可扩展标记语言(XML) 1.0

W3C 建议 1998 年 2 月 10 日

本版本:
http://www.w3.org/TR/1998/REC-xml-19980210
http://www.w3.org/TR/1998/REC-xml-19980210.xml
http://www.w3.org/TR/1998/REC-xml-19980210.html
http://www.w3.org/TR/1998/REC-xml-19980210.pdf
http://www.w3.org/TR/1998/REC-xml-19980210.ps

最新版本:
http://www.w3.org/TR/REC-xml

上一版本:
http://www.w3.org/TR/PR-xml-971208

编者:
Tim Bray (Textuality and Netscape) <tbray@textuality.com>
Jean Paoli (Microsoft) <jeanpa@microsoft.com>
C. M. Sperberg-McQueen (University of Illinois at Chicago) <cmsmcq@uic.edu>

摘要

本文档完整地描述了可扩展标记语言(Extensible Markup Language，XML)，它是标准通用标记语言(Standard Generic Markup Language，SGML)的一个子集。其目的在于使得在 Web 上能以现有超文本标记语言(Hypertext Markup Language，HTML)的使用方式提供，接收和处理通用的 SGML 成为可能。XML 的设计既考虑了实现的方便性，同时也顾及了与 SGML 和 HTML 的互操作性。

本文档的状态

本文档已由 W3C 组织成员和其他相关各方审阅，并已被组织理事批准为 W3C 建议。这是一个稳定的文档，可以用作参考材料，也可以作为其他文档的正式参考文献。W3C 在建议制定过程中的作用是吸引对本规范的注意并促进它的广泛使用。这能增强 Web 的功能和互操作性。

本文档规定了一种用于 World Wide Web 的语法，此语法是通过取一个业已存在并已广泛使用的文本处理国际标准(标准通用标记语言，经增补和更正的 ISO 8879:1986(E))的子集而创建的。它是 W3C XML 行动组(XML Activity)的工作成果，关于 XML 行动组的详细信息可以在 http://www.w3.org/XML 找到。在 http://www.w3.org/TR 可以找到现有 W3C 建议和其他技术文档的一个列表。

本规范中使用了[Berners-Lee 等人]定义的一个术语 URI，他们正在从事的的工作将更新[IETF RFC1738]和[IETF RFC1808]。

本规范的已知错误列表可以在 http://www.w3.org/XML/xml-19980210-errata 找到。

请将本文档中的错误报告给 xml-editor@w3.org。

1. 绪论

可扩展标记语言，缩写为 XML，描述了一类称为 XML 文档的数据对象，同时也部分地描述了处理这些数据对象的计算机程序的行为。XML 是 SGML(标准通用标记语言[ISO 8879])针对应用的一个子集，或者说是 SGML 的一种受限形式。根据定义，XML 文档是合乎规范的 SGML 文档。

XML 文档由称为实体的存储单元组成，实体包含解析数据或未解析数据。解析数据由字符组成，其中一些字符组成字符数据，另一些字符组成标记。标记中包含了对文档存储格式(storage layout)和逻辑结构的描述。XML 提供了一种机制用于约束存储格式和逻辑结构。

称为 XML 处理器的软件模块用于读取 XML 文档，存取其中的内容和结构。XML 处理器被设想为是为另一个称为应用的模块作处理。本规范从 XML 处理器应如何读取 XML 数据以及应向应用提供哪些信息的这两个方面，描述了要求 XML 处理器作出的动作。

1.1 开发者和开发目标

XML 由 XML 工作组(原先的 SGML 编辑审查委员会)开发，此工作组由 World Wide Web Consortium(W3C)在 1996 年主持成立。工作组由 Sun Microsystems 的 Jon Bosak 负责，同样由 W3C 组织的 XML SIG(Special Interest Group)(原先的 SGML 工作组)积极参与了 XML 工作组的工作。XML 工作组的成员在附录中给出。工作组与 W3C 的联系人是 Dan Connolly。

XML 的设计目标如下：

XML 应该可以直接在因特网(Internet)中使用。
XML 应该支持大量不同的应用。
XML 应该与 SGML 兼容。
处理 XML 文档的程序应该容易编写。
XML 中的可选项应尽可能少，理想状况下应为零。
XML 文档应该清晰明了，可读性强。
XML 应易于设计。
XML 的设计应该正式而且简洁。
XML 文档应易于创建。
XML 标记的简洁性较为次要。

本规范与其他相关的标准一起(Unicode 和 ISO/IEC 10646 定义了字符集，Internet RFC1766 定义了语言识别码，ISO 639 定义了语言名称代码，ISO 3166 定义了国家名称代码)，提供了理解 XML 版本 1.0 和创建相应计算机处理程序所需的所有信息。

1.2 术语

用于描述 XML 文档的术语在此规范的正文中定义。在这些定义中以及描述一个 XML 处理器的动作时，使用了下表中的术语：

可以(may)
允许合乎规范的文档和 XML 处理器按所描述的方式工作，但不要求必须如此。

必须(must)
要求合乎规范的文档和 XML 处理器按所描述的方式工作; 否则出现错误。

错误(error)
对本规范中的规则的违反; 其结果不确定。合乎规范的软件可以检测和报告错误，并可以从中恢复。

严重错误(fatal error)
合乎规范的 XML 处理器必须检测到，并向应用报告的一类错误。在遇到严重错误之后，处理器可以继续处理数据以发现更多的错误并可以向应用报告这些错误。为了支持错误的更正，处理器可以向应用提供文档中未经处理的数据(字符数据和标记的混合体)。但是，一旦检测到一个严重错误，处理器必须停止正常的处理(也就是说，它必须停止以正常的方式向应用提供与文档逻辑结构有关的数据和信息)。

由用户选择(at user option)
合乎规范的软件可以或者必须(取决于句子中的情态动词)按所描述的方式工作; 如果它满足这个条件，它必须同时提供用户一种手段，使得用户能够启用和禁用所描述的工作方式。

有效性约束(validity constraint)
适用于所有有效的 XML 文档的一种规则。违反有效性约束属于错误; 由用户选择，进行验证的 XML 处理器必须报告这些错误。

格式约束(well-formedness constraint)
适用于所有有效的 XML 文档的一种规则。违反格式约束属于严重错误。

匹配(match)
(对于字符串和名字：)被比较的两个字符串或名字必须完全相同。在 ISO/IEC 10646 中有多种可能表示方式的字符(例如，既有预定义(precomposed)形式和基字符(base)+变音符形式的字符)只在两个字符串中的表示方式相同时才匹配。由用户选择，处理器可以将这些字符规范成某种规范形式。不进行字符的大小写转换。(对于句法中的字符串和规则：)如果一个字符串属于一个句法产生式产生的语言，则它匹配这个产生式。(对于内容和内容模型：)当一个元素符合"元素有效性"约束中的描述时，它匹配其声明.

兼容性考虑(for compatibility)
仅用于保证与 SGML 兼容的 XML 特性。

互操作性考虑(for interoperability)
是一个不具约束性的建议，目的是增加 XML 文档能被在 ISO 8879 的 WebSGML 改编附件之前已有的 SGML 处理器处理的可能性。

2. 文档

如果一个数据对象满足本规范中格式良好的要求时，它是一个 XML 文档。一个规范的 XML 文档如果满足某些进一步的约束，它将更为有效。

每一个 XML 文档都有逻辑和物理结构。物理上而言，文档由称为实体的单元组成。一个实体可以引用(refer)其他实体，将它们包含在文档中。文档开始于"根(root)"或文档实体中。逻辑上而言，文档由声明、元素、注释、字符引用和处理指令组成，所有这些都在文档中用显式标记指明。逻辑和物理结构必须如"4.3.2 格式良好的解析实体"中所描述那样严格地嵌套。

2.1 格式良好的(Well-Formed)XML 文档

一个文本对象如果满足以下条件，它将是一个格式良好的 XML 文档：

1.作为一个整体，它匹配文档(document)产生式。
2.它满足本规范中定义的所有格式约束。
3.此文档中直接或间接引用的每一个解析实体都是格式良好的。

文档
[1] document ::= prolog element Misc*

匹配 document 产生式意味着：

1.它包含一个或多个元素.
2.有且仅有一个称为根(root)或文档元素的元素，它不出现在其他任何元素的内容(content)中。对于其他所有元素，如果起始标签在另一个元素的内容中，则其结束标签也在同一元素的内容中。换一个更简单的说法，以起始标签和结束标签为界的各个元素，必须严格地嵌套。

这样做的结果是，对于每一个非根的元素 C，文档中另有一个元素 P，C 在 P 的内容中，而不在其他任何被 P 所包含的元素的内容中。P 被称为 C 的父元素(parent)，而 C 被称为 P 的子元素(child)。

2.2 字符

一个解析实体包含文本(text)，文本是一个字符(character)序列，可以表示标记或字符数据。一个字符是 ISO/IEC 10646[ISO/IEC 10646]中定义的文本最小单元。合法的字符包括制表符，回车，换行以及 Unicode 和 ISO/IEC 10646 中定义的合法的图形字符。不提倡使用[Unicode]6.8 节中定义的"兼容字符(compatibility characters)"。

字符范围
[2] Char ::= #x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]
/* 除了替代块(surrogate block)，FFFE 和 FFFF 以外的任意 Unicode 字符。*/

将字符代码编码成位模型的机制各个实体间可能会有所不同。所有的 XML 处理器必须接受 10646 中的 UTF-8 和 UTF-16 编码；用于指出所用编码或指定使用其他编码的机制在后面的"4.3.3 实体中的字符编码"中讨论。

2.3 通用句法成分

本节中定义了一些在句法中广泛使用的符号。

S(空白域)包括一个或多个空格字符(#x20)，回车，换行或制表符。

空白
[3] S ::= (#x20 | #x9 | #xD | #xA)+

为方便起见，字符被分为字母，数字和其他字符三类。字母可以是字母表中的字母，或是一个音节基字符(syllabic base character)后跟一个或多个组合字符，也可以是一个表意字符。在"B. 字符的分类"中给出了每一类字符的特定定义。

名字(name)是以一个字母或某一标点符号开头的记号，后跟字母，数字，连字符，下划线，冒号或句号，这些符号统称为命名字符(name character)。以"xml"或其他任何以 (('X'|'x') ('M'|'m') ('L'|'l')) 的字符串开头的名字，被保留用于本规范的此版本或后续版本的标准化。

注意：XML 名字中的冒号被保留用于名域(name space)实验。它的含义有待于日后标准化，那时那些将冒号用于实验目的的文档有可能需要更新。(不保证 XML 采用的任何名字空间机制实际会采用冒号作为定界符。)实际上，这意味着除非用于名字空间实验，XML 文档编者不应该在 XML 名字中使用冒号，但 XML 处理器应该接受冒号作为一个命名字符。

Nmtoken(名字记号，name token)是任何命名字符的混合体。

名字和记号
[4] NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar | Extender

[5] Name ::= (Letter | '_' | ':') (NameChar)*
[6] Names ::= Name (S Name)*
[7] Nmtoken ::= (NameChar)+
[8] Nmtokens ::= Nmtoken (S Nmtoken)*

字面数据是任何用引号括起的字符串，不包括用作定界符的引号。字面数据用于指明内部实体的内容(EntityValue)，属性值(AttValue)，以及外部标识符(SystemLiteral)。注意，对 SystemLiteral 的解析可以不扫描标记。

字面数据
[9] EntityValue ::= '"' ([^%&"] | PEReference | Reference)* '"'
| "'" ([^%&'] | PEReference | Reference)* "'"
[10] AttValue ::= '"' ([^<&"] | Reference)* '"'
| "'" ([^<&'] | Reference)* "'"
[11] SystemLiteral ::= ('"' [^"]* '"') | "'" [^']* "'")
[12] PubidLiteral ::= '"' PubidChar* '"' | "'" (PubidChar - "'")* "'"
[13] PubidChar ::= #x20 | #xD | #xA | [a-zA-Z0-9] | [-'()+,./:=?;!*#@$_%]

2.4 字符数据和标记

文本由字符数据和标记混合构成。标记包括起始标记、结束标记、空元素标记、实体引用、字符引用、注释、CDATA 段定界符、文档类型声明和处理指令。

其他所有非标记的文本组成文档的字符数据。

"and"号(&)和左尖括号(<)只有作为标记定界符，或在注释，处理指令，或 CDATA 段中时才能以字面形式出现。它们在一个内部实体声明的字面实体数值中也是合法的，参见"4.3.2 格式良好的解析实体"。如果在其他地方需要用到这两个字符，它们必须用数值式字符引用来转义或分别用字符串"&"和"<"表示。右尖括号(>)可以用">"表示，而当它在内容中的字符串"]]>"中出现，但此字符串不表示一个 CDATA 段的结束时，出于兼容性考虑，必须用">"或一个字符引用转义得到。

在一个元素的内容中，字符数据可以是不包括任何标记的起始定界符的任意字符串。在一个 CDATA 段中，字符数据可以是不包括 CDATA 段结束定界符"]]>"的任意字符串。

为了允许在属性值中包含单引号和双引号，省略符或称单引号(')可以被表示为"'"，而双引号(")可以被表示为"""。

字符数据
[14] CharData ::= [^<&]* - ([^<&]* ']]>' [^<&]*)

2.5 注释

注释可以在其他标记之外的文档中的任何位置出现。另外，它们可以在文档类型声明中语法允许的地方出现。它们不是文档字符数据的一部分，XML 处理器可以，但不必须，允许一个应用检索注释文本。出于兼容性考虑，字符串"--"(双连字符)不能在注释中出现。

注释
[15] Comment ::= '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'

注释的一个例子：

<!-- declarations for <head> & <body> -->

2.6 处理指令

处理指令(PI)允许文档中包含由应用来处理的指令。

处理指令
[16] PI ::= '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>'
[17] PITarget ::= Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))

PI 不是文档字符数据的一部分，但必须传递给应用。PI 以用于指示传递给哪个应用的目标(PITarget)开头，目标名字"XML"，"xml"，等等，保留用于本规范的此版本或后续版本的标准化。XML 符号机制可以用于 PI 目标的形式化声明。

2.7 CDATA 段

CDATA 段可以出现在字符数据可以出现的任何地方，它们用于转义包含会被识别为标记的字符串的文本块。CDATA 段以字符串"<![CDATA["开始，以字符串"]]>"结束：

CDATA 段
[18] CDSect ::= CDStart CData CDEnd
[19] CDStart ::= '<![CDATA['
[20] CData ::= (Char* - (Char* ']]>' Char*))
[21] CDEnd ::= ']]>'

在一个 CDATA 段内，只有 CDEnd 字符串被识别为标记，因此左尖括号和"&"可以以它们的字面形式出现，不需要(也不能)被换码为"<"和"&"。CDATA 段不能嵌套。

一个 CDATA 段的例子，其中"<greeting>"和"</greeting>"被识别为字符数据，而不是标记：

<![CDATA[<greeting>Hello, world!</greeting>]]>

2.8 序言(prolog)和文档类型声明

XML 文档可以，也应该以一个 XML 声明开始，其中指明了所用 XML 的版本。 例如，以下是一个完整的 XML 文档，它是格式良好的，但不是有效的：

<?xml version="1.0"?>
<greeting>Hello, world!</greeting>

下面这个也同样：

<greeting>Hello, world!</greeting>

版本号"1.0"应该用于表明对与本规范的此版本相一致，如果使用了值"1.0"但又与本规范的此版本不一致，那么这是文档的一个错误。XML 工作组打算赋予本规范的后续版本不同于"1.0"的数值，但这并不代表开发后续版本的承诺，也不代表如果有后续版本，会使用任何特殊的命名方案的承诺。因为不排除有后续版本的可能性，提供了本构造(construct)作为一旦需要时进行自动版本识别的手段。当处理器收到的文档标有它们不支持的版本时，可以给出一个错误。

XML 文档中标记的功能是描述文档的存储格式和逻辑结构，并将属性-值对和逻辑结构关联起来。XML 提供一种称为文档类型声明的机制，用于定义对逻辑结构的约束，支持预定义存储单元的使用。如果一个 XML 文档有相应的文档类型声明并且它遵循其中的约束，则称它是有效的(valid)。

文档类型声明必须位于文档第一个元素之前。

序言
[22] prolog ::= XMLDecl? Misc* (doctypedecl Misc*)?
[23] XMLDecl ::= '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'
[24] VersionInfo ::= S 'version' Eq (' VersionNum ' | " VersionNum ")
[25] Eq ::= S? '=' S?
[26] VersionNum ::= ([a-zA-Z0-9_.:] | '-')+

[27] Misc ::= Comment | PI | S

XML 文档类型声明包含或指向标记声明，标记声明提供某一类文档的语法。这种语法被称为文档类型定义(document type difinition，DTD)。文档类型定义可以指向一个外部子集(一种特殊类型的外部实体)，或者可以在一个内部子集中直接包含标记声明，或者两者兼用。一个文档的文档类型定义由这两个子集合在一起组成。

标记声明可以是元素类型声明，属性表声明，实体声明，或是符号声明。这些声明可以如下面规范性和有效性约束中所述，全部或部分地包含在参数实体中，完整的信息参见"4. 物理结构"。

文档类型定义
[28] doctypedecl ::= '<!DOCTYPE' S Name (S ExternalID)? S? ('[' (markupdecl | PEReference | S)* ']' S?)? '>' [VC: 根元素类型]
[29] markupdecl ::= elementdecl | AttlistDecl | EntityDecl | NotationDecl | PI | Comment [VC: 严格的声明/PE 嵌套] [WFC: 内部子集中的 PE]

标记声明可以全部或部分地由参数实体的置换文本组成。本规范后面的各个非终结符(elementdecl，AttlistDecl，等等)产生式描述的是在所有的参数实体被包含(include)之后的声明。

有效性约束: 根元素类型(Root Element Type)
文档类型声明中的 Name 必须匹配根元素的类型。

有效性约束: 严格的声明/PE 嵌套
参数实体的置换文本必须用标记声明严格嵌套。即，如果一个标记声明(上面的 markupdecl)的第一个或最后一个字符被包含于一个参数实体引用的置换文本中，两者必须都在此置换文本中。

格式约束: 内部子集中的 PE
在内部 DTD 子集中，参数实体引用只能出现在标记声明出现的地方，而不能在标记声明内部出现。(这个约束不适用于出现在外部参数实体内的引用，也不适用于外部子集。)

同内部子集一样，外部子集和任何 DTD 中引用的外部参数实体，必须由一系列被非终结符 markupdecl 所允许的完整的标记声明组成，其中可以夹杂空白字符或参数实体引用。但是，外部子集和外部参数实体的部分内容可以通过使用条件段(conditional section)被有条件地忽略，在内部子集中则不允许这么做。

外部子集
[30] extSubset ::= TextDecl? extSubsetDecl
[31] extSubsetDecl ::= (markupdecl | conditionalSect | PEReference | S)*

外部子集和外部参数实体与内部实体不同之处还在于：在外部子集和外部参数实体内，参数实体引用不仅可以出现在标记声明间，还可以出现在标记声明内。

有文档类型声明的 XML 文档的例子：

```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

系统标识符"hello.dtd"给出了文档 DTD 的 URI。

声明也可以如同下面这个例子一样直接(locally)给出：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE greeting [
  <!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```

如果同时使用外部和内部子集，内部子集被看成出现在外部子集之前，这意味着内部子集中的实体和属性表声明的优先级要比在外部子集中的高。

2.9 独立文档声明

当文档从 XML 处理器递给应用时，标记声明可以影响它的内容，属性缺省值和实体声明是其中的例子。可以作为 XML 声明成分的独立文档声明，指明了对于文档实体而言，是否存在外部的声明。

独立文档声明
[32] SDDecl ::= S 'standalone' Eq (("'" ('yes' | 'no') "'") | ('"' ('yes' | 'no') '"')) [VC: 独立文档声明]

在一个独立文档声明中，值"yes"表示对于文档实体没有外部标记声明(不论是在 DTD 外部子集中，还是在由内部实体引用的外部参数实体中)会影响从 XML 处理器传递给应用的信息。值"no"表示有或可能有这样的外部标记声明。注意独立文档声明只是表示外部声明的存在，如果文档中存在对外部实体的引用，而这些实体已在内部声明时，不影响它的独立状态。

如果不存在外部标记声明，独立文档声明没有意义。如果存在外部标记声明，但没有独立文档声明，就假定取值"no"。

某些网络传输应用也许需要独立的文档，任何满足 standalone="no"的 XML 文档可以通过一

定的算法转换为独立文档。

有效性约束: 独立文档声明
独立文档声明必须取值为"no"，如果任何外部标记声明中包含：

- 有缺省值的属性声明，如果适用这些属性的元素出现在文档中而又没有给这些属性赋值的话。
- (除了 amp，lt，gt，apos，quot 的)实体声明，而对这些实体的引用出现在文档中的话。
- 需要规范化的属性声明，这些出现在文档中的属性的值会因规范化而改变。
- 具有元素内容的元素类型声明，如果在这些类型的任一实例中直接出现空白域的话。

具有独立文档声明的 XML 声明的例子：

<?xml version="1.0" standalone='yes'?>

2.10 空白域处理

在编辑 XML 文档时，使用"空白域"(空格，制表符，空行，在本规范中用非终结符 S 表示)来分开标记以获得更好的可读性是很方便的。通常在文档的交付版本中不想包含这些空白域。另一方面，必须保留在交付版本中的有意义的空白域是很常见的，如在诗歌和源码中的空白域。

XML 处理器必须始终把不是标记的所有字符传递给应用。一个进行验证的 XML 处理器必须同时通知应用这些字符中的那一些组成了出现在元素内容中的空白域。

可以在元素中附加一个名为 xml:space 的特殊属性，以通知应用应该保留此元素中的空白域。在有效的文档中，此属性和其他属性一样，使用时必须声明。它必须被声明为枚举类型，只有"default"和"preserve"两个可能的值。例如：

```
<!ATTLIST poem   xml:space (default|preserve) 'preserve'>
```

"default"表示可以对此元素使用应用的缺省空白域处理模式，"preserve"表示应用应该保留所有的空白域。这适用于其所处元素的内容中的所有元素，除非被另一个 xml:space 属性的实例所覆盖。

任何文档的根元素被认为对应用的空白域处理方式不作要求，除非它给此属性赋了值或将此属性声明为带缺省值。

2.11 行尾处理

为编辑的方便起见，存储 XML 已析实体的计算机文档经常用行来组织。通常这些行用回车

符(#xD)和换行符(#xA)的一些组合来分隔。

为了使应用的工作简单化，对于一个外部已析实体或内部已析实体的字面实体值中包含的任何双字符序列"#xD#xA"或单独的#xD，XML 处理器都应换成#xA 传递给应用。(这可以通过在进行解析前将所有行定界符规范成#xA 而方便地实现。)

2.12 语言标识

在进行文档处理时，标识出其内容所使用的自然或形式化语言经常是很有用的。可以在文档中插入一个名为 xml:lang 的特殊属性用于指出 XML 文档中任何元素的内容和属性所使用的语言。在有效的文档中，此属性和其他属性一样，使用时必须声明。此属性的值是[IETF RFC 1766]，"语言标识码"中定义的语言标识符：

语言标识
[33] LanguageID ::= Langcode ('-' Subcode)*
[34] Langcode ::= ISO639Code | IanaCode | UserCode
[35] ISO639Code ::= ([a-z] | [A-Z]) ([a-z] | [A-Z])
[36] IanaCode ::= ('i' | 'I') '-' ([a-z] | [A-Z])+
[37] UserCode ::= ('x' | 'X') '-' ([a-z] | [A-Z])+
[38] Subcode ::= ([a-z] | [A-Z])+

Langcode 可以是下列值：

- [ISO 639]，"语言名称的表示码"中定义的双字母语言码。
- 在 Internet Assigned Numbers Authority [IANA]注册的语言标识码，以前缀"i-"(或"I-")开头。
- 用户指定或经各方同意的专用语言标识符，必须以前缀"x-"或"X-"开头，以保证它们不会和以后经 IANA 标准化或在 IANA 注册的名字相冲突。

可以有任意多个子代码段（subcode），如果第一个子代码段存在，并且子代码由两个字母组成，那么此子代码必须是[ISO 3166]，"国家名称表示码"中定义的国家代码。如果第一个子代码多于两个字母，那么它必须是在 IANA 注册的语言代码所表示的语言的子代码，除非它 Langcode 以前缀"x-"或"X-"开头。

习惯上用小写字母给出语言代码，用大写字母给出国家代码(如果有的话)。注意这些值与 XML 文档中的其他名字不同，是大小写无关的。

举例如下：

```
<p xml:lang="en">The quick brown fox jumps over the lazy dog.</p>
<p xml:lang="en-GB">What colour is it?</p>
<p xml:lang="en-US">What color is it?</p>
```

```
<sp who="Faust" desc='leise' xml:lang="de">
  <l>Habe nun, ach! Philosophie,</l>
  <l>Juristerei, und Medizin</l>
  <l>und leider auch Theologie</l>
  <l>durchaus studiert mit hei 逻 m Bem 黨'n.</l>
  </sp>
```

xml:lang 所表示的语言选择适用于它所处元素的所有属性和内容，除非被此内容中的元素内的另一个 xml:lang 的实例所覆盖。

xml:lang 的一个简单声明可以采用如下形式：

```
xml:lang    NMTOKEN    #IMPLIED
```

但是如果合适的话，也可以给出特定的缺省值。在一本供英国学生使用的法文诗歌集中，评注和注解使用英语，xml:lang 属性可以这样声明：

```
    <!ATTLIST poem      xml:lang NMTOKEN 'fr'>
    <!ATTLIST gloss     xml:lang NMTOKEN 'en'>
    <!ATTLIST note      xml:lang NMTOKEN 'en'>
```

3. 逻辑结构

每个 XML 文档包含一个或多个元素，它们的边界用起始标记和结束标记定界，或者，对于空元素，用一个空元素标记分隔。每一个元素有一个用名字标识的类型，有时称之为它的"通用标识符(generic identifier)"(GI)，同时它可以有一个属性说明(attribute specification)集。每个属性说明有一个名字和一个值。

元素

[39]　element ::=　EmptyElemTag | STag content ETag
　　[WFC: 元素类型匹配] [　VC: 元素有效]

除了那些开头匹配(('X'|'x')('M'|'m')('L'|'l'))的名字保留用于本规范的此版本和后继版本的标准化外，本规范不对元素类型和属性的语义，用法和名字(语法之外)作出限制。

格式约束: 元素类型匹配
元素结束标记中的 Name 必须和起始标记中的元素类型相匹配。

有效性约束: 元素有效
如果有一个与 elementdecl 相匹配的声明的 Name 与元素类型相匹配，且下述之一成立时，称此元素是有效的：

1. 此声明与 EMPTY 相匹配，同时此元素没有内容。

2．此声明与 children 相匹配，同时子元素的序列属于内容模型中的正则表达式所产生的语言，在每对子元素间允许有空白域(匹配非终结符 S 的字符)。

3．此声明与 Mixed 相匹配，同时内容由其类型匹配内容模型中的名字的字符数据和子元素组成。

4．此声明与 ANY 相匹配，同时每个子元素的类型均已声明。

3.1 起始标记，结束标记和空元素标记

每一个非空 XML 元素以一个起始标记作为开始的标记。

起始标记
[40] STag ::= '<' Name (S Attribute)* S? '>' [WFC: 唯一的属性说明]
[41] Attribute ::= Name Eq AttValue [VC: 属性值类型]
[WFC: 无外部实体引用]
[WFC: 在属性值中没有<]

起始标记和结束标记中的 Name 给出了元素的类型。Name-AttValue 对被统称为元素的属性说明，其中每一对中的 Name 被称为属性名，AttValue 的内容(在定界符'或"间的文本)被称为属性值。

格式约束: 唯一的属性说明
一个属性名只能在同一个起始标记或空元素标记中出现一次。

有效性约束: 属性值类型
属性必须被声明，其值必须是所声明的类型。(属性类型参见"3.3 属性表声明"。)

格式约束: 无外部实体引用
属性值不能包含对外部实体直接或间接的实体引用。

格式约束: 在属性值中没有<
在一个属性值中直接或间接引用的实体的置换文本(除了"<")不能包含<。

起始标记的一个例子：

<termdef id="dt-dog" term="dog">

由一个起始标记开始的每一个元素必须用一个结束标记标记其结束，结束标记中的名字必须与起始标记中给出的元素类型相同：

结束标记
[42] ETag ::= '</' Name S? '>'

结束标记的一个例子：

</termdef>

在起始标记和结束标记中的文本被称为元素的内容：

元素的内容
[43] content ::= (element | CharData | Reference | CDSect | PI | Comment)*

如果一个元素为空，它必须表示为一个起始标记紧跟一个结束标记或空元素标记。一个空元素标记采用一种特殊的形式：

空元素标记
[44] EmptyElemTag ::= '<' Name (S Attribute)* S? '/>' [WFC: 唯一的属性说明]

不论元素是否用关键字 EMPTY 声明，空元素标记都可以用于任何没有内容的元素。出于互操作性考虑，空元素必须用于，且只能用于声明为 EMPTY 的元素。

空元素的例子：

```
<IMG align="left"
  src="http://www.w3.org/Icons/WWW/w3c_home" />
<br></br>
<br/>
```

3.2 元素类型声明

出于验证的目的，可以用元素类型和属性表声明限制 XML 文档中元素的结构。元素类型声明限制了元素的内容。

元素类型声明通常限制了子元素的类型。由用户选择，当声明提到的元素类型没有相应的声明时，XML 处理器可以给出警告，但这不是一个错误。

元素类型声明形式如下：

元素类型声明
[45] elementdecl ::= '<!ELEMENT' S Name S contentspec S? '>' [VC: 唯一的元素类型声明]
[46] contentspec ::= 'EMPTY' | 'ANY' | Mixed | children

其中 Name 给出了所声明的元素类型。

有效性约束: 唯一的元素类型声明
元素类型只能声明一次。

元素类型声明的例子：

```
<!ELEMENT br EMPTY>
<!ELEMENT p (#PCDATA|emph)* >
<!ELEMENT %name.para; %content.para; >
<!ELEMENT container ANY>
```

3.2.1 元素内容

当某一类型的元素只能包含用可选空白域(匹配非终结符 S)分隔的子元素(无字符数据)时，此元素类型具有元素内容。在这种情况下，有内容模型作为类型限制之一，内容模型是决定子元素类型和子元素出现顺序的一种简单语法。此语法用内容粒子(cp)构建，内容粒子由名字，内容粒子的选择表(choice list)或内容粒子的序列表(sequence list)组成：

元素内容的模型
[47] children ::= (choice | seq) ('?' | '*' | '+')?
[48] cp ::= (Name | choice | seq) ('?' | '*' | '+')?
[49] choice ::= '(' S? cp (S? '|' S? cp)* S? ')' [VC: 严格的组/PE 嵌套]
[50] seq ::= '(' S? cp (S? ',' S? cp)* S? ')' [VC: 严格的组/PE 嵌套]

其中每一个 Name 是可以作为子元素的元素的类型。选择表中出现的任意内容粒子在元素内容中允许出现的位置对应于选择表在语法中的位置。序列表中出现的所有内容粒子必须以相同的顺序出现在元素内容中。在名字或表之后的可选字符(optional character)决定了表中元素或内容粒子可以出现一次或多次(+)，还是零次或多次(*)，或是零次或一次(?)。没有这样一个操作符意味着元素或内容粒子必须恰好出现一次。这种句法和意义和本规范中的产生式中所使用的相同。

当且仅当一个元素的内容可以通过满足内容模型中的选择，序列和重复操作符得到，并且内容中的每一个元素与内容模型中的一种元素类型相匹配时，称此元素的内容与一个内容模型相匹配。出于兼容性考虑， 如果文档的某个元素可以和内容模型中的一种元素类型多次匹配，这是一个错误。 更详细的信息参见"E. 确定型内容模型".

有效性约束: 严格的组/PE 嵌套
参数实体的置换文本用括号括起的组严格嵌套。即，如果 choice，seq 或 Mixed 成分的开始或结束括号出现在某个参数实体的置换文本中，两者必须同在此置换文本中。出于互操作性考虑，如果一个参数实体引用出现在 choice，seq 或 Mixed 成分中时，它的置换文本不应为空，同时其置换文本的第一个和最后一个非空字符不应为一个连接符(|或,)。

元素内容模型的例子：

```
<!ELEMENT spec (front, body, back?)>
<!ELEMENT div1 (head, (p | list | note)*, div2*)>
<!ELEMENT dictionary-body (%div.mix; | %dict.mix;)*>
```

3.2.2 混合型内容(Mixed Content)

当某元素类型可以包含字符数据，其间可以随意穿插子元素时，称此元素类型具有混合型内容。在这种情况下，对子元素的类型可能有所限制，但对它们的次序和出现次数没有限制：

混合型内容声明

[51] Mixed ::= '(' S? '#PCDATA' (S? '|' S? Name)* S? ')*' | '(' S? '#PCDATA' S? ')'
[VC: 严格的组/PE 嵌套] [VC: 无重复类型]

其中 Name 给出了子元素的元素类型。

有效性约束: 无重复类型
同一名字在单个混合型内容声明中只能出现一次。

混合内容声明的例子：

```
<!ELEMENT p (#PCDATA|a|ul|b|i|em)*>
<!ELEMENT p (#PCDATA | %font; | %phrase; | %special; | %form;)* >
<!ELEMENT b (#PCDATA)>
```

3.3 属性表声明

属性用于联系名字-值对和元素。属性说明只能在起始标记和空元素标记中出现；因此，用于识别它们的产生式可参看"3.1 起始标记，结束标记和空元素标记"中。属性表声明可以用于：

- 定义与一给定元素类型有关的属性集。
- 确定这些属性的类型限制。
- 提供属性的缺省值。

属性表声明详细说明了与给定元素类型相关联的每一个属性的名字，数据类型和缺省值(如果有的话)：

属性表声明

[52] AttlistDecl ::= '<!ATTLIST' S Name AttDef* S? '>'

[53]　AttDef ::=　S Name S AttType S DefaultDecl

AttlistDecl 规则中 Name 是元素的类型。由用户选择，当元素类型中的属性没有被声明时，XML 处理器可以给出一个警告，但这不是一个错误。AttDef 规则中的 Name 是属性的名字。

当与某个给定元素类型相关的 AttlistDecl 超过一个时，这些声明中的内容被合并在一起。当给定元素类型的某个属性的定义超过一个时，绑定第一个定义，其余定义被忽略。出于互操作性考虑，DTD 的作者可以选择一个给定的元素类型至多有一个属性表声明，一个给定的属性名至多有一个属性定义，以及每个属性表声明至少有一个属性定义。出于互操作性考虑，当一个给定元素有超过一个的属性表声明或一个给定属性有超过一个的属性定义时，由用户选择，XML 处理器可以给出警告，但这不是一个错误。

3.3.1 属性类型

XML 属性有三种类型：字符串类型，一组记号化类型和枚举类型。字符串类型可以以任意字面字符串为值; 各个记号化类型有不同的词法和语义约束，如下：

属性类型
[54]　AttType ::=　StringType | TokenizedType | EnumeratedType
[55]　StringType ::=　'CDATA'
[56]　TokenizedType ::= 'ID' [VC: ID] [VC: 每种元素类型一个 ID] [VC: ID 属性的缺省值]
　　| 'IDREF' [　VC: IDREF]
　　| 'IDREFS' [　VC: IDREF]
　　| 'ENTITY' [　VC: 实体名]
　　| 'ENTITIES' [　VC: 实体名]
　　| 'NMTOKEN' [　VC: 名字记号]
　　| 'NMTOKENS' [　VC: 名字记号]

有效性约束: ID
ID 类型的值必须匹配 Name 产生式。作为此类型值的名字只能在 XML 文档中出现一次;即，ID 类型的值必须能唯一标识元素。

有效性约束: 每种属性类型一个 ID
每种属性类型只能有一个 ID 属性。

有效性约束: ID 属性的缺省值
ID 属性必须有一个声明为#IMPLIED 或#REQUIRED 的缺省值。

有效性约束: IDREF
IDREF 类型的值必须匹配 Name 产生式，IDREFS 类型的值必须匹配 Names 产生式；每一个 Name 必须匹配 XML 文档中某些元素 ID 属性的值；也就是说，IDREF 类型的值必须匹配某些 ID 属性的值。

有效性约束: 实体名
ENTITY 类型的值必须匹配 Name 产生式，ENTITIES 类型的值必须匹配 Names 产生式；每一个 Name 必须匹配 DTD 中声明的未析实体的名字。

有效性约束: 名字记号
NMTOKEN 类型的值必须匹配 Nmtoken 产生式;NMTOKENS 类型的值必须匹配 Nmtokens 产生式。

枚举类型的属性可以在声明中提供的取值表中取值。有两种枚举类型：

枚举属性类型
[57] EnumeratedType ::= NotationType | Enumeration
[58] NotationType ::= 'NOTATION' S '(' S? Name (S? '|' S? Name)* S? ')'
[VC: 符号属性]
[59] Enumeration ::= '(' S? Nmtoken (S? '|' S? Nmtoken)* S? ')' [VC: 枚举]

一个 NOTATION 类型的属性标识了一种用于解释与此属性相关的元素的符号，此符号用相关系统或公共标识符在 DTD 中声明。

有效性约束: 符号属性
此类型的值必须与声明中所包含的符号名之一相匹配;声明中的所有符号名都必须声明。

有效性约束: 枚举
此类型的值必须与声明中所包含的 Nmtoken 记号之一相匹配。

出于互操作性考虑，同一 Nmtoken 只能在单个元素类型的枚举属性类型中出现一次。

3.3.2 属性缺省值

属性声明提供的信息指明了某属性是否必须出现，同时指明了在被声明的属性不是必须出现而文档中没有出现此属性的情况下，XML 处理器应如何处理。

属性缺省值
[59] DefaultDecl ::= '#REQUIRED' | #IMPLIED' | (('#FIXED' S)? AttValue)
[VC: 必须的属性] [VC: 合法的属性缺省值]
[WFC: 在属性值中无<] [VC: 固定的属性缺省值]

在一个属性声明中，#REQUIRED 表示必须总是提供此属性，#IMPLIED 表示不提供缺省值。如果声明既不是#REQUIRED，也不是#IMPLIED，那么 AttValue 值包含了所声明的缺省值;关键字#FIXED 规定此属性必须总是有缺省值。如果声明了一个缺省值，当 XML 处理器遇

到一个被省略的属性时，它将当成此属性以缺省值出现

有效性约束: 必须的属性
如果缺省值声明是关键字#REQUIRED，那么在所有此类型元素的属性表声明中必须有此属性。

有效性约束: 合法的属性缺省值
被声明的属性缺省值必须满足被声明的属性类型的词法约束。

有效性约束: 固定的属性缺省值
如果某属性的缺省值用关键字#FIXED 声明，此属性的所有实例必须匹配该缺省值。

属性表声明的例子：

```
<!ATTLIST termdef
          id          ID          #REQUIRED
          name        CDATA       #IMPLIED>
<!ATTLIST list
          type        (bullets|ordered|glossary)    "ordered">
<!ATTLIST form
          method    CDATA     #FIXED "POST">
```

3.3.3 属性-值对的规范化(Attribute-Value Normalization)

在将属性的值传给应用或检验有效性之前，XML 处理器必须将其规范化：

- 对字符引用的处理是将被引用的字符附加在属性值之后
- 对实体引用的处理是递归地处理实体的置换文本
- 对空白字符(#x20，#xD，#xA，#x9)的处理是将#x20 附加在规范化的值之后，例外是对作为外部已析实体或内部已析实体字面实体值一部分的"#xD#xA"序列只附加一个#x20。
- 对于其他字符的处理是将它们附加与规范化的值之后

如果被声明的值不是 CDATA，那么 XML 处理器必须继续处理规范化后的值，去掉其前导和尾随空格(#x20)字符，并将空格(#x20)字符序列替换成单个空格(#x20)字符。

不进行验证的语法分析器应该将所有尚未读到声明部分的属性当成被声明为 CDATA。

3.4 条件段(Conditional Sections)

条件段是文档类型声明外部子集的一部分，取决于相应的关键字，它们或被包含在 DTD 逻

辑结构之内，或被排除在 DTD 逻辑结构之外。

条件段
[61] conditionalSect ::= includeSect | ignoreSect
[62] includeSect ::= '<![' S? 'INCLUDE' S? '[' extSubsetDecl ']]>'
[63] ignoreSect ::= '<![' S? 'IGNORE' S? '[' ignoreSectContents* ']]>'
[64] ignoreSectContents ::= Ignore ('<![' ignoreSectContents ']]>' Ignore)*
[65] Ignore ::= Char* - (Char* ('<![' | ']]>') Char*)

同内部或外部 DTD 子集一样，条件段可以包含一个或多个完整的声明，注释，处理指令，或嵌套的条件段，其间可以夹杂空白域。

如果条件段的关键字是 INCLUDE，那么条件段的内容是 DTD 的一部分，如果条件段的关键字是 IGNORE，那么条件段的内容逻辑上不是 DTD 的一部分。注意对于可靠的解析过程，即使被忽略的条件段的内容也必须被读取以检测嵌套的条件段，保证最外层(被忽略)的条件段的结尾被恰当地检测到。如果一个关键字为 INCLUDE 的条件段出现在更大的关键字为 IGNORE 的条件段中，内外两个条件段都被忽略。

如果条件段的关键字是一个参数实体引用，处理器在决定是否包含或忽略此条件段前，必须先将该参数实体置换成其内容。

一个例子：

```
<!ENTITY % draft 'INCLUDE' >
<!ENTITY % final 'IGNORE' >

<![%draft;[
<!ELEMENT book (comments*, title, body, supplements?)>
]]>
<![%final;[
<!ELEMENT book (title, body, supplements?)>
]]>
```

4. 物理结构

一个 XML 文档可能包含一个或多个存储单元。它们被称为实体(entity);它们都具有内容并且都用名字(name)进行标识(除了下面要提到的文档实体，和外部 DTD 子集之外)。每一个 XML 文档有一个称为文档实体的实体，它作为 XML 处理器处理的起点并可能包含了整个文档。

实体可以是已析的或未析的。已析实体(parsed entity)的内容被称为它的置换文本;此文本被看成是文档整体的一部分。

未析实体(unparsed entity)是一种资源，其内容可以是也可以不是文本，并且如果是文本的话，可以不是 XML。每一个未析实体有一个相关联的用名字标识的符号。除了要求 XML 处理器能向应用提供实体和符号的标识符之外，XML 对未析实体的内容不作任何限制。

已析实体以实体引用的方式使用名字来调用;未析实体用 ENTITY 或 ENTITIES 属性中给出的名字调用。

通用实体(general entity)是那些在文档内容中使用的实体。在本规范中，在不致引起混淆的情况下,普通实体有时用未修饰的术语 entity 来表示。参数实体是用于 DTD 内的已析实体。这两类实体用不同形式的引用，在不同的上下文中识别。另外，它们使用不同的名域;具有相同名字的参数实体和通用实体是两个截然不同的两个实体。

4.1 字符和实体引用(Character and Entity References)

一个字符引用引用 ISO/IEC 10646 字符集中的一个特定字符。例如一个不能用输入设备直接输入的字符。

字符引用
[66]　CharRef ::=　'&#' [0-9]+ ';'
　　| '&#x' [0-9a-fA-F]+ ';' [　WFC: 合法字符]

格式约束: 合法字符
用字符引用引用的字符必须匹配 Char 产生式。

如果字符引用以"&#x"开头的数字和字母(直到终结)提供了某字符在 ISO/IEC 10646 中代码的一个十六进制表示。如果它仅以"&#"开头的数字(直到终结)提供了某字符的代码的十进值表示。

实体引用(entity reference)引用一个命名实体的内容。对已析普通实体的引用使用"and"号(&)和分号(;)作为定界符。参数实体引用则使用百分号(%)和分号(;)作为定界符。

实体引用
[67]　Reference ::=　EntityRef | CharRef
[68]　EntityRef ::=　'&' Name ';' [　WFC: 声明实体]
　　[　VC: 声明实体]
　　[　WFC: 已析实体]
　　[　WFC: 无递归]
[69]　PEReference ::=　'%' Name ';' [　VC: 声明实体]
　　[　WFC: 无递归]
　　[　WFC: 在 DTD 内]

格式约束: 声明实体

在一个没有任何 DTD 的文档，或一个只有不包含参数实体引用的内部 DTD 子集的文档，或一个"standalone='yes'"的文档内，在实体引用中给出的 Name 必须与实体声明中所给出的相匹配，但格式良好的文档不需要声明以下的这些实体：amp，lt，gt，apos 和 quot。参数实体的声明必须先于任何对它的引用。类似地，通用实体的声明必须先于任何在属性表声明中的缺省值中出现的对它的引用。注意对于在外部子集或外部参数实体中声明的实体，不进行验证的处理器不必要读取和处理它们的声明;对这些文档，仅当 standalone='yes'时，实体必须被声明的规则才是一个格式约束。

有效性约束: 声明实体
在一个有外部子集或外部参数实体且"standalone='no'"的实体中，实体引用中给出的 Name 必须与实体声明中所给出的相匹配。出于互操作性考虑，有效的文档应该以"4.6 预定义实体"中的简化形式声明实体 amp，lt，gt，apos 和 quot。参数实体的声明必须先于任何对它的引用。类似地，通用实体的声明必须先于任何在属性表声明中的缺省值中出现的对它的引用。

格式约束: 已析实体
实体引用不能包含一个未析实体的名字。未析实体只能在声明为 ENTITY 或 ENTITIES 的属性值中引用。

格式约束: 无递归
已析实体不能直接或间接地包含对自身的递归引用。

格式约束: 在 DTD 内
参数实体引用只能在 DTD 中出现。

字符引用和实体引用的例子：

```
Type <key>less-than</key> (&#x3C;) to save options.
This document was prepared on &docdate; and
is classified &security-level;.
```

参数实体引用的例子：

```
<!-- declare the parameter entity "ISOLat2"... -->
<!ENTITY % ISOLat2    SYSTEM "http://www.xml.com/iso/isolat2-xml.entities" >
<!-- ... now reference it. -->
%ISOLat2;
```

4.2 实体声明(Entity Declaration)

实体以如下方式声明：

实体声明
[70] EntityDecl ::= GEDecl | PEDecl

[71] GEDecl ::= '<!ENTITY' S Name S EntityDef S? '>'
[72] PEDecl ::= '<!ENTITY' S '%' S Name S PEDef S? '>'
[73] EntityDef ::= EntityValue | (ExternalID NDataDecl?)
[74] PEDef ::= EntityValue | ExternalID

实体引用中的 Name 标识了该实体;对于未析实体，ENTITY 或 ENTITIES 属性的值标识了该实体。如果同一实体被声明了不止一次，绑定第一个声明。由用户选择，如果实体被多次声明，XML 处理器可以给出警告。

4.2.1 内部实体(Internal Entities)

如果实体定义是一个 EntityValue，被定义的实体被称为内部实体。内部实体没有单独的物理存储对象，实体的内容在声明中给出。注意字面实体值中一些实体和字符引用的处理可能要求产生正确的置换文本：参见"4.5 内部置换文本的构造"。

内部实体是已析实体。

内部实体声明的例子：

```
<!ENTITY Pub-Status "This is a pre-release of the
 specification.">
```

4.2.2 外部实体(External Entities)

如果实体不是内部的，那么它是一个外部实体，声明如下：

外部实体声明
[75] ExternalID ::= 'SYSTEM' S SystemLiteral
| 'PUBLIC' S PubidLiteral S SystemLiteral
[76] NDataDecl ::= S 'NDATA' S Name [VC: 声明符号]

如果有 NDataDecl，那么这是一个通用未析实体;否则它是一个已析实体。

有效性约束: 声明符号
Name 必须与符号的名字相匹配。

SystemLiteral 被称为该实体的系统标识符。这是一个 URI，可以用于查找此实体。注意井号(#)和 URI 中常用的片断标识符形式上而言不是 URI 的一部分；如果一个片断标识符作为系统标识符的部分给出，XML 处理器可以给出一个错误。除非在本规范范围之外另外给出(如，

一个特殊 DTD 定义的专用 XML 元素类型，或一个特殊应用规范中定义的处理指令)，相对 URI 指相对于实体声明所在资源的位置。因此，一个 URI 可能是相对于文档实体，或相对于包含外部 DTD 子集的实体，或相对于其他一些外部参数实体。

XML 处理器处理 URI 中的非 ASCII 字符时，将 UTF-8 中的字符用一个或多个字节表示，然后将这些字符用 URI 转义机制转义(即，将每个字节转换成%HH，其中 HH 是字节值的十六进制符号)。

除了系统标识符之外，外部标识符还可以包含公共标识符。试图查找实体内容的 XML 处理器可以用公共标识符试着产生一个可选 URI。如果处理器无法做到这一点，它必须使用系统常量中定义的 URI。在试着匹配之前，公共标识符中所有空白字符串必须被规范为单个空格字符(#x20)，同时必须去掉前导和尾随空白。

外部实体声明的例子：

```
<!ENTITY open-hatch
         SYSTEM "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY open-hatch
         PUBLIC "-//Textuality//TEXT Standard open-hatch boilerplate//EN"
         "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY hatch-pic
         SYSTEM "../grafix/OpenHatch.gif"
         NDATA gif >
```

4.3 已析实体(Parsed Entities)

4.3.1 文本声明(Text Declaration)

每个外部已析实体可以以文本声明作为开始。

文本声明

[77] TextDecl ::= '<?xml' VersionInfo? EncodingDecl S? '?>'

文本声明必须以字面形式给出，而不能使用已析实体的引用。文本声明只能在外部已析实体的开头出现，不允许在其他任何地方出现。

4.3.2 格式良好的已析实体(Well-Formed Parsed Entities)

如果文档实体匹配 document 产生式，那么它是格式良好的。如果外部通用已析实体匹配 extParsedEnt 产生式，那么它是格式良好的。如果外部参数实体匹配 extPE 产生式，那么它是格式良好的。

格式良好的外部已析实体
[78] extParsedEnt ::= TextDecl? content
[79] extPE ::= TextDecl? extSubsetDecl

如果内部普通已析实体的置换文本匹配 content 产生式，那么它是格式良好的。根据定义，所有内部的参数实体都是格式良好的。

实体符合格式要求的一个结果是 XML 文档的逻辑和物理结构是严格嵌套的;起始标记，结束标记，空元素标记，元素，注释，处理指令，字符引用，或实体引用都不能在一个实体中开始而在另一个实体中结束。

4.3.3 实体中的字符编码(Character Encoding in Entities)

XML 文档中的每个外部已析实体都可以对其字符采用一种不同的编码方案。所有 XML 处理器必须能读编码为 UTF-8 或 UTF-16 的实体。

以 UTF-16 编码的实体必须以 ISO/IEC 10646 增补 E 和 Unicode 附录 B(零宽度不间断空格字符，#xFEFF)中所描述的字节次序标记(Byte Order Mark)开头。这是一个编码签名，即不是 XML 文档中标记的一部分，也不是 XML 文档字符数据的一部分。XML 处理器必须能用此字符区分 UTF-8 编码和 UTF-16 编码的文档。

虽然 XML 处理器只被要求能读取 UTF-8 和 UTF-16 编码的实体，普遍认为国际上还有其他编码方案。有时可能想让 XML 处理器读取以那些编码方案编码的实体。以不同与 UTF-8 和 UTF-16 的编码方案存储的实体必须以包含编码声明的文本声明开头：

编码声明
[80] EncodingDecl ::= S 'encoding' Eq ('" EncName '" | ' EncName ')
[81] EncName ::= [A-Za-z] ([A-Za-z0-9._] | '-')* /* 编码方案的名字只包含拉丁字母 */

在文档实体中，编码声明是 XML 声明的一部分。EncName 是所用编码方案的名字。

在一个编码声明中，值"UTF-8"，"UTF-16"，"ISO-10646-UCS-2"和"ISO-10646-UCS-4"应该用于表示 Unicode 或 ISO/IEC 10646 中的各种不同编码和变换方案，值"ISO-8859-1"，"ISO-8859-2"，... "ISO-8859-9"应该用于表示 ISO 8859 的各个部分，而值"ISO-2022-JP"，

"Shift_JIS"和"EUC-JP"应该用于表示 JIS X-0208-1997 的各种编码。XML 处理器可以识别其他编码方案;建议对于在 Internet Assigned Numbers Authority [IANA]注册的字符编码方案(以字符集(charset)的方式)，除了以上所列的之外，引用时应使用其注册名。注意这些注册名定义为大小写敏感，因此欲与之匹配的处理器要以大小写敏感的方式进行匹配。

在缺少外部传输协议(如 HTTP 或 MIME)所提供的信息时，以下情况均是错误：XML 处理器接收到的实体的编码方案与实体所含编码声明中指出的编码方案不同，编码声明不在外部实体的开头，既不以字节次序标记开头也不以编码声明开头的实体使用了不同于 UTF-8 的编码。注意因为 ASCII 是 UTF-8 的一个子集，以普通 ASCII 编码的实体不严格需要编码声明。

当 XML 处理遇到的实体使用了它不能处理的编码时，是一个严重错误。

编码声明的例子：

<?xml encoding='UTF-8'?>
<?xml encoding='EUC-JP'?>

4.4 XML 处理器对实体和引用的处理

下表汇总了字符引用，实体引用，和对未析实体的调用可以出现的上下文，以及每种情况下 XML 处理器要求的动作。最左边一列的标签记指明了识别时的上下文：

内容中的引用
可以在元素的起始标记之后，结束标记之前的任何地方以引用形式出现，对应于非终结符 content。
属性值中的引用
可以在起始标记内的属性值中，或属性声明内的缺省值中以引用形式出现;对应于非终结符 AttValue。
作为属性值
可以以 Name 而不是以引用的形式出现，作为声明为 ENTITY 类型的属性的值，或可以作为声明为 ENTITIES 类型的属性值中的以空白分隔的记号之一。
实体值中的引用
可以在参数中或内部实体的实体声明内的字面实体值中以引用形式出现;对应于非终结符 EntityValue。
DTD 中的引用
可以在 DTD 的内部或外部子集中以引用形式出现，但须在 EntityValue 和 AttValue 之外。

	实体类型				字符
	参数	内部通用	外部已析通用	未析	
内容中的引用	不被识别	被包含	进行验证时被包含	被禁止	被包含
属性值中的引用	不被识别	作为常量	被禁止	被禁止	被包含

		被包含			
作为属性值	不被识别	被禁止	被禁止	通知	不被识别
实体值中的引用	作为常量被包含	不处理	不处理	被禁止	被包含
DTD 中的引用	作为 PE 被包含	被禁止	被禁止	被禁止	被禁止

4.4.1 不被识别(Not Recognized)

在 DTD 之外，百分号字符%没有特殊含义;因此在 DTD 中的参数实体引用在 content 中不被当成标记识别。类似地，除非未析实体的名字出现在已适当声明的属性的值中，否则它们不被识别。

4.4.2 被包含(Included)

当一个实体的置换文本代替引用,被当成引用文档的一部分一样被查找和处理时，称此实体被包含。其置换文本可以包含字符数据和标记(不包括参数实体)，其中标记必须以通常的方式识别，但用于转义标记定界符(实体 amp，lt，gt，apos 和 quot)的实体的置换文本总是被当成数据。(字符串"AT&T;"展开为"AT&T;"尚存的"and"号&不被识别为实体引用的定界符。)当被表示的字符代替引用被处理时，称此字符引用被包含。

4.4.3 进行验证时被包含(Included If Validating)

当 XML 处理器识别出一个对已析实体的引用，为了验证该文档，处理器必须包含此实体的置换文本。如果实体是外部的，而处理器不试图验证该 XML 文档，那么处理器可以，但不是必须，包含此实体的置换文本。如果一个不验证的解析器不包含此置换文本，它必须通知应用它识别出但没有读取此实体。

这条规范基于这样一个共识：由 SGML 和 XML 的实体机制提供的起初设计用于支持模块化创作的自动包含不一定适合于其他应用，尤其是文档浏览。例如，当浏览器遇到一个外部已析实体引用时，可能选择用可视方式表示其存在但只在被请求时才查找它进行显示。

4.4.4 被禁止(Forbidden)

以下情况被禁止，并构成一个严重错误：

- 出现对未析实体的引用。
- 在 DTD 中出现任何字符或通用实体引用，除非它们出现在 EntityValue 或 AttValue 中。
- 属性值中出现对外部实体的引用。

4.4.5 被包含在常量中(Included in Literal)

当实体引用出现在属性值中或参数实体引用出现在字面实体值中时，它们的置换文本代替引用被当成引用文档的一部分一样被查找和处理，但是置换文本中的单双引号总是被当成正常的数据字符而不会结束此常量。例如，下面的例子是格式良好的：

```
<!ENTITY % YN '"Yes"' >
<!ENTITY WhatHeSaid "He said &YN;" >
```

而这个例子不是：

```
<!ENTITY EndAttr "27" >
<element attribute='a-&EndAttr;>
```

4.4.6 通知(Notify)

当未析实体名字作为记号在声明为 ENTITY 或 ENTITIES 类型的属性的值中出现时，进行验证的处理器必须将此实体和它的相关符号的系统和公共(如果有的话)标识符通知给应用。

4.4.7 不处理(Bypassed)

当实体声明内一个通用兵实体引用出现在 EntityValue 中时，它不被处理，保持不变。

4.4.8 作为 PE 被包含(Included as PE)

和外部已析实体一样，参数实体只需在进行验证时被包含。当参数实体引用在 DTD 中被识别并被包含时，它的置换文本被前后各加上一个空格字符;其目的在于强制参数实体的置换文本在 DTD 中包含完整的的语法记号。

4.5 内部实体置换文本的构建(Construction of Internal Entity)

在讨论内部实体的处理时，区分两种形式的实体值是有帮助的。字面实体值(literal entity value)是实际出现在实体声明中用引号括起的字符串。对应于非终结符 EntityValue。置换文本(replacement text)是置换了字符引用和参数实体引用后的实体内容。

在内部实体声明(EntityValue)中给出的字面实体值可以包括字符引用，参数实体引用和通用实体引用。这些引用必须被整个包含于字面实体值中。如前述方式被包含的实际置换文本必

须包含所有被引用的参数实体的置换文本，同时在字面实体值中必须包含所有代替字符引用的字符。但通用实体的引用必须保持不变，不被展开。例如，如果有以下的声明：

```
<!ENTITY % pub       "&#xc9;ditions Gallimard" >
<!ENTITY     rights "All rights reserved" >
<!ENTITY     book     "La Peste: Albert Camus, &#xA9; 1947 %pub;. &rights;" >
```

那么实体"book"的置换文本为：

La Peste: Albert Camus, ?nbsp;1947 ditions Gallimard. &rights;

一旦引用"&book;"出现在文档的内容或属性值中时，通用实体引用"&rights;"应该被展开。

这些简单的规则将可能会有复杂的相互作用;参见"D. 实体和字符引用的展开"中对一个难的例子的详细讨论。

4.6 预定义实体(Predefined Entities)

实体和字符引用都可以用于转义左尖括号，"and"号(&)和其他定界符。通用实体集合(amp，lt，gt，apos，quot)专门用于此目的。也可以使用数值字符引用; 一旦被识别，它们立即被展开，同时它们必须被当成字符数据，因此数值字符引用"<"和"&"可以用于转义出现在字符数据中的<和&。

不管这些实体是否被声明，所有的 XML 处理器必须能识别它们。出于互操作性考虑，如其他实体一样，有效的 XML 文档应该在使用这些实体前先声明它们。如果声明的话，这些实体必须被声明为内部实体，其置换文本是被转义的单个字符或指向这个字符的字符引用。如下所示。

```
<!ENTITY lt       "&#38;#60;">
<!ENTITY gt       "&#62;">
<!ENTITY amp      "&#38;#38;">
<!ENTITY apos     "&#39;">
<!ENTITY quot     "&#34;">
```

注意在"lt"和"amp"的声明中，<和&被两次转义，这是为了满足实体置换的格式要求。

4.7 符号声明(Notation Declarations)

符号用名字标识了未析实体的格式，具有符号属性的元素的格式以及处理指令所针对的应用的格式。

符号声明赋予符号一个名字用于实体,属性表声明和属性说明中，同时也给出了一个符号的外部标识符使得 XML 处理器或它的客户应用可以定位能以给定符号处理数据的助理应用。

符号声明
[82] NotationDecl ::= '<!NOTATION' S Name S (ExternalID | PublicID) S? '>'
[83] PublicID ::= 'PUBLIC' S PubidLiteral

XML 处理器必须向应用提供任何在属性值中，属性定义中或实体声明中定义或引用的符号的名字和外部标识符。它们还可以将外部标识符解析成系统标识符，文档名，或是应用调用相应处理器处理给定符号格式的数据的所需的其他信息。(但如果 XML 处理器或应用所运行的系统中没有处理 XML 文档声明和引用的符号的相应应用的情况，不是一个错误。)

4.8 文档实体(Document Entity)

文档实体(document entity)是实体树的根和 XML 处理器的处理起点。本规范没有规定 XML 如何定位文档实体;与其他实体不同，文档实体没有名字，而且可以完全不带任何标识地出现在处理器的输入流中。

5. 一致性(Conformance)

5.1 进行验证和不进行验证的处理器(Validating and Non-Validating Processors)

合乎规范的 XML 处理器可以分为两类：进行验证的和不进行验证的。

进行验证和不进行验证的处理器都必须报告在文档实体的内容中和任何其他它们读到的已析实体中对格式约束的违反。

进行验证的处理器必须报告违反 DTD 声明中所述约束的情况以及不满足本规范中给出的有效性约束的情况。要完成这一点，进行验证的 XML 处理器必须读取和处理整个 DTD 和所有在文档中引用的外部已析实体。

不进行验证的处理器只被要求检查文档实体和整个内部 DTD 子集的格式。虽然它们不被要求检查文档的有效性,但它们必须处理它们读取的所有内部 DTD 子集和参数实体中的声明，直到遇到第一个没有读取的参数实体的引用;也就是说，它们必须根据这些声明中的信息规范化属性值，包含内部实体的置换文本，并提供缺省属性值。它们在遇到第一个没有读取的

参数实体的引用后，不应处理其后的实体声明或属性表声明，因为此实体中包含的声明可能覆盖前面的声明。

5.2 使用 XML 处理器

进行验证的处理器的行为是高度可预测的;它必须读取文档的所有部分，报告所有对格式和有效性的违反。对一个不进行验证的处理器的要求要低一点;它不需要读取文档实体以外的任何文档部分。这对 XML 的处理器的用户而言可能会有两个重要的影响：

- 某些格式错误，尤其是那些要求读取外部实体的，可能不会被不进行验证的处理器检测到。例如称为声明实体，已析实体和无递归的约束，以及"4.4 XML 处理器对实体和引用的处理"中描述为被禁止的一些情况。
- 取决于处理器是否读取参数和外部实体，从处理器传给应用的信息可能会有所不同。例如，不进行验证的处理器可能不规范化属性值，不包含内部实体的置换文本，或不提供缺省属性值，这些动作要求先读取外部或参数实体中的声明。

为了使不同 XML 处理器间的互操作有最大的可靠性，使用不进行验证的处理器的应用不应依赖于不要求这些处理器具备的动作。那些要求使用如缺省值或在外部实体中声明内部实体等功能的应用应该使用进行验证的 XML 处理器。

6. 符号(Notation)

本规范中 XML 的正式句法用一种简单的扩展巴科斯范式(Extended Backus-Naur Form，EBNF)给出。句法中的每一条规则定义了一个记号，形式如下：

symbol ::= expression

如果记号用正则表达式定义，则它以大写字母开头，否则以小写字母开头。字符串(literal strings)用引号括起。

在规则右边的表达式中，以下表达式用于匹配一个或多个字符的字符串：

#xN
N 是一个十六进制的整数，当 ISO/IEC 10646 中某个字符的规范(UCS-4)代码值作为无符号二进制数与 N 相等时，此表达式匹配这个字符。#xN 中的前导 0 没有意义，在相应的代码值中的前导 0 的个数则由所用字符编码方案决定，对 XML 没有意义。
[a-zA-Z]， [#xN-#xN]
与其值在指定范围内的任何字符相匹配(含界，inclusive)。
[^a-z]， [^#xN-#xN]
与其值在指定范围之外的任何字符相匹配。
[^abc]， [^#xN#xN#xN]

与任何不在给定字符集内的字符相匹配。
"string"
与双引号中字符串相匹配。
'string'
与单引号中字符串相匹配。
这些符号可以按下列方式组合，以匹配更复杂的模式，其中 A 和 B 表示简单表达式：

(expression)
expression 被当成一个单元，可以如本表描述进行组合。
A?
与零个或一个 A 相匹配，即 A 可选。
A B
与 A 后跟 B 的模式相匹配。
A | B
与 AB 之一相匹配，但不同时匹配。
A - B
与任何匹配 A 但不匹配 B 的字符串相匹配。
A+
与一个或多个 A 相匹配。
A*
与零个或多个 A 相匹配。
其他在产生式中使用的符号有：

/* ... */
注释
[wfc: ...]
格式约束;用名字标识一个对与某个产生式相关联的格式良好的文档的约束。
[vc: ...]
有效性约束;用名字标识一个对与某个产生式相关联的有效文档的约束。

附录

A. 参考文献

A.1 标准参考文献

IANA
(Internet Assigned Numbers Authority) Official Names for Character Sets， ed. Keld Simonsen et al. See ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets.

IETF RFC 1766
IETF (Internet Engineering Task Force). RFC 1766: Tags for the Identification of Languages， ed. H. Alvestrand. 1995.

ISO 639
(International Organization for Standardization). ISO 639:1988 (E). Code for the representation of names of languages. [Geneva]: International Organization for Standardization， 1988.

ISO 3166
(International Organization for Standardization). ISO 3166-1:1997 (E). Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes [Geneva]: International Organization for Standardization， 1997.

ISO/IEC 10646
ISO (International Organization for Standardization). ISO/IEC 10646-1993 (E). Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane. [Geneva]: International Organization for Standardization， 1993 (plus amendments AM 1 through AM 7).

Unicode
The Unicode Consortium. The Unicode Standard，Version 2.0. Reading，Mass.: Addison-Wesley Developers Press， 1996.

A.2 其他参考文献

Aho/Ullman
Aho， Alfred V., Ravi Sethi， and Jeffrey D. Ullman. Compilers: Principles, Techniques, and Tools. Reading: Addison-Wesley, 1986, rpt. corr. 1988.

Berners-Lee et al.
Berners-Lee, T., R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax and Semantics. 1997. (Work in progress; see updates to RFC1738.)

Br 黝 gemann-Klein
Br 黝 gemann-Klein, Anne. Regular Expressions into Finite Automata. Extended abstract in I. Simon, Hrsg., LATIN 1992, S. 97-98. Springer-Verlag, Berlin 1992. Full Version in Theoretical Computer Science 120: 197-213, 1993.

Br 黝 gemann-Klein and Wood
Br 黝 gemann-Klein, Anne, and Derick Wood. Deterministic Regular Languages. Universit 鋘 Freiburg, Institut f 黵 Informatik, Bericht 38, Oktober 1991.

Clark

James Clark. Comparison of SGML and XML. See http://www.w3.org/TR/NOTE-sgml-xml-971215.

IETF RFC1738
IETF (Internet Engineering Task Force). RFC 1738: Uniform Resource Locators (URL), ed. T. Berners-Lee, L. Masinter, M. McCahill. 1994.

IETF RFC1808
IETF (Internet Engineering Task Force). RFC 1808: Relative Uniform Resource Locators, ed. R. Fielding. 1995.

IETF RFC2141
IETF (Internet Engineering Task Force). RFC 2141: URN Syntax, ed. R. Moats. 1997.

ISO 8879
ISO (International Organization for Standardization). ISO 8879:1986(E). Information processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML). First edition -- 1986-10-15. [Geneva]: International Organization for Standardization, 1986.

ISO/IEC 10744
ISO (International Organization for Standardization). ISO/IEC 10744-1992 (E). Information technology -- Hypermedia/Time-based Structuring Language (HyTime). [Geneva]: International Organization for Standardization, 1992. Extended Facilities Annexe. [Geneva]: International Organization for Standardization, 1996.

B. 字符的分类(Character Classes)

根据 Unicode 标准中定义的特征，字符被分为基类字符(其中包括没有变音符的拉丁字母)，表意字符和组合字符(其中包括大多数的变音符);这些类合起来组成了字母类。数字和扩展符(extender)也各自被分成类。

字符
[84] Letter ::= BaseChar | Ideographic
[85] BaseChar ::= [#x0041-#x005A] | [#x0061-#x007A] | [#x00C0-#x00D6] | [#x00D8-#x00F6] | [#x00F8-#x00FF] | [#x0100-#x0131] | [#x0134-#x013E] | [#x0141-#x0148] | [#x014A-#x017E] | [#x0180-#x01C3] | [#x01CD-#x01F0] | [#x01F4-#x01F5] | [#x01FA-#x0217] | [#x0250-#x02A8] | [#x02BB-#x02C1] | #x0386 | [#x0388-#x038A] | #x038C | [#x038E-#x03A1] | [#x03A3-#x03CE] | [#x03D0-#x03D6] | #x03DA | #x03DC | #x03DE | #x03E0 | [#x03E2-#x03F3] | [#x0401-#x040C] | [#x040E-#x044F] | [#x0451-#x045C] | [#x045E-#x0481] | [#x0490-#x04C4] | [#x04C7-#x04C8] | [#x04CB-#x04CC] | [#x04D0-#x04EB] | [#x04EE-#x04F5] | [#x04F8-#x04F9] | [#x0531-#x0556] | #x0559 | [#x0561-#x0586] | [#x05D0-#x05EA] | [#x05F0-#x05F2] | [#x0621-#x063A] | [#x0641-#x064A] | [#x0671-#x06B7] | [#x06BA-#x06BE] | [#x06C0-#x06CE] | [#x06D0-#x06D3] | #x06D5 | [#x06E5-#x06E6] | [#x0905-#x0939] | #x093D | [#x0958-#x0961] | [#x0985-#x098C] | [#x098F-#x0990] | [#x0993-#x09A8] | [#x09AA-#x09B0] | #x09B2 |

[#x09B6-#x09B9] | [#x09DC-#x09DD] | [#x09DF-#x09E1] | [#x09F0-#x09F1] | [#x0A05-#x0A0A] | [#x0A0F-#x0A10] | [#x0A13-#x0A28] | [#x0A2A-#x0A30] | [#x0A32-#x0A33] | [#x0A35-#x0A36] | [#x0A38-#x0A39] | [#x0A59-#x0A5C] | #x0A5E | [#x0A72-#x0A74] | [#x0A85-#x0A8B] | #x0A8D | [#x0A8F-#x0A91] | [#x0A93-#x0AA8] | [#x0AAA-#x0AB0] | [#x0AB2-#x0AB3] | [#x0AB5-#x0AB9] | #x0ABD | #x0AE0 | [#x0B05-#x0B0C] | [#x0B0F-#x0B10] | [#x0B13-#x0B28] | [#x0B2A-#x0B30] | [#x0B32-#x0B33] | [#x0B36-#x0B39] | #x0B3D | [#x0B5C-#x0B5D] | [#x0B5F-#x0B61] | [#x0B85-#x0B8A] | [#x0B8E-#x0B90] | [#x0B92-#x0B95] | [#x0B99-#x0B9A] | #x0B9C | [#x0B9E-#x0B9F] | [#x0BA3-#x0BA4] | [#x0BA8-#x0BAA] | [#x0BAE-#x0BB5] | [#x0BB7-#x0BB9] | [#x0C05-#x0C0C] | [#x0C0E-#x0C10] | [#x0C12-#x0C28] | [#x0C2A-#x0C33] | [#x0C35-#x0C39] | [#x0C60-#x0C61] | [#x0C85-#x0C8C] | [#x0C8E-#x0C90] | [#x0C92-#x0CA8] | [#x0CAA-#x0CB3] | [#x0CB5-#x0CB9] | #x0CDE | [#x0CE0-#x0CE1] | [#x0D05-#x0D0C] | [#x0D0E-#x0D10] | [#x0D12-#x0D28] | [#x0D2A-#x0D39] | [#x0D60-#x0D61] | [#x0E01-#x0E2E] | #x0E30 | [#x0E32-#x0E33] | [#x0E40-#x0E45] | [#x0E81-#x0E82] | #x0E84 | [#x0E87-#x0E88] | #x0E8A | #x0E8D | [#x0E94-#x0E97] | [#x0E99-#x0E9F] | [#x0EA1-#x0EA3] | #x0EA5 | #x0EA7 | [#x0EAA-#x0EAB] | [#x0EAD-#x0EAE] | #x0EB0 | [#x0EB2-#x0EB3] | #x0EBD | [#x0EC0-#x0EC4] | [#x0F40-#x0F47] | [#x0F49-#x0F69] | [#x10A0-#x10C5] | [#x10D0-#x10F6] | #x1100 | [#x1102-#x1103] | [#x1105-#x1107] | #x1109 | [#x110B-#x110C] | [#x110E-#x1112] | #x113C | #x113E | #x1140 | #x114C | #x114E | #x1150 | [#x1154-#x1155] | #x1159 | [#x115F-#x1161] | #x1163 | #x1165 | #x1167 | #x1169 | [#x116D-#x116E] | [#x1172-#x1173] | #x1175 | #x119E | #x11A8 | #x11AB | [#x11AE-#x11AF] | [#x11B7-#x11B8] | #x11BA | [#x11BC-#x11C2] | #x11EB | #x11F0 | #x11F9 | [#x1E00-#x1E9B] | [#x1EA0-#x1EF9] | [#x1F00-#x1F15] | [#x1F18-#x1F1D] | [#x1F20-#x1F45] | [#x1F48-#x1F4D] | [#x1F50-#x1F57] | #x1F59 | #x1F5B | #x1F5D | [#x1F5F-#x1F7D] | [#x1F80-#x1FB4] | [#x1FB6-#x1FBC] | #x1FBE | [#x1FC2-#x1FC4] | [#x1FC6-#x1FCC] | [#x1FD0-#x1FD3] | [#x1FD6-#x1FDB] | [#x1FE0-#x1FEC] | [#x1FF2-#x1FF4] | [#x1FF6-#x1FFC] | #x2126 | [#x212A-#x212B] | #x212E | [#x2180-#x2182] | [#x3041-#x3094] | [#x30A1-#x30FA] | [#x3105-#x312C] | [#xAC00-#xD7A3]

[86] Ideographic ::= [#x4E00-#x9FA5] | #x3007 | [#x3021-#x3029]

[87] CombiningChar ::= [#x0300-#x0345] | [#x0360-#x0361] | [#x0483-#x0486] | [#x0591-#x05A1] | [#x05A3-#x05B9] | [#x05BB-#x05BD] | #x05BF | [#x05C1-#x05C2] | #x05C4 | [#x064B-#x0652] | #x0670 | [#x06D6-#x06DC] | [#x06DD-#x06DF] | [#x06E0-#x06E4] | [#x06E7-#x06E8] | [#x06EA-#x06ED] | [#x0901-#x0903] | #x093C | [#x093E-#x094C] | #x094D | [#x0951-#x0954] | [#x0962-#x0963] | [#x0981-#x0983] | #x09BC | #x09BE | #x09BF | [#x09C0-#x09C4] | [#x09C7-#x09C8] | [#x09CB-#x09CD] | #x09D7 | [#x09E2-#x09E3] | #x0A02 | #x0A3C | #x0A3E | #x0A3F | [#x0A40-#x0A42] | [#x0A47-#x0A48] | [#x0A4B-#x0A4D] | [#x0A70-#x0A71] | [#x0A81-#x0A83] | #x0ABC | [#x0ABE-#x0AC5] | [#x0AC7-#x0AC9] | [#x0ACB-#x0ACD] | [#x0B01-#x0B03] | #x0B3C | [#x0B3E-#x0B43] | [#x0B47-#x0B48] | [#x0B4B-#x0B4D] | [#x0B56-#x0B57] | [#x0B82-#x0B83] | [#x0BBE-#x0BC2] | [#x0BC6-#x0BC8] | [#x0BCA-#x0BCD] | #x0BD7 | [#x0C01-#x0C03] | [#x0C3E-#x0C44] | [#x0C46-#x0C48] | [#x0C4A-#x0C4D] | [#x0C55-#x0C56] | [#x0C82-#x0C83] | [#x0CBE-#x0CC4] | [#x0CC6-#x0CC8] | [#x0CCA-#x0CCD] | [#x0CD5-#x0CD6] | [#x0D02-#x0D03] | [#x0D3E-#x0D43] | [#x0D46-#x0D48] |

[#x0D4A-#x0D4D] | #x0D57 | #x0E31 | [#x0E34-#x0E3A] | [#x0E47-#x0E4E] | #x0EB1 | [#x0EB4-#x0EB9] | [#x0EBB-#x0EBC] | [#x0EC8-#x0ECD] | [#x0F18-#x0F19] | #x0F35 | #x0F37 | #x0F39 | #x0F3E | #x0F3F | [#x0F71-#x0F84] | [#x0F86-#x0F8B] | [#x0F90-#x0F95] | #x0F97 | [#x0F99-#x0FAD] | [#x0FB1-#x0FB7] | #x0FB9 | [#x20D0-#x20DC] | #x20E1 | [#x302A-#x302F] | #x3099 | #x309A

[88] Digit ::= [#x0030-#x0039] | [#x0660-#x0669] | [#x06F0-#x06F9] | [#x0966-#x096F] | [#x09E6-#x09EF] | [#x0A66-#x0A6F] | [#x0AE6-#x0AEF] | [#x0B66-#x0B6F] | [#x0BE7-#x0BEF] | [#x0C66-#x0C6F] | [#x0CE6-#x0CEF] | [#x0D66-#x0D6F] | [#x0E50-#x0E59] | [#x0ED0-#x0ED9] | [#x0F20-#x0F29]

[89] Extender ::= #x00B7 | #x02D0 | #x02D1 | #x0387 | #x0640 | #x0E46 | #x0EC6 | #x3005 | [#x3031-#x3035] | [#x309D-#x309E] | [#x30FC-#x30FE]

在此定义的字符类可以从 Unicode 字符库中如下导出：

- 名字的起始字符必须属于 Ll, Lu, Lo, Lt, Nl 中的一类。
- 除了起始字符之外的命名字符必须属于 Mc，Me，Mn，Lm 或 Nd 中的一类。
- 兼容区(即代码大于#xF900，小于#xFFFE 的字符)中的字符不允许在 XML 名字中出现。
- 不允许出现具有字体或兼容性分解的字符(即数据库中第 5 项有以"<"开始的"compatibility formatting tag"的字符)。
- 下列字符被当成名字起始字符而非名字字符，因为特性文件中将它们归于字母类：[#x02BB-#x02C1]，#x0559，#x06E5，#x06E6。
- 不允许出现字符#x20DD-#x20E0(与 Unicode 的 5.14 节保持一致)。
- 字符#x00B7 被分为扩展符，因为特性文件中对它是这么标识的。
- 字符#x0387 被当成一个命名字符，因为#x00B7 是它的等价规范形式。
- 字符':'和'_'可以作为名字起始字符。
- 字符'-'和'.'可以作为命名字符。

C. XML 和 SGML(非标准)

XML 被设计为 SGML 的一个子集，表现在每一个有效的 XML 文档应该也是一个合乎规范的 SGML 文档。对 XML 在 SGML 之外对文档所加的限制的详细讨论参见[Clark]。

D. 实体和字符引用的展开(非标准)

本附录中举例说明了在 "4.4 XML 处理器对实体和引用的处理"一节中规定的实体和字符引用的识别和展开的次序。

如果声明包含在 DTD 中

<!ENTITY example "<p>An ampersand (&) may be escaped
numerically (&) or with a general entity

(&). " >

那么 XML 处理器将在对实体声明进行解析时识别出字符引用，并在将下面的字符串存为实体"example"的值前解析这些字符引用:

An ampersand (&) may be escaped
numerically (&) or with a general entity
(&).

文档中对"&example;"的引用会导致对文本的重新分析，此时元素"p"的起始和结束标记被识别，三个引用被识别和展开，其结果是一个包含下面内容(所有数据，无定界符或标记)的"p"元素：

An ampersand (&) may be escaped
numerically (&) or with a general entity
(&).

一个更复杂的例子可以完整地说明这些规则和它们的作用。在下面的例子中，行号仅仅是为了方便说明。

1 <?xml version='1.0'?>
2 <!DOCTYPE test [
3 <!ELEMENT test (#PCDATA) >
4 <!ENTITY % xx '%zz;'>
5 <!ENTITY % zz '<!ENTITY tricky "error-prone" >' >
6 %xx;
7]>
8 <test>This sample shows a &tricky; method.</test>

这个例子会导致下列动作：

- 在第 4 行，对字符 37 的引用会被立即展开，参数实体"xx"以值"%zz;"存于记号表中。因为置换文本不被再次扫描，对参数实体"zz"的引用不会被识别。(而且如果它被识别的话则是一个错误，因为"zz"还没有被声明。)
- 在第 5 行，字符引用"<"被立即展开，而参数实体"zz"以置换文本"<!ENTITY tricky "error-prone" >"被存储，此置换文本是一个格式良好的实体声明。
- 在第 6 行，对"xx"的引用被识别，"xx"的置换文本(即"%zz;")被解析。对"zz"的引用随后被识别，它的置换文本("<!ENTITY tricky "error-prone" >")被解析。此时通用实体"tricky"被声明，它的置换文本是"error-prone"。
- 在第 8 行，对通用实体"tricky"的引用被识别，并被展开，因此"test"元素的全部内容

为一个自我描述的(不合语法)字符串。此例表明了一种有错误倾向的方法。

E. 确定型内容模型(非标准)

出于兼容性考虑，要求元素类型声明中的内容模型是确定型的。

SGML 要求内容模型是确定型的(它称为"无歧义的"); 用 SGML 系统生成的 XML 处理器可能会把非确定型内容模型标为错误。

例如，内容模型((b, c) | (b, d))是非确定型的，因为给定一个初始 b，解析器没有在向前看以知道 b 后是什么元素之前，无法知道匹配模型中的哪个 b。在这种情况下，两个对 b 的引用可以简化成单个的引用，使得模型成为(b,(c | d))。此时初始的 b 只和内容模型中的一个名字明确匹配。解析器不需要向前看其后的内容。c 或 d 都能被接受。

更正式的说法：使用 Aho，Sethi 和 Ullman 所著[Aho/Ullman]3.9 节中的标准算法 3.5，可以从内容模型构造出一个有限状态自动机。在很多这样的算法中，对应正则表达式中的每一个位置(即正则表达式的语法树中的每个叶子节点)，都构造一个随集(follow set)；如果任一位置的随集中不止一个后继位置被标为同一元素类型时，那么此内容模型出错，并且可以被报为错误。

存在将许多但不是所有非确定型内容模型自动规约为等价的确定型模型的算法；参见 Br 黵 gemann-Klein 1991 [Br 黵 gemann-Klein].

F. 字符编码的自动检测(非标准)

XML 编码声明在实体中以内部标签的方式工作，用于指出使用了何种字符编码。然而，在 XML 处理器能读取这个内部标签前，显然它必须知道当前使用的是何种字符编码－而这正是此内部标签要试图指出的。通常情况下，这是一种无法解决的情况。但在 XML 中并非如此，因为 XML 在两个方面对这种情形作出了限制：假定每一种实现只支持一个有限的字符编码集，并且，为了使得正常情况下自动检测每个实体中所用字符编码成为可能，限制了 XML 编码声明的位置和内容。同时，很多情况下除了 XML 数据流本身之外，另外还有可用的信息源。根据 XML 实体交给处理器时没有或有任何的附带(外部)信息，可以区分出两种情况。我们先考虑第一种情况。

因为每一个非 UTF-8 或 UTF-16 格式的 XML 实体必须以 XML 编码声明开头，其开始的几个字符必须为'<?xml'，任何合乎规范的处理器可以在两到四个八位组的输入后，检测出适用于下列何种情况。在读这张表时，知道这些是有帮助的：在 UCS-4 中，'<'是"#x0000003C"，'?'是"#x0000003F"，UTF-16 数据流的字节次序标记要求为"#xFEFF"。

- 00 00 00 3C: UCS-4，big-endian 编码的计算机(1234 次序)
- 3C 00 00 00: UCS-4，little-endian 编码的计算机(4321 次序)
- 00 00 3C 00: UCS-4，异常的八位组次序(2143)

- 00 3C 00 00: UCS-4，异常的八位组次序(3412)
- FE FF: UTF-16，big-endian
- FF FE: UTF-16，little-endian
- 00 3C 00 3F: UTF-16，big-endian，无字节次序标记(因此严格说来出错)
- 3C 00 3F 00: UTF-16，little-endian，无字节次序标记(因此严格说来出错)
- 3C 3F 78 6D: UTF-8，ISO 646，ASCII，ISO 8859 的一些部分，Shift-JIS，EUC，及其他任何 7 位，8 位或混合宽度的能保证 ASCII 字符有它们正常的位置，宽度，取值的编码;具体其中哪一个适用需读取实际的编码声明来检测确定，但因为所有这些编码中 ASCII 字符的位模式相同，所以能够可靠地读取编码声明本身。
- 4C 6F A7 94: EBCDIC(在某些变种中，完整的编码声明必须能用于确定使用了哪一代码页)
- 其他：无编码声明的 UTF-8，或是数据流已损坏，不完整或被包含在某种外层数据中。

这种层次的自动检测足以用于读取 XML 编码声明和解析字符编码标识符。字符编码标识符仍然是必须的，它用于区分编码方案集中的单个成员(例如从 8859 中区分出 UTF-8，8859 各个部分间的相互区分，以及区分所用的特定 EBCDIC 代码页，等等)。

因为编码声明的内容限于 ASCII 字符，一旦处理器检测到使用的是哪一个编码方案集，它能够可靠地读取整个编码声明。因为在实际中，所有广泛使用的字符编码都可以归于上述种类中，XML 编码声明保证了可靠的内嵌(in-band)字符编码标注，即使是在操作系统或传输协议级的外部信息源并不可靠的情况下。

一旦处理器检测到所用的字符编码，它就可以作出合适的动作，或是针对每种情况调用单独的输入例程，或是对每个输入的字符调用版本合适的转换函数。

和任何自标注(self-labeling)的系统一样，一旦任何软件改变了实体的字符集或其编码而没有相应修改编码声明的话，XML 的编码声明将无法工作。字符编码方案的实现者必须小心，以保证用于标注实体的内部和外部信息的正确性。

第二种可能的情况是 XML 实体有附带的信息，如在一些文档系统和网络协议中。当具有多个信息源时，它们间的相对优先级和首选冲突处理方法必须在传输 XML 的高层协议中给出。例如，内部标签和外部文档头中的 MIME 类型标签的相对优先级应该是定义 text/xml 和 application/xml MIME 类型的 RFC 文档的一部分。然而出于互操作性考虑，建议使用下列规则。

如果 XML 实体是在一个文档中，用字节次序标记和编码声明 PI(如果有的话)来确定字符编码。所有其他信息源和推断都仅仅用于错误恢复。
如果 XML 实体传递时标为 text/xml MIME 类型，那么 MIME 类型的 charset 参数决定了字符编码方法;所有其他信息源和推断都仅仅用于错误恢复。
如果 XML 实体传递时标为 application/xml MIME 类型，那么用字节次序标记和编码声明 PI(如果有的话)来确定字符编码。所有其他信息源和推断都仅仅用于错误恢复。
这些规则只适用于缺少协议级文档时的情况；特别是，当相关 RFC 中定义了这些 text/xml 和 application/xml MIME 类型时，RFC 中的建议取代这些规则。

G. W3C XML 工作组(非正式)

本规范由 W3C XML 工作组(WG)完成并批准发表。工作组批准了本规范并不一定表示工作组的所有成员一致同意本规范。现有和以前的 XML 工作组成员包括：

Jon Bosak, Sun (Chair); James Clark (Technical Lead); Tim Bray, Textuality and Netscape (XML Co-editor); Jean Paoli, Microsoft (XML Co-editor); C. M. Sperberg-McQueen, U. of Ill. (XML Co-editor); Dan Connolly, W3C (W3C Liaison); Paula Angerstein, Texcel; Steve DeRose, INSO; Dave Hollander, HP; Eliot Kimber, ISOGEN; Eve Maler, ArborText; Tom Magliery, NCSA; Murray Maloney, Muzmo and Grif; Makoto Murata, Fuji Xerox Information Systems; Joel Nava, Adobe; Conleth O'Connell, Vignette; Peter Sharpe, SoftQuad; John Tigue, DataChannel