

附录A XML 1.0规范

本附录选自 1998年2月10日发布的W3C建议，可以从以下地址获得完整的文档：

<http://www.w3.org/TR/REC-xml>

版权信息 (c) 1995-1998 WWW协会，(Massachusetts Institute of Technology，Institut National de Recherche en Informatique et en Automatique，Keio University)。版权所有。

<http://www.w3.org/Consortium/Legal/>。

本附录已参照 1999年2月17日发布的勘误表进行修正，可以从以下地址获得勘误表：

<http://www.w3.org/XML/xml-19980210-errata>。

编辑：

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

可扩展标记语言 (Extensible Markup Language，XML) 是SGML的子集，本文对XML进行了完整而详尽的描述。XML的目标是将通用的SGML变为像HTML一样能够在Web上传输和处理的语言。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。这是一个可靠的文档，可以用作参考材料，或者作为其他文档的正式参考文献。在建议制定过程中，W3C力求吸引各界对本规范的注意，并促进规范的推广。这一举措增强了Web的功能和互操作性。

本文档规定了一种用于WWW的语法，此语法是通过取一个业已存在并已广泛使用的国际文本处理标准 (通用标记语言标准，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>获得。在<http://www.w3.org/TR>上列举了现有的W3C建议及其他技术文档。

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

本规范的已知错误清单可以在<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。

可扩展标记语言 (XML) 1.0

目录

1. 简介

- 1.1 起源和目标
- 1.2 术语

2. 文档

- 2.1 格式正规的XML文档
- 2.2 字符
- 2.3 通用语法结构
- 2.4 字符数据和标记
- 2.5 注释
- 2.6 处理指令
- 2.7 CDATA部分
- 2.8 序言和文档类型声明
- 2.9 独立文档声明
- 2.10 空白处理
- 2.11 行尾处理
- 2.12 语言标识

3. 逻辑结构

- 3.1 起始标记、结束标记和空元素标记
- 3.2 元素类型声明
 - 3.2.1 元素内容
 - 3.2.2 混合型内容
- 3.3 属性列表声明
 - 3.3.1 属性类型
 - 3.3.2 属性缺省值
 - 3.3.3 属性值的规格化
- 3.4 条件部分

4. 物理结构

- 4.1 字符引用和实体引用
- 4.2 实体声明
 - 4.2.1 内部实体
 - 4.2.2 外部实体
- 4.3 解析实体
 - 4.3.1 文本声明
 - 4.3.2 格式正规的解析实体

- 4.3.3 实体中的字符编码
- 4.4 XML处理器对实体和引用的处理
 - 4.4.1 不识别
 - 4.4.2 包含
 - 4.4.3 验证有效性时包含
 - 4.4.4 禁止
 - 4.4.5 以文字形式包含
 - 4.4.6 通知
 - 4.4.7 忽略
 - 4.4.8 作为PE包含
- 4.5 内部实体置换文本的结构
- 4.6 预定义实体
- 4.7 表示法声明
- 4.8 文档实体
- 5. 一致性
 - 5.1 验证有效性和不验证有效性的处理器
 - 5.2 使用XML处理器
- 6. 表示法
- 附录
 - 附录A 参考文献
 - A.1 标准化的参考文献
 - A.2 其他参考文献
 - 附录B 字符的分类
 - 附录C XML和SGML (未标准化)
 - 附录D 实体引用和字符引用的展开 (未标准化)
 - 附录E 确定的内容模型 (未标准化)
 - 附录F 字符编码的自动检测 (未标准化)
 - 附录G W3C XML工作组 (非正式)
 - 1. 简介

可扩展标记语言 (简称 XML) 描述了一类称为 XML 文档的数据对象, 与此同时, 它还描述了用于处理这些文档的计算机程序的部分行为。XML 是应用程序的数据文件, 它是 SGML (通用标记语言标准) 的子集, 或称 SGML 的受限形式。从结构上讲, XML 文档是合乎规范的 SGML 文档。

XML 文档是由被称为实体的存储单元构成的, 实体中包含解析数据或未解析数据。解析数据是由字符组成的, 若干字符组合在一起可以形成字符数据或标记。标记描述了文档的存储布局 and 逻辑结构。XML 提供了用于约束存储布局和逻辑结构的机制。

名为 XML 处理器的软件模块用于读取 XML 文档, 并提供对文档内容和结构的访问。实际

上，XML处理器是代表另一个软件模块——应用程序来完成该任务的。本规范通过定义 XML 处理器读取XML数据的方式以及它必须提供给应用程序的信息，描述了 XML处理器必须具备的功能。

1.1 起源和目标

XML是由XML工作组（原称 SGML编辑审议委员会）开发的，该工作组是在 WWW协会（W3C）的支持下于1996年成立的。工作组由Sun Microsystems公司的Jon Bosak领导，并且得到了W3C组建的另一个工作组——XML特殊兴趣组（XML Special Interest Group，原称SGML工作组）的热情帮助。附录中列出了 XML工作组成员名单。Dan Connolly负责工作组与W3C的联络与协调工作。

XML有以下设计目标：

- XML应该可以直接应用于Internet。
- XML应该支持多种应用。
- XML应该与SGML兼容。
- 用于处理XML文档的程序应该易于编写。
- XML的可选特征应该极少，最好没有。
- XML文档应该清晰明了，可读性强。
- XML应该易于设计。
- XML的设计应该正式而简明。
- XML文档应该易于创建。
- XML标记是否简练不甚重要。

本规范以及其他相关标准（定义字符集的统一码和 ISO/IEC 10646，定义语言识别标记的 Internet RFC 1766，定义语言名称代码的 ISO 639，以及定义国家名称代码的 ISO 3166）为理解 XML 1.0以及构建处理XML文档的计算机程序提供了所有必需的信息。

在保证规范完整性和合法性的前提下，XML 1.0规范可以免费发行。

1.2 术语

本规范定义了用于描述 XML文档的术语。在定义 XML文档以及描述XML处理器的行为时，我们将用到下列术语：

可以

表示合乎规范的文档和XML处理器允许出现规范中描述的行为，但是并不强行要求。

必须

表示合乎规范的文档和XML处理器需要具有规范中描述的行为；否则视为错误。

错误

违反本规范定义的规则；其导致的结果没有明确定义。表示合乎规范的软件可以检测并报告错误，或者修复错误。

致命错误

XML处理器必须检测并报告给应用程序的错误。遇到致命错误后，处理器可以继续处理数据以寻找出更多的错误，并报告给应用程序。为了支持纠错功能，处理器可以根据提供给应用

程序的文档（字符数据与标记的混合体）生成未处理的数据。然而，一旦检测到致命错误，处理器必须停止正常的处理过程（例如，它不能以常规的方式继续将字符数据和有关文档逻辑结构的信息传递给应用程序）。

根据用户的选择

表示合乎规范的软件可以或者必须（取决于句子中的情态动词）具备所描述的行为；如果确实如此，它必须为用户提供允许或禁止该方法的方法。

有效性约束

应用于所有有效的 XML 文档的规则。违反有效性约束将导致错误；它们必须根据用户的选择报告给验证有效性的 XML 处理器。

格式正规约束

应用于所有格式正规的 XML 文档的规则。违反格式正规约束将产生致命错误。

匹配

（对于字符串或名称：）两个被比较的字符串必须完全相同。对于 ISO/IEC 10646 定义的可能有多种表示法的字符（例如：预定义（precomposed）形式和基字符加变音符形式属于两种不同的字符形式），仅当它们在两个字符串中具有相同的表示法时，才被认为是匹配的。根据用户的选择，处理器可以将这些字符规格化为某种规范的形式。不进行字符的大小写转换。（对于字符串和语法中的规则：）当字符串属于由语法产生式产生的语言时，认为字符串和该语法产生式匹配。（对于内容和内容模型：）当元素符合“元素有效性”约束中描述的形式时，认为元素与其声明匹配。

出于兼容性考虑

仅仅是为了保证 XML 与 SGML 兼容而包含的 XML 特征。

出于互操作性考虑

是一个不具约束性的建议，其目的是增加 XML 文档能被在 ISO 8879 的 WebSGML 改编附件（WebSGML Adaptations Annex）之前已有的 SGML 处理器处理的可能性。

2. 文档

如果一个数据对象根据本规范的定义被认为是格式正规的，它就可以称为 XML 文档。如果格式正规的 XML 文档能够满足进一步的约束条件，它就是有效的。

每个 XML 文档都有逻辑结构和物理结构。从物理结构上讲，文档是由被称为实体的单元组成的。一个实体可以引用其他实体，从而通过这种方式将这些实体也包含到文档中。文档是以“根”或称文档实体开始的。从逻辑结构上讲，文档是由声明、元素、注释、字符引用和处理指令构成的，它们均由显式的标记来表示。逻辑结构和物理结构都必须按照“4.3.2 格式正规的解析实体”中描述的要求正确嵌套。

2.1 格式正规的 XML 文档

满足以下条件的文本对象被认为是格式正规的 XML 文档：

- 从整体上来看，它与 document 产生式匹配。
- 它满足本规范定义的所有格式正规约束。
- 文档中直接或间接引用的每个解析实体都是格式正规的。

表 A-1

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

与document产生式匹配意味着：

- 它包含一个或多个元素。
- 有且只有一个称为根或文档元素的元素，其他所有元素均包含在该元素中。对于其他所有元素，如果起始标记出现在另一个元素的内容中，则相应的结束标记也必须位于同一元素中。简而言之，元素是由起始标记和结束标记定界的，各个元素必须正确地嵌套。

因此，对于文档中的任意非根元素 C，如果它包含在文档的另一个元素 P 中，且不包含在 P 中的任何其他元素中，则 P 称为 C 的父元素，C 称为 P 的子元素。

2.2 字符

解析实体包含文本，文本是由一系列字符构成的，它可以代表标记或字符数据。根据 ISO/IEC 10646 [ISO/IEC 10646] 的定义，字符是组成文本的原子单元。制表符、回车、换行，以及统一码和 ISO/IEC 10646 中定义的合法的图形字符都是合法的字符。本规范不提倡使用 [Unicode] 中 6.8 节定义的“兼容字符”。

表 A-2

字符范围				
[2]	Char	::=	#x9 #xA #xD [#x20- #xD7FF] [#xE000-#xFFFD] [#x10000-#x10FFFF]	/*除了替代块、FFFE和FFFF 之外的任何统一码字符。*/

产生式[2]是规格化的；实际上，这意味着 Euro (€ €) 等新增的统一码字符在XML文档中是合法的。

各种实体可能采用不同的字符编码机制。所有 XML 处理器必须接受 10646 的 UTF_8 和 UTF-16 编码；我们将在“ 4.3.3 实体中的字符编码 ”一节讨论如何标识这两种编码，以及如何指定使用其他编码机制。

2.3 通用语法结构

本节定义了一些常用的语法规符号。

S (空白) 包含一个或多个空格 (#x20) 字符、回车、换行或制表符。

表 A-3

空白			
[3]	S	::=	(#x20 #x9 #xD #xA)+

为了便于使用，字符被分为字母、数字和其他字符。字母可以是字母表中的字母字符，或者是音节基字符后跟一个或多个组合字符，也可以是表意字符。“ B. 字符的分类 ”一节提供了每类字符的完整定义。

名称是一种以字母或某个特定的字符开头，后面跟随着若干名称字符的记号，其中名称字

符包括：字母、数字、连字符、下划线、冒号和句点。在本规范以及后续版本的规范中，任何以字符串“ xml ”或者其他与((‘ X ’ | ‘ x ’)(‘ M ’ | ‘ m ’)(‘ L ’ | ‘ l ’)) 匹配的字符串开头的名称都是为了标准化而保留的。

注意 XML名称中的冒号字符是为试验命名空间而保留的。这意味着它将来可能被标准化，到那时所有将冒号用作试验目的文档都要被更新。（事实上，XML采用的命名空间机制并不一定将冒号作为分隔符。）因此，除非作者确实想表示命名空间，否则不应该在XML名称中使用冒号，但是XML处理器应该将冒号视作名称字符。

Nmtoken（名称记号）是任意名称字符的混合体。

表 A-4

名称和记号			
[4]	NameChar	::=	Letter Digit ‘ ’ ‘ ’ ‘ ’ ‘ ’ CombiningChar Extender
[5]	Name	::=	(Letter ‘ ’ ‘ ’) (NameChar)*
[6]	Names	::=	Name (S Name)*
[7]	Nmtoken	::=	(NameChar)+
[8]	Nmtokens	::=	Nmtoken (S Nmtoken)*

文字数据是指字符串中被引用的部分，它不包含作为字符串定界符的引号。文字数据用于指定内部实体的内容（ EntityValue ） 属性值（ AttValue ） 和外部标识符（ SystemLiteral ）。需要注意的是，解析SystemLiteral时可以不扫描标记。

表 A-5

文字数据			
[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

文本是由字符数据和标记混合而成的。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部分定界符、文档类型声明和处理指令。

所有非标记的文本构成了文档的字符数据。

“与”符号（&）和左尖括号（<）只能在作为标记定界符，或者位于注释、处理指令和CDATA部分中时，才能以原来的文字形式出现。它们也可以用在内部实体声明的文字实体值中；参见“4.3.2 格式正规的解析实体”。如果其他地方需要这两个字符，必须用相应的转义字符，例如：数字字符引用，或者“&”和“<”串。右尖括号（>）可以用“>”串表示，但是，出于兼容性考虑，当它出现在内容中的字符串“]]>”中，且不表示CDATA部分的结束时，必须用“>”或字符引用来替代。

在元素内容中，字符数据是不包含任何标记的起始定界符的字符串。在CDATA部分中，字符数据是任何不包含CDATA部分结束定界符“]]>”的字符串。

在属性值中，单引号和双引号也要用转义字符表示，单引号（'）表示为“'”，双引号（"）表示为“"”。

表 A-6

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

2.5 注释

在文档中，注释可以出现在标记之外的任何位置。另外，它们也可以出现在文档类型声明中语法允许的地方。注释不属于文档的字符数据；XML处理器可以为应用程序提供获取注释文本的机制，但这并不是必须的。出于兼容性考虑，字符串“--”（双连字符）绝对不能出现在注释中。

表 A-7

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

下面是一个注释的例子：

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

2.6 处理指令

处理指令（Processing Instruction，PI）允许文档包含要传递给应用程序的指令。

表 A-8

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

PI不是文档的字符数据的一部分，但是它必须传递给应用程序。PI以目标（PITarget）开始，目标用于标识要接收指令的应用程序。在本规范及后续版本的规范中，“XML”、“xml”等目标名称都是为标准化而保留的。XML表示法机制可以用于PI目标的正式声明。

2.7 CDATA部分

要是可以出现字符数据的位置就可以有CDATA部分；它用于转义文本块中的某些字符，以免它们被识别为标记。CDATA部分以“<![CDATA[”串作为开始，以“]]>”串作为结束。

表 A-9

CDATA部分			
[18]	CDsect	::=	CDStart CData CEnd
[19]	CDStart	::=	'<![CDATA['
[20]	CData	::=	(Char*- (Char*']>' Char*))
[21]	CEnd	::=	']>'

在CDATA部分中，只有CEnd串被识别为标记，因此左尖括号和“与”符号可以以原有的文字形式出现在CDATA部分中；它们不需要（也不能）替换为“<”和“&”。CDATA部分不能嵌套。

在下面的CDATA部分，“<greeting>”和“</greeting>”被识别为字符数据，而不是标记：

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

2.8 序言和文档类型声明

XML文档可以且应该以XML声明开头，声明指定了所使用的XML版本。例如，下面是一个完整的XML文档，它是格式正规的，但不是有效的：

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

它等同于下面的文档：

```
<greeting>Hello, world!</greeting>
```

版本号1.0表示文档符合此版本的规范；如果不遵守此版本的规范的文档使用了值“1.0”就会产生错误。XML工作组决心制定更高版本的规范，但是这并不意味着将来一定会有新的XML版本诞生，即使产生新的规范，也不一定会使用任何特定的编号方式。由于有可能出现新的版本，XML声明为自动版本识别提供了可能性，因此该语句本应该成为必须的。如果处理器不支持文档所标识的版本，它将产生错误通知。

在XML文档中，标记的作用是描述文档的存储布局 and 逻辑结构，并且将属性-值对与其逻辑结构相关联。XML提供的文档类型声明能够定义对逻辑结构的约束，并支持使用预定义的存储单元。如果XML文档有相关的文档类型声明，且文档符合声明中的约束，则认为XML文档是有效的。

文档类型声明必须位于文档的第一个元素之前。

表 A-10

序言			
[22]	prolog	::=	XMLDecl? Misc* (doctypeDecl Misc*)?
[23]	XMLDecl	::=	'<?xml' VersionInfo EncodingDecl? SDDDecl? 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，标记声明提供了某一类文档的语法。该语法又称为文档类型定义（Document Type Definition，DTD）。文档类型声明可以指向包含标记声明的外部子集（特殊类型的外部实体），也可以直接在内部子集中包含标记声明，或者两者兼而有之。文档的DTD是由这两种类型的子集构成的。

标记声明可以是元素类型声明、属性列表声明、实体声明或者表示法声明。正如下面介绍格式正规约束和有效性约束时所描述的，这些声明可以全部或部分包含在参数实体中。要了解更完整的信息，参见“4. 物理结构”。

表 A-11

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：正确的声明/PE嵌套] [WFC：内部子集中的PE]

标记声明可以全部或部分由参数实体的置换文本构成。本规范后面要定义的各个非终结符（elementdecl，AttlistDecl，等等）产生式描述了在所有的参数实体被包含之后的声明。

有效性约束：根元素类型

文档类型声明中的Name必须与根元素的元素类型匹配。

有效性约束：正确的声明/PE嵌套

参数实体的置换文本必须与标记声明正确嵌套。这意味着如果标记声明（即：上面的markupdecl）的第一个字符或最后一个字符包含在参数实体引用的置换文本中，则整个标记声明必须包含在同一置换文本中。

格式正规约束：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可以出现的位置，而不能位于标记声明中。（这条约束不适用于外部参数实体中的引用或外部子集中的引用。）

与内部子集类似，DTD引用的外部子集和外部参数实体必须由一系列完整的标记声明构成，其中可以夹杂空白字符或参数实体引用，非终结符markupdecl定义了所允许的标记声明类型。然而，利用条件部分结构，可以根据特定的条件忽略外部子集或外部参数实体的部分内容；而这在内部子集中是不允许的。

表 A-12

外部子集				
[30]	extSubset	::=	TextDecl? extSubsetDecl	

(续)

外部子集

[31]	extSubsetDecl	::=	(markupdecl conditionalSect PEReference S)*
------	---------------	-----	--

外部子集和外部参数实体与内部子集还存在着一些差异，它们的参数实体引用不仅可以出现在标记声明之间，而且可以位于标记声明内。

下面是一个含文档类型声明的 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” 指定了文档 DTD 的 URL。

声明也可以直接包含在文档中，例如下面的例子：

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

如果文档同时使用了内部子集和外部子集，系统认为内部子集出现在外部子集之前。这意味着内部子集中的实体和属性列表声明具有更高的优先级。

2.9 独立文档声明

当 XML 处理器将文档传递给应用程序时，标记声明会影响文档的内容，例如：属性缺省值和实体声明。独立文档声明可以是 XML 声明的一部分，它用于说明文档实体之外是否有这类标记声明。

表 A-13

独立文档声明

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

在独立文档声明中，值 “yes” 表示文档实体没有外部标记声明（无论在 DTD 外部子集中，还是在内部子集的外部参数实体引用中），因此没有外部声明会影响 XML 解析器传递给应用程序的信息。值 “no” 表示可能存在于这类外部标记声明。独立文档声明仅仅说明是否存在外部声明；但是如果文档引用了内部声明的外部实体，它不影响文档的独立状态。

如果不存在外部标记声明，独立文档声明就没有任何意义。如果文档存在外部标记声明，但没有独立文档声明，则假设取值为 “no”。

某些在网络上传输的应用程序可能需要独立的文档，从算法角度讲，任何 standalone="no" 的 XML 文档都能够转化为独立的文档。

有效性约束：独立文档声明

如果文档的外部标记声明包含以下类型的声明，则独立文档声明的值必须设置为 “no”：

- 含缺省值的属性声明，且具有该属性的元素在文档中没有指定属性值。
- （除amp、lt、gt、apos和quot之外的）实体声明，且文档中引用了该实体。
- 属性值受规格化限定的属性声明，且文档中的属性值将因规格化而修改。
- 具有元素内容的元素类型声明，且这些类型的任何实例中直接出现空白。

下面的例子显示了含独立文档声明的 XML 声明：

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

2.10 空白处理

在编辑 XML 文档时，为了增强文档的可读性，经常要使用“空白”（空格、制表符或空行，本规范中用非终结符 S 表示）来分隔标记。这些空白通常不需要出现在文档的发布版本中。另一方面，有些“重要的”空白应该保留，例如诗歌和源代码中的空白。

XML 处理器必须将文档中的所有非标记字符传递给应用程序。验证有效性的 XML 处理器必须告知应用程序元素内容中的哪些字符构成了空白。

元素中可以附加一个名为 xml:space 的特殊属性，它用于通知应用程序应该保留此元素中的空白。在有效的文档中，与其他属性一样，该属性必须在使用之前进行声明。该属性必须声明为枚举类型，它的可选值只有“default”和“preserve”。例如：

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

值“default”表明对于该元素可以使用应用程序缺省的空白处理模式；值“preserve”表明应该为应用程序保留所有空白。该声明将作用于它所在的元素内容中的所有元素，除非被另一个 xml:space 属性实例所覆盖。

任何文档的根元素通常被认为对应用程序的空白处理方式不作要求，除非它设置了 xml:space 属性的值，或者将该属性声明为带缺省值的。

2.11 行尾处理

XML 解析实体一般保存在计算机文件中，为了编辑方便，它们是按行组织的。这些行通常用某些字符的组合来分隔，例如：回车（#xD）和换行（#xA）。

为了简化应用程序的工作，如果外部解析实体或内部解析实体的文字实体值包含字符序列“#xD#xA”或独立的字符 #xD，XML 处理器必须将它们替换为单字符 #xA 传递给应用程序。（在解析文档之前，只要将所有换行规格化为 #xA，就能够简便地实现该行为。）

2.12 语言标识

在处理文档时，标识出内容所用的自然语言或形式语言是非常有必要的。为此，可以在文档中插入名为 xml:lang 的特殊属性，它能够指定 XML 文档中任意元素的内容和属性值所用的语言。在有效的文档中，与其他属性一样，该属性必须在使用之前进行声明。 [IETF RFC 1766]（“语言识别标记”）定义了该属性值可以选择的语言标识符：

表 A-14

语言标识			
[33]	LanguageID	::=	Langcode ('-' Subcode)*
[34]	Langcode	::=	ISO639Code IanaCode UserCode
[35]	ISO639Code	::=	([a-z] [A-Z]) ([a-z] [A-Z])

(续)

语言标识			
[36]	IanaCode	::=	('I' 'T') '-' ([a-z] [A-Z])+
[37]	UserCode	::=	('x' 'X') '-' ([a-z] [A-Z])+
[38]	Subcode	::=	([a-z] [A-Z])+

Langcode可以取以下值：

- [ISO 639] (“语言名称的表示码”) 中定义的双字母的语言代码。
- IANA (Internet Assigned Numbers Authority) 注册的语言标识符；它们是以前缀“i-” (或“I-”) 开头的。
- 用户设置的语言标识符，或者各方经协商同意的语言标识符；为了不与标准化的名称或IANA注册的名称冲突，它们必须以前缀“x-”或“X-”开头。

Subcode段的数量不受限制；如果存在第一个子代码段且它包含两个字母，则它必须是 [ISO 3166] (“国家名称的表示码”) 中定义的国家代码。如果第一个子代码包含的字母多于两个，则它必须是IANA注册的语言的子代码，除非Langcode是以前缀“x-”或“X-”开头的。

习惯上，语言代码用小写字母表示，国家代码（如果存在的话）用大写字母表示。需要注意的是，与XML文档中的其他名称不同，这些值是不区分大小写的。

例如：

```
<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 heißem Bemüh'n.</l>
</sp>
```

属性声明xml:lang将应用于它所在的元素的所有属性和内容，除非被内容中的另一个元素的xml:lang实例所覆盖。

简单的xml:lang属性声明可能具有以下形式：

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

如果需要的话，也可以给出特定的缺省值。在下面的例子中，有一本要给英国学生阅读的法文诗集，它的注释和说明都是英文的，它可以有以下xml:lang属性声明：

```
<!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文档都包含一个或多个元素，它们通过起始标记和结束标记定界，或者对于空元素，使用空元素标记来表示元素的边界。每个元素都有一个用名称标识的类型，有时称之为“通用标识符 (Generic Identifier, GI)，元素还可以有属性定义集合。属性定义是由名称和值构成的。

表 A-15

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

本规范对元素类型和属性的语义、用法和（除了语法之外的）名称没有特殊限制，但是在本规范的当前版本及后续版本中，以((‘X’|‘x’)(‘M’|‘m’)(‘L’|‘l’))开头的名称都是专为标准化而保留的。

格式正规约束：元素类型匹配

元素的结束标记中的名称必须与起始标记中的元素类型匹配。

有效性约束：元素有效性

对于一个与elementdecl匹配的声明，如果声明的 Name与元素类型匹配，且满足以下条件之一，则认为元素是有效的：

- 声明与EMPTY匹配，且元素没有内容。
- 声明与children匹配，且子元素序列属于根据内容模型中的正则表达式产生的语言，每对子元素之间可以有空白（与非终结符 S匹配的字符）。
- 声明与Mixed匹配，且内容包含字符数据和子元素，其中子元素的类型与内容模型中的名称匹配。
- 声明与ANY匹配，且任何子元素的类型都已经声明。

3.1 起始标记、结束标记和空元素标记

每个非空XML元素都是以起始标记作为开始的。

表 A-16

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：属性值中不含<]

起始标记和结束标记中的Name指定了元素的类型。Name-AttValue对被称为元素的属性说明，其中的Name代表属性名称，AttValue的内容（定界符'或"之间的文本）是属性值。

格式正规约束：唯一的属性说明

任何属性名称在起始标记或空元素标记中至多只能出现一次。

有效性约束：属性值类型

属性必须提前声明；它的值必须是所声明的类型。（“ 3.3 属性列表声明 ”一节详细定义了属性类型。）

格式正规约束：无外部实体引用

属性值不能包含对外部实体的直接或间接的实体引用。

格式正规约束：属性值中不含 <

属性值直接或间接引用的实体的置换文本绝对不能包含 <，但是可以使用“<”。

下面是一个起始标记的例子：

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

对于以起始标记开始的元素，必须以结束标记终止，结束标记中的名称应该与起始标记中的元素类型相同。

表 A-17

结束标记			
[42]	ETag	::=	'<' Name S? '>'

下面是一个结束标记的例子：

```
</termdef>
```

起始标记和结束标记之间的文本称为元素的内容：

表 A-18

元素内容			
[43]	content	::=	(element CharData Reference CDSEct PI Comment)*

如果元素为空，它必须以连续的起始标记和结束标记表示，或者以空元素标记表示。空元素标记的格式如下：

表 A-19

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

空元素标记可以应用于任何无内容的元素，无论该元素在声明时是否使用了关键字 EMPTY。出于互操作性考虑，空元素标记必须且只能用于声明为 EMPTY 的元素。

下面是空元素的例子：

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

3.2 元素类型声明

为了保证文档的有效性，XML 文档的元素结构要使用元素类型和属性列表声明进行约束。元素类型声明用于限制元素的内容。

元素类型声明指定了该元素可以包含的子元素类型。根据用户的选择，当声明中涉及的元素类型没有声明时，XML 处理器可以产生警告，但这并不是错误。

元素类型声明的格式如下：

表 A-20

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是所声明的元素类型。
有效性约束：唯一的元素类型声明
元素类型至多声明一次。
下面是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

当元素只能包含以空白（与非终结符 S 匹配的字符）分隔的子元素（不含字符数据）时，我们称元素类型具有元素内容。在这种情况下，使用内容模型对元素进行约束，内容模型是用于指定子元素类型及顺序的简单语法。该语法是建立在内容粒子（Content Particle，CP）基础上的，内容粒子包含名称、内容粒子的选择列表或顺序列表：

表 A-21

元素内容模型				
[47]	children	::=	(choice seq) ('?' '*' '+')?	
[48]	cp	::=	(Name choice seq) ('?' '*' '+')?	
[49]	choice	::=	'(' S? cp (S? ' ' S? cp)* S? ')'	[VC：正确的组/PE 嵌套]
[50]	seq	::=	'(' S? cp (S? ',' S? cp)* S? ')'	[VC：正确的组/PE 嵌套]

其中Name是子元素的类型。选择列表中的任何内容粒子都可以出现在元素内容中，它的位置应该是选择列表在语法中出现的位置；顺序列表中的内容粒子必须按照列表中指定的顺序出现在元素内容中。名称或列表后的可选字符用于控制列表中的元素或内容粒子出现的次数，例如：+表示一次或多次，*表示零次或多次，?表示零次或一次。如果缺少这类运算符，则意味着元素或内容粒子必须出现一次，且只能出现一次。上述语法和含义可以应用于本规范的所有产生式。

当且仅当元素内容能够与内容模型吻合，遵守顺序、选择和重复运算符，且内容中的每个元素与内容模型中的元素类型匹配，则称元素内容与内容模型匹配。出于兼容性考虑，如果某个元素能够与内容模型中一个元素类型的多次出现匹配，系统会认为出现错误。要了解更详细的信息，参见“E. 确定的内容模型”。

有效性约束：正确的组/PE嵌套

参数实体的置换文本必须与用括号括起来的组正确嵌套。换言之，如果 choice、seq或Mixed结构中的开始或结束括号包含在参数实体的置换文本中，则这两个括号必须同时包含在该置换文本中。出于互操作性考虑，如果 choice、seq或Mixed结构中出现参数实体引用，它的置换文本不应该为空，而且置换文本的第一个和最后一个非空字符都不应该是连接符（|或,）。

下面是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

当元素中可以包含字符数据，且其间能够随意穿插子元素时，称元素类型具有混合型内容。在这种情况下，子元素的类型要受到约束，但是它们的顺序或出现的次数不受限制。

表 A-22

混合型内容声明				
[51]	Mixed	::=	'(' S? '#PCDATA' (S? ' ' S? Name)*S? ')*' '(' S? '#PCDATA' S? ')'	[VC：正确的组/PE嵌套] [VC：无重复类型]

其中，Name代表元素中将出现的子元素的类型。

有效性约束：无重复类型

在一个混合型内容声明中，同一名称至多出现一次。

下面是混合型内容声明的例子：

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

3.3 属性列表声明

属性用于将名称-值对与元素相关联。属性说明只能出现在起始标记和空元素标记中；“ 3.1 起始标记、结束标记和空元素标记 ”一节介绍了用于识别它们的产生式。属性列表声明可以有以下用途：

- 定义适用于特定元素类型的属性集合。
- 为属性建立类型约束。
- 提供属性的缺省值。

属性列表声明指定了与特定元素类型相关的每个属性的名称、数据类型和缺省值（如果有缺省值的话）。

表 A-23

属性列表声明				
[52]	AttlistDecl	::=	'<!ATTLIST' S Name AttDef*S? '>'	
[53]	AttDef	::=	S Name S AttType S DefaultDecl	

AttlistDecl定义中的Name指定了元素的类型。根据用户的选择，如果属性中声明的元素类型

未被声明，XML处理器可以发出警告，但这并不是错误。AttDef定义中的Name代表属性的名称。

如果特定的元素类型有多个 AttlistDecl时，它们的内容将被合并。对于特定元素类型的同一属性，如果有多个定义，XML处理器将采用第一个声明，而忽略后续的声明。出于互操作性考虑，DTD的作者可以选择为特定的元素类型至多提供一个属性列表声明，为特定的属性名称至多提供一个属性定义，而且每个属性列表声明中至少有一个属性定义。出于互操作性考虑，当特定的元素类型有多个属性列表声明，或者特定的属性有多个属性定义时，XML处理器可以根据用户的选择发出警告，但这并不是错误。

3.3.1 属性类型

XML定义了以下三种属性类型：字符串类型、记号化类型和枚举类型。字符串类型的值可以是任何文字串；记号化类型受到许多不同的词汇和语义约束，下面的表格显示了这些类型的定义：

表 A-24

属性类型			
[54]	AttType	::=	StringType TokenizedType EnumeratedType
[55]	StringType	::=	'CDATA'
[56]	TokenizedType	::=	'ID' [VC : ID] [VC : 每个元素类型 一个ID] [VC : ID属性缺省值] 'IDREF' [VC : IDREF] 'IDREFS' [VC : IDREF] 'ENTITY' [VC : 实体名称] 'ENTITIES' [VC : 实体名称] 'NMTOKEN' [VC : 名称记号] 'NMTOKENS' [VC : 名称记号]

- 有效性约束：ID
- ID类型的值必须与Name产生式匹配。在一个XML文档中，作为该类型值的名称至多出现一次，即：ID值必须是相应元素的唯一标识符。
- 有效性约束：每个元素类型一个ID
- 任何元素类型至多有一个ID属性。
- 有效性约束：ID属性缺省值
- ID属性必须有声明为#IMPLIED或#REQUIRED的缺省值。
- 有效性约束：IDREF
- IDREF类型的值必须与Name产生式匹配，IDREFS类型的值必须与Names产生式匹配；每个Name必须与XML文档中某个元素的ID属性值匹配；即：IDREF的值必须与某个ID属性的值匹配。
- 有效性约束：实体名称
- ENTITY类型的值必须与Name产生式匹配，ENTITIES类型的值必须与Names产生式匹配；每个Name都必须与DTD中声明的未解析实体的名称匹配。

有效性约束：名称记号

NMTOKEN类型的值必须与Nmtoken产生式匹配，NMTOKENS类型的值必须与Nmtokens匹配。

枚举属性的值应该从声明中的值列表选择。XML定义了两种枚举类型：

表 A-25

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

NOTATION属性用于标识一种表示法，它是在 DTD中声明的，与系统和 /或公共标识符相关联，它用于解释与属性相关的元素。

有效性约束：表示法属性

该类型的值必须与声明中的某个表示法名称匹配；属性列表声明中的表示法名称必须是已经声明的。

有效性约束：每个元素类型一种表示法。

没有一个元素类型可以有一个以上指定的 NOTAION属性。

有效性约束：枚举

该类型的值必须与声明中的某个 Nmtoken记号匹配。

出于互操作性考虑，在一个元素类型的枚举属性类型中，同一 Nmtoken至多出现一次。

3.3.2 属性缺省值

属性声明指明了属性是否必须出现，以及当文档中不包含已声明的属性时，XML处理器应该如何应对。

表 A-26

属性缺省值				
[60]	DefaultDecl	::=	'#REQUIRED' '?#IMPLIED' ((' #FIXED' S)? AttValue)	[VC：必需的属性] [VC：属性缺省值的合法性] [WFC：属性值中不含<] [VC：固定的属性缺省值]

在属性声明中，#REQUIRED意味着必须提供属性，#IMPLIED表示属性可有可无。如果声明不是#REQUIRED或#IMPLIED，则AttValue包含声明的缺省值；#FIXED关键字说明属性必须有缺省值。如果声明了缺省值，当XML处理遇到被省略的属性时，它认为文档中存在该属性，且具有所声明的缺省值。

有效性约束：必需的属性

如果缺省值声明使用了关键字 #REQUIRED，元素必须说明该类型的属性。

有效性约束：属性缺省值的合法性

所声明的缺省值必须满足声明的类型的词汇约束。

有效性约束：固定的属性缺省值

如果属性声明使用了关键字 #FIXED，且指定了缺省值，则该属性的实例必须与缺省值匹配。

下面是属性列表声明的例子：

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

3.3.3 属性值的规格化

在XML处理器将属性值传递给应用程序或者检查其有效性之前，必须对它执行以下规格化操作：

- 对于字符引用，将被引用的字符插入属性值中。
- 对于实体引用，以递归的方式处理实体的置换文本。
- 对于空白字符（#x20、#xD、#xA、#x9），将#x20添加到规格化的值中，对于外部解析实体或内部解析实体的文字实体值中的“#xD#xA”序列需要特别处理，仅仅添加一个#x20。
- 对于其他字符，直接将它们添加到规格化的值中。

如果所声明的值不是CDATA，XML处理器必须对规格化的值做进一步处理，首先删除前后的空格字符（#x20），然后用一个空格字符（#x20）替换若干连续的空格字符（#x20）。

不具备有效性验证功能的解析器会将所有未声明的属性作为已声明为CDATA的属性处理。

3.4 条件部分

只有文档类型声明外部子集中才能有条件部分，条件部分是否包含在 DTD的逻辑结构中取决于用于控制它的关键字的值。

表 A-27

条件部分			
[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*)

与内部和外部 DTD 子集类似，条件部分也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部分，且其间可以夹杂空白。

如果条件部分的关键字为 INCLUDE，则条件部分的内容将成为 DTD 的一部分。如果条件部分的关键字为 IGNORE，则条件部分的内容从逻辑上不包含在 DTD 中。为了实现可靠的解析，即使对于被忽略的条件部分，也必须读取它的内容，以便发现嵌套的条件部分，并确保正确检测出（被忽略的）条件部分最外层的结束标记。如果关键字为 INCLUDE 的条件部分包含在关键字为 IGNORE 的条件部分中，外层和内层的条件部分都将被忽略。

如果条件部分的关键字是参数实体引用，处理器首先必须用所引用的内容替换参数实体，然后再决定包含或忽略条件部分。

下面是一个条件部分的例子：

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

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

4. 物理结构

XML 文档可以包含一个或多个存储单元。它们被称为实体；实体都有内容，并且都通过实体名称进行标识（除了文档实体和外部 DTD 子集之外）。每个 XML 文档都有一个称为文档实体的实体，它作为 XML 处理器的起始点，可以包含整个文档。

实体可以是解析的或未解析的。解析实体（parsed entity）的内容称为置换文本；这些文本被看作是文档的一部分。

未解析实体是一种资源，它的内容不一定是文本，如果内容是文本，也可能不是 XML。每个未解析实体都有相关的表示法，表示法是以名称标识的。XML 处理器需要为应用程序提供实体和表示法的标识符，除此之外，XML 对未解析实体的内容没有任何限制。

解析实体是以实体引用的方式通过名称调用的；未解析实体是通过在 ENTITY 或 ENTITIES 属性值中指定名称实现引用的。

通用实体是指在文档内容中使用的实体。在本规范中，在不引起混淆的情况下，通用实体有时用未修饰的术语“实体”来表示。参数实体是指在 DTD 中使用的解析实体。这两种类型的实体采用不同的引用形式，并且应用于不同的环境。另外，它们使用不同的命名空间；具有相同名称的参数实体和通用实体实际上是两个截然不同的实体。

4.1 字符引用和实体引用

字符引用是引用 ISO/IEC 10646 字符集中的特殊字符，例如无法直接从输入设备输入的字符。

表 A-28

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

格式正规约束：合法字符

字符引用中所引用的字符必须与 Char产生式匹配。

如果字符引用以“&#x”开头，则它后面直至终止符，之间的数字和字母构成了 ISO/IEC 10646中字符代码的十六进制表示。如果它以“&#”开头，它后面直至终止符，之间的数字构成了字符代码的十进制表示。

实体引用是引用命名实体中的内容。对解析通用实体的引用以与号（&）和分号（;）作为定界符。参数实体引用使用百分号（%）和分号（;）作为定界符。

表 A-29

实体引用				
[67]	Reference	::=	EntityRef CharRef	
[68]	EntityRef	::=	'&' Name ';'	[WFC：声明实体] [VC：声明实体] [WFC：解析实体] [WFC：无递归]
[69]	PEReference	::=	'%' Name ';'	[VC：已声明的实体] [WFC：无递归] [WFC：在DTD中]

格式正规约束：声明实体

对于没有DTD的文档，或者只有内部DTD子集且其中不含参数实体引用的文档，或者独立文档声明为“standalone='yes'”的文档，除了amp、lt、gt、apos和quot是格式正规的文档不需要声明的实体，其他实体引用中的 Name必须与实体声明中的 Name匹配。参数实体必须先声明后引用。同样，如果属性列表声明的缺省值中包含对通用实体的引用，必须在此之前声明该通用实体。

需要注意的是，如果实体是在外部子集或外部参数实体中声明的，不具备有效性验证功能的处理器没有义务读取和处理这些声明；对于这类文档，仅当 standalone='yes'时，实体必须声明的规则才是一个格式正规约束。

有效性约束：声明实体

对于有外部子集或外部参数实体且“standalone='no'”的文档，实体引用中的 Name必须与实体声明中的 Name匹配。出于互操作性考虑，有效的文档应该依据“4.6 预定义实体”一节中指定的形式声明实体amp、lt、gt、apos和quot。参数实体必须先声明后引用。同样，如果属性列表声明的缺省值中包含对通用实体的引用，必须在此之前声明该通用实体。

格式正规约束：解析实体

实体引用不能包含未解析实体的名称。只有在声明为 ENTITY或ENTITIES类型的属性的值中才能引用未解析实体。

格式正规约束：无递归

解析实体不能包含直接或间接的对自身的递归引用。

格式正规约束：在DTD中

参数实体引用只能出现在 DTD中。

下面是字符引用和实体引用的例子：

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

下面是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

实体声明的定义如下：

表 A-30

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

实体引用中使用 Name 标识实体；在 ENTITY 或 ENTITIES 属性值中，同样使用 Name 标识未解析实体。如果同一实体被多次声明，则遇到的第一个声明被采纳；根据用户的选择，如果实体被多次声明，XML 处理器可以发出警告。

4.2.1 内部实体

如果实体定义为 EntityValue，则被定义的实体称为内部实体。它没有独立的物理存储对象，实体的内容是在声明中给出的。值得注意的是，处理某些文字实体值中的实体引用和字符引用时，需要产生正确的置换文本，参见“4.5 内部实体置换文本的结构”。

内部实体是解析实体。

下面是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

如果实体不是内部的，它就是外部实体，它的声明如下：

表 A-31

外部实体声明			
[75]	ExternalID	::=	'SYSTEM' S SystemLiteral 'PUBLIC' S PubidLiteral S SystemLiteral
[76]	NDataDecl	::=	S 'NDATA' S Name [VC：声明表示法]

如果出现 NDataDecl，说明这是通用未解析实体；否则，它是解析实体。
有效性约束：声明表示法

上述声明中的Name必须与表示法所声明的名称匹配。

SystemLiteral称为实体的系统标识符。它是一个用于获取实体的URI。根据正式的定义，URI中经常出现的井号(#)和标识符片断并不是URI的一部分；如果系统标识符中包含标识符片断，XML处理器可以将它视作错误。除非在本规范之外另有说明（例如：特定的DTD定义的专用XML元素类型，或者由特殊的应用程序规范定义的处理指令），相对URI是以实体声明所在的资源的位置为基准的。因此，URI可能相对于文档实体、包含外部DTD子集的实体，或者某些其他外部参数实体。

XML处理器应该对URI中的非ASCII字符进行处理，将UTF-8字符表示为一个或多个字节，然后利用URI的转义机制替换这些字节（即：将每个字节变为 %HH的形式，其中HH是字节值的十六进制表示）。

除了系统标识符，外部标识符也可以包含公共标识符。试图获取实体内容的XML处理器可以利用公共标识符生成可选的URI。如果处理器不能完成上述操作，它必须使用SystemLiteral中指定的URI。在进行匹配之前，公共标识符中的所有空白字符必须规格化为单一的空格字符（#x20），而且必须删除字符串首尾的空白。

下面是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 "../grafx/OpenHatch.gif"
NDATA gif >
```

4.3 解析实体

4.3.1 文本声明

外部解析实体可以以文本声明开头。

表 A-32

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

文本声明必须以文字形式提供，它不能引用解析实体。文本声明只能出现在外部解析实体的开始。

4.3.2 格式正规的解析实体

如果文档实体与 document匹配，它被认为是格式正规的。如果外部通用解析实体与 extParsedEnt匹配，它被认为是格式正规的。如果外部参数实体与 extPE匹配，它被认为是格式正规的。

表 A-33

格式正规的外部解析实体			
[78]	extParsedEnt	::=	TextDecl? Content
[79]	extPE	::=	TextDecl? extSubsetDecl

如果内部通用解析实体的置换文本与 content 产生式匹配，它被认为是格式正规的。根据定义，所有内部参数实体都是格式正规的。

格式正规的实体将使得 XML 文档的逻辑结构和物理结构能够正确嵌套；起始标记、结束标记、空元素标记、元素、注释、处理指令、字符引用和实体引用都不能跨越实体。

4.3.3 实体中的字符编码

XML 文档中的每个外部解析实体都可以采用不同的字符编码机制。所有 XML 处理器都必须接受 UTF-8 或 UTF-16 的实体。

采用 UTF-16 编码的实体必须以 ISO/IEC 10646 附录 E 和统一码附录 B 中描述的字节顺序标记（零宽度不间断空格字符，#xFEFF）开始。这是一个编码信号，而不是 XML 文档标记或字符数据的一部分。XML 处理器必须使用该字符区分采用 UTF-8 和 UTF-16 编码的文档。

虽然本规范仅要求 XML 处理器能够读取以 UTF-8 和 UTF-16 编码的实体，但是它也可以识别其他编码机制，并读取采用这些编码的实体。如果解析实体不是以 UTF-8 或 UTF-16 编码的，它必须以包含编码声明的文本声明开始。

表 A-34

编码声明			
[80]	EncodingDecl	::=	S 'encoding' Eq ("" EncName "" "" EncName "")
[81]	EncName	::=	[A-Za-z] ([A-Za-z0-9._] '-') [*] /* 编码名称只能包含拉丁字符 */

在文档实体中，编码声明是 XML 声明的一部分。EncName 是所采用的编码的名称。

在编码声明中，值“UTF-8”、“UTF-16”、“ISO-10646-UCS-2”和“ISO-10646-UCS-4”对应于各种统一码和 ISO/IEC 10646 编码和相应的变种，值“ISO-8859-1”、“ISO-8859-2”……“ISO-8859-9”对应于 ISO 8859 编码的各个部分，值“ISO-2022-JP”、“Shift_JIS”和“EUC-JP”对应于 JIS X-0208-1997 的各种编码形式。XML 处理器可以识别其他编码；本规范建议除了上面列出的编码之外，对于 IANA 注册的字符编码（字符集），应该使用它们的注册名称来引用。需要注意的是，这些注册名称是不区分大小写的，因此处理器也应该以这种方式处理它。

在缺少外部传输协议（例如，HTTP 或 MIME）所提供的信息的情况下，如果包含编码声明的实体采用的编码方式与所声明的不同，或者实体既没有以字节顺序标记开始，又没有声明使用非 UTF_8 编码，则 XML 处理器会认为出现了错误。注意，由于 ASCII 码是 UTF_8 的子集，因此从严格意义上来说，普通的 ASCII 实体并不需要编码声明。

当 TextDecl 不是出现在外部实体的开始处时，将产生一个错误。

当 XML 处理器无法处理实体的编码时，将产生致命错误。

下面是编码声明的例子：

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

4.4 XML 处理器对实体和引用的处理

下面的表格总结了字符引用、实体引用和未解析实体引用可能出现的位置，以及对于每种

情况XML处理器应该具备的处理功能。最左列中的标签描述了它们所在的上下文环境。

内容中的引用
作为引用出现在元素的起始标记和结束标记之间；对应于非终结符 content。

属性值中的引用
作为引用出现在起始标记中的属性值中或属性声明的缺省值中，对应于非终结符 AttValue。

作为属性值出现
作为名称而不是引用，它可以作为 ENTITY类型的属性值，或者作为 ENTITIES类型的属性值中以空格分隔的一个记号。

实体值中的引用
作为引用出现在参数实体声明或内部实体声明的文字实体值中；对应于非终结符 EntityValue。

DTD中的引用
作为引用出现在DTD的内部或外部子集中，但是在 EntityValue或AttValue之外。

表 A-35

	实体类型 参数	内部通用	外部解析通用	未解析	字符
内容中的引用	不识别	包含	验证有效性时包含	禁止	包含
属性值中的引用	不识别	以文字形式包含	禁止	禁止	包含
作为属性值出现	不识别	禁止	禁止	通知	不识别
实体值中的引用	以文字形式包含	忽略	忽略	禁止	包含
DTD中的引用	作为PE包含	禁止	禁止	禁止	禁止

- 4.4.1 不识别

在DTD之外，%字符没有特殊的含义；因此，DTD中的参数实体引用在 content中不会被识别为标记。类似，对于未解析实体的名称，除非出现在已声明的适当的属性值中，否则无法被识别。
- 4.4.2 包含

当XML处理器获取并处理实体的置换文本，用它取代实体引用本身，使之似乎成为文档的一部分时，我们称实体被包含。置换文本可以包括字符数据和标记（除了参数实体之外），它们必须以常规的方式被识别，但是为了转义标记分隔符而使用的实体（实体 amp、lt、gt、apos和 quot）的置换文本总是被视作数据。（字符串“ AT&T;”经过处理后成为“ AT&T;”，剩余的与号不作为实体引用分隔符。）对于字符引用，当被表示的字符取代引用被处理时，我们称字符引用被包含。
- 4.4.3 验证有效性时包含

当XML处理器发现对解析实体的引用时，为了验证文档的有效性，处理器必须包含相应的置换文本。如果实体是外部的，且处理器不准备验证 XML文档的有效性，处理器可以，但不是必须，包含实体的置换文本。如果不验证有效性的解析器没有包含置换文本，它必须通知应用

程序它能够识别但未读取实体。

本规则是基于 SGML 和 XML 实体机制提供的自动包含特征，它的主要目标是为了支持模块化设计，但是它对于某些应用程序来说不一定合适，特别是文档浏览。例如，浏览器遇到外部解析实体引用时，可以选择提供实体的可视化表示，并且仅在需要显示时才获取实体的内容。

4.4.4 禁止

以下情况是禁止的，并且会产生致命错误：

- 出现对未解析实体的引用。
- 在 DTD 的 Entity Value 或 Att Value 之外出现字符引用或通用实体引用。
- 在属性值中引用外部实体。

4.4.5 以文字形式包含

当实体引用出现在属性值中，或者参数实体引用出现文字实体值中，要用置换文本取代引用本身，并对置换文本进行处理，但是置换文本中的单引号或双引号字符总是被视作普通的数据字符，而不能用作文字的结束。例如，下面的代码是格式正规的：

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

而下面的代码不是格式正规的：

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

4.4.6 通知

当未解析实体的名称作为记号出现在声明为 ENTITY 或 ENTITIES 类型的属性的值中时，验证有效性的处理器必须将实体和相关的表示法的系统标识符和公共标识符（如果存在的话）通知给应用程序。

4.4.7 忽略

当通用实体引用出现在实体声明的 Entity Value 中时，它被忽略。

4.4.8 作为 PE 包含

与外部解析实体类似，仅当验证有效性时才包含参数实体。当 DTD 中发现参数实体引用并被包含时，要在它的置换文本的前后各增加一个空格字符（#x20）；其目的是约束参数实体的置换文本，使 DTD 中能够包含完整的语法记号。

4.5 内部实体置换文本的结构

在讨论如何处理内部实体时，有必要分清两种形式的实体值。文字实体值是真正出现在实体声明中用引号括起的字符串，与之对应的是非终结符 Entity Value。当置换文本取代了字符引用和参数实体引用之后，它将成为实体的内容。

内部实体声明中的文字实体值（Entity Value）可以包含字符引用、参数实体引用和通用实体引用。这些引用必须完全包含在文字实体值中。对实体进行包含处理时，必须用参数实体的置换文本取代相应的引用，用被引用的字符取代字符引用；但是，通用实体引用要保留。以下的声明为例：

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

实体 “ book ” 将有如下的置换文本：

```
La Peste: Albert Camus,
© 1947 Éditions Gallimard. &rights;
```

当引用 “ &book; ” 出现在文档的内容或属性值中时，通用实体引用 “ &rights; ” 才会被展开。

这些简单的规则可能会导致复杂的操作；“ D. 实体引用和字符引用的展开 ” 讨论了一个繁琐的例子。

4.6 预定义实体

实体引用和字符引用都能够用于转义左尖括号、与号及其他定界符。为此，专门定义了一组通用的实体（amp、lt、gt、apos和quot）。实际上，也可以使用数字字符引用；XML处理器一旦发现这类引用，会立即将它们展开，并将它们视作字符数据，因此当字符数据中出现数字字符引用 “ < ” 和 “ & ” 时，它们是用来转义<和&符号的。

无论声明与否，XML处理器必须识别这些实体。出于互操作性考虑，有效的 XML文档应该首先声明这些实体，然后再引用它们。如果声明实体，它们必须被声明为内部实体，其中置换文本是要转义的单字符，或者对该字符的字符引用。例如：

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

“ lt ” 和 “ amp ” 声明的置换文本都是双字符，其目的是为了满足实体置换的格式正规需求。

4.7 表示法声明

表示法可以由以下因素标识：未解析实体的格式，具有表示法属性元素的格式，或者处理指令指定的应用程序。

表示法声明定义的表示法名称可以用在实体和属性列表声明以及属性说明中，声明中定义的表示法外部标识符使得 XML处理器或客户端应用程序能够定位用于处理采用特定表示法的数据的应用程序。

表 A-36

表示法声明			
[82]	NotationDecl	::=	'<!NOTATION' S Name S (ExternalID PublicID) S? '>'
[83]	PublicID	::=	'PUBLIC' S PubidLiteral

XML处理器必须为应用程序提供所声明的以及在属性值、属性定义或实体声明中引用的表示法的名称和外部标识符。它们还可以将外部标识符解析为系统标识符、文件名或者其他信息，以便应用程序调用适当的处理程序处理表示法所描述的数据。（然而，对于XML文档声明和引用

的表示法，即使 XML 处理器或应用程序所运行的系统上没有处理该表示法的应用程序，也不会产生错误。)

4.8 文档实体

文档实体是实体树的根，也是 XML 处理器处理的起始点。本规范并未指定 XML 处理器如何定位文档实体；与其他实体不同的是，文档实体没有名称，在处理器的输入流中不会有任何标识。

5. 一致性

5.1 验证有效性和不验证有效性的处理器

合乎规范的 XML 处理器分为两类：验证有效性的和不验证有效性的。

对于文档实体内容和读到的其他解析实体中违反本规范定义的格式正规约束之处，验证有效性和不验证有效性的处理器都必须报告。

验证有效性的处理器必须报告违反 DTD 中声明所定义的约束的情况，以及违反本规范定义的有效性约束的情况。为了实现上述功能，验证有效性的 XML 处理器必须读取并处理整个 DTD 以及文档中引用的所有外部解析实体。

不验证有效性的处理器只需要检查包括整个内部 DTD 子集在内的文档实体是否符合格式正规约束。虽然它们不必检查文档的有效性，但是它们需要处理内部 DTD 子集和读到的参数实体中的所有声明，直至遇到第一个没有读取的参数实体的引用；换言之，它们必须使用这些声明中的信息规格化属性值，包含内部实体的置换文本，以及提供缺省属性值。它们不能处理没有读到的参数实体引用之后出现的实体声明或属性列表声明，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

验证有效性的 XML 处理器的行为是高度可预测的；它必须读取文档的每个部分，报告违反格式正规约束和有效性约束之处。对于不验证有效性的处理器的要求略微低一些；它不必读取整个文档，只要将注意力集中于文档实体即可。这对于 XML 的处理器用户而言可能会产生以下两个重要的影响：

- 某些违反格式正规约束的错误，特别是需要读取外部实体才能发觉的错误，可能不会被不验证有效性的处理器检测到。例如：实体声明约束、解析实体约束和无递归约束，以及“4.4 XML 处理器对实体和引用的处理”的“禁止”一节中描述的某些情况。
- 根据处理器是否读取参数和外部实体，从处理器传递给应用程序的信息有可能发生变化。例如，规格化属性值、包含内部实体的置换文本，或者提供缺省属性值，如果上述操作需要依赖于外部或参数实体中的声明，不验证有效性的处理器可能无法执行。

为了实现不同 XML 处理器之间最大限度的可靠的互操作，使用不验证有效性的处理器的应用程序不能依赖于这类处理器的非必需行为。需要更高功能的应用程序，例如要使用外部实体中声明的缺省属性或内部实体，应该使用验证有效性的 XML 处理器。

6. 表示法

本规范使用简单的扩展巴克斯-诺尔范式 (Extended Backus-Naur Form, EBNF) 定义了 XML 的正式语法。语法中的每条规则都采用以下形式定义了一个符号：

`symbol ::= expression`

如果符号是用正规表达式定义的，则它的第一个字母大写，否则第一个字母小写。文字串都是带引号的。

在规则右侧的表达式中，以下表达式用于与含一个或多个字符的字符串匹配：

`#xN`

其中N是十六进制整数，表达式与 ISO/IEC 10646 中的字符匹配，该字符的规范（UCS-4）代码值解释为无符号二进制数时恰好与表达式的值相等。`#xN` 格式中数字前面的零无关紧要；相应的代码值中前面的零是由所采用的字符编码控制的，对 XML 来说没有任何意义。

`[a-zA-Z], [#xN-#xN]`

与值在指定范围内的任意字符匹配（包含范围的边界）。

`[^a-z], [^#xN-#xN]`

与值在指定范围之外的任意字符匹配。

`[^abc], [^#xN#xN#xN]`

与任何不在给定字符集内的字符匹配。

`"string"`

与由双引号包含的文字串匹配。

`'string'`

与由单引号包含的文字串匹配。

以上符号可能通过组合构成更加复杂的形式，下面描述了可能出现的组合，其中 A 和 B 代表简单表达式：

`(expression)`

`expression` 被视作一个单元，它可以按照下面列表描述的方式进行组合。

`([-'()+,./:=?;!*#@$_$])`

与该列表匹配的字符可被认为是表达式。

`A?`

与零个或一个 A 匹配，即：A 是可选的。

`A B`

与 A 后接 B 匹配

`A | B`

与 A 或 B 匹配，但不能同时和两者匹配。

`A - B`

与任何与 A 匹配但不与 B 匹配的字符串匹配。

`A+`

与一个或多个 A 匹配。

`A*`

与零个或多个 A 匹配。

本产生式还用到了以下表示法：

`/*...*/`

注释。

[wfc : ...]

格式正规约束；它通过名称标识格式正规的文档应有的约束。

[vc : ...]

有效性约束；它通过名称标识有效的文档应有的约束。

附录A 参考文献

A.1 标准化的参考文献

(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](http://ftp.isi.edu/in-notes/iana/assignments/character-sets).

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

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

(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 (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).

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

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

Brüggemann-Klein, Anne. *Formal Models in Document Processing*. Habilitationsschrift. Faculty of Mathematics at the University of Freiburg, 1993, available at [ftp://ftp.informatik.uni-freiburg.de/documents/papers/brueggem/habil.ps](http://ftp.informatik.uni-freiburg.de/documents/papers/brueggem/habil.ps).

Brüggemann-Klein, Anne, and Derick Wood. *Deterministic Regular Languages*. Extended abstract in A. Finkel, M. Jantzen, Hrsg., STACS 1992, S. 173-184. Springer-Verlag, Berlin 1992. Lecture Notes in Computer Science 577. Full version titled *One-Unambiguous Regular Languages* in Information and Computation 140 (2): 229-253, February 1998.

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

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

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

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

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 (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 Annex*. [Geneva]: International Organization for Standardization, 1996.

附录B 字符的分类

根据统一码标准定义的特征，字符可以分为：基字符（包括拉丁字母表中没有变音符的字母）、表意字符和组合字符（包括大多数有变音符的字母）；以上三种类型构成了字母的分类。除此之外，还有数字和混合字符。

表 A-37

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-

(续)

字符

```

#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] Combining Char ::= [#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-

(续)

字符

			#x0B6F]		[#x0BE7-#x0BEF]		[#x0C66-#x0C6F]	
				[#x0CE6-#x0CEF]		[#x0D66-#x0D6F]		[#x0E50-
			#x0E59]		[#x0ED0-#x0ED9]		[#x0F20-#x0F29]	
[89]	Extender	::=	#x00B7		#x02D0		#x02D1	
			#x0E46		#x0EC6		#x3005	
				[#x309D-#x309E]		[#x30FC-#x30FE]		[#x3031-#x3035]

在此定义的字符类可以从统一码字符库中如下导出：

- 名称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名称字符必须属于 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）中的字符不允许出现在 XML 名称中。
- 不允许出现具有字体或兼容分解（即：字符库第 5 区有“兼容格式化标记”的字符——“<”标志着 5 区的开始）的字符。
- 下列字符被当成名称起始字符，而不是名称字符，因为属性文档中将它们归于字母类：
[#x02BB-#x02C1], #x0559, #x06E5, #x06E6。
- 不允许出现字符 #x20DD~#x20E0（与统一码的 5.14 节保持一致）。
- 字符 #x00B7 被归入混合字符类，因为属性文档中对它是这么标识的。
- 字符 #x0387 被作为名称字符，因为 #x00B7 是它的等价规范形式。
- 字符 ‘:’ 和 ‘_’ 可以作为名称起始字符。
- 字符 ‘-’ 和 ‘.’ 可以作为名称字符。

附录C XML和SGML（未标准化）

XML 是 SGML 的子集，因此有效的 XML 文档也应该是正确的 SGML 文档。要详细了解 XML 对文档的附加限制，参见 [Clark]。

附录D 实体引用和字符引用的展开（未标准化）

本附录包含的例子用于说明“4.4 XML 处理器对实体和引用的处理”一节中规定的实体引用和字符引用识别和展开的顺序。

如果 DTD 包含以下声明：

```
<!ENTITY example "<p>An ampersand (&#38;#38;) may be escaped  
numerically (&#38;#38;#38;) or with a general entity  
(&amp;amp;). </p>" >
```

当 XML 解析实体声明时，它会发现字符引用，并处理这些引用，然后再将以下字符串保存为实体“example”的值：

```
<p>An ampersand (&#38;) may be escaped  
numerically (&#38;#38;#38;) or with a general entity  
(&amp;amp;).</p>
```

当文档中出现引用“&example;”时，文本会被再次解析，此时 XML 处理器会发现元素“p”的起始标记和结束标记，文本中的三个引用也会被识别并展开，产生包含以下内容的元素“p”（全部是数据，没有定界符或标记）：

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

下面这个更加复杂的例子将完整地解释实体引用和字符引用的规则和效果。在下面的例子中，行号只是为了便于表述。

```

1 <?xml version='1.0'?>
2 <!DOCTYPE test [
3 <!ELEMENT test (#PCDATA) >
4 <!ENTITY % xx '&#37;zz;'>
5 <!ENTITY % zz '&#60;!ENTITY tricky "error-prone" >' >
6 %xx;
7 ]>
8 <test>This sample shows a &tricky; method.</test>

```

XML处理器将对上述代码执行以下处理：

- 在第4行，对字符 37 的引用被立即展开，参数实体 “ xx ” 保存在符号表中，它的值是 “ %zz; ”。由于不重新扫描置换文本，因此对参数实体 “ zz ” 的引用不被发现。（假如发现对参数实体 “ zz ” 的引用的话会产生错误，因为 “ zz ” 尚未声明。）
- 在第5行，字符引用 “ < ” 被立即展开，XML处理器还将保存参数实体 “ zz ”，它的置换文本为 “ <!ENTITY tricky "error-prone" > ”，一个格式正规的实体声明。
- 在第6行，发现对 “ xx ” 的引用，“ xx ” 的置换文本（即：“ %zz; ”）被解析。此时，对 “ zz ” 的引用被发现，它的置换文本（“ <!ENTITY tricky "error-prone" > ”）被解析。现在，通用实体 “ tricky ” 被声明，它的置换文本是 “ error-prone ”。
- 在第8行，发现对通用实体 “ tricky ” 的引用，XML处理器将展开它，因此 “ test ” 元素的完整内容是一个自描述（且不合语法）的字符串 This sample shows a error-prone method。

附录E 确定的内容模型（未标准化）

出于兼容性考虑，元素类型声明中的内容模型应该是确定的。

SGML要求内容模型是确定的（它称为“明确的”）；利用SGML系统生成的XML处理器会将不确定的内容模型标记为错误。

例如，内容模型 $((b, c) | (b, d))$ 是不确定的，因为对于给定的 b ，如果解析器不查看 b 之后的元素，就无法确定应该将它与模型中的哪个 b 匹配。在这种情况下，两次 b 的引用可以合并为一次引用，即：将模型改为 $(b, (c | d))$ 。现在，很明显元素 b 只能与内容模型中的一个名称匹配。解析器不必查看它后面的元素； c 或 d 都是可接受的。

如果采用更加正式的表述方法：可以使用标准的算法根据内容模型构造有限状态自动机，例如：Aho、Sethi和Ullman的[Aho/Ullman]一文3.9节描述的算法3.5。在许多这类算法中，规则表达式中的每个位置（例如，规则表达式的语法树中的每个叶节点）都构造一个跟随集合（follow set）；如果任何位置的跟随集中有多个后续的位置标有相同的元素类型名称，则认为内容模型出错，并且会被报告。

有些算法能够将大部分（但并非全部）不确定的内容模型自动规约为等价的确定模型；参见Brüggemann-Klein 1991 [Br ü ggemann-Klein]。

附录F 字符编码的自动检测（未标准化）

XML编码声明作为实体的内部标签，说明实体采用的字符编码。然而，在XML处理器能够读取内部标签之前，显然需要知道所使用的字符编码——这也正是内部标签要说明的。通常情况下，这简直是可望不可及。但是在XML中，这并非完全不可能，因为XML通过两种方式限制了通常的情况：假设每一种实现仅支持有效的字符编码集，并且XML编码声明的位置和内容是受限的，它使自动检测每个实体所用的字符编码成为可能。另外，在许多情况下，除了XML数据流本身，还有许多其他可用的信息源。根据XML实体提交给处理器时是否提供附加（外部）信息，可以分为两种情况。我们首先来考虑第一种情况：无附加信息。

由于不采用UTF-8或UTF-6格式的所有XML实体必须以XML编码声明开头，它的前几个字符必然是‘<?xml’，只要提供两至四个字节的输入，任何合乎规范的XML处理器都能够检测出适用于下列哪种情况。在读取该字节序列时，可以使用提前掌握的编码知识，例如：在UCS—4中，‘<’表示为“#x0000003C”，‘?’表示为“#x0000003F”，UTF-16数据流需要的字节顺序标记表示为“#xFEFF”。

- 00 00 00 3C：UCS-4，big-endian编码的计算机（1234顺序）
- 3C 00 00 00：UCS-4，little-endian编码的计算机（4321顺序）
- 00 00 3C 00：UCS-4，异常的字节顺序（2143）
- 00 3C 00 00：UCS-4，异常的字节顺序（3412）
- FE FF：UTF-16，big-endian
- FF FE：UTF-16，little-endian
- 00 3C 00 3F：UTF-16，big-endian，无字节顺序标记（因此从严格意义上讲是错误的）
- 3C 00 3F 00：UTF-16，little-endian，无字节顺序标记（因此从严格意义上讲是错误的）
- 3C 3F 78 6D：UTF-8，ISO 646，ASCII，ISO 8859的某些部分，Shift-JIS，EUC，或者其他7位、8位或混合长度的编码，它们能够保证ASCII字符具有正常的位置、长度和值；XML处理器必须读取真正的编码声明以检测到底采用了哪种编码，但是由于这些编码中的ASCII字符采用同样的位模式，因此编码声明本身是可读的
- 4C 6F A7 94：EBCDIC（对于某些变种，XML处理器必须读取完整的编码声明以确定到底使用了哪页编码）
- 其他：无编码声明的UTF-8，或者数据流已损坏，不完整或被包含在某种外层数据中。

这种程度的自动检测机制足以读取XML编码声明，以及解析字符编码标识符，以便区分出特定编码族中的某个编码（例如，从8859中识别出UTF-8，识别出8859编码的各个部分，或者辨别出所使用的EBCDIC代码页，等等）。

由于编码声明的内容仅限于ASCII字符，因此只要处理器检测出所用的编码族，就能够可靠地读取整个编码声明。实际上，所有广泛使用的字符编码必然归入以上某种类型，XML编码声明能够相当可靠地标识字符编码，即使当操作系统或传输协议级的外部信息源不够可靠时，也能够根据编码声明判断出所用的字符编码。

一旦处理器检测出所用的字符编码，它就能够执行适当的操作，针对每种情况调用不同的输入例程，或者调用适当的转换函数处理每个输入字符。

与任何自标识的系统一样，如果软件改变了实体的字符集或编码，而没有更新编码声明，

XML处理器无法根据编码声明判断所用的编码。字符编码的实现者应该格外谨慎，他应该保证用于标识实体的内部和外部信息的准确性。

第二种情况是XML实体有附加的编码信息，如同在某些文件系统和网络协议中一样。如果有多种信息源，用于传递XML的高层协议应该指定这些信息源的相对优先级，以及处理冲突的方法。例如，定义text/xml和application/xml MIME类型的RFC文档应该定义确定内部标签和外部头中的MIME类型标签相对优先级的规则。然而，出于互操作性考虑，本规范推荐以下规则。

- 如果XML实体包含在文件中，字节顺序标记和编码声明 PI（如果存在的话）用于确定字符编码。所有其他启发式方法和信息源仅仅用作错误恢复。
- 如果XML实体是随text/xml类型的MIME一起传递的，则MIME类型的charset参数用于确定字符编码方法；所有其他启发式方法和信息源仅仅用作错误恢复。
- 如果XML实体是随application/xml类型的MIME一起传递的，则字节顺序标记和编码声明 PI（如果存在的话）用于确定字符编码。所有其他启发式方法和信息源仅仅用作错误恢复。

仅当不存在协议级的文档时，才应用以上规则；特别是在定义了 MIME类型text/xml和application/xml的情况下，相关RFC中的建议将取代以上规则。

附录G W3C XML工作组（非正式）

本规范由W3C XML工作组（Working Group，WG）完成并批准发表。工作组赞成本规范并不代表所有工作组成员都对该规范满意。XML工作组现在和过去的成员有：

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
Copyright © 1998 W3C (MIT, INRIA, Keio), All Rights Reserved. W3C liability, trademark, document use and software licensing rules apply.