第4章　无线标记语言规范

4.1　范围

无线应用协议（Wireless Application Protocol，WAP）是无线通信网络发展应用的必然结果，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。为了给运营商和生产商提供一个面对先进业务、多种类业务和快速灵活业务生成的商机， WAP定义了一系列用于传输层、安全层、事务处理层、会话层和应用层的协议。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]。

这个标准定义了无线标记语言（Wireless Markup Language，WML）。WML是基于[XML]的标记语言，用来定义窄带设备中用到的内容和用户接口，这些窄带设备包括蜂窝电话和寻呼等。

WML的设计受到小型的窄带设备的限制，这些限制是：

- 小型显示屏幕和受限的用户输入设备。
- 窄带网络连接。
- 有限的内存和计算的资源。

WML包括四个主要的功能域：

- 文本显示和布局　WML支持文本和图像，包括各种格式和排版命令。例如，可以定义粗体字。
- 页面/卡片有组织的比喻　所有WML的信息都被组织在一系列的卡片和页面内，卡片指定一个或更多的用户交互单元（例如，菜单选择、文本屏或文本输入域）。逻辑上，用户可在一系列的WML卡片中导航，检查每项内容、输入要求的信息、作出选择或移动到另一张卡片上。

把多张卡片组合在一起可构成页面， WML页面类似于一个由URL标识的HTML页面，是内容传输的单位。

- 卡片间的导航和链接　WML不仅支持卡片和页面之间的导航管理，而且还支持设备中的事件处理，这些处理可能用于导航或程序脚本的执行。同时它还支持锚点的链接，类似的情况在[HTML4]中可以见到。
- 字符串参数化和状态管理　借助于状态模型，所有的WML页面都可以实现参数化，这种参数化有助于提高网络资源的使用效率。变量可以在字符串的位置上使用并且在运行时被替代。

设备类型

WML的设计目的是为了满足数量众多的小型窄带设备的需要。这些设备主要有四个方面的特点：

- 显示尺寸　较小的屏幕尺寸和清晰度。像电话一类小巧的移动设备只能显示几行文本，每行包含8-12个字符。
- 输入设备　受到限制的或为了特殊目的的输入设备。典型的电话有一个数字键盘和几个额外的特殊功能键，一些更复杂的设备可能有软件可编程的按钮，但没有鼠标或其他点击设备。
- 计算资源　低功率的CPU和较小的内存，通常受功率的限制。
- 窄带网络连接　较窄的带宽和较长的延迟。有300bps至10kbps的网络连接和5秒至10秒往返延迟的设备还不普遍。

本文使用下列术语来定义设备功能的广泛性：

电话　典型的显示尺寸是2行至10行，输入通常由数字键和一些额外的功能键实现。特别是与用途广泛的计算机设备相比，它的计算资源和网络吞吐量是有限的。

PDA　个人数字助理是一种有较广泛功能的设备，在本文中使用时，这个术语特指带有额外显示和输入特征的设备。它的显示分辨率通常在160×100像素范围内，并且设备可能支持点击设备、手写识别和其他的许多先进的功能。

这些术语是为了定义一些非常概括性的描述原则，并用来阐明本文中给定的例子。

4.2 WML和URL

万维网是一个信息和设备的网络。万维网规范的三个领域保证了其普遍的互操作性。

- 统一的命名模式命名由统一资源定位器（URL）实现，它为命名任何网络资源提供了标准的方式，参见[RFC1738]。
- 传输信息的标准协议（如HTTP）。
- 标准的内容类型（如HTML、WML）。

WML采用与HTML和万维网相同的参考结构。其内容用URL命名，并用与HTTP有相同语义的标准协议传输（如[WSP]）。URL在[RFC1738]中定义，表示URL的字符集也在[RFC1738]中定义。

在WML中，URL用在下面的情况中：

- 当指定导航时（如超链接）。
- 当指定外部资源时（如一个图像或一个脚本）。

4.2.1 URL方案

WML浏览器必须实现[WAE]中规定的URL方案。

4.2.2 字段锚

对于一个资源的命名定位，WML事实上采用了HTML的标准。它的字段锚由URL文件定义，后面紧随着井号（#），再后面是字段标识符。在一个WML页面中，WML借助字段锚识别独立的卡片，如果没有指定字段，URL则为整个页面。在某些情况下，页面URL的默认的是页面中的第一张卡片。

4.2.3 相对URL

WML使用在[RFC1808]中定义的相对的URL。[RFC1808]规定了在WML页面的上下文中

解析相对URL的方法。一个WML页面的基本URL是标识这个页面的URL。

4.3 WML字符集

WML是一种XML语言并且继承了XML文档的字符集。在SGML命名法中，一个文档字符集是该文档类型包含的所有逻辑字符（例如，字母‘ T ’和识别这个字母的固定整数）。一个SGML或XML文档就是一系列这样的整数记号，这些记号放在一起构成了一个文档。

XML和WML的文档字符集是ISO/IEC-1046（[ISO10646]）的通用字符集，目前，这个字符集与Unicode 2.0[UNICODE]保持一致。WML将对[XML]和[ISO10646] 规范做进一步的修改和增强。在本文中，术语ISO10646和Unicode可以互换，指的是同一个文档字符集。

WML页面不要求使用完整的Unicode编码（如UCS-4）。Unicode字符集的任何一个恰当的子集（如，US-ASCII、ISO-8859-1等）都可以使用，未使用 UTF-8或UTF-16编码的文档必须像在XML中一样声明其编码方式。

4.3.1 参考处理模型

WML的参考处理模型如下。用户代理必须实现这个处理模型，或者实现一个与之类似的模型。

- 用户代理在处理文档之前，必须用某种方法将文档的外部字符编码正确地映射成Unicode编码。
- 任何实体的处理都在这个文挡的字符集中进行。

一个给定的具体实现可以选择一种方便的内部表示方法。

4.3.2 字符实体

WML支持命名的和数字的字符实体，参考处理模型的一个重要结果就是所有数字的字符实体都参考文档字符集（Unicode）而不参考当前的文档编码（charset）。

这意味着&#302总是引用同样的逻辑字符，这些逻辑字符与当前的字符编码无关。

WML支持下列字符实体格式：

- 已命名的字符实体，例如&amp和&lt。
- 十进制的数字字符实体，例如 &#32。
- 十六进制的数字字符实体，例如 &#x20。

在处理WML时，有七个命名的字符实体特别的重要：

```
<! ENTITY quot  "&#34;">       <!- quotation mark ->
<! ENTITY amp  "&#38;#38;">    <!- ampersand ->
<! ENTITY apos  "&#39;">        <!-apostrophe ->
<! ENTITY lt     "&#38;#69">     <!- less than ->
<! ENTITY gt    "&#62;">        <!- greater than->
<! ENTITY nbps   " ">      <!-non-breaking space ->
<! ENTITY shy    "&#173;">      <!- soft hyphen (discretionary hyphen) ->
```

4.4 WML语法

WML从XML继承了许多它的语法的结构。关于语法问题深入的信息请参考 [XML]。

4.4.1 实体

WML文本可以包含数字或被命名的字符实体，这些实体在文档字符集中定义了一些特定的字符，用于在文档字符集中规定字符。此文档字符集在 WML中必须被转义，否则很难进入文本编辑器，例如，&号用命名的实体&amp代替。所有的实体由&开头，由分号结尾。

WML是一种XML语言，这意味着 &号和较少的字符以文本数据形式使用时一定被转义（例如，这些字符在注解中只作为标记分隔符使用时，才会显示成文字的形式）。有关更多的信息，请参阅[XML]。

4.4.2 元素

元素规定了有关 WML页面的所有标记和结构信息，它包含一个开始标签、内容和一个结束标签。元素可以由下面的两种结构定义：

```
<tag> content </tag>
```

或

```
<tag/>
```

元素所包含的内容位于开始标签（ <tag>）和结束标签（ </tag>）之间。空元素标签（<tag/>）定义的是没有内容的元素。

4.4.3 属性

WML属性规定了关于元素的指定附加信息。确切的讲，属性定义了关于元素的非内容部分的信息，它总在元素的开始标签中规定。例如：

```
<tag attr="abcd"/>
```

属性名是一个XML名（NAME）并且区分大小写。

XML要求所有的属性值用双引号（“ ）或单引号（‘ ）表示。当属性值由双引号表示时，单引号可以包含在属性值内，反之亦然。字符实体可包含在一个属性值内。

4.4.4 注解

WML注解服从XML的注解式样并有如下的语法：

```
<!- a comment ->
```

WML作者可以使用注解，客户代理不显示注释， WML注解不能嵌套。

4.4.5 变量

WML卡片和页面可用变量实现参数化。当卡片或页面用一个变量替代时，使用下列语法：

```
$identifier
$(identifier)
$(identifier:conversion)
```

如果变量结束没有空白空间，就需要使用圆括号。变量语法在 WML中有最高的优先权；任何变量语法合法的地方，符号‘ $ ’代表一个变量替换。在所有的 PCDATA和由vdata实体类

型定义的属性值中，索引都是合法的。（见5.3节）。

两个“ $$ ”代表一个美圆符号字符“ $ ”。

有关变量的语法和语义的详细信息见4.7.3节。

4.4.6 区分大小写

XML是一种区分大小写的语言， WML继承了这种特性。当分析WML页面时不使用字符交迭，这意味着所有的WML标签和属性都是区分大小写的。此外，任何枚举的属性值都区分大小写。

4.4.7 CDATA部分

CDATA用于转义文本块，而且在PCDATA中也是合法的。例如，在一个元素中， CDATA部分用字符串“ <! [CDATA[”表示开始,而用“]]> ”表示结尾。例如：

```
<! [CDATA[ this is <B> a test ]]>
```

任何在CDATA中的文本都作为正文对待，并且不用于标记的解析。 CDATA在任何使用文字文本地方都是有用的。

关于CDATA部分的更多信息，请参阅[XML]规范。

4.4.8 处理指令

WML不会使用超出XML规范明确定义的XML处理指令。

4.4.9 错误

[XML]规范定义了具有良好格式的XML文档的概念，未按这种文档格式定义的WML页面都是错误的。有关信息，见4.11.2节中“ 文档的有效性 ”。

4.5 核心WML数据类型

4.5.1 字符数据

WML的所有字符数据（Character Data）都是根据XML数据类型定义的。总结如下：

CDATA 包含数字字符实体或命名字符实体的正文，它只在属性值中使用。

PCDATA 包含数字字符实体或命名字符实体的正文，该正文可以包含标签（ PCDATA是“ 解析的 CDATA ”）。PCDATA只在元素中使用。

NMTOKEN 一个名字记号，与XML规范定义一样，它可以包含任何名字字符的混合。

更多的信息，见[XML]。

4.5.2 长度

```
<! ENTITY % length "CDATA">  <! - [0-9] + for pixels or [0-9]+ "%" for
percentage length->
```

长度(Length)类型可以表示画布（屏幕、纸）的像素数目，可作为水平或垂直空间的一个百分数。例如，数值“ 50 ”表示50个像素。对于宽度而言，数值“ 50% ”表示可用的水平空

间（在一个画布两条边缘之间的空间）的一半。对于高度而言，数值“ 50% ”表示用的垂直空间（在当前的窗口或画布中）的一半。

整数值包含一位或多位的十进制数字（ 0～9），其后跟着一个可选的百分号（ %）。长度类型只在属性值中使用。

4.5.3 Vdata

```
<! ENTITY% vdata "CDATA">  <!- attributevalue possiblycontaining
variable references ->
```

vdata类型代表一个包含变量索引的字符串（见 4.7.3节），这种类型只在属性值中使用。

4.5.4 流和内联

```
<! ENTITY %  layout  "BR">
<! ENTITY %  inline  "%text;  | %layout;">
<! ENTITY %  flow    "%inline;  |  IMG  |  A">
```

流（Flow）类型代表“ 卡级 ”的信息。内联（Inline）类型代表“ 文本级 ”的信息。通常，流类型可在任何地方使用，普通的标记可以包含在内，内联类型表明了只处理纯文本或变量引用的区域。

4.5.5 URL

```
<! ENTITY %  URL    "%vdata;">     <! - URL or URN表示超文本节点，可以表示变量的引用 ->
```

URL类型指一种相对或绝对的统一资源定位器 [RFC1738]。更多信息，见第4.2节。

4.5.6 布尔型

```
<! ENTITY % boolle@n  "(TRUE | FALSE)">
```

布尔型（Boolean）的逻辑值为TRUE或FALSE。

4.5.7 数字类型

```
<! ENTITY % number  "NMTOKEN">        <+个数字，从 0到9]+ ->
```

数字类型（Number）表示一个大于或等于零的整数值。

4.6 事件和导航

4.6.1 导航和事件处理

WML包括导航和事件处理模型。有关元素允许作者定义用户代理进行的处理，事件可能由作者绑定在任务上：当一个事件发生时，绑定的任务被执行。可以指定多种任务，例如导航到作者规定的URL。事件绑定由一些元素声明，包括 DO和ONEVENT。

4.6.2 历史

WML包括一个简单的导航历史记录模型，以允许作者用一种方便和有效的方式管理逆向

导航。用户代理历史记录是一个 URL的堆栈，表示在用户到达当前卡片之前，经历过的导航路径。对历史堆栈执行三种操作：

- Reset 历史堆栈可以复位到只包含当前卡片的状态。更多的信息，见 4.7.2NEWCONT-EXT属性节。
- Push 一个新的URL被压到历史记录堆栈上，作为导航至一个新卡片结果。
- Pop 当前卡片的URL（堆栈顶）被弹出，作为一个逆向导航的结果。

用户代理必须实现导航历史记录。当每个卡片通过一个明确定义的 URL（例如，GO中一个URL属性）被访问时，卡片的 URL被加到历史记录堆栈中。在历史记录中，用户代理必须提供一个让用户能逆向导航回到前一个卡片的方法，作者可以利用允许用户在历史记录中逆向导航的已有的用户接口结构。当执行一个 PREV任务时，用户代理必须将用户逆向导航至历史记录中的前一个卡片（见 6.3节），执行PREV将当前卡片的 URL从历史堆栈中弹出。有关PREV更多的语义信息，请参阅4.9.5节中“ PREV任务 ”。

4.6.3 VAR元素

```
<! ELEMENT VAR EMPTY>
<! ATTLIST VAR
NAME        %vdata  #REQUIRED
VALUE       %vdata  #REQUIRED
>
```

作为执行一个任务的影响， VAR元素规定了变量在当前浏览器上下文中的设置。如果NAME属性在运行时没有作为一个合法的变量名，则必须忽略元素（见4.7.3节）。关于设置变量的更多细节见4.7.3节中“ 变量赋值 ”。

属性

NAME =*vdata*

NAME属性指定变量名。

VALUE =*vdata*

VALUE属性指定分配给变量的值。

4.6.4 任务

```
<! ENTITY  % task  "GO | PREV | NOOP | REFRESH">
```

任务规定了响应事件的处理过程。它被绑定到 DO、ONEVENT和A元素的事件上。

1. GO元素

```
<!ELEMENT GO (VAR)*>
<!ATTLIST GO
  URL               %URL;           #REQUIRED
  SENDREFERER       %boolean;       "FALSE"
  METHOD            (POST|GET)      "GET"
  ACCEPT-CHARSET    CDATA           #IMPLIED
  POSTDATA          %vdata;         #IMPLIED
  >
```

GO元素声明一个GO任务，表示导航至一个URL，如果这个URL命名了一个WML卡片或页面，它将被显示。GO对历史记录堆栈执行一个“ push ”操作（见4.6.2节）。

有关GO语义更多的信息，参见4.9.5节中“ GO任务 ”。

属性

URL= *URL*

URL属性指定目的地URL（例如，要显示卡片的URL）。

SENDREFERER= *boolean*

如果这个属性是TRUE，为了使服务方便起见，用户代理必须指定包含这个任务的页面的URL（也就是引用的页面）。这使一个服务器可以基于与ULR连接的页面，完成对URL的接入控制。如果URL是相对的话，那必须尽可能是最小的相对的URL。例如，如SENDREFERER=TRUE，一个基于HTTP的用户代理将在HTTP“ Referer ”的请求报头中指示当前页面的URL [RFC2068]。

METHOD = *(POST | GET)*

这个属性指定HTTP的提交方式。目前，GET和POST值可以接受并且引起用户代理独立地执行一个HTTP的GET或POST，如果没有指定METHOD，用户代理必须使用GET方式。如果存在POSTDATA属性，用户代理必须使用POST方式。

ACCEPT-CHARSET= *cdata*

这个属性规定了用于数据的字符编码的列表，这些数据是源服务器在处理输入时必须接受的。这些属性值是用逗号或是空格分开的字符编码名称（ charset）的列表，字符编码在[RFC2045]和[RFC2068]中给出了定义。IANA字符集注册表定义了关于字符集取值的公共注册表，这个列表是一个“ 异或 ”逻辑列表，换句话说，服务器必须接受所有可接受的字符编码中的一种。

这个属性的默认值是保留字符串UNKNOWN。用户代理必须解释这个值，以作为一种字符编码，这个字符编码在传输包含该属性的WML页面时使用。

POSTDATA= *vdata*

这个属性规定了传递给服务器的数据，这个数据作为一个 application/x-www-form-urlencoded实体传送给服务器，并使用在[RFC738]中定义的URL转义机制，它们被格式变成一串由八位组构成的字节流。

特别地，当下列情况发生时：

- 用户代理应当将输入数据转换成正确的字符集中的代码，字符集可以由 ACCEPT-CHARSET明确地规定，也可由文档编码隐含地定义。
- 使用URL转义机制，对数据进行转义。任何在合法URL字符集之外的字符都可以被转换成%XX序列，这里XX是用十六进制数表示的八位组。
- 在一个/x-www-form-urlencoded应用实体中，将结果字符串传送给服务器，用参数charset指出这个字符编码。

如果METHOD属性值是GET，则这个属性被忽略。

2. PREV元素

```
<! ELEMENT PREV (VAR) *>
```

PREV元素声明一个PREV任务，指出导航至历史记录堆栈中的前一个 URL，它在历史记录堆栈上执行一个“ pop ”操作（见4.6.2节）。

有关PREV语义的更多信息，参见9.5.2节。

3. REFRESH元素

```
<! ELEMENT PREV (VAR) +>
```

REFRESH元素声明一个REFRESH任务，指出按照VAR元素的规定，更新用户代理的上下文。在REFRESH任务进行时，用户可以见到状态改变的发生（如，屏幕显示的改变）。

关于REFRESH语义的更多信息，参见4.9.5节中“ REFRESH 任务 ”。

4. NOOP元素

```
<! ELEMENT NOOP EMPTY>
```

NOOP元素指定什么也不做，换句话说，就是“ 没有操作 ”。

关于NOOP语义的更多信息，参见4.9.5节中“ NOOP任务 ”。

4.6.5 卡片/页面间的任务遮盖

大量的元素可用来生成绑定在一个卡片上的事件，这些绑定也可以在页面级中声明：

Card-level 事件处理元素可出现在CARD元素里，对于那个特殊卡片，规定了事件的处理行为。

Deck-level 事件处理元素可出现在TEMPLATE元素里，对于一个页面中的所有卡片，规定了事件的处理行为。一个页面级的事件处理元素等同于在每个卡片中指定事件处理元素。

如果卡片和页面都定义了相同的事件，卡片级的事件处理元素将覆盖页面级的事件处理元素。如果两者有相同的 TYPE，卡片级ONEVENT元素将覆盖页面的ONEVENT元素；如果两者有相同的NAME，卡片级DO元素将覆盖页面DO元素。

如果一个卡片级元素遮盖了一个页面元素，并且卡片级元素指定了一个 NOOP任务，则分配给该卡片的事件将被完全遮盖。在这种情况下，卡片级和页面级元素将被忽略并且当传递该事件时不会发生影响，同时，用户代理不应该将元素暴露给用户（例如，移交一个 UI控制）。实际上，NOOP将元素从卡片中移走。

在下面的例子中，一个页面的 DO元素表示当收到一个特殊的用户动作时，将会执行一个PREV任务，第一张卡片继承 TEMPLATE元素中指定的DO元素并将其显示给用户；第二张卡片用一个NOOP遮盖了页面级的 DO元素，用户代理在显示第二张卡片时，将不显示 DO 元素；第三张卡片遮盖了页面级的 DO元素，如果DO被选定时将引起用户代理显示可替代的标签并执行GO任务。

```
<WML>
  <TEMPLATE>
    <DO TYPE="OPTIONS" NAME="do1" LABEL="default">
      <PREV/>
    </DO>
  </TEMPLATE>
  <CARD NAME="first">
    <!- 页面级的DO不被遮盖。这个卡片显示页面级的DO，作为当前卡片的一部分。 ->
    <!-rest of card ->
```

```
...
  </CARD>
<CARD NAME="second">
    <!- 页面级的DO用NOOP 遮盖，不被显示给用户 ->
    <DO TYPE="OPTIONS" NAME="do1">
      <NOOP/>
    </DO>
    <!- 卡片的其他部分 ->
    ...
  </CARD>
<CARD NAME="third">
    <!- 页面级的DO被遮盖，它被卡片级的DO代替 ->
    <DO TYPE="OPTIONS" NAME="do1" LABEL="options">
      <GO URL="/options"/>
    </DO>
    <!-rest of card ->
    ...
  </CARD>
 </WML>
```

4.6.6 DO元素

```
<!ENTITY % task    "GO | PREV | NOOP | REFRESH">
<!ELEMENT DO (%task;)>
<!ATTLIST DO
  TYPE          CDATA          #REQUIRED
  LABEL         %vdata;        #IMPLIED
  NAME          NMTOKEN        #IMPLIED
  OPTIONAL      %boolean;      "FALSE"
  >
```

DO元素为用户对当前卡片的操作提供一种通用机制，换句话说，是一个卡片级的用户接口元素。DO元素表示与用户代理有关，作者只能假设该元素仅可以被映射到一个用户接口widget，用户可以激活这个接口。例如，窗口widget可能是一个用图形表示的显示按钮、一个软键或功能键、一个话音激活命令序列或任何其他不需要持续状态配合动作就有简单激活操作的接口。

TYPE属性按照作者的设计，对于元素的使用，向用户代理提供了一个暗示，这个属性将被用户代理使用，提供一个到物理用户接口结构的适当映射。 WML作者不能依靠单独的TYPE值语义、行为或一个特定物理结构的TYPE映射。

DO元素可以会出现在卡片或页面级中：

Card-level DO元素可以出现在CARD元素内部，并可以位于文本流的任何地方。如果用户代理想显示内联的DO元素（也就是，在文本流中），就必须用元素的锚点作为显示点。WML的作者不必依靠DO元素的内联显示，也不必依赖于对元素内联显示的正确定位。

Deck-level DO元素可以出现在TEMPLATE元素中，表示一个页面级的DO元素。一个页面的DO元素用于这个页面的所有卡片（也就是说，等同于给每个卡片指定一个DO元素）。为了显示内联的元素，用户代理必须就像页面级的DO元素位于卡片文本流的末端一样进行操作。

如果有相同的NAME，一个卡片级的DO元素遮盖一个页面的DO 元素（更多信息，节见4.6.5节）。对于一张单独的卡片，激活的 DO元素在给定卡片内定义，并加入任何在页面TEMPLATE中指定并在卡片中没有被遮盖的 DO元素。有NOOP任务的所有激活的DO元素不会展现在用户面前，所有非激活的 DO元素也一定不会出现在用户面前，所有的带有非 NOOP任务的DO元素一定会以某种方式被用户访问。换句话说，当卡片包含激活的 DO 元素时，用户一定可以激活这个用户接口项，当用户激活 DO元素时，相关的任务被执行。

属性

TYPE= *cdata*

DO元素类型向用户提供了作者使用这个元素的企图，以及元素如何被映射到物理用户接口结构上。所有的类型都保留着，除了那些标记成试用的类型之外。

用户代理必须能接受任意的类型，可以把任何无法识别的类型看成是 UNKNOWN类型。

在表4-1中，*字符代表任任意的字符串；例如， Test*表示任意以Test开始的字符串。

LABEL= *vdata*

表4-1 预定义的DO类型

类 型	描 述
ACCEPT	肯定的确认（接受）
PREV	逆向历史记录导航
HELP	帮助请求，可能是与上下文有关的
RESET	清除或重置状态
OPTIONS	对于选项或附加操作上下文相关的要求
DELETE	删除项目或选择
UNKNOWN	一般的DO元素，等同于一个空字符串（举例来说，TYPE= ""）
X-*，x-*	实验的类型，这个设置不保留
vnd.*, VND.*和任意的[Vv] [Nn] [Dd].*的组合	卖方特定或用户代理特定的类型，这个设置不保留。卖方应当用格式 VND指定名称。 CO-TYPE，这里CO是公司名缩写，TYPE是DO元素类型。更多的信息参见[RFC2045]

如果用户代理能够动态地标记用户接口的窗口部件（ widget），这个属性规定了适合于这个被标志的文本串，用户代理必须尝试标记 UI窗口部件并使标志（label）按照窗口部件的约束（例如，截短字符串）。如果元素不能被动态地标识，这个属性可以被忽略。

为了可以在各种用户代理上更好地工作，标志 (label)字符长度应当为六个或更短。

NAME=nmtoken

这个属性规定了 DO事件绑定的名称。如果两个 DO元素指定了同样的名字，则它们指的是同一个绑定；如果在卡片级（在一个 CARD元素内）和页面级（在 TEMPLATE元素内）都指定了DO元素并有相同的名字，页面级的 DO元素将被忽略。在单独的卡片或 TEMPLATE元素内用相同名字定义两个或更多 DO元素是错误的。有空值的 NAME等同于一个未指定的NAME属性，未定义的NAME默认为TYPE属性值。

OPTIONAL= *boolean*

如果这个属性值为TRUE，用户代理可以忽略这个元素。

4.6.7 A元素

```
<! ELEMENT A ( %inline; | GO | PREV | REFRESH ) *>
<! ATTLIST  A
   TITLE              %vdata;          #IMPLIED
   >
```

A元素定义了链接的头，链接的结束被定义成其他元素的一部分（如，一个卡片名字属性）。嵌套的锚链接是错误的。

除了OPTION元素中的文本，锚可以出现在任何文本流中（也就是，除了在 OPTION元素中之外，在任意格式化文本的任何地方都是合法的）。锚的链接有一个相关的任务，规定了当锚被选中时，应该表现的行为。在一个 A元素内，定义一个以上的任务元素（例如 GO、PREV、或REFRESH）是错误的。

属性

TITLE= *vdata*

这个属性定义了一个标识链接的短文本字符串，用户代理可以用多种方式显示它，包括用一个按钮或按键的动态标志，或是用一个工具提示、语音提示等等。用户代理可以借助导航的用户接口特性，截短或忽略这个属性。为了在数量众多的用户代理上更好地工作，作者应该限定所有标志的长度不超过六个字符。

4.6.8 内部事件

某些WML元素与用户交互时可以产生事件，这些所谓的“内在事件”指明了用户代理内部的状态转换。独立的元素定义了可由元素自身产生的事件。 WML在表4-2中定义了内在事件。

表4-2 WML内部事件

事 件	元 素	描 述
ONTIMER	CARD，TEMPLATE	在计时器超时， ONTIMER事件发生。用TIMER元素来定义计时器（见4.8.7节）
ONENTERFORWARD	CARD，TEMPLATE	通过GO任务或有相同语义的其他方式，当用户代理进入到一个卡片时， ONENTERFORWARD事件将发生。这包括由脚本程序或用户代理特定机制引起的进入卡片事件，例如直接进入或导航至一个URL ONENTERFORWARD内部事件既可以在卡片级又可以在页面级中定义。在 TEMPLATE元素中定义的事件绑定适用于给定页面中的所有卡片，而且可以像4.6.5节规定的那样被遮盖
ONENTERBACKWARD	CARD，TEMPLATE	借助一个PREV 任务或有相同语义的其它方式，当用户代理导航进入一个卡片时， ONENTERBACKWARD事件发生。换句话说，通过一个从历史记录堆栈中取回的 URL，可使用户代理导航至一个卡片，此时 ONENTERBACKWARD事件发生。这包括由一个脚本程序或用户代理指定机制引起的导航

（续）

事 件	元 素	描 述
		ONENTERBACKWARD内部事件可以在卡片级和页面级中定义。TEMPLATE元素中定义的事件绑定在给定页面的所有卡片中都有效，而且可以像4.6.5节中规定的那样被遮盖
ONCLICK	OPTION	当用户选择或取消这项时，ONCLICK事件发生

作者可以规定当一个内部事件发生时，执行什么样的特定任务，这个规范可以采用两种格式之一。

第一种格式定义了在特定事件发生时导航到 URL。该事件绑定可以在定义好的给定元素的属性中定义，且等同于一个GO任务。例如：

```
< CARD ONENTERFORWARD="/url" > Hello </ CARD >
```

这个属性值可以只规定一个URL。

第二种格式是前一种的扩充方案，使作者能对用户代理的行为有更多的控制。一个ONEVENT元素在父元素中声明，并为特定的内部事件规定完全的事件绑定。例如，下面给出了等同于前一个例子的又一种实现：

```
<CARD>
  <ONEVENT TYPE="ONENTERFORWARD">
     <GO URL="/url"/>
  </ONEVENT>
  Hello
</CARD>
```

用户代理必须把属性语法看作是 ONEVENT元素的缩短格式，这里属性名映射成ONEVENT类型。

一个内部事件绑定的作用域是其所声明的元素。例如，一个在卡片中声明的事件绑定于这个卡片上，在一个元素中声明的任何事件绑定只能在这个元素中被激活。在发生事件绑定冲突时，在子元素中定义的事件绑定优先级高于在父元素定义的事件绑定。在一个元素中，冲突的事件绑定是一个错误。

1. ONEVENT元素

```
<!ENTITY % task  "GO | PREV | NOOP | REFRESH">
<!ELEMENT ONEVENT (%task;)>
<!ATTLIST ONEVENT
   TYPE          CDATA       #REQUIRED
   >
```

在ONEVENT的直接封装元素中，ONEVENT元素把一个任务绑到于一个特殊的内部事件上，换句话说，在“ XYZ ”元素内定义的一个ONEVENT元素,将“ XYZ ”元素与一个内部的事件绑定相关联。

如果ONEVENT元素规定的类型与直接封装元素的合法内部事件不符，则用户代理必须忽略这个ONEVENT元素。

属性

TYPE= *cdata*

TYPE属性指明了内部事件的名称。

2. 卡片/页面内部事件

ONENTERFORWARD和ONENTERBACKWARD事件都可以在卡片级和页面级中定义，并符合在4.6.5节中规定的遮盖语法。不管规定内部事件的语法如何，内部事件可能会被覆盖。ONEVENTFORWARD属性可以覆盖一个用ONEVENT元素定义的页面级事件处理器，反之亦然。

4.7 状态模型

WML支持用户代理状态管理，包括：

变量一种参数，可以改变WML卡片或页面的特性和内容。

历史导航的历史记录，是逆向导航可以利用的有效工具。

与实现有关的状态与用户代理的实现和动作行为有关的其他状态。

4.7.1 浏览器上下文

WML状态存储在一个独立的区域内，被称做浏览器上下文。它用于管理所有的参数和用户代理状态，包括变量、导航记录和其他的与当前用户代理状态、具体实现有关的信息。

4.7.2 NEWCONTEXT属性

CARD元素的NEWCONTEXT属性可将浏览器上下文初始化到一个定义好的状态（见4.8.5节）。这个属性表明浏览器的上下文应该重新初始化，而且必须完成下述的操作：

- 重设（删除）当前浏览器上下文中定义的全部变量。
- 清除导航历史记录状态。
- 重置特定实现的状态到一个已知的值。

NEWCONTEXT只作为GO任务的一部分被完成。有关在导航中进行状态处理的更多信息，参见4.9.5节。

4.7.3 变量

所有WML内容可以参数化，使作者具有更大的灵活性，能生成具有良好的缓存行为和感觉上交互性更好的卡片和页面。 WML变量可以代替字符串，并且可以在运行时用它的当前值替换。

如果变量值不为空字符串，则变量要被置初值。如果变量值等于空字符串、未知或在当前浏览器内容中未被定义则变量值不用被设置。

1. 变量替换

变量值可以被替换成一个卡片的文本（#PCDATA）和WML元素的%vdata和%URL属性值。只有文本信息可以被替换，元素或属性不能被替换。在用户代理中，变量替换在运行时发生，它被定义成一个字符串替换操作，而且不会影响变量的当前值。如果未被定义的变量被引用，结果被替换成一个空字符串。

WML变量名包含一个US-ASCII字母，或后面跟着零或多个字母、数字的下划线，或只是

下划线，任何其他的字母都是非法的。变量名区分大小写。

下面是一个变量替换语法的BNF式描述。除了用字符“|”表示可选之外，这个描述使用[RFC822]中建立的惯例。简单地讲，用“(”和“)”来组合元素，而可选的元素括在“[”和“]”内。元素前可用<N>*定义，表示这个元素N次或更多次的重复（未指定时N默认为零）。

```
var     = ( "$" varname ) |
          ( "$(" varname [ conv ] ")" )
conv    = ":" ( escape | noesc | unesc )
escape  = ("E" | "e") [ ( "S" | "s" ) ( "C" | "c" )
                       ( "A" | "a" ) ( "P" | "p" )
                       ( "E" | "e" ) ]
noesc   = ( "N" | "n" ) [ ( "O" | "o" ) ("E" | "e")
                          ( "S" | "s" ) ( "C" | "c" ) ]
unesc   = ( "U" | "u" ) [ ( "N" | "n" ) ("E" | "e")
                          ( "S" | "s" ) ( "C" | "c" ) ]
varname = ( "_" | alpha ) *[ "_" | alpha | digit ]
alpha   = lalpha | halpha
lalpha  = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" |
          "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" |
          "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
halpha  = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
          "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
          "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
digit   = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
          "8" | "9"
```

若从周围的上下文中不能推断出某变量，则在该变量结尾处要加上圆括号（例如，一个如空白的非法字符）。

例如：

```
This is a $var
This is another $(var).
This is an escaped $(var:e).
Long form of escaped $(var:escape).
Long form of unescape $(var:unesc).
Short form of no-escape $(var:N).
Other legal variable forms: $_X $X32 $Test_9A
```

变量值被替换时可以转换成其他不同的格式，转换可在变量引用中定义，跟在冒号后面。表4-3概括了当前的转换和它们的合法缩写：

表4-3 变量扩展方式

转 换	作 用
无扩展	没有改变变量值
扩展	URL扩展变量值
未扩展	URL未扩展变量值

在变量替换时的转换不会影响变量的实际值。

[RFC1738]中详细描述了URL的转义。像在[RFC1738]中规定的一样，所有在WML中定义

的词法敏感的字符必须被转义，包括所有保留的和不安全的 URL字符。

如果没有指定转换类型，将使用适合上下文的变换格式替换变量。在没有进行转换的地方，ONENTERBACKWARD、ONENTERFORWARD、URL和SRC属性默认为扩展转换。按照规定，nosc转换将使一个变量的上下文相关转义被禁止。

2. 变量替换语法分析

在所有的 XML分析完成之后，再进行变量替换语法（如 $x）分析。在 XML术语中，XML处理程序分析完文件并向 XML应用程序提供了分析结果报表后，再进行变量替换分析。在该规范的上下文中，WML语法分析程序和用户代理是 XML应用程序。

这意味着在分析了 XML结构（如标签和实体）之后，再进行所有变量语法分析。在变量分析的上下文中，所有的 XML语法比变量语法有更高的优先权，例如，实体替换发生在变量替换语法分析前。下面的例子等同于名为 x的变量：

```
$x
&#x24;x
$&#x58
&#36;&#x58
```

3. $符号

分析规则的一个副作用是文字中的美圆符号必须编码成一对 $符号实体，单独的 $实体，甚至诸如&#x24这样的定义，将导致一个变量替换。

为了在WML页面中包括一个$符号，必须对其进行转义。这可用下面的语法实现：

一行中的两个$符号被单独的$符号代替。例如：

```
This is a $$ character.
```

这将显示为：

```
This is a $ character.
```

为了使URL扩展字符串包括$符号，可以用URL扩展形式定义它：

```
%24
```

4. 变量赋值

有多种方法为变量赋值。当为一个已经在浏览器上下文中定义的变量赋值时，这个变量被赋予当前值。

作为导航的另一个作用， VAR元素允许作者设置变量的状态。 VAR可以在任务元素中定义，包括GO、PREV和REFRESH。VAR元素指定变量名和变量值，例如：

```
<VAR   NAME="location"   VALUE="$(X)"/>
```

作为导航的另一个作用， NAME属性中定义好的变量可以被赋值（如： location）。有关VAR元素的处理过程更多的信息，参见对事件处理的讨论（第 4.6节和第4.9.5节）。

输入元素建立根据用户输入信息的 KEY属性设置的变量。例如， INPUT元素把输入的文本分配给变量，而SELECT元素则负责分配所选OPTION元素的VALUE属性值。

当用户把输入提交给INPUT或SELECT元素时，用户的输入便被写进了变量。提交输入是一个与MMI有关的概念，而且WML的作者不必依赖特定的用户接口，例如，有些实现用每个进入INPUT元素的字符更新变量，而其他的一些实现会推迟变量更新，直到 INPUT元素失去焦点。在执行任何任务之前，用户代理必须更新所有的变量。在设置变量时，用户代理可能

重新显示当前卡片，但是作者不必假设这个动作一定发生。

4.8 WML页面结构

WML数据构成了卡片的集合群，一个单独的卡片集合被称为一个 WML页面，每张卡片包含结构化的内容和导航规则。从逻辑上讲，一个用户通过一系列的卡片进行导航，浏览每个卡片内容，输入要求的信息，并作出选择是导航到另一个卡片还是返回到以前访问的卡片。

4.8.1 文档序言

一个有效的WML页面也就是一个有效的 XML文件，因此必须包括一个 XML声明以及一个文件类型声明（有关定义一个有效文档的更详细内容，参见 [XML]）。一个典型的文档序言包括：

```
<?xml version="1.0"?>
<!DOCTYPE WML PUBLIC "-//WAPFORUM//DTD WML 1.0//EN"
                      http://www.wapforum.org/    DTD/wml.xml>
```

省略文档序言是错误的。

4.8.2 WML元素

```
<!ELEMENT WML ( HEAD? TEMPLATE? , CARD+)>
<!ATTLIST WML
 xml:lang            NMTOKEN         #IMPLIED
 >
```

WML元素定义了一个页面和包括在这个页面中的所有信息及卡片。

属性

xml:lang= *nmtoken*

xml:lang属性规定了写文档使用的自然语言或正式语言。关于属性值的语法和规范，详细内容请参阅[XML]。如果xml:lang属性被指定，它就具有比文档语言（也就是，传输元数据）中任何其他规范都高的优先级。

一个WML示例

以下是一个包含两张卡片的页面，每个卡片用一个 CARD元素表示（有关卡片的信息，参见4.8.5节）。装入页面之后，一个用户代理显示第一张卡，如果用户激活了 DO元素，用户代理就会显示第二张卡。

```
<WML>
   <CARD>
      <DO TYPE="ACCEPT">
          <GO URL="#card2"/>
      </DO>
      Hello world!
      This is the first card...
   </CARD>
   <CARD NAME="card2">
```

```
        This is the second card.
        Goodbye 。
    </CARD>
</WML>
```

4.8.3 HEAD元素

```
<!ELEMENT HEAD (ACCESS | META)+>
```

HEAD元素包含了与一个页面有关的总体信息，包括元数据和接入控制元素。

1. ACCESS元素

```
<!ELEMENT ACCESS EMPTY>
<!ATTLIST ACCESS
  DOMAIN      CDATA         #IMPLIED
  PATH        CDATA         #IMPLIED
  PUBLIC      %boolean;     "FALSE"
  >
```

ACCESS元素为整个页面规定了接入控制信息，一个页面若包含多于一个的 ACCESS元素会产生错误。

属性

DOMAIN= *cdata*
PATH= *cdata*

页面的DOMAIN和PATH属性定义了可以访问这个页面的其他的页面。当用户代理从一个页面导航到另一个页面时，它执行接入控制检查来决定目标页面是否允许当前的页面访问。

如果一个页面有 DOMAIN属性或PATH属性（或二者都有），则这个页面相应的 URL必须与这些属性值相匹配。具体的匹配方法如下：访问域与相应 URL域名部分的后缀匹配，访问路径与相应URL路径部分的前缀匹配。

应使用每一个子域的全部元素进行 DOMAIN后缀匹配，且必须准确匹配每一个元素（例如，www.Wapforum . org 与wapforum .org匹配，但与forum .org不匹配）。同样，应使用完整的路径元素进行PATH的前缀匹配，且必须准确匹配每一个元素（例如， /X/Y与/X匹配，但与/XZ不匹配）。

DOMAIN属性的默认值是当前页面的域。 PATH属性的默认值是“／”。

对于不知道当前页面绝对路径的应用，为了简化开发， PATH属性也接受相对URL。用户代理能将相对路径转化为绝对路径，然后再对 PATH属性进行前缀匹配。

例如，给定下列访问控制属性：

DOMAIN="wapforum.org"
PATH="/cbb"

则以下这些URL被允许访问这个页面：

http://wapforum.org/cbb/stocks.cgi
https://www.wapforum. org/cbb/bonds.cgi
http://www.wapforum.org/cbb/demos/alpha/packages.cgi?x=123&y=456

以下这些URL不被允许访问这个页面：

http://www.test.net/cbb

http://www.wapforum.org/internal/foo.wml

DOMAIN和PATH遵循URL的大小写规则。

PUBLIC= *boolean*

该属性说明页面访问控制对这个页面是否可用。若不可用（即， PUBLIC=”TRUE”被指定），在任何页面中的卡片都可以接入此页面；若可用，则要根据 DOMAIN和PATH属性决定哪些卡片或页面能够访问这个页面。访问控制的默认值是可用。

2. META元素

```
<!ELEMENT META EMPTY>
<!ATTLIST META
  HTTP-EQUIV     CDATA     #IMPLIED
  NAME           CDATA     #IMPLIED
  USER-AGENT     CDATA     #IMPLIED
  CONTENT        CDATA     #REQUIRED
  SCHEME         CDATA     #IMPLIED
 >
```

META元素包括与WML页面有关的一般元信息，元信息由性质名称和取值定义，这种规范既没有定义任何性质，也没有规定用户代理应该如何解释这些元数据。用户代理不需要支持元数据机制。

一个META元素不能包括多于一个指明性质名称的属性，否则会产生错误，也就是说，一个META元素不能包括下列集合中的多于一个的属性： NAME,HTTP-EQUIV和USER-AGENT。

属性

NAME= *cdata*

该属性指定性质名称，用户代理必须忽略任何以这个属性命名的元数据，而且网络服务器不能发出携带有以这个属性命名的元数据的 WML内容。

HTTP-EQUIV= *cdata*

该属性可以取代NAME，表明这个性质应作为一个HTTP报头来进行解释（见[RFC2068]）。如果内容在到达用户代理之前被作了标记，用这个属性命名的元数据应被转化为一个 WSP或HTTP响应头。

USER-AGENT= *cdata*

该属性可以取代NAME。这个元数据只能被传到用户代理，而不能被任何网络的中间媒体删除。

CONTENT= *cdata*

该属性规定性质的取值。

SCHEME= *cdata*

该属性规定了一个解释性质取值的格式或结构。方案值根据元数据类型的不同而变化。

4.8.4 TEMPLATE元素

```
<!ENTITY % navelmts "DO | ONEVENT">
```

```
<!ELEMENT TEMPLATE (%navelmts;)*>
<!ATTLIST TEMPLATE
  %cardev;
  >
```

TEMPLATE元素为页面中的卡片声明一个模板。在 TEMPLATE元素中声明的绑定事件（如DO和ONEVENT）可用于页面中的所有卡片，它规定一个绑定事件与在每一个卡片元素中规定是等价的。一个卡片元素可以覆盖在 TEMPLATE元素中定义的行为。特别是：

- 在TEMPLATE元素中声明的DO元素可以被在一个单独卡片中定义的 DO元素覆盖，要求两个元素有相同的NAME属性值。更详细的内容，参见4.6.5节。
- 在TEMPLATE元素中规定的内部事件绑定可以被在卡片元素中规定的绑定事件覆盖，更详细的内容，参见4.6.8节。

关于卡片级的内部事件的定义，参见4.8.5节的cardev实体。

在其他地方定义的属性

以下任务的属性在4.8.5节“卡片内部事件”中定义。

%cardev

4.8.5 卡片元素

一个WML页面包含一个卡片的集合。有多种卡片类型，每种类型定义了一种不同的用户交互模式。

1. 卡片内部事件

```
<!ENTITY % cardev
 "ONENTERFORWARD        %URL;        #IMPLIED
  ONENTERBACKWARD       %URL;        #IMPLIED
  ONTIMER               %URL;         #IMPLIED"
  >
```

下列属性出现在CARD元素和TEMPLATE元素中。

属性

ONENTERFORWARD= *URL*

ONENTERFORWARD事件发生，当用户使用用户代理利用 GO任务导航进入一个卡片时。

ONENTERBACKWARD= *URL*

ONENTERBACKWARD事件发生，当用户使用用户代理利用 PREV任务导航进入一个卡片时。

ONTIMER= *URL*

当一个计时器（TIMER）超时时，ONTIMER事件发生。

2. CARD元素

```
<!ENTITY % fields "%flow; | INPUT | SELECT | FIELDSET">
<!ELEMENT CARD (%fields; | %navelmts; | TIMER)*>
<!ATTLIST CARD
  NAME                  NMTOKEN               #IMPLIED
  TITLE                 %vdata;               #IMPLIED
  NEWCONTEXT            %boolean;             "FALSE"
```

```
STYLE                (LIST|SET)          "LIST"
%cardev;
>
```

CARD元素是一个文本和输入元素的容器，它可使各种设备的显示和版面安排具有充分的灵活性，并具有各式各样的显示和输入特性。 CARD元素只是说明了一般的版面安排和所需的输入字段，并没有在版面安排及用户输入方面过多地限制用户代理的应用实现。例如，一个CARD既可以在一个大屏幕的设备中显示为单独一页，也可以在一个小屏幕的设备中以多页显示。

一个CARD可以包括标记、输入字段以及说明这个卡片结构的元素，在卡片中的元素的顺序是很重要的，用户代理应引起重视。

属性

NAME= *nmtoken*

该属性给出了一个卡片名称，一个卡片的名称被用作一个片段的锚。更多的信息，参见4.2.2节。

TITLE= *vdata*

TITLE属性定义了有关卡片的建议信息，用户代理可以用很多方式显示标题（如，建议的书签名称、可自动弹起的工具提示等）。

NEWCONTEXT= *boolean*

该属性说明当前浏览器上下文在进入此卡片前应被重新初始化。更多的内容，参见 4.7.2节。

STYLE=(*LIST|SET*)

该属性为用户代理提供了有关卡片内容组织方式的一个提示，这个提示可用来组织卡片的内容表示或其他的影响这个卡片版面安排的内容。

LIST卡片被自然地组织成一个字段元素的线性序列。如，一个由用户按指定的顺序排列自动处理的问题或字段集，这种格式最适用于没有可选字段的短格式（如，发送一个e-mail消息命令，需要一个To字段：填写地址、题目、消息，这些字段都是按逻辑顺序指定的）。

如果希望在一个小屏幕设备中LIST组可以被表示成一系列的小显示屏，可以利用各字段间或字段集间的屏幕翻转（screen flip）来实现，其他的用户代理可以选择同时显示所有的字段。

SET这个卡片是一个由无序字段元素组成的集合。对于用户更新单独的输入字段，这些包含可选部分、或无序的部分、或简单记录数据的字段的集合是非常有用的。如果希望在小屏幕设备中SET组会以分层或树形结构显示，那么在这些显示类型中，每个字段和字段组的TITLE属性将被用来定义在顶层摘要卡片中显示给用户的名称。

用户代理可以根据与设备能力（如屏幕的大小及输入设备）相适应的方法解释格式属性。另外，用户代理会采用用户接口规范，这样能以最适合设备输入模式的方法处理对输入元素的编辑。

例如，一个以STYLE=SET显示CARD的电话级设备可以用一个软件定义的功能键或按钮来选择编辑或所要查看的单个字段。一个 PDA级的设备可以应要求创建一个软件按钮，也可以在屏幕上显示所有字段以进行直接的操作。

对于具有有限显示能力的设备来说，通常有必要在字段之间插入屏幕跳转或用户接口转换，随后，用户代理需要在字段之间判定合适的界限。用户代理可以用以下内容来决定屏幕跳转位置的选择：

- FIELDSET定义字段之间的逻辑界限。
- 字段（如，INPUT）可以单独显示。显示之后，在此字段之前的标记（流）应看作一个字段提示并和输入元素一起显示。

在其他地方定义的属性

以下任务属性在4.8.5节“卡片内部事件”中定义。

%cardev

- 一个CARD示例

以下是一个嵌入WML页面中的简单CARD元素，这个卡片包括由用户代理显示的文本。另外，这个示例说明了一个简单的DO元素的用法，这个DO元素定义在页面级中。

```
<WML>
  <TEMPLATE>
    <DO TYPE="ACCEPT" LABEL="Exit">
      <PREV/>
    </DO>
  </TEMPLATE>
  <CARD>
    Hello World!
  </CARD>
</WML>
```

4.8.6 控制元素

1. TABINDEX属性

TABINDEX=number

此属性规定了当前元素的制表位置。制表位置说明了在单个WML卡片中制表时，元素来回移动的相对顺序。数值上大一点的TABINDEX值表示在表中位置相对靠后的元素，数值小一点的TABINDEX值表示相对靠前的元素。

对于卡片中的每一个输入元素(如，INPUT和SELECT)，卡片的制表序列都为其安排了位置，此外，用户代理还可以为其他的元素分配一个制表位置。 TABINDEX属性说明一个给定元素的制表位置，对于作者没有指明的制表位置的元素，可由用户代理为其分配一个。在制表序列中，由用户代理指定的制表位置一定比由作者指定的制表位置靠后。

制表是一个导航加速器，对于所有的有用户代理而言，它都是选择性的，因此作者不能指望用户代理自己会实现制表。

2. 选择列表

选择列表是一个输入元素，为用户指定一个选择列表，单选和多选列表都支持。

(1) SELECT元素

```
<!ELEMENT SELECT (OPTGROUP|OPTION)+>
<!ATTLIST SELECT
```

```
TITLE          %vdata;          #IMPLIED
KEY            NMTOKEN          #IMPLIED
DEFAULT        %vdata;          #IMPLIED
IKEY           NMTOKEN          #IMPLIED
IDEFAULT       %vdata;          #IMPLIED
MULTIPLE       %boolean;        "FALSE"
TABINDEX       %number;         #IMPLIED
>
```

SELECT元素让用户在一个选项列表中进行选取，每个选项由一个 OPTION元素指定，每个OPTION元素可以有一行格式化的文本（若文本太长，会被用户代理打包或截断）。通过OPTGROUP元素可将OPTION元素组织在一个分层的组中。

属性

MULTIPLE= *boolean*

该属性说明选择列表将接受多项选择，若不设置此属性，则选择列表只能接受单项选择。

KEY= *nmtoken*

DEFAULT= *vdata*

KEY属性指出随选择结果设定的变量的名称，此变量被赋给选定的 OPTION元素的字符串值，而OPTION元素的值由VALUE属性指定。变量KEY的值用来对选择列表中的选项进行预选。

DEFAULT属性指定由KEY属性命名的变量的默认值。如果元素被显示，且由 KEY属性命名的变量没有赋值，则 KEY变量的值就是在属性 DEFAULT中给出的值；如果KEY变量已经有值，则忽略DEFAULT属性。在使用变量KEY的值进行列表的预选之前，对默认值进行处理。

若该元素允许进行多选，用户选择的结果是一个所有被选值的列表，用分号分隔， KEY变量用这个选择结果赋值。此外，属性 DEFAULT被解释成一个由分号分隔的所有预选择项的列表。

IKEY= *nmtoken*

IDEFAULT= *vdata*

IKEY属性指出用选择索引结果进行了赋值的变量名称。索引结果是在选择列表中当前被选定的OPTION的位置，当没有OPTION被选定时，索引值为0，索引值从1开始单调递增。

IDEFAULT属性指出了默认选择的OPTION元素。显示OPTION元素的时候，如果由IKEY属性命名的变量没有赋值，则此变量指向默认的选定记录；如果这个变量已经被赋值，则可忽略IDEFAULT属性。若没指定IKEY属性，则在每次显示这个元素时，都采用 IDEFAULT值。

若该元素允许进行多选，用户选择的结果是一个所有被选值的列表，用分号分隔 （如，“1；2”），IKEY变量用这个结果赋值。此外，属性 IDEFAULT被解释成一个由分号分隔的所有预选择项的列表（如，“1;4”）。

TITLE= *vdata*

TITLE属性指定显示这个对象时使用的元素标题。

当进入一个含SELECT元素的卡片时，用户代理必须采取以下的方式选择初始项：

- 若IKEY属性存在，就用以IKEY属性命名的变量中的索引选择选项。若指定的变量没有赋值，则假设索引为 1。若索引值比选择列表中的选项数目还要大，则选择最后一个选项。

• 若IKEY属性不存在而KEY属性存在，用KEY指定的变量值作为选择选项。若由 KEY指定的变量没有赋值，或所有的OPTION属性与该变量值都不匹配，则选择第一个选项。一旦选定OPTION，则由KEY命名的变量的值更新为选定选项的值。

KEY和IKEY，或者DEFAULT和IDEFAULT都可以被指定。 IDEFAULT的优先级高于DEFAULT，IKEY的优先级高于KEY。

(2) OPTION元素

```
<!ELEMENT OPTION (%text; | ONEVENT)*>
<!ATTLIST OPTION
  VALUE      %vdata;    #IMPLIED
  TITLE      %vdata;    #IMPLIED
  ONCLICK    %URL;      #IMPLIED
  >
```

该元素在选择元素中指定一个单独的选择选项。

属性

VALUE= *vdata*

在为KEY变量赋值时，使用VALUE属性给出值，在用户选择该选项时， VALUE属性指定的结果值被赋给SELECT元素的KEY变量。

VALUE属性可以包括变量的引用。在变量被赋值之前，计算这些引用。

TITLE= *vdata*

该属性为这个元素指定了一个向用户显示对象的标题。

ONCLICK= *URL*

当用户选择或取消一个选项时，触发一个ONCLICK事件。只要用户选择或取消一个选项，多项选择列表就都会触发一个ONCLICK事件；对于单项选择，只有在用户选择选这个选项时，才会触发一个ONCLICK事件，也就是说，取消原来选定的选项不触发任何事件。

(3) OPTGROUP元素

```
<!ELEMENT OPTGROUP (OPTGROUP|OPTION)+ >
<!ATTLIST OPTGROUP
  TITLE      %vdata;    #IMPLIED
  >
```

OPTGROUP元素允许作者对有关的OPTION元素进行分组，构成一个层次结构，用户代理利用这种层次结构可以非常容易的在多种设备中安排版面和提供显示。

属性

TITLE= *vdata*

该属性为元素指定一个用于显示对象的标题。

(4) 选择列表示例

在这个示例中，给出了一个单选择列表。如果用户选择了“ Dog ”选项，则给变量X赋值“ D ”。

```
<WML>
  <CARD>
    Please choose your favourite animal:
```

```
    <SELECT KEY="X">
      <OPTION VALUE="D">Dog</OPTION>
      <OPTION VALUE="C">Cat</OPTION>
    </SELECT>
  </CARD>
</WML>
```

在这个示例中，给出了一个单选择列表。若用户选择“ Cat ”选项，则变量“ I ”被赋值“ 2 ”。此外，如果变量“ I ”没有被预先赋值，则会预选“ Dog ”选项。

```
<WML>
  <CARD>
    Please choose your favourite animal:
    <SELECT IKEY="I" IDEFAULT="1">
      <OPTION VALUE="D">Dog</OPTION>
      <OPTION VALUE="C">Cat</OPTION>
    </SELECT>
  </CARD>
</WML>
```

在这个示例中，给出了一个有多项选择的列表。若用户想要选择“ Cat ”和“ House ”选项，则变量X被用“ C；H ”赋值，变量“ I ”被用“ 1；3 ”赋值。另外，如果变量“ I ”没有被预先赋值，则会预选“ Dog ”和“ Cat ”选项。

```
<WML>
  <CARD>
    Please choose <I>all</I> of your favourite animals:
    <SELECT KEY="X" IKEY="I" IDEFAULT="1;2" MULTIPLE="TRUE">
      <OPTION VALUE="D">Dog</OPTION>
      <OPTION VALUE="C">Cat</OPTION>
      <OPTION VALUE="H">Horse</OPTION>
    </SELECT>
  </CARD>
</WML>
```

3. INPUT元素

```
<!ELEMENT INPUT EMPTY>
<!ATTLIST INPUT
  KEY         NMTOKEN           #REQUIRED
  TYPE        (TEXT|PASSWORD)   "TEXT"
  VALUE       %vdata;           #IMPLIED
  DEFAULT     %vdata;           #IMPLIED
  FORMAT      CDATA             #IMPLIED
  EMPTYOK     %boolean;         "FALSE"
  SIZE        %number;          #IMPLIED
  MAXLENGTH   %number;          #IMPLIED
  TABINDEX    %number;          #IMPLIED
  TITLE       %vdata;           #IMPLIED
  >
```

INPUT元素指定一个文本输入对象。FORMAT属性约束用户输入，但它又是可选的。

属性

KEY= *nmtoken*
DEFAULT= *vdata*
VALUE= *vdata*

KEY属性指定一个变量名，此变量的值由用户的文本输入的结果设置。KEY变量的值用来预加载文本输入对象。

DEFAULT属性给出了用KEY属性命名变量的默认值。当这个元素被显示，用KEY属性命名的变量没有赋值，则KEY变量的值由DEFAUULT属性给出；若KEY变量已经有值，则忽略DEFAULT属性；若DEFAULT属性指定的值与FORMAT属性指定的输入模式不一致，则用户代理必须忽略DEFAULT属性。

DEFAULT和VALUE属性在行为和语法上是相同的。

TYPE= *(TEXT|PASSWORD)*

该属性指定了文本输入区域的类型，其默认类型是TEXT，它允许以下取值：

TEXT一个文本输入框。要以可读的格式输入并向用户显示，同时以与用户代理相适应的方式回显每个字符。

PASSWORD一个文本输入框，输入的每个字符应以模糊或难辨认的方式回显。例如，用户代理可以用“*”来显示用户输入的字符。一般情况下，PASSWORD输入模式常用于输入密码或其他私人数据。应注意PASSWORD输入并不十分安全，在关键的应用中不能完全依靠PASSWORD输入。

在两种情况下，KEY变量应用于用户的输入。

FORMAT= *cdata*

FORMAT属性为用户输入内容提供输入模式。字符串由模式控制字符和在输入区域显示的静态文本构成。用户代理可利用这种模式格式，使数据输入更方便。

格式控制字符指出了用户输入数据的格式，默认格式为“*M”。格式码如下：

A 可输入大写字母或标点符号（即，大写的非数字字符）。

a 可输入小写字母或标点符号（即，小写的非数字字符）。

N 可输入数字字符。

X 可输入大写字母。

x 可输入小写字母。

M 可输入任意大写字符。为简化数据输入，用户代理选择字符以大写输入，但必须允许任何字符输入。

m 可输入任意小写字符。为简化数据输入，用户代理选择字符以小写输入，但必须允许任何字符输入。

*f 可输入任何数字字符。f是上面所说的格式码之一，指定可以输入的字符类型。注意：这种格式只能指定一次且必须出现在格式化字符串的末尾。

nf 可输入n个字符，这里n从1到9。f是上面所说的格式码之一，指定可以输入的字符类型。注意：这种格式只能指定一次且必须出现在格式字符串的末尾。

\c 显示输入字段的下一个字符‘c’，允许引用格式码使它们能在输入区域中显示。

为最大地发挥受到限制的输入语言和字符集的作用，用户代理应力求使用格式码。若输入语言和字符集对数字和字符的情况有清楚的定义，则必须遵循所给定义，作者不能依赖在给定的语言中对特殊格式码进行解释。

EMPTYOK= *boolean*

EMPTYOK属性说明尽管已经指定了非空格式的字符串，此 INPUT元素接受到的仍然是空的输入。一般情况下，这个属性常用来说明可选的格式化输入字段，在默认情况下，对于定义了FORMAT属性的INPUT元素，要求用户以FORMAT属性规定的格式输入数据。

SIZE= *number*

此属性指定了文本输入区域的字符宽度，用户代理可以忽略这个属性。

MAXLENGTH= *number*

此属性规定了用户可以在文本输入区域内输入的最大字符数。默认情况下，不限制最大字符数。

TITLE= *vdata*

此属性为本元素规定显示时的标题。

INPUT元素示例

本示例中将给出一个 INPUT元素，这个元素可接受任何字符，它将输入以可读的形式向用户显示，其允许输入的最大字符数为 32，并将输入的结果赋给变量 X。

```
<INPUT KEY="X" TYPE="TEXT" MAXLENGTH="32"/>
```

下面的示例向用户请求输入，并将结果赋给变量 NAME。文本字段的默认值为“ Robert ”。

```
<INPUT KEY="NAME" TYPE="TEXT" DEFAULT="Robert"/>
```

下面的示例是一个提醒输入用户姓名（名、姓）和年龄的卡片。

```
<CARD>
  First name: <INPUT TYPE="TEXT" KEY="first"/><BR/>
  Last name: <INPUT TYPE="TEXT" KEY="last"/><BR/>
  Age: <INPUT TYPE="TEXT" KEY="age" FORMAT="*N"/>
</CARD>
```

4. FIELDSET元素

```
<!ELEMENT FIELDSET (%fields;)* >
<!ATTLIST FIELDSET
  TITLE           %vdata;      #IMPLIED
  >
```

FIELDSET元素可将相关字段和文本分组，这种分组为用户代理提供信息，可使排版和浏览更优化。FIELDSET元素允许嵌套，给用户提供了一种可以指定跨越不同种类设备的行为的方法。关于FIELDSET元素如何影响版面安排和浏览的更多信息，参见4.8.5节中“ CARD元素 ”。

属性

TITLE= *vdata*

此属性指定FIELDSET元素在显示时的标题。

FIELDSET元素示例

下述示例定义了一个WML页面，向用户请求输入基本的身份信息和个人信息。该页面被分成多个字段的集合，向用户代理说明了首选的字段分组。

```
<WML>
  <CARD>
    <DO TYPE="ACCEPT">
      <GO URL="/submit?f=$(fname)&amp;l=$(lname)&amp;s=$(sex)&amp;a=$(age)"/>
    </DO>
    <FIELDSET TITLE="Name">
      First name: <INPUT TYPE="TEXT" KEY="fname" MAXLENGTH="32"/><BR/>
      Last name: <INPUT TYPE="TEXT" KEY="lname" MAXLENGTH="32"/><BR/>
    </FIELDSET>
    <FIELDSET TITLE="Info">
      <SELECT KEY="sex">
        <OPTION VALUE="F">Female</OPTION>
        <OPTION VALUE="M">Male</OPTION>
      </SELECT>
      <BR/>
      Age: <INPUT TYPE="TEXT" KEY="age" FORMAT="*N"/>
    </FIELDSET>
  </CARD>
</WML>
```

4.8.7 TIMER元素

```
<!ELEMENT TIMER EMPTY>
<!ATTLIST TIMER
  KEY          NMTOKEN      #IMPLIED
  DEFAULT      %vdata;      #REQUIRED
  >
```

TIMER元素声明了一个卡片计时器，这个计时器用来处理休眠或空闲时间。计时器在进入卡片时被初始化并启动，在退出卡片时终止，进入卡片就是导致卡片被激活某个任务或用户行为。例如，在卡片中导航。退出卡片意味着某一个任务的完成（见 4.6.3节和4.9.5节）。计时器的值从初始值递减，当从1减到0时触发ONTIMER内部事件。若计时器超时，用户还没有退出卡片，一个ONTIMER内部事件被发往这张卡片。

计时器分辨率与具体的实现有关，计时器与用户代理的用户接口以及其他基于时间或异步设备的功能性的交互也取决于实现。一张卡片中只能有一个 TIMER元素。

TIMER的时间间隔值以1/10秒为单位。作者不要期望有特殊的计时器分辨率，而是应该为用户提供另一种调用计时器的方式。若时间间隔不是正整数，则用户代理必须忽略 TIMER元素；若时间间隔为0，则使计时器无效。

属性

KEY= *nmtoken*

KEY属性指定一变量名，该变量根据计时器的值取值， KEY属性值用来在计时器初始化的时候设定时间间隔。当退出卡片或计时器的值减到 0时，将计时器的当前值赋给由 KEY属性命名的变量。例如，当计时器的值减到 0时，由KEY属性指定的变量的值被置为 0。

DEFAULT= *vdata*

DEFAULT属性给出了用KEY属性规定的变量的默认值。当计时器被初始化而由 KEY指定的变量的值没有设置时，变量被赋予默认值；若 KEY变量已经有值，则忽视DEFAULT属性；若没有指定KEY属性，通常将时间间隔初始化为 DEFAULT属性指定的值。

TIMER示例

下面的页面将一条文本消息显示约 10秒，然后进入URL /next：

```
<WML>
  <CARD ONTIMER="/next">
       <TIMER DEFAULT="100"/>
       Hello World!
  </CARD>
</WML>
```

以上示例还可以实现如下：

```
<WML>
  <CARD>
    <ONEVENT TYPE="ONTIMER">
      <GO URL="/next"/>
    </ONEVENT>
    <TIMER DEFAULT="100"/>
    Hello World!
  </CARD>
</WML>
```

以下示例说明一个计时器如何被初始化和恢复计数的。每次进入卡片时，计时器被设置为变量t的值，若t没被赋值，计时器被设置成 5秒。

```
<WML>
  <CARD ONTIMER= "/next" >
    <TIMER KEY= "t" DEFAULT= "50" />
    Hello World!
  </CARD>
</WML>
```

4.8.8 文本

在本节中定义与文本有关的元素和构造。

1. 空格

WML空格和断行处理是基于 [XML]的，且假定了默认的空格的处理规则。 WML用户代理忽略所有无关紧要的空格，正如 XML规范中定义的。另外，所有其他的空白序列都必须被压缩到一个单独的字间距中。

用户代理使用与方法有关的方式处理字间距，所用语言不同，处理字间距的方式也不同。

2. 重点

```
<!ELEMENT EM     (%flow;)*>
<!ELEMENT STRONG (%flow;)*>
<!ELEMENT B      (%flow;)*>
<!ELEMENT I      (%flow;)*>
<!ELEMENT U      (%flow;)*>
<!ELEMENT BIG    (%flow;)*>
<!ELEMENT SMALL  (%flow;)*>
```

重点元素给出了强调的文本标记信息。

EM **加黑显示**（Render with emphasis）。

STRONG **更黑加重显示**（Render with strong emphasis）。

B **黑体显示**（Render with a bold font）。

I **斜体显示**（Render with an italic font）。

U **下画线显示**（Render with underline）。

BIG **大字体显示**（Render with a large font）。

SMALL **小字体显示**（Render with a small font）。

作者应该在所有可能的地方使用 STRONG**和**EM**元素，除了在需要特别控制文本显示的地方，尽可能的不要使用**B、I**和**U**元素。**

3. 换行

```
<!ENTITY % TAlign    "(LEFT|RIGHT|CENTER)" >
<!ENTITY % BRMode    "(WRAP|NOWRAP)" >
<!ELEMENT BR EMPTY>
<!ATTLIST BR
  ALIGN   %TAlign;   "LEFT"
  MODE    %BRMode;    #IMPLIED
>
```

WML**有两种自动换行模式**:**断行或不断行。在断行模式下**,**换行符应当被插入在文本流中**,**作为在个体设备上的适当表示，任何字之间的空间都是合法的换行点；在不断行模式下** ,**文本行不能被自动地换行。**

不可断行的空间实体 (or)**表明一个不能被用户代理作为字间位置的空格，作者应该使用** **，以防止不期望的换行。软连字符实体** (or)**表明一个可以被用户代理用来作为换行符的位置。如果在软连字符处发生换行** ,**用户主体必须在行尾插入一个连字符**($#45;)**，在其他任何操作中**,**软连字符实体都应该被忽略。一个用户代理可以在格式化文本行中选择彻底地忽略软连字符。**

BR**元素建立了一个新行开始，并规定了新行的换行和行对齐参数。如果换行模式没有被确定**,**将采取本卡片的上一行的换行模式；若文本对齐格式没有确定** ,**将采取默认的模式，也就是左对齐。**

一个 WML**卡片有一个换行和对齐模式。一张卡片初始化时的换行模式是** : MODE=“WRAP”(**断行**)**，初始的文本对齐模式是**:ALIGN=“LEFT”(**左对齐**)**。若一个卡片中的第一个非空格标识是** BR**元素**,**那么从** BR**开始卡片的第一行。若一个卡片中的第一个非空格标识不是**BR**元素**,**那么一个新行是以隐含的默认换行模式和对齐模式开始。**

如何对待一个过长、无法在屏幕上全部显示的行，在当前行的换行模式中有具体阐述。若指定MODE=“WRAP”，则这一行会被安排到多行上去；若指定MODE=“LINE”,那么这一行不换行。用户代理必须提供一种机制，并用它来观看整个行（如水平滚动或其他用户代理确认的机制）。

属性

ALIGN= *(LEFT|RIGHT|CENTER)*

此属性规定了行文本的对齐模式，向用户显示时，文本可以是中心对齐、左对齐或者右对齐。左对齐是默认的对齐方式，若没有特别指出，文本对齐模式是默认模式。例如，一个简单的
元素开始一个新行，并采用左对齐模式。

MODE= *(WRAP|NOWRAP)*

此属性规定了后续文本行的换行模式。WRAP规定了断开文本模式，NOWRAP规定了不断开模式，若没有具体指出，这一行的换行模式与文本流中前一行使用的模式相同。例如，一个简单的
元素开始一个新行，但不改变当前行的换行模式。

换行示例

下述示例将表明BR元素是如何影响文本的对齐和换行模式的：

```
<WML>
  <CARD>
    line 1, three-line card       <!-left alignment, breaking mode ->
    <BR ALIGN="RIGHT"   />line 2 <!-right alignment, breaking mode ->
    <BR MODE="NOWRAP"/>line 3     <!-left alignment, non-breaking mode ->
  </CARD>
  <CARD>
    <BR ALIGN="CENTER"/>
    line 1, one-line card         <!-centre alignment, breaking mode ->
  </CARD>
  <CARD>
    <BR MODE="NOWRAP"/>
    <BR ALIGN="CENTER" />
    line 2, two-line card         <!-centre alignment, non-breaking mode ->
  </CARD>
</WML>
```

下面的这个示例给出了一个更复杂的卡片、文本的对齐和换行模式：

```
<WML>
  <CARD>
    <FIELDSET>
      line 1                      <!-left alignment, breaking mode ->
      <BR ALIGN="NOWRAP"/>line 2 <!-left alignment, non-breaking mode ->
      <BR MODE="RIGHT"/>line 3    <!-right alignment, non-breaking mode ->
    </FIELDSET>
    <FIELDSET>
      Choose:                     <!-right alignment, non-breaking mode ->
      <SELECT KEY="X">
        <OPTION VALUE="1">One</OPTION>
```

```
        <OPTION VALUE="2">Two</OPTION>
      </SELECT>
      continuation of line 3      <!-right alignment, non-breaking mode ->
    </FIELDSET>
    <FIELDSET>
      still on line 3            <!-right alignment, non-breaking mode ->
      <INPUT KEY="Y"/>
      <BR MODE="WRAP"/>line 4    <!-left alignment, breaking mode ->
    </FIELDSET>
  </CARD>
</WML>
```

4. TAB元素

下面的元素定义了TAB栏:

```
<!ENTITY % tab       "TAB">
<!ENTITY % TAlign    "(LEFT|RIGHT|CENTER)" >
<!ELEMENT TAB EMPTY>
<!ATTLIST TAB
  ALIGN   %TAlign;   "LEFT"
  >
```

TAB元素用来在行中产生一个对齐的栏，它把标记分割成栏，但不规定栏的宽度。栏是按行的顺序给出，一个给定的行是由BR或其他的非文本 （%text）结束。元素可以结束一行。

一个栏组被定义为包含在TAB元素中的连续行的最大集合。 TAB元素可在文本流的任意一点形成，依靠显示特性，用户代理可以为每一个栏组产生对齐的栏，或使一张卡片中的所有栏组使用同一个对齐的栏。为确保最窄的显示宽度，用户代理应当根据那一栏的文本和图像的最大宽度决定每一栏的宽度。一个非空的分隔槽被用来分割每一个非空的栏。

在栏组中所有行的空栏可以被忽略，给定行可能比它的栏组中的其他行有更少的 TAB元素,此时它右边的栏被视为空。

属性

ALIGN= *(LEFT|RIGHT|CENTER)*

这个属性规定了在栏中的文本布局，在向用户显示时 ,文本可以被设为中心对齐、左对齐或者右对齐。左对齐是默认的模式。

TAB示例

下面的示例包含一张有单一栏组的卡片 ,此栏组有两栏三行：

```
<WML>
  <CARD>
    One <TAB/> Two <BR/>
    1     <TAB/>  <BR/>
    A     <TAB/> B      <BR/>
  </CARD>
</WML>
```

这个卡片接收到的版面为:

```
One    Two
```

```
1
A       B
```

下面的示例包含一张有两个栏组的卡片，它用一行文本分开，表明多栏组可以有独立的栏宽。

```
<WML>
  <CARD>
    alpha <TAB/> beta <BR/>
    gamma <TAB/> epsilon <BR/>
    this is a test<BR/>
    1 <TAB/> 2 <BR/>
    3 <TAB/> 4 <BR/>
  </CARD>
</WML>
```

这个卡片接收到的版面为：

```
alpha  beta
gamma  epsilon
this is a test...
1   2
3   4
```

下面的示例包含一张卡片，这个卡片有一个栏组、各种栏对齐方式和一个空的初始化栏：

```
<WML>
  <CARD>
   <TAB/>                       alpha  <TAB/>                beta         <BR/>
   <TAB/>                       gamma  <TAB/>                 epsilon     <BR/>
   <TAB/>                       <TAB ALIGN= "RIGHT"/>       2            <BR/>
   <TAB ALIGN= "CENTER"/> 3 <TAB ALIGN= "CENTER"/>     4            <BR/>
  </CARD>
</WML>
```

此卡片接收到的版面显示为：

```
alpha       beta
gamma       epsilon
                 2
    3            4
```

4.8.9 图像

```
<!ENTITY % IAlign "(TOP|MIDDLE|BOTTOM)" >
<!ELEMENT IMG EMPTY>
<!ATTLIST IMG
  ALT           %vdata;           #IMPLIED
  SRC           %URL;             #IMPLIED
  LOCALSRC      %vdata;            #IMPLIED
  VSPACE        %length;          "0"
  HSPACE        %length;          "0"
  ALIGN         %IAlign;          "BOTTOM"
```

```
HEIGHT          %length;          #IMPLIED
WIDTH           %length;          #IMPLIED
>
```

IMG元素定义了包含在文本流中的图像，图像的安排通常是在文本的排版中完成的。

属性：

ALT= *vdata*

该属性定义了一种图像的文本表示方法。当图像无法显示（如，用户代理不支持图像，或没有找到图像内）时，使用这种文本表示。

SRC= *URL*

该属性指定了图像的URL。如果浏览器支持图像格式，从给定的URL下载图像，并在显示文本的时候渲染这个图像。

LOCALSRC= *vdata*

该属性定义了一种图像的内部显示方法。如果该属性存在，则这个显示方法被使用；否则，图像从SRC属性指定的URL中下载。由LOCALSRC参数指定的图像比SRC参数指定的图像优先级高。

VSPACE= *length*
HSPACE= *length*

这些属性定义了在一个图像或对象的左右（HSPACE）、上下（VSPACE）插入的空白格的数量，这个属性没有规定默认值，但通常是小的、非零长度。如果长度以百分比给出，最终的数量与可用的水平和垂直空间有关，而不取决图像本身的大小。这些属性对用户代理是隐含的，可以被忽略。

ALIGN=(*TOP* | *MIDDLE* | *BOTTOM*)

该属性规定了在文本流中图像的对齐方式，或相对于当前插入点的对齐方式。ALIGN有三种可能值：

BOTTOM 图像底与当前基线垂直对齐，这是默认值。

MIDDLE 图像中心应与当前文本线中心垂直对齐。

TOP 图像顶部应与当前文本线的顶部垂直对齐。

HEIGHT= *length*
WIDTH= *length*

这些属性为用户代理提供了了解图像或对象的尺寸的方法，以便能够为其预留出空间，并能在等待图像数据时，连续地显示卡片，用户代理可衡量对像或图像是否与这些属性恰当的匹配。如果以百分比定义长度，最终的尺寸与可用的水平和垂直空间有关，而不取决于与图像本身的大小。这些属性对用户代理是隐含的，可以被忽略。

4.9 用户代理的语义

4.9.1 页面接入控制

在WML中引入变量，可能会出现在其他的标记语言，如HTML中不会存在的、潜在的安全性问题，特别是一些变量的状态可被认为是用户私有的。用户通常希望向一个安全服务发

送一个私有信息时，不安全和恶意的服务不能用其他的手段从用户代理中得到这个信息。

符合规范的WML用户代理必须能执行页面级的访问控制，包括 ACCESS元素、PUBLIC、SENDREFERER、DOMAIN和PATH属性。

WML的作者应该从浏览器的上下文中清除私有和敏感的信息，这可以通过清除包含这些信息的变量完成。

4.9.2 低存储特性

WML的目标是要在硬件资源有限的设备上使用，这包括存储容量受到极大限制的设备。因此，在出现错误的情况下（包括由于缺乏内存引起的错误），作者对设备特性的正确估计是非常重要的。

1. 历史记录的限制

用户代理可以限制历史记录堆栈的大小，如历史记录导航信息的深度，在存放历史记录的内存被耗尽的情况下，用户代理应该删除最近使用过的历史信息。建议所有用户代理使用的最小历史堆栈尺寸是10条。

2. 浏览器上下文大小的限制

在一些情况下，创作者或许在浏览器的上下文中定义了过多的变量，导致内存耗尽。

在这种情况下，用户代理可以通过4.9.2节“历史记录的限制”中描述的收回缓存和历史记录内存的方法，试着获得附加的内存。如果无效，用户代理耗尽了所有的内存，会向用户通报发生了错误，这时必须根据对用户状态的预测，重新设置用户代理。例如，浏览器可以被终止，或上下文被清除，或浏览器被重新设置到已知的状态。

4.9.3 错误处理

符合规范的用户代理必须强制执行在本规范中规定的错误条例，不能试图通过推断作者或源服务器的意图来掩盖错误。

4.9.4 未知的DTD

用其他DTD编码的WML页面可以包含用户代理无法识别的元素和属性。用户代理应该能在这种情况下显示这个页面，就好像无法识别的标签和属性不存在一样，具有无法识别元素的内容也应该被显示。

4.9.5 卡间导航的参考处理特性

下面过程描述了用于WML卡间导航的参考模型，所有用户代理必须执行该过程，或是一个与它相类似的过程。

1. GO任务

执行GO任务的过程包含以下的步骤：

1) 如果源任务包含了VAR元素，每个VAR元素的变量名称和值被转换成通过替换所有的参考变量而得到的一个字串。变量名称和值的最终结果存储在一个临时的内存中，以便以后处理。有关变量替代的更多信息，参见4.7.3节。

2) 目标URL由用户代理标别和获取，URL的属性值被转换成通过替换所有的参考变量而

得到的一个单独的字串。

3) 按4.8.3节“ ACCESS元素 ”中规定的方法，处理获取到的页面访问控制参数。

4) 使用在URL中给定的片段名称来定位目标卡片。

- 如果片段名称没有作为URL的一部分被指定，页面中的第一张卡片就作为目标卡片。
- 如果片段名称被指定并且卡片的NAME属性和该片段名称是一样的，那么这张卡片就是目标卡片。
- 如果片段名称不能与给定的卡片相关联，页面中的第一张卡片就是目标卡片。

5) 如果目标卡片中包含了NEWCONTEXT属性，按4.7.2节描述的方法重新初始化当前的浏览器上下文。

6) 从步骤(1)（VAR元素）处理结果得到的变量分配应该用在当前浏览器的上下文中。

7) 目标卡片被压入历史记录栈。

8) 如果目标卡片规定了ONENTERFORWARD的内部事件绑定，那么与事件绑定相关的任务或者被执行，或者过程停止。更多的信息，参见4.6.8节。

9) 如果目标卡片包含了TIMER元素，按4.8.7节中规定的那样启动计时器。

10) 利用当前变量的状态来显示目标卡片，然后过程结束。

2. PREV任务

执行PREV任务的过程包含以下步骤：

1) 如果源任务包含了VAR元素，每个VAR元素的变量名称和值被转换成通过替换所有的参考变量而得到的一个字串。变量名称和值的最终结果存储在一个临时的内存中，以便以后处理。有关变量替代的更多信息，参见4.7.3节。

2) 目标URL由用户代理标别和获取。历史记录栈被弹出，目标 URL被置于历史记录栈顶层，如果在历史记录栈中没有先前的卡片，过程停止。

3) 按4.8.3节“ ACCESS元素 ”中规定的方法，处理获取到的页面的访问控制参数。

4) 使用在URL中给定的片段名称来定位目标卡片。

- 如果片段名称没有作为URL的一部分被指定，页面中的第一张卡片就作为目标卡片。
- 如果片段名称被指定并且卡片的NAME属性和该片段名称是一样的，那么这张卡片就是目标卡片。

5) 从步骤(1)（VAR元素）处理结果得到的变量分配应该用在当前浏览器的上下文中。

6) 如果目标卡片规定了ONENTERFORWARD的内部事件绑定，那么与事件绑定相关的任务或者被执行，或者过程停止。更多的信息，参见4.6.8节。

7) 如果目标卡片包含了TIMER元素，按4.8.7节中规定的那样启动计时器。

8) 利用当前变量的状态来显示目标卡片，然后过程结束。

3. NOOP任务

没有任何过程被用来完成NOOP任务。

4. REFRESH任务

执行REFRESH任务的过程包含了以下几步：

1) 对于每一个VAR元素，通过替换所有的参考变量，它的变量名称和值被转换而得到一个字串。变量名称和值的最终结果存储在一个临时的内存中，以便于今后的处理。有关变量替代的更多信息，参见4.7.3节。

2) 从步骤(1)（VAR元素）处理结果得到的变量分配应该用在当前浏览器的上下文中。

3) 利用当前变量的状态再次显示当前的卡片，然后过程结束。

5. 任务执行失败

如果一个任务获取其目标 URL失败，或者访问控制限制了卡间转换的成功，用户代理必须向用户通报，并且采取以下措施：

- 保持当前的卡片。
- 保持浏览器上下文不变，包括任何未决的变量分配或 NEWCONTEXT处理。
- 不执行任何内部的事件绑定。

4.10 WML参考信息

WML是[XML]1.0版本的应用。

4.10.1 文档标识符

这些标识符还没有向IANA或ISO 90070进行注册。

1. SGML公共标识符

```
-//WAPFORUM//DTD WML 1.0//EN
```

2. WML媒体类型

文本格式：

```
text/x-wap.wml
```

标记格式：

```
application/x-wap.wmlc
```

这些标识还没有用IANA注册，因此是实验性的媒体类型。

4.10.2 文档类型定义

```
<!-
Wireless Markup Language (WML) Document Type Definition.
WML is an XML language. Typical usage:
   <?xml version="1.0"?>
   <!DOCTYPE WML PUBLIC "-//WAPFORUM//DTD   WML   1.0//EN"
                        "http://www.wapforum.org/DTD/wml.xml">
   <WML>
   ...
   </WML>
->
<!ENTITY % length      "CDATA">      <!-[0-9]+ for pixels or [0-9]+"%" for
percentage length ->
<!ENTITY % vdata       "CDATA">      <!-attribute value possibly containing
variable references ->
<!ENTITY % URL         "%vdata;">    <!-URL or URN designating a hypertext
node. May contain variable->
references ->
```

```
<!ENTITY % boolean "(TRUE|FALSE)">
<!ENTITY % number  "NMTOKEN">  <!-a number, with format [0-9]+ ->
<!ENTITY % emph    "EM | STRONG | B | I | U | BIG | SMALL">
<!ENTITY % tab     "TAB">
<!ENTITY % layout  "BR">
<!ENTITY % text     "#PCDATA | %emph; | %tab;">
<!ENTITY % inline   "%text; | %layout;">
<!-flow covers "card-level" elements, such as text and images ->
<!ENTITY % flow     "%inline; | IMG | A">
<!-Task types ->
<!ENTITY % task   "GO | PREV | NOOP | REFRESH">
<!-Navigation and event elements ->
<!ENTITY % navelmts "DO | ONEVENT">
<!-============== Decks and Cards ================-->
<!ELEMENT WML ( HEAD?, TEMPLATE?, CARD+ )>
<!ATTLIST WML
xml:lang        NMTOKEN#IMPLIED
  >
<!-card intrinsic events ->
<!ENTITY % cardev
 "ONENTERFORWARD          %URL;        #IMPLIED
  ONENTERBACKWARD         %URL;        #IMPLIED
  ONTIMER                 %URL;        #IMPLIED"
  >
<!-CARD field types ->
<!ENTITY % fields  "% flow; | INPUT | SELECT | FIELDSET">
<!ELEMENT CARD (%fields; | %navelmts; | TIMER)*>
<!ATTLIST CARD
  NAME                    NMTOKEN      #IMPLIED
  TITLE                   %vdata;      #IMPLIED
  NEWCONTEXT              %boolean;    "FALSE"
  STYLE                   (LIST|SET)   "LIST"
  %cardev;
  >
<!-============== Event Bindings ================-->
<!ELEMENT DO (%task;)>
<!ATTLIST DO
  TYPE          CDATA       #REQUIRED
  LABEL         %vdata;     #IMPLIED
  NAME          NMTOKEN     #IMPLIED
  OPTIONAL      %boolean;   "FALSE"
  >
<!ELEMENT ONEVENT (%task;)>
<!ATTLIST ONEVENT
TYPE             CDATA       #REQUIRED
  >
<!-============== Deck-level declarations ================-->
<!ELEMENT HEAD ( ACCESS | META )+>
<!ELEMENT TEMPLATE (%navelmts;)*>
```

```
<!ATTLIST TEMPLATE
%cardev;
  >
<!ELEMENT ACCESS EMPTY>
<!ATTLIST ACCESS
  DOMAIN                CDATA          #IMPLIED
  PATH                  CDATA          #IMPLIED
  PUBLIC                %boolean;      "FALSE"
  >
<!ELEMENT META EMPTY>
<!ATTLIST META
  HTTP-EQUIV            CDATA          #IMPLIED
  NAME                  CDATA          #IMPLIED
  USER-AGENT            CDATA          #IMPLIED
  CONTENT               CDATA          #REQUIRED
  SCHEME                CDATA          #IMPLIED
  >
<!-============== Tasks ================-->
<!ELEMENT GO (VAR)*>
<!ATTLIST GO
  URL                   %URL;          #REQUIRED
  SENDREFERER           %boolean;      "FALSE"
  METHOD                (POST|GET)     "GET"
  ACCEPT-CHARSET        CDATA          #IMPLIED
  POSTDATA              %vdata;        #IMPLIED
  >
<!ELEMENT PREV (VAR)*>
<!ELEMENT REFRESH (VAR)+>
<!ELEMENT NOOP EMPTY>
<!-============== VAR ================-->
<!ELEMENT VAR EMPTY>
<!ATTLIST VAR
  NAME                  %vdata;        #REQUIRED
  VALUE                 %vdata;        #REQUIRED
  >
<!-============== CARD Fields ================-->
<!ELEMENT SELECT (OPTGROUP|OPTION)+>
<!ATTLIST SELECT
  TITLE                 %vdata;        #IMPLIED
  KEY                   NMTOKEN        #IMPLIED
  DEFAULT               %vdata;        #IMPLIED
  IKEY                  NMTOKEN        #IMPLIED
  IDEFAULT              %vdata;        #IMPLIED
  MULTIPLE              %boolean;      "FALSE"
  TABINDEX              %number;       #IMPLIED
  >
<!ELEMENT OPTGROUP (OPTGROUP|OPTION)+ >
<!ATTLIST OPTGROUP
TITLE                   %vdata;        #IMPLIED
```

```
  >
<!ELEMENT OPTION (%text; | ONEVENT)*>
<!ATTLIST OPTION
  VALUE                 %vdata;       #IMPLIED
  TITLE                 %vdata;       #IMPLIED
  ONCLICK               %URL;         #IMPLIED
  >
<!ELEMENT INPUT EMPTY>
<!ATTLIST INPUT
  KEY                NMTOKEN          #REQUIRED
  TYPE               (TEXT|PASSWORD)  "TEXT"
  VALUE              %vdata;          #IMPLIED
  DEFAULT            %vdata;          #IMPLIED
  FORMAT             CDATA            #IMPLIED
  EMPTYOK            %boolean;        "FALSE"
  SIZE               %number;         #IMPLIED
  MAXLENGTH          %number;         #IMPLIED
  TABINDEX           %number;         #IMPLIED
  TITLE              %vdata;          #IMPLIED
  >
<!ELEMENT FIELDSET (%fields;)* >
<!ATTLIST FIELDSET
TITLE                 %vdata;       #IMPLIED
  >
<!ELEMENT TIMER EMPTY>
<!ATTLIST TIMER
  KEY                 NMTOKEN       #IMPLIED
  DEFAULT             %vdata;       #REQUIRED
  >
<!-================ Images ================->
<!ENTITY % IAlign "(TOP|MIDDLE|BOTTOM)" >
<!ELEMENT IMG EMPTY>
<!ATTLIST IMG
  ALT                 %vdata;       #IMPLIED
  SRC                 %URL;         #IMPLIED
  LOCALSRC            %vdata;       #IMPLIED
  VSPACE              %length;      "0"
  HSPACE              %length;      "0"
  ALIGN               %IAlign;      "BOTTOM"
  HEIGHT              %length;      #IMPLIED
  WIDTH               %length;      #IMPLIED
  >
<!-================ Anchor ================->
<!ELEMENT A ( %inline; | GO | PREV | REFRESH )*>
<!ATTLIST A
TITLE                 %vdata;       #IMPLIED
  >
<!-================ Text layout and line breaks ================->
<!-Text alignment attributes ->
```

```
<!ENTITY % TAlign   "(LEFT|RIGHT|CENTER)" >
<!ELEMENT TAB EMPTY>
<!ATTLIST TAB
ALIGN   %TAlign;   "LEFT"
  >
<!ELEMENT EM            (%flow;)*>
<!ELEMENT STRONG        (%flow;)*>
<!ELEMENT B             (%flow;)*>
<!ELEMENT I             (%flow;)*>
<!ELEMENT U             (%flow;)*>
<!ELEMENT BIG           (%flow;)*>
<!ELEMENT SMALL         (%flow;)*>
<!ENTITY % BRMode       "(WRAP|NOWRAP)" >
<!ELEMENT BR EMPTY>
<!ATTLIST BR
  ALIGN                 %TAlign;      "LEFT"
  MODE                  %BRMode;      #IMPLIED
  >
<!ENTITY quot           "&#34;">      <!-quotation mark ->
<!ENTITY amp            "&#38;#38;">  <!-ampersand ->
<!ENTITY apos           "&#39;">      <!-apostrophe ->
<!ENTITY lt             "&#38;#60;">  <!-less than ->
<!ENTITY gt             "&#62;">       <!-greater than ->
<!ENTITY nbsp           " ">     <!-non-breaking space ->
<!ENTITY shy            "&#173;">     <!-soft hyphen (discretionary hyphen) ->
```

4.11 WML紧凑二进制表示

WML可使用简洁的二进制格式进行编码，该内容格式是基于 WAP二进制XML文本格式[WBXML]。

4.11.1 扩展记号

1. 全局扩展记号

[WBXML]全局扩展记号被用来表示WML变量，变量的引用会出现在WML页面中的许多地方（参见4.7.3节）。有几种代码用来表示变量的替代，每个代码有不同的转义（ escape）语义（例如，直接替代、转义替代和非转义替代）。变量名用当前文档字符集编码，并且必须使用源文档中规定的方式编码（如，变量名不能被截断、映射或作其他的改变）。例如：全局转义扩展标记EXT_I_0代表了用内联变量名进行的一种转义变量替代。

2. 标签记号

WML定义了一整套的单字节记号，与定义在 DTD中的标签相对应，所有的这些记号定义在零代码页中。

3. 属性记号

WML定义了一整套的单字节记号，与定义在 DTD中的属性名和属性值相对应，所有的这些记号定义在零代码页中。

4.11.2 编码语义

1. 编码变量

所有变量引用必须被转换成变量引用记号（如， EXT_I_0）。

2. 文档的有效性

在标记WML页面的过程中，XML文档有效性的问题（见[XML]）会出现，文档有效性的确认必须以WML页面中的DOCTYPE声明为基础。在确认源文本时，如果文档作为 WML媒体类型，标记过程必须能接受任何的 DOCTYPE或公共标识（见4.10.1节中“ WML媒体类型 ”）。

标记过程在源页面中检测到的任何格式和有效性错误，应该向用户通报。

(1) 有效的 %length;

WML标记化过程必须确认被定义为 %length；的属性值，它包含了一个 NMTOKEN，或者是在NMTOKEN后面跟着一个百分号字符。例如，下面的属性是合法的：

```
VSPACE="100%"
HSPACE="123"
```

利用普通的属性值编码方法，可以对 %length；数据进行编码。

(2) 有效 %vdata；

WML标记化过程必须确认被定义为 %vdata;的属性值，它包含变量和其他的 CDATA属性值没有的变量，没有在DTD中定义的属性值必须使用变量引用。

4.11.3 数字常量

1. WML扩展记号分配

在表4-4中的全局扩展记号用在 WML中，且在全局记号范围内占用了文档类型说明的记号位置，与全局范围内所有记号一样，这些代码必须在每个代码页中保留，其所有的数字都是十六进制的。

2. 标签记号

在表4-5中的记号代码表示零代码页中的标签，其中所有的数字都是十六进制的。

3. 属性开始记号

在表4-6中的记号代码表示零代码页中的属性开始，其中所有的数字都是十六进制的。

4. 属性值标记

在表4-7中的记号代码表示零代码页中的属性值，其中所有的数字都是十六进制的。

表4-4 全局扩展记号的分配

记号名称	记 号	描 述
EXT_I_0	40	变量替代，被转义。变量名是内联，后面跟着 termstr记号
EXT_I_1	41	变量替代，不能被转义。变量名是内联，后面跟着 termstr记号
EXT_I_2	42	变量替代，不能转换。变量名是内联，后面跟着 termstr记号
EXT_T_0	80	变量替代，被转义。变量名被编码成一个对字串表的引用
EXT_T_1	81	变量替代，不能被转义。变量名被编码成一个对字串表的引用
EXT_T_2	82	变量替代，不能转换。变量名被编码成一个对字串表的引用
EXT_0	C0	保留以备用
EXT_1	C1	保留以备用
EXT_2	C2	保留以备用

表4-5 标签记号

标签名称	记号值	标签名称	记号值
A	22	NOOP	31
ACCESS	23	PREV	32
B	24	ONEVENT	33
BIG	25	OPTGROUP	34
BR	26	OPTION	35
CARD	27	REFRESH	36
DO	28	SELECT	37
EM	29	SMALL	38
FIELDSET	2A	STRONG	39
GO	2B	TAB	3A
HEAD	2C	TEMPLATE	3B
I	2D	TIMER	3C
IMG	2E	U	3D
INPUT	2F	VAR	3E
META	30	WML	3F

表4-6 属性开始记号

属性名称	属性值前缀	记号
ACCEPT-CHARSET		5
ALIGN	BOTTOM	6
ALIGN	CENTER	7
ALIGN	LEFT	8
ALIGN	MIDDLE	9
ALIGN	RIGHT	A
ALIGN	TOP	B
ALT		C
CONTENT		D
DEFAULT		E
DOMAIN		F
EMPTYOK	FALSE	10
EMPTYOK	TRUE	11
FORMAT		12
HEIGHT		13
HSPACE		14
IDEFAULT		15
IKEY		16
KEY		17
LABEL		18
LOCALSRC		19
MAXLENGTH		1A
METHOD	GET	1B
METHOD	POST	1C
MODE	NOWRAP	1D
MODE	WRAP	1E
MULTIPLE	FALSE	1F

(续)

属性名称	属性值前缀	记 号
MULTIPLE	TRUE	20
NAME		21
NEWCONTEXT	FALSE	22
NEWCONTEXT	TRUE	23
ONCLICK		24
ONENTERBACKWARD		25
ONENTERFORWARD		26
ONTIMER		27
OPTIONAL	FALSE	28
OPTIONAL	TRUE	29
PATH		2A
POSTDATA		2B
PUBLIC	FALSE	2C
PUBLIC	TRUE	2D
SCHEME		2E
SENDREFERER	FALSE	2F
SENDREFERER	TRUE	30
SIZE		31
SRC		32
STYLE	LIST	33
STYLE	SET	34
TABINDEX		35
TITLE		36
TYPE		37
TYPE	ACCEPT	38
TYPE	DELETE	39
TYPE	HELP	3A
TYPE	PASSWORD	3B
TYPE	ONCLICK	3C
TYPE	ONENTERBACKWARD	3D
TYPE	ONENTERFORWARD	3E
TYPE	ONTIMER	3F
TYPE	OPTIONS	45
TYPE	PREV	46
TYPE	RESET	47
TYPE	TEXT	48
TYPE	vnd.	49
URL		4A
URL	http://	4B
URL	https://	4C
USER-AGENT		4D
VALUE		4E
VSPACE		4F
WIDTH		50
xml:lang		51

表4-7 属性值记号

属 性 值	记 号 值	属 性 值	记 号 值
.com/	85	NOWRAP	94
.edu/	86	ONCLICK	95
.net/	87	ONENTERBACKWARD	96
.org/	88	ONENTERFORWARD	97
ACCEPT	89	ONTIMER	98
BOTTOM	8A	OPTIONS	99
CLEAR	8B	PASSWORD	9A
DELETE	8C	RESET	9B
HELP	8D	SET	9C
http://	8E	TEXT	9D
http://WWW.	8F	TOP	9E
https://	90	UNKNOWN	9F
https://WWW.	91	WRAP	A0
LIST	92	www.	A1
MIDDLE	93		

4.11.4 WML编码示例

有关其他的示例，请参阅[WBXML]。

下面是标记WML页面的另一个例子，它演示了变量编码、属性编码、字串表的使用。源页面为：

```
<WML>
   <CARD NAME="abc" STYLE="LIST">
   <DO TYPE="ACCEPT">
      <GO URL="http://xyz.org/s"/>
   </DO>
   X: $(X)<BR/>
   Y: $(&#x59;)<BR MODE="NOWRAP"/>
   Enter name: <INPUT TYPE="TEXT" KEY="N"/>
   </CARD>
</WML>
```

标记格式（十六进制数）如下：这个例子仅使用了内联的字符串并且假定字符编码使用NULL结束字符串格式。假定编码字符集是UTF-8。

```
00   02   04   'X'   00   'Y'   00   7F   E8   21   03   'a'  'b'  'c'  00
33   01   E9   38    01   AD    4B   03   'x'  'y'  'z'  00   88   03   's'
00   01   03   ' '   'X'  ':'   ' '  00   82   00   27   03   ' '  'Y'  ':'
' '  00   82   02    A7   1D    01   03   ' '  'E'  'n'  't'  'e'  'r'  ' '
'n'  'a'  'm'  'e'   ':'  ' '   00   B1   48   18   03   'N'  00   01   01
01
```

有关格式扩展和注释的格式，参见表4-8。

表4-8 记号页面示例

记 号 流	描 述
00	WBXML版本号
02	WML公共ID
04	字符串表的长度
‘X’ ,00,‘ Y’ ,00	字符串表
7F	带有内容的WML
E9	带有内容和属性的CARD
21	NAME=
03	后面跟着内联字符串
‘a’,‘ b’ ,‘c’ ,00	字符串
33	类型=“ LIST ”
01	END（CARD属性列表）
EA	带有内容和属性的DO
38	类型=ACCEPT
01	END（DO属性列表）
AD	带有属性的GO
4B	URL=“ http:// ”
03	后面跟着内联字符串
‘x’ ,' y' ,' :' ,' ',00	字符串
88	“ .org/ ”
03	后面跟着内联字符串
‘0	字符串
01	END（DO的元素）
03	后面跟着内联字符串
‘ ‘,' x' ,' :' ,' ',00	字符串
82	直接变量引用（EXT_T_2）
00	变量偏移值0
28	BR
03	后面跟着内联字符串
‘ ’ ,' Y',':',' ',00	字符串
82	直接变量引用（EXT_T_2）
02	变量偏移值2
A7	带属性的BR
1D	模式=“ NOWRAP ”
01	END（BR属性列表中的）
03	后面跟着内联字符串
‘ ’,'E','n','t','e','r',' ','n','a','m','e',':',' ',00	字符串
B1	带有属性的INPUT
48	类型="TEXT"
18	KEY=
03	后面跟着内联字符串
‘ N ’ ,00	字符串
01	END（INPUT属性列表中的）
01	END（CARD 元素的）
01	END（WML元素的）

4.12 术语定义

本规范采用了下列术语：

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。

卡片(Card) 一个单独的WML导航和用户接口的单元。它可能包含显示给用户的信息、收集用户输入的指令等等。

客户端（Client） 客户端是向服务器发出连接请求的设备或应用程序。

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。

内容编码（Content Encoding） 当被用作动词时，内容编码指的是把数据对象从一种格式转换为另外一种格式的行为。通常，目标格式需要的物理空间比原格式要少，更易于处理或存储，和/或被加密。当被用作名词时，内容编码指的是一种特殊的格式或编码的标准或处理。

内容格式（Content Format） 内容的实际表示。

页面（Deck） 一组WML卡片的集合。一个WML页面也即一个XML文件。

设备（Device） 一个网络实体，能够发送和接收信息包的，并且有一个唯一的地址。在一个给定的上下文或跨越多重上下文，一个设备既可作为客户端，也可作为服务器。例如，一个设备作为其他服务器的客户端时可充当其他客户端的服务器。

Java脚本（JavaScript） Java脚本是一种实际的标准语言，用于向 HTML文档添加动态行为，它是ECMA脚本（ECMAScript）的起源技术之一。

人机界面（Man-Machine Interface） 与用户接口同义。

源服务器（Origin server） 它作为一种服务器，是给定资源（或称内容）存储或将被生成的地方，通常被看作是Web服务器或HTTP服务器。

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。

服务器（Server） 是一种被动地等待一个或多个客户端连接请求的设备（或应用程序）它可以接受或拒绝来自客户端的连接请求。

标准通用标记语言（SGML） （Standardized Generalized Markup Language）（它被定义于[ISO8879]）是一种为特定领域标记语言开发的通用语言。

终端(Terminal) 终端向用户提供了用户代理的能力，它具有发出请求和接收信息的能力，也被称作移动终端或移动台。

代码转换(Transcode) 是从一种字符集到另外一种字符集的转换行为（如从 UCS-2到UTF-8的转换）。

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） 用户代理是可以解释WML、WMLScript 或其他内容的软件和设备，它包括文本浏览器、语音浏览器和搜索引擎等。

无线标记语言脚本（WMLScript） 用来对移动设备进行编程的一种脚本语言，它是JavaScript脚本语言的扩展子集。

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

4.13 缩略语

在本规范用到了如下缩写：

BNF	Backus-Naur Form	Backus-Naur窗体格式
HDML	Handheld Markup Language [HDML2]	手持标记语言
HTML	HyperText Markup Language [HTML4]	超文本标记语言
HTTP	HyperText Transfer Protocol [RFC2068]	超文本传输协议
IANA	Internet Assigned Number Authority	因特网域名分配权威机
MMI	Man-Machine Interface	人-机接口
PDA	Personal Digital Assistant	个人数字助理
RFC	Request For Comments	请求注释
SGML	Standardized Generalized Markup Language [ISO8879]	标准通用标记语言
UI	User Interface	用户接口
URL	Uniform Resource Locator [RFC1738]	统一资源定位器
URN	Uniform Resource Name	统一资源域名
W3C	World Wide Web Consortium	万维网联盟
WAE	Wireless Application Environment [WAE]	无线应用开发环境
WAP	Wireless Application Protocol [WAP]	无线应用协议
WSP	Wireless Session Protocol [WSP]	无线会话协议
XML	Extensible Markup Language [XML]	可扩展标记语言

4.14 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

[RFC1738] "Uniform Resource Locators (URL)",T. Berners-Lee, et al., December 1994
URL: ftp://ds.internic.net/rfc/rfc1738.txt

[RFC1808] "Relative Uniform Resource Locators",R. Fielding, June 1995
URL: ftp://ds.internic.net/rfc/rfc1808.txt

[RFC2045] "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies",N. Freed, et al., November 1996
URL: ftp://ds.internic.net/rfc/rfc2045.txt

[RFC2048] "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures",N. Freed, et al., November 1996
URL: ftp://ds.internic.net/rfc/rfc2048.txt

[RFC2068] "Hypertext Transfer Protocol—HTTP/1.1,"R. Fielding, et al., January

1997
URL: ftp://ds.internic.net/rfc/rfc2068.txt

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

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

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

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

[WBXML] "WAP Binary XML Content Format",WAP Forum, 30-April-1998
URL: http://www.wapforum.org/

[WSP] "Wireless Session Protocol",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

4.15 参考资料

[HDML2] "Handheld Device Markup Language Specification",P. King, et al., April 11, 1997
URL: http://www.uplanet.com/pub/hdml_w3c/hdml20-1.html

[HTML4] "HTML 4.0 Specification, W3C Recommendation 18-December-1997, REC-HTML40-971218",D. Raggett, et al., September 17, 1997
URL: http://www.w3.org/TR/REC-html40

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