT EMPTY>
<!ATTLIST INPUT TYPE (TEXT|PASSWORD)'TEXT'>
<!ATTLIST INPUT KEY NMTOKEN #IMPLIED>
<!ENTITY nbsp " ">
]>
<!-This is a comment ->
<XYZ>
   <CARD NAME="abc" STYLE="LIST">
    <DO TYPE="ACCEPT"URL="Http://xyz.org/"s/>
    \Enter name: <INPUT TYPE="TEXT"KEY="N/>
   </CARD>
</XYZ>
```

标签代码空间的记号定义如下：

标签名称	记号
CARD	5
INPUT	6
XYZ	7
DO	8

属性开始记号定义如下：

属性名称	属性值前缀	记号
STYLE	LIST	5
TYPE		6
TYPE	TEXT	7
URL	http://	8
NAME		9
KEY		B

属性值记号定义如下：

属性值	记号
.org	85
ACCEPT	86

经过标记的格式 (数字用十六进制格式表示)如下，该例子假设使用了 UTF-8**字符编码和** NULL**结束字符串。**

```
00  01  12  'a'  'b'  'c'  00  ' '  'E'  'n'  't'  'e'  'r'  ''  'n'
'a' 'm' 'e' ':'  ' '  00   47  C5   09   03   00   05   01   88  06
86  08  03  'x' 'y'  'z'  00  85   03   '/'  's'  00   01   83  04
01  83  04  86  07   0A   03  'N'  00   01   01   01
```

对于更详细的内容和注释形式，见表 5-7**。**

表5-7 经过标记的Deck的例子

记 号	描 述
00	版本号，即表示WBXML版本1.0
01	未知的标识符
12	字符串表长度
‘a’, ‘b’, ‘c’, 00, ‘ ’, ‘E’, ‘n’, ‘t’, ‘e’, ‘r’, ‘ ‘, ‘n’, ‘a’, ‘m’, ‘e’, ‘:’, ‘ ‘, 00	字符串表
47	XYZ，带有内容
C5	CARD，带有内容和属性
09	NAME=
83	后面跟着字符串表引用
00	字符串表索引
05	STYLE="LIST"
01	END（CARD属性列表的结束）
88	DO，带有属性
06	TYPE=
86	ACCEPT
08	URL="http://"
03	后面跟着内联字符串
‘x’, ‘y’, ‘z’, 00	字符串
85	“.org”
03	后面跟着内联字符串
‘/’, ‘s’, 00	字符串
01	END(DO属性列表的结束)
83	后面跟着字符串表引用
04	字符串表索引
86	INPUT，带有属性
07	TYPE="TEXT"
0A	KEY=
03	后面跟着内联字符串
‘N’, 00	字符串
01	END(INPUT属性列表的结束)
01	END(CARD元素的结束)
01	END(XYZ元素的结束)

5.6 术语定义

在整个规范中，我们使用了下列术语和习惯用法。

作者（Author） 作者是一个人或是一个应用程序，它编写或生成了无线标记语言 WML、无线标记语言脚本WMLScript或其他的内容。

内容（Content） 内容是源服务器生成或存储的数据（或事件）。典型的是，在响应用户请求时，内容由用户代理显示或解释

资源（Resource） 它是一个可以被URL识别的网络数据对象或服务，可以用多种表述格式所表达（例如，多种语言、数据格式、数据块尺寸和分辨率）或以其他方式进行变化。

标准通用标记语言（SGML） 标准通用标记语言（定义在 [ISO8879]中）是一种通用语

言，专门用在标记语言领域中。

用户（User） 用户是一个通过用户代理观看、聆听或使用资源的人。

用户代理（User Agent） 用户代理是对无线标记语言 WML、无线标记语言脚本（WMLScript）、无线电话应用接口（WTAI）或其他资源进行解释的软件或设备，它包括文本浏览器、语音浏览器和搜索引擎等。

可扩展标记语言（XML） 可扩展标记语言是一个万维网联盟 (W3C)的标记语言标准，WML是其中的一种。XML是SGML的一个有限子集。

5.7 缩略语

该规范使用了下列缩略语：

API	Application Programming Interface	应用编程接口
BNF	Backus-Naur Form	Backus-Naur窗体标识语法
LSB	Least significant Bits	最低有效位
MSB	Most Significant Bits	最高有效位
MSC	Mobile Switch Center	移动交换中心
RFC	Request For Comments	请求注释
SGML	Standardized Generalized Markup Language[ISO8879]	标准通用标记语言[ISO8879]
UCS-4	Universal Character Set—4byte[ISO10646]	4字节通用字符集[ISO10646]
URL	Universal Resource Locator	统一资源定位器
UTF-8	UCS Transformation Format 8 [ISO10646]	UCS传输格式[ISO10646]
W3C	World Wide Web Consortium	万维网联盟
WAP	Wireless Application Protocol[WAP]	无线应用协议[WAP]
WML	Wireless Markup Language[WML]	无线标记语言[WML]
XML	Extensible Markup Language[XML]	可扩展标记语言[XML]

5.8 参考标准

[ISO10646] "Information Technology—Universal Multiple-Octet Coded Character Set (UCS)—Part 1: Architecture and Basic Multilingual Plane", ISO/IEC 10646-1:1993

[RFC822] "Standard for the Format of ARPA Internet Text Messages",STD 11, RFC 822, D. Crocker, August 1982
URL: ftp://ds.internic.net/rfc/rfc822.txt

[RFC2119] "Key words for Use in RFCs to Indicate Requirement Levels",S. Bradner, March 1997
URL: ftp://ds.internic.net/rfc/rfc2119.txt

[WAP] "Wireless Application Protocol Architecture Specification",WAP Forum, 30-April-1998
URL: http://www.wapforum.org/

[XML] "Extensible Markup Language (XML), W3C Proposed Recommendation 10-February-1998, REC-xml-19980210", T. Bray, et al., February 10, 1998
URL: http://www.w3.org/TR/REC-xml

5.9 参考资料

[ISO8879] "Information Processing—Text and Office Systems—Standard Generalized Markup Language (SGML)"，ISO 8879:1986

[UNICODE] "The Unicode Standard: Version 2.0",The Unicode Consortium, Addison-Wesley Developers Press, 1996
URL: http://www.unicode.org/

[WML] "Wireless Markup Language",WAP Forum, 30-April-1998
URL: http://www.wapforum.org/