ICS 35.240.30
L 74

中华人民共和国国家标准化指导性技术文件

GB/Z 21025—2007

XML 使用指南

XML user's guide

2007-06-29 发布

中华人民共和国国家质量监督检验检疫总局
中国国家标准化管理委员会 发布

中 华 人 民 共 和 国
国家标准化指导性技术文件
XML 使用指南
GB/Z 21025—2007
*
中 国 标 准 出 版 社 出 版 发 行
北京复兴门外三里河北街 16 号
邮政编码:100045
网址 www. spc. net. cn
电话:68523946 68517548
中国标准出版社秦皇岛印刷厂印刷
各地新华书店经销
*
开本 880×1230 1/16 印张 3.25 字数 92 千字
2007 年 11 月第一版 2007 年 11 月第一次印刷
*
书号:155066·1-30020 定价 34.00 元

目　次

前　言

本指导性技术文件仅供参考。有关对本指导性技术文件的建议和意见，请向国务院标准化行政主管部门反映。

本指导性技术文件的附录A和附录B为资料性附录。

本指导性技术文件由中华人民共和国信息产业部提出。

本指导性技术文件由中国电子技术标准化研究所归口。

本指导性技术文件起草单位：中国电子技术标准化研究所、北京信息工程学院、万达信息技术公司、北京航天航空大学、方正电子技术公司。

本指导性技术文件主要起草人：李宁、顾晓毅、林学练、王国印、吴志刚、赵菁华。

引　言

本指导性技术文件规定了使用可扩展置标语言(XML)应该遵循的原则和注意事项,适用于XML应用的开发者、管理者、使用者和其他关心XML使用的人员。

在GB/T 18793—2002《信息技术　可扩展置标语言(XML)1.0》中,描述了可扩展置标语言(eXtensible Markup Language,XML)。它是标准通用置标语言(Standard Generic Markup Language,SGML)的一个子集,其目的是使SGML文档在web应用中可以像HTML文档一样进行发送、接收和处理。为此,XML的设计力求易于实现,并能与SGML和HTML很好地互操作。它针对web应用,简化了一些SGML的不常用的内容。

GB/T 18793—2002所定义的XML是一组用来构造语义置标的规则集合,通过这些置标文档的各个部分可按预先定义的语义结构组织起来并进行结构化验证。GB/T 18793—2002的重点在于对以DTD为核心的XML本身的语法进行描述,并不包括Schema的内容,具体的使用方法在那里也少有涉及,这些将在本指导性技术文件中做出解释,以帮助XML的推广使用。本指导性技术文件还给出了部分使用XML的例子。本指导性技术文件为设计一致性的、合理的DTD和Schema提供了一个指导性框架,将有助于更好地理解XML,重用XML组件,并达到良好的互操作性。

本指导性技术文件的重点在于如何采用元语言标准和基础标准来定义应用标准,核心是行业应用词汇表的设计。在本指导性技术文件中,部分内容仅适用于Schema或仅适用于DTD,将特别标出。未被特殊标出的部分两者都适合。另外为了查阅方便,本指导性技术文件的大部分条目标题直接采用了原则性陈述的形式。

XML 使用指南

1 范围

本指导性技术文件给出了指导 XML 文档的编写的原则，包括标准的符合性原则，国际化和本地化原则，组件命名原则，命名空间使用原则，词汇表编写原则，类型、元素与属性的使用原则，版本与注释的使用原则，实例的编写原则，XML 解析器及选择，应用开发过程和注册规程等内容。

本指导性技术文件适用于 XML 的各类开发人员和使用人员。

2 规范性引用文件

下列文件中的条款通过本指导性技术文件的引用而成为本指导性技术文件的条款。凡是注日期的引用文件，其随后所有的修改单(不包括勘误的内容)或修订版均不适用于本指导性技术文件，然而，鼓励根据本指导性技术文件达成协议的各方研究是否可使用这些文件的最新版本。凡是不注日期的引用文件，其最新版本适用于本指导性技术文件。

GB/T 1988—1998 信息技术 信息交换用七位编码字符集(eqv ISO 646:1991)

GB 2312—1980 信息交换用汉字编码字符集 基本集

GB/T 13000.1 信息技术 通用多八位编码字符集(UCS) 第一部分:体系结构与基本多文种平面(GB/T 13000.1—1993,idt ISO/IEC 10646-1:1993)

GB/T 14814—1993 信息技术 文本和办公系统 标准通用置标语言(SGML)(idt ISO 8879:1986)

GB 18030—2000 信息技术 信息交换用汉字编码字符集 基本集的扩充

GB/T 18391(所有部分) 信息技术 数据元的规范和标准化

GB/T 18793—2002 信息技术 可扩展置标语言(XML)1.0(neq W3C RFC-xml-19980210:1998)

3 术语和定义

GB/T 18793—2002 中确定的以及下列术语和定义适用于本指导性技术文件。

3.1

文档对象模型 Document Object Model;DOM

W3C 制定的 XML 应用程序接口。它将 XML 文档表示成一个树形的结构。DOM 规定了一系列编程指令，允许应用程序多次访问并操作文档树的组件。

3.2

元数据 metadata

定义和描述其他数据的数据。

3.3

资源目录描述语言 Resource Directory Description Language;RDDL

一种符合 XHTML 格式的资源目录和描述，用于为用户提供目标资源的相关信息。

3.4

XHTML

W3C 为 HTML 制定的 XML 词汇表。

3.5

XML 的简单 API　Sample API for XML;SAX

为序列化存取 XML 文档信息所制定的一种接口。SAX 处理器在处理 XML 文档的时候，随着遇见开、关标记和字符数据等内容，不断触发事件，调用应用程序(事件处理程序)处理，能够达到比较快的速度和效率。

3.6

式样单　stylesheet

一套指令集合，主要用于规定 XML 文档显现格式，也可以将一个 XML 文档转换成另一个文档。

3.7

XML 组件　XML component

XML 元素、元素属性和 XML 词汇表的统称。

3.8

XML　schema/Schema

一种用于限定文档结构(如元素的顺序、出现次数、属性等)的机制，用于描述一类实例文档的结构。解析器可以根据 schema 来验证文档。本指导性技术文件中，用小写字母开头的 schema 统称这一概念，其中也包括 DTD。用大写字母开头的 Schema 特指 W3C 制定的 Schema 标准(REC-xmlschema-0-20010502 Part 0～2)。

3.9

XML 命名空间　XML namespace

为了解决命名冲突，为元素和属性命名引入的逻辑空间，是在 XML 文档中通过 URI 引用声明的，并采用限定性前缀将元素和属性与命名空间联系起来。

3.10

XML 词汇表　XML vocabulary

在特定领域给特定用户群使用、有确定的功能的数据元集合以及数据模型，是表示一类文件的结构的 schema 的统称。

3.11

XML 作品　XML artifacts

具有独立保存价值的各种 XML 数据，包括 schema、式样单以及 XML 实例文档等。

3.12

可扩展式样语言　eXtensible Stylesheet Language;XSL

由 W3C 组织制定的用于定义 XML 文档转换和显现的系列标准，包括：XSLT、XPath、XSL-FO 三部分。

3.13

可扩展式样语言转换　eXtensible Stylesheet Language Transformations;XSLT

由 W3C 组织制定的用于转换 XML 文档的语言。

3.14

统一建模语言　Unified Modeling Language;UML

为创建商业和技术模型定义的一种语言和图形表示法，它定义了多种模型种类，涵盖了从功能需求定义、事务处理活动工作流模型到逻辑和物理层次等软件开发的各个方面。

3.15

上驼峰形式大小写　Upper Camel Case;UCC

把单词拼接在一起的一种方式，不使用含下划线“_”和句点“.”等的连字符，每个单词的首字母大写，其余字母均小写。在有大写缩写字母或数字的时候，后面的单词首字母小写。

例如：NameUsedInXMLschema。

3.16

下驼峰形式大小写 Lower Camel Case；LCC

LCC 与 UCC 的唯一区别是整个名称的首字母用小写，例如：attributeValue。

3.17

统一建模方法 Unified Modeling Methodology；UMM

UN/CEFACT 为事务处理建模，支持下一代电子数据交换（Electronic Data Interchange，EDI）开发而推出的一套建议的方法。它基于 Rational 统一过程理论，使用 UML 作为建模语言。

3.18

Unicode

由 Unicode 协会（Unicode consotium）制定的通用字符。其主要目的是为纯文本内容提供一套无歧义的编码，以方便全球各种语言文字的转换。

3.19

统一资源定位符/统一资源指示符/统一资源名称 URL（Uniform Resource Locators）/URI（Uniform Resource Indicators）/URN（Uniform Resource Names）

网络环境下三种不同的，但又相关的引用资源的统一的方法。

4 标准的符合性原则

所有 XML 文档的编制和所有的处理工具或处理器，包括解析器、文档生成器、验证工具，以及所有使用 XML 的软件应符合 GB/T 18793—2002，同时也应该考虑符合国际权威标准化组织订立的其他正式标准。对于标准未涉及到的，而应用中需要的对 XML 的扩充，原则上仅限于在组织内部使用，不宜公开传播，除非经过正式的审查和注册。

5 国际化和本地化原则

5.1 XML 文档编码

GB/T 18793—2002 规定，XML 文档可以使用以下编码字符集：

——GB 18030—2000；

——GB 13000.1；

——GB 2312—1980；

——其他 XML 处理器支持的编码字符集。

在 GB/T 18793—2002 中，缺省字符集规定为 GB 13000.1，亦称为通用字符集（Universal Character Set，UCS）。

注：Unicode 是由 Unicode 协会（Unicode Consotium）制定的通用字符集。其主要目的是为纯文本内容提供一套无歧义的编码，以方便全球各种语言文字的转换。在 W3C 的 XML 1.0 中，大量使用了 Unicode。其 2004 年发布的 XML 1.1 中作了更新，使 XML 不再依赖于 Unicode 的特定版本。由于 GB/T 13000.1—1993 与 Unicode（2.0 版本以上）是完全兼容的，本指南中除非特殊需要不涉及 Unicode。

在 XML 文档交换中经常用到 UCS-4、UCS-2 以及 UTF-8、UTF-16 等编码形式，简要介绍如下。

UCS 的双八位的 BMP 形式（UCS-2）规定每个字符用两个字节编码，这种形式仅适用于基本多语种平面。如“一”的双八位形式为 4E00。

UCS 的肆八位的正则形式（UCS-4）规定每个字符用四个字节编码，例如：汉字“一”的正则形式为 0000 4E00。

在 UCS 中，编码点在 0～65535 的字符归属第 0 平面，也称基本多语种平面（Basic Multilingual Plane，BMP）。这个平面中包含大部分全世界正在使用的公用字符，包括来自罗马字母、西里尔字母、阿

拉伯语、希腊语、希伯来语、常用汉字和其他语言的文字。编码点在65536～131071的字符归属第1平面。这个平面包括音乐符号、数学符号和一些已经不再使用的语言(如古意大利语)的文字。编码点在131072～196607的字符归属第2平面,收录了很多不常用汉字。第14平面包含了一些语言标记,因为XML有xml:lang属性可用而完全不需理会这些标记。其他的平面至今都没有很好地定义。

在实际使用中更多采用的是UTF-8和UTF-16。制定UTF-8的目的是为了与原8比特系统向下兼容;制定UTF-16的目的是为了向上发展和扩充。

UTF-8(Unicode Transformation Format, 8-bit encoding form)是一种变长编码。编码点为0～127的每一个字符(GB/T 1988—1998字符)占一个字节,编码点为128～4095的每一个字符占据两个字节,第0平面的其他字符每一个占据3个字节,从第1至第15平面的每一个字符占据4个字节。UTF-8有很多优点,列举如下:

——它是GB/T 1988—1998的超集,因此对于纯英文的文本,一个UTF-8文件与GB/T 1988—1998文件完全一样,非常利于兼容,因此XML把UTF-8选作缺省的编码形式。

——所有的GB/T 1988—1998字符都不会成为其他字符编码的一部分,因此非常容易分辨GB/T 1988—1998字符。

——UTF-8与字节顺序无关。在计算机系统中,大数值类型(如整型)使用多个字节表示,不同体系结构采用的字节排列顺序不同。其中,部分采用由高字节到低字节的排列顺序,称为big-endian;其他则采用由低字节到高字节的排列顺序,称little-endian。对于大多数big-endian的UNIX系统和little-endian的Windows系统,对同一个文档UTF-8可以做到每个字符一一对应,因此,没有必要在XML文档开头放置字节顺序标记。

——从单一字节就可以判断字符边界。只观察单一字节,程序就可以判断该字节是下列哪种情形之一:单字节字符、双字节字符的第一个字节,三字节字符的第一、二、三字节。

——对于常见字符组成的文档,UTF-8占用空间最节省。

UTF-16(Unicode Transformation Format, 16-bit encoding form)也是一种变长编码。在UTF-16中,编码点为0～65535的字符使用单一的16位编码单元表示;而编码点为65536～1114111的字符使用一对16位编码单元表示(RC-element或surrogate pair)。UTF-16最大的好处是优化了基本多语种平面的字符表示,每个字符只需要2个字节,可作为定长编码来有效使用。对于大量使用中、日、韩文字的文本,其占用空间比UTF-8约节省1/3。然而比起UTF-8,UTF-16丧失了很多优点,最主要的是UTF-16是字节顺序相关的,为解决字节顺序问题,要在XML文档开头加一个字节顺序标志(＃xFEFF)。如果程序读出的是FE和FF,则可以断定文档的编码是big-endian UTF-16;如果读出的是FF和FE,则文档的编码是little-endian UTF-16。由于＃xFEFF不是一个合法的GB/T 13000字符,所以不会与其他内容混淆。UTF-16的另一个缺点是检测字符边界比较麻烦。

除了UTF-8和UTF-16,GB/T 13000.1还有UTF-32编码形式,与UCS-4一致。然而常用的只有UTF-8和UTF-16。一般原则是,如果文档不含大量的中、日、韩文字,XML应该使用UTF-8作为缺省的编码,否则应该使用UTF-16。如果难以判断,仍可采用UTF-8。

但是,除非特别必要,XML置标标记(包括元素名和属性名)应该尽量采用GB/T 1988—1998字符集,以适应当前很多工具对GB 13000.1和其他编码字符集支持不够完善的现况。尽量不要使用GB 13000.1以外的字符集,如果确实必要应该考虑采用编码转换,将其他字符集编码映射到GB 13000.1再处理。

注:在使用GB 13000的时候,应该遵照XML 1.1标准的建议先将XML文本规格化。因为在GB 13000中,一些文本成分即可使用静态的预先组合好的形式,也可使用动态组合的形式。例如"é"可以表达为单个字符"＃xE9",也可以表达为两个字符连用,即"＃x65"和"＃x301"。为了进行字符比较,需要进行规格化,即使用一种规范化的、单一的GB/T 13000文本形式来表示这些成分。Unicode定义了四种规范化形式:Normalization Form D (NFD),Normalization Form KD (NFKD),Normalization Form C (NFC)和Normalization Form KC(NFKC)。

其中 NFD 和 NFKD 将可能的字符进行分解，而 NFC 和 NFKC 将可能的字符进行组合。XML 1.1 规定规范化的文本应该采用 NFC 的形式。

5.2 URI 字符集使用

统一资源指示器(URI)用于定位系统中的某项资源。统一资源定位器 (URL)(较 URI 更为人所熟知)是 URI 的子集。XML 通常使用 URI 指定链接中的资源，定义命名空间等。以往的 URI 基于 GB/T 1988—1998 的基本集，无法在 URI 中直接表示扩展字符，但可以通过转义机制来进行。

对于 URI 中的保留字符、空白字符以及其他不满足条件或非安全的字符(包括 GB/T 1988—1998 中编码点 127 之后的字符、空白符、控制符，如“{”、“}”、“|”、“、”、“～”、“[”、“]”等)都应该编码成“%hh”形式，其中，hh 是该字符在字符集中的 16 进制的编号。

以下是一个尚未编码的 URL 例子：

```
http://www.company.com/中国
```

转换后成为：

```
http://www.company.com/%E4%B8%AD%E5%9B%BD
```

W3C 和 IETF 正在制定基于 GB/T 13000 的国际化的 URI 标准。在正式标准发布之前，仍然需要转义非安全的字符。

5.3 谨慎使用多语种文档

多语种文档有两类：一类是在内容级别上使用多种语言，即元素内容使用不同的语言，另一类是在结构级别上使用多种语言，即用不同语言表示一个元素。多语言文档不易处理，特别是后者，不易保持几种语言版本的一致性，不易为检索等应用建立等价元素关系，因而需要尽力避免使用多语言文档。如果确实需要，应该尽量采用工具进行转换，并通过标识符建立元素之间的关联。

5.4 XML 中多语种的使用

除非在其他标准、规范或既定应用中明确规定了须使用其他语种，在 XML 的以下成分中可以使用中文：

——元素名和元素内容；

——属性名和属性值；

——枚举值；

——唯一标识符；

——记法(Notation)；

——文本字符串。

面向公共使用的词汇表应该首先准备一个语种的主版本。为了利于使用其他语种的地区和国家使用，可以将该词汇表主版本翻译成其他语种的副版本，例如：由英文主版本翻译成中文或汉语拼音版本。建议通过一套 XSL 式样单(stylesheet)或程序进行词汇表之间的转换。经过转换的词汇表应该通过注释指明原始词汇表的出处。另外，在词汇表的不同版本中，应该使用标识符(建议使用“locID”)来标识等价的元素，并使用特定的属性(建议使用“attrList”)来标识属性顺序，以利在一个词汇表的不同语种实例之间能仍然保持严格的元素和属性的语义关联，并方便 XML 检索等处理机制识别。这方面的一个实例见 14.1。

[适用于 Schema] 某些时候，也可采用元素替换(substitutionGroup)的方式，在一套 Schema 中指定多个语种的等价元素或属性名称。例如：

```
<xsd:element name="Subway" type="xsd:string"/>
<xsd:element name="Metro" substitutionGroup="Subway" type="xsd:string"/>
<xsd:element name="地铁" substitutionGroup="Subway" type="xsd:string"/>
```

这种方法要求 Schema 的编码方法可以编码各个所需的语种，而且 XML 处理器可以处理这样的编码。另外，substitutionGroup 中的替换元素必须是全局的。它的优点是能够保证多个语种文档的一致性。

通过 Schema 的命名空间可以建立多个版本的联系，例如：

```
〈xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    targetNamespace="http://www.books.org/zh-cn-py"
                    xmlns=http://www.books.org/zh-cn-py〉
```

在 DTD 中，可以通过前导说明部分的 DOCTYPE 声明来建立联系：

```
〈! DOCTYPE Book PUBLIC "universal/Publishing/Book/ZH-CN-PY"〉
```

在进行词汇表语种转换的过程中，要注意妥善处理好命名空间的对应关系。

XML 注册机构最好能提供查询机制以方便检索同一词汇表的不同编码版本。

5.5 使用 xml:lang 整合语种声明

明确指示文档内容（如时间、日期、货币符号和数字）所使用的语言，可以有助于采用适合的显现式样，或正确地解释与语种无关的数据。所有与语种相关的元素或属性内容，均应使用该属性进行声明。

对于 schema 中使用多语种元素的情况，应该为所有与方言相关的元素指定 xml:lang 属性。例如：

```
〈ProductName id="id1"〉
  〈var xml:lang="en"〉disc〈/var〉
  〈var xml:lang="zh-cn"〉光盘〈/var〉
  〈var xml:lang="zh-tw"〉光碟〈/var〉
〈/ProductName〉
```

xml:lang 所使用的语种代码，与中文相关的有如下几个：

——zh　　中文（简体）；

——zh-cn　中文-中国（简体）；

——zh-hk　中文-香港（繁体）；

——zh-mo 中文-澳门（简体）；

——zh-sg　中文-新加坡（简体）；

——zh-tw　中文-台湾（繁体）。

如果上述代码不能满足需要，可以经权威机构注册后加以扩充，如用“zh-cn-py” 表示汉语拼音。

5.6 在内容中使用本地化属性

为了便于本地化，需要规定一些用于本地化的属性，例如：

——translate：指明哪些内容需要翻译；

——localize：指明是否需要做本地化处理；

——locID：为本地化应用建立元素的语义关联；

——locNote：通过它给翻译人员提供为理解原文应查看的重要信息。

例如，元素的内容可能既有可翻译的部分，也有不可翻译的部分，采用本地化属性可指明如下：

```
〈p〉
  〈span translate="yes"〉The XML specification is maintained by the 〈/span〉
  〈span translate="no"〉World Wide Web〈/span〉
〈/p〉
```

指示“World Wide Web”不被翻译。

另外，在非 XML 文件中，也可使用类似的本地化机制，例如，在 JavaScript 中插入：

```
/ * loc:span localize='no' * / … / * /loc:span * /
```

5.7 属性值中避免使用可翻译的文本

属性值的翻译会带来一些问题。首先，指定属性的本地化信息十分困难；其次，属性没有显式的 ID 供等价的属性建立关联；再者，属性值中与语言相关的空白不易处理；另外，元素 xml:lang 的作用域不覆盖属性值。

5.8 避免条件翻译

在设计 schema 时，应避免使用这样的元素：其内容是否需要翻译取决于某个属性值、父元素或同级元素的内容以及其他条件。

6 组件命名原则

6.1 拼写规则

所有的名称应该使用 UCC 规则（某些情况下，属性名称可采用 LCC），而枚举值可以使用 LCC 规则。当然枚举值中的该有的大小写规则可不受此限制，例如“ChinaGNP”。

6.2 缩写规则

可读性比起标记的长度更为重要，在元素、属性和类型名称中应尽量少用缩略语和首字母缩略语，除非这些缩略语已经为世人公认，或者在注册库中有明确的定义。例如：不要使用 DptID 作为元素或属性的名称，而应该使用 DepartmentID。如果使用缩略语，应该通过注释给出其非缩写的形式。当然也要避免使用太长的名称。

6.3 通用化名称使用的限制

在为全局（Global）元素或属性、枚举值命名时，除非它们确实是较高抽象级别上的概念，不要使用过于通用化的名称（例如：“Name”、“Address”等），应该把名称加上特定的上下文，使其更符合当前的含义（如“ReceiverName”、“EmailAddress”）。

6.4 用 URI 作为名称型数值的定义

用 URI 来唯一标识某个名称，可给它赋予更准确的内涵（这种内涵可以在专门的文档中详细描述）。例如：用“http://www.books.org/owner”来为一个“Owner”角色命名。

6.5 避免发明新名称

在为元素、属性、枚举值命名时，必须使用标准或其他规范中定义的标准名称，或者是其他外部标准文档中定义的词汇，尽量少创造新的名称。在 schema 中使用到的外部词汇也必须在注册库中注册。

如果必须创造新的名称（元素、属性、类型、枚举值等的名称），这些名称必须在行业领域专家的指导下定义，应该经过各方用户同意，并在注册库中注册。

对于 DTD 中的实体引用的命名，也应避免随意使用新的名称，应该首先考虑使用 GB/T 14814 等标准给出的实体名称。

6.6 尽量重用已有组件

schema 的制定者应尽量使用在注册库中已经定义好的标准组件（例如：元素、属性、类型等），避免制定新的雷同标准。因此，在制定 schema 之前，应该先检索注册库，看是否有可用组件存在。如果使用了注册库之外的其他标准化组织的 XML schema，也应该把它们注册到注册库中。

可使用〈include〉或〈import〉来重用已有的组件。也可使用 schema 的继承机制通过扩展已有的元素类型来定义新的元素，以提高设计效率。

为了方便重用，每个单一的 schema 不宜太大。

在 schema 设计中，如果多次用到相同的类型、元素结构、内容模型片断、属性组合等，应该将它们定义成命名的全局类型、元素、模型组和属性组，并使用通用的名称来命名。类型的定义应考虑如何方便后续的重用。

6.7 XML 组件的命名

XML 组件的命名应该遵循 GB/T 18391 的规定：XML 元素和属性名称以及 XML 数据类型名称都应遵从 GB/T 18391 相应的命名方法。另外，GB/T 18391 的名称需要经过适当转换才能用于 XML 组件命名，即除了要去除空格和分割点之外，还需将名称转换成 UCC 的形式。

简单数据类型的名称应以“……类型”或“…Type”结尾，复杂数据类型的名称应以“……结构”或“…Structure”结尾，以区别简单类型、复杂类型和元素。

6.8 元素和属性的名称应不依赖显现

元素的名称应该反映逻辑语义，而不是其显现形式。合适的名称将有助于正确理解文档并对其作适当处理。例如：

```
〈Paragraph〉〈Light〉This text is 〈/Light〉〈Bold〉important〈/Bold〉〈/Paragraph〉
```

这种表示方法就不如：

```
〈Paragraph〉〈Regular〉This text is 〈/Regular〉〈Emphasis〉important 〈Emphasis〉〈/Paragraph〉
```

因为翻译成本地化语言(比如中文)后，重点强调的内容可能不是用粗体来表示，而是加着重号来表示。显然第一种表示方法就不适合这么做。

7 命名空间使用原则

7.1 所有的 XML 文档都应使用命名空间

命名空间是在一个 XML 文档中使用多词汇表的标准的方法。即使没有命名冲突，也能使程序快速无误地确定哪个元素属于哪个应用。新开发的 XML 文档都应该为元素定义命名空间，即使词汇表的作者将来并不打算将这个词汇表用于其他应用，但不能保证词汇表的用户不会这么做。

7.2 使用统一的命名空间命名规则

应制定统一的、结构清晰的命名空间命名规则。命名空间应该用绝对路径表示的 URI 形式来指定，例如：“http://www.w3.org/TR/2001/REC-xmlSchema-2-20010502”，不建议命名空间 URI 使用收尾的斜线(“/”)。不建议使用 URN 作为命名空间的名称。应规定统一的 Schema 存放位置(schemaLocation)，并将其注册到注册库中。

命名空间字符串的比较应该按大小写相关的方式逐个字符比较两个 URI 字符串(不处理 16 进制转义序列)。如果完全相同，才认为两个命名空间是相同的。

尽管命名空间 URI 不一定要对应一个真实的资源(网页、主机、域名)，但如有可能，应该在那里放置一个 RDDL 文档，使命名空间的用户能获得关于该命名空间的必要信息。

7.3 使用一致的命名空间前缀

对来自外部的命名空间，使用规定的前缀，例如：

——xml　　(在 XML 标准中定义)；

——xmlns　　(在 XML Namespaces 标准中定义)；

——xsd/xs　　http://www.w3.org/2001/XMLSchema ；

——xsi　　http://www.w3.org/2001/XMLSchema-instance；

——xlink/xl　　http://www.w3.org/1999/xlink；

——xsl　　http://www.w3.org/1999/transform。

7.4 ［适用于 Schema］使用缺省的命名空间

在 Schema 文档中，如果存在目标(target)命名空间，应该将缺省(default)命名空间指定为目标命名空间。不要使用 W3C Schema 的命名空间作为缺省命名空间。

7.5 ［适用于 Schema］不要使用不带命名空间属性的导入(import)

所有导入(import)的内容必须带有相应的命名空间，以方便调试和阅读。尽可能避免使用空的、无属性的〈import〉，这会导致 Schema 难以调试和更新。

7.6 ［适用于 Schema］隐藏或暴露命名空间的准则

通过将 elementFormDefault 设置为 unqualified 或 qualified，可以实现对命名空间的隐藏或暴露。但由于全局元素是不能隐藏命名空间的，因此，如果想通过 elementFormDefault 作为暴露或隐藏命名空间的“开关”，则应尽量少用全局元素。

当实例文档的简洁性和可读性是第一位的时候，最好隐藏命名空间。因为元素附加的命名空间信息会给不熟悉 XML 的用户带来困惑。另外，当需要 Schema 有一定的灵活性，即要求对 Schema 的某些改动不影响实例文档的情况下，也可以通过隐藏命名空间来做到。例如，将原先外部命名空间定义的元素改为在 schema 内部定义(inline 方式)，这时，采用隐藏命名空间的方式就不需要改动实例文档。

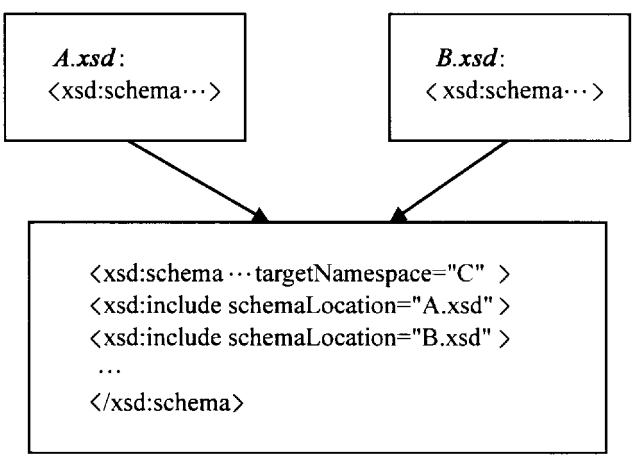
当实例文档需侧重考虑元素的拥有/承袭关系的时候(如版权的需要)，最好使用暴露的命名空间。另外，当元素同名不同义的时候，需要暴露命名空间以消除歧义。再有，当元素的处理需要命名空间的知识作辅助的时候，也应暴露命名空间。

在应用中，通常将 elementFormDefault 设置为 qualified，将 attributeFormDefault 设置为 unqualified。根据命名空间规范，没有命名空间限定的属性并不是缺省地属于其元素的命名空间，所以需要时应该显式地(explicitly)在属性前使用命名空间。

7.7 正确设计命名空间

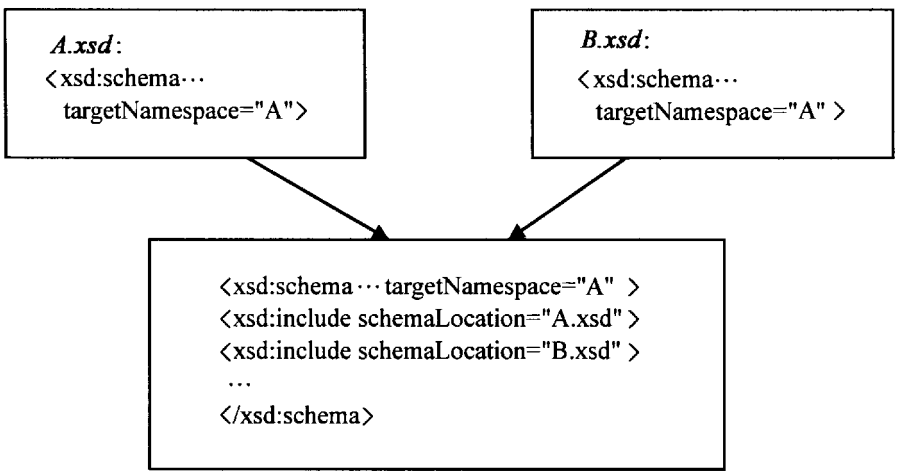
命名空间的设计大致有三种方法：

方法 1：多个 schema 中有一个“主”schema，其他的为辅助 schema。只指定主 schema 的 targetNamespace。例如：



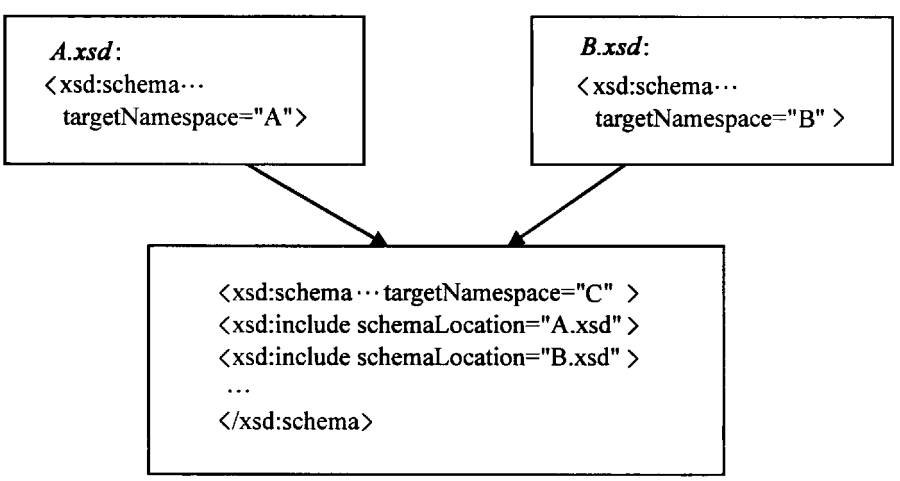
如果某些 schema 仅包含类型定义而不含元素定义，则那些 schema 最好定义成上述不带 targetNamespace 的辅助 schema。

方法 2：所有 schema 用一个 targetNamespace。如：



这种设计适合下列情况：所有的 schema 在逻辑上是紧密相关的；实例文档中没有必要显式区分元素和属性的来源及其关联，亦即不需要对元素或属性划分类别。

方法 3：所有的 schema 都有不同的 targetNamespace。如：



这种设计适合下列情况：当有多个元素具有相同的名称，为防止名称冲突；实例文档中有必要显式区分元素和属性的来源及其关联。

7.8 [适用于 Schema] 未定因素尽量靠后处理

不要把所设计的 Schema 硬性作为“targetNamespace”，一开始可考虑设计没有“targetNamespace”的 Schema，而让 Schema 的用户来决定对于具体应用应该用什么“targetNamespace”，应将 Schema 和 targetNamespace 的绑定尽量延后。

如果要导入(import)带有命名空间的 Schema，先不要硬性指定它。例如，假定有一个声明的元素使用了另一个命名空间的类型，即：〈xsd：element name＝“MyElement” type＝“n：MyElementType”/〉，其中 MyElementType 来自于需要〈import〉的另一个命名空间。而一旦指定了〈import〉的 schemaLocation 属性，意味着定义 MyElementType 的命名空间被严格确定了。但是，实际上没必要这样做。如果在 Schema 中不指定 schemaLocation，而让实例文档的作者来指定 MyElementType 所在的 Schema，会提高 Schema 的动态适应能力。因此，可以把类型引用和类型定义之间的绑定留到最后时

刻——在验证 Schema 之前来确定。

8 词汇表编写原则

8.1 DTD 到 Schema 的过渡

文件类型定义(DTD)用来描述一个特定目的的置标语言(词汇表),亦即定义一类文档的整体结构以及语法。DTD 源于 SGML,是 XML1.0 标准的重要组成部分。DTD 有很强的形式化特点,结构精简,但由于 DTD 采用非 XML 的语法规则,数据类型不丰富,扩展性较差等等,目前,越来越多的词汇表采用 Schema 来描述(包括 W3C 以外的 Schema)。Schema 本身是 XML 的一种应用,它使用 XML 语言来定义 DTD 的内容,充分发挥出 XML 自描述性的优势。与 DTD 相比,W3C Schema 具有一致性、扩展性、易用性、规范性和可互换性等优点。因此,文档词汇表的编写应该尽量使用 W3C Schema。但是,由于 DTD 是 XML1.0 标准所支持的,历史较早、存在大量的应用,今后在相当长的时间内,应该允许 DTD 和 Schema 并存,XML 的处理工具应该能够同时支持这两者。而在新建词汇表的时候,以及将已有 DTD 更新的时候,建议逐渐过渡到 Schema 的形式。

8.2 尽量使用 W3C Schema

除了 W3C 制定的 Schema,还存在多种 schema,例如:DTD、RELAX NG、Schematron 等。它们各自都有优缺点。但是,为了方便信息共享,最好仅使用一种 schema。一般情况下,XML 应用应该尽量使用 W3C Schema 和 DTD。

8.3 尽量使用简单的 schema 语法

XML DTD 和 Schema 都提供了强大而灵活的功能。要描述一类文档,schema 可能很简单,也可能很复杂。而大多数 schema 的用户可能都不是 XML 方面的专家,因此应尽量使用最简单的语法来描述文档,尽量不采用 schema 中不常用的机制,这对文档验证也有好处。因为有些验证工具不支持不常用的东西。

8.4 XML 文档的设计要反映信息的结构

置标语言的初衷是将文档的内容和表现形式分离。文档的内容最为人们所关心,需要能够定义文档的逻辑结构(信息模型),验证一个具体的文档正确与否,能够检索特定的文档组成部分,或者将文档的部分或全部进行传送。这些都是 XML 所擅长处理的。而与 XML 文档的结构相比,文档表现形式或文档用途是易变的。DTD 和 Schema 的设计要着重反映文档的逻辑结构,而不是文档的显现与处理。

8.5 文档的 XML 粒度

一个文档的 XML 粒度(即 XML 置标文档内容彻底与否)取决于该类文档信息建模的粒度。例如,对于一个通过三维图形交互的应用,XML 应该能够描述该图形中的各个对象,即需要 schema 能够描述图形元素,因而 XML 文档中需要加入类似 SVG 的代码。而对于产品图录这样的应用来说,一般不需要对图形进行刻画和理解,因而可以将图形数据转换成二进制数据插入到 XML 文档之中。对于数据建模不关心的部分尽可以使用 CDATA 机制或通过链接外部资源来处理。类似的情况还包括表格、代码和各种多媒体对象。一般来说,将一个文档的结构分解过细的结果要好于分解过粗,因为对于经过置标的文档结构,如果不需要处理细节,可以直接处理父元素或祖先元素节点,或合并子元素,不会有什么问题。但是对于分解不彻底的置标文档结构,如果需要处理细节就会很困难。

8.6 尽量由词汇表来规定 XML 文档的合法性

元素的各种关系、数据类型、取值范围等应该尽量使用 schema 来规定。有些内容 schema 确实无法表达时,才求助于外部应用程序。

8.7 采用 XSLT 和其他 schema 来表达事务处理规则

W3C Schema 并不适合表达复杂的事务处理规则,可以考虑通过以下三种途径来解决:

——采用其他的 schema，如 Schematron；

——编写脚本代码表达附加的限定；

——将限定规则表达为 XSLT/XPath 式样单。这种情况下，采用 schema 检验尽量多的限定条件，而采用式样单检验其余的限定条件。如果 schema 的有效性验证工具和 XSL 处理器都产生正确的结果，则实例文档是有效的。

8.8 尽量提高词汇表的可读性

schema 不仅用来验证 XML 文档，更是一种严谨的信息交流机制，系统开发人员往往需要通过阅读 schema 来全面理解和使用 XML 文档，因此必须尽可能地提高 schema 的可读性。例如：

——在名称中尽量少用缩略语。

——枚举数值也应该使用名称，而不是数值。元素和属性的命名规则同样适用于枚举值。

——避免使用元素或属性的缺省值。

——使正则表达式具有更好的可读性，例如：使用“[0-9]”，而不是“\d”。

——避免使用 xsd:redefine，以避免重用组件的副作用，提高透明性和可读性。

8.9 词汇表应尽量做到与应用程序无关

schema 的任何地方都不应存放与特定处理程序相关的配置信息，例如：SQL 语句片断、调用函数名称等。这样做会妨碍 XML 在异构系统中的使用。schema 的设计应该尽量使之能用于多种应用场景。

8.10 不宜使用处理指令来表达数据信息

尽管 XML 处理指令(Processing Instructions)提供了一种控制程序的捷径，但它毕竟只能针对特定的程序，不利于在程序间交换。而且它不能使用标准的 XML 语汇来表达信息，即处理指令的内容是不能由 Schema 来规定的。处理指令应该仅限于局部的个别的处理，处理指令不应与文档内容联系紧密，作为单一指令也不应有结构性的扩展。在考虑使用处理指令之前，应首先考虑使用元素置标。

8.11 使 XML 文档具有可扩展性

在很多情况下，标准化的 XML 文档的更新速度远远慢于现实的变化。因此，经常会发生在 XML 文档中写入了新元素，而这些元素还没有被相应标准正式承认的情况。schema 的设计要容纳开放的内容，即在实例文档中允许包含 schema 没有声明的内容。可以使用 any 和 anyAttribute 来容纳未知的元素和属性，提高 schema 的可扩展性。

在 Schema 中使用〈any〉时，要注意避免在实例文档中出现非确定性内容模型。例如，假设有以下 schema：

```
〈xsd:complexType name="Book"〉
  〈xsd:sequence〉
    〈xsd:any namespace="##any" minOccurs="0" maxOccurs="2"/〉
    〈xsd:element name="Title" type="xsd:string"/〉
    〈xsd:element name="Author" type="xsd:string"/〉
    〈xsd:element name="Date" type="xsd:string"/
    〈xsd:element name="ISBN" type="xsd:string"/〉
    〈xsd:element name="Publisher" type="xsd:string"/〉
  〈/xsd:sequence〉
〈/xsd:complexType〉
```

对于如下实例文档：

```
〈Book〉
  〈Title〉XML 手册〈/Title〉
  〈Author〉Goldfarb,H. F〈/Author〉
  〈Date〉2003〈/Date〉
  〈ISBN〉7-5053-8555-0〈/ISBN〉
  〈Publisher〉电子工业出版社〈/Publisher〉
〈/Book〉
```

有效性验证工具在读入〈Author〉之前将无法判别〈Book〉中出现的〈Title〉是来自〈xsd:element name="Title" …/〉还是来自〈any〉。

为避免出现非确定性内容模型，一般规定〈any〉来自于与缺省命名空间不同的其他命名空间，并放置在内容模型的最后，即如下所示：

```
〈xsd:complexType name="Book"〉
  〈xsd:sequence〉
    〈xsd:element name="Title" type="xsd:string"/〉
    〈xsd:element name="Author" type="xsd:string"/〉
    〈xsd:element name="Date" type="xsd:string"/〉
    〈xsd:element name="ISBN" type="xsd:string"/〉
    〈xsd:element name="Publisher" type="xsd:string"/〉
    〈xsd:element name="other" minOccurs="0"〉
      〈xsd:any namespace="##any" maxOccurs="2"/〉
    〈/xsd:element〉
  〈/xsd:sequence〉
〈/xsd:complexType〉
```

8.12 对允许扩展的元素尽量不限定其出现次数

对于允许扩展的元素，如果不必要，尽量不要限定其出现次数，例如，可将 minOccurs 设为“0”，将 maxOccurs 设为“unbound”。

8.13 [适用于 DTD] 参数化 DTD

在 DTD 中无法重写属性表或元素声明，但是可以改写实体(entity)定义。使用实体是使 DTD 可扩展的关键。如果属性表和元素声明都定义成对参数实体的引用，如需改变属性表和元素声明，只需重新定义参数实体即可。参数化 DTD 依靠的是内部参数实体。当构造一个参数化的 DTD 时，几乎可以把任何内容定义成参数实体的引用，例如：元素名、属性名、元素内容、属性类型、属性表、命名空间 URI 以及命名空间前缀等。另外，还可以在元素内容中用空的实体定义一个用于扩展的子元素。将来只要重定义空实体就可以实现元素内容的扩展。

但是，在 XML 文档中使用实体增加了不必要的复杂性，并妨碍了 XML 文档的可读性。如果不是不得已，不要使用实体机制，但是，XML 的 5 个标准实体(&、<、>、'、")，以及数值实体(&#nnn;)都是允许使用的。

8.14 [适用于 DTD] 模块化 DTD

一个很大的 DTD 文件阅读起来往往会难以理解。尽管跟程序比起来，DTD 一般不算很大，但也不妨将一个大的 DTD 分解成模块，每个模块存成一个文件。这样也利于在一个应用中选择使用不同的模块。模块化就是把一个 DTD 分解成多个独立的功能单元，一个主 DTD 将不同的单元混合起来形成一个整体。与参数化 DTD 不同，模块化 DTD 依靠的是外部参数实体。通过重定义实体，就可以达到

在某处包含某个模块的目的。

8.15 [适用于 Schema] 用 subsititutionGroup 适应动态的变化

将经常发生变化的内容封装起来是一个很好的办法。例如:对于经常更换纸型的打印应用,可以将打印介质作为抽象元素,在实际使用时用"subsititutionGroup"来替换它。示例如下:

```
〈xsd:element name="PrintMedium" abstract="true" type=" PrintMediumType"/〉
〈xsd:element name="A4" substitutionGroup=" PrintMedium" type="A4Type"/〉
〈xsd:element name="A3" substitutionGroup=" PrintMedium" type="A3Type"/〉
〈xsd:element name="B5" substitutionGroup=" PrintMedium" type="B5Type"/〉
```

8.16 standalone 宣声明为"no"

XML 中 standalone 属性缺省为"no"。如果声明为"yes",则表示外部 schema 的任何声明都与文档内容无关,即元素没有缺省属性值,实例文档中从未定义实体引用,属性值不需要规格化,所有元素都不含可忽略的空白。实际上这些经常做不到。因此,除非特别注重性能,一般将 standalone 属性声明为"no"。

8.17 [适用于 Schema] processsContents 的属性值应为 lax

属性 processsContents 指出如何验证来自外部命名空间的 XML 元素。将该属性值设为"lax",可使解析器尝试根据外部 Schema 来解析外部命名空间的 XML 元素,如果无法解析,解析器并不报错,这样不影响应用程序的处理。

8.18 谨慎对待空白

XML 将下列 GB/T 13000.1 字符定义为空白字符:

——空格符(0x20);

——回车符(0x0D);

——换行符(0x0A);

——制表符(0x09);

——上述符号的组合。

在 XML 文档中空白是有意义的,不能随便忽略。例如:

```
〈Name〉John〈/Name〉
```

与

```
〈Name〉
  John
〈/Name〉
```

在解析器看来是不同的,解析器会如实报告后者的 Name 元素内容中多了两个换行符和一个制表符。尽管应用程序可能会忽略这种异同,但是这是应用程序的事,而解析器应该忠于职守。

由于不同语言和应用对空白有不同要求,应该用 xml:space 属性来指定要保留空白(preserve)还是合并空白(default)。如果不指定 xml:space,XML 处理工具会自动合并空白。

在 Schema 中,可以通过制定刻面(facet)whiteSpace 来指示元素或属性中的空白是否需要保留、替换或紧缩。但这只是给应用程序的提示,XML 解析器仍会报告每个空白。

XML 属性中对空白的处理有些特殊。根据不同的属性类型,XML 解析器会在向应用程序报告结果之前规格化这些属性值。首先对所有的属性值,将回车符、换行符、制表符各自都变成一个空白,如果属性值有声明的类型并且类型不是 CDATA,解析器还将把所有的空白行程合并成单个空白,最后会将属性值的前导和后续空白全去掉。在处理时需要特别注意。

9 类型、元素与属性的使用原则

9.1 定义数据类型还是定义元素的准则

许多情况下，一个可重用的组件既可以定义成数据类型也可以定义成元素。在下列情况下，一个组件应该定义成数据类型：

——在不同语境中，以多个名称使用该组件。例如，“地址”在不同语境中可能意味着邮政地址、电子邮件地址、存储地址等，应该将地址定义成数据类型。

——从该组件有可能进一步派生出其他数据类型。

——在实例文档中该组件不打算作为元素出现。

——在实例文档中该组件打算作为元素出现，但是有时可以为空有时不能为空。

在下列情况下，一个组件应该定义成元素：

——不打算从该组件派生出其他组件；

——该组件如果作为元素使用不会改变名称；

——在实例文档中该组件打算作为元素出现，同时允许用其他元素替换之。

总之，元素重用的前提是有确定不变的语义，而类型的重用在一定程度上是允许修改的。定义类型有助于提高元素的可重用性和可扩展性。遇上难以判断的情形，尽可使用类型，因为如果需要的话，总可以从类型创建元素。

9.2 优先使用元素和类型

在设计 schema 时，应该尽量多地使用类型和元素，而不是属性，因为属性不像元素那样可以承载结构化的数据。因此应该用元素作为容纳信息的主要手段，而尽量避免使用属性作为存储信息的手段。可以采用以下准则：

——属性适用于表达描述整个元素的数据元，例如某个数量值的单位。如果该元素存在子元素，则上层的属性也应适合下层元素。

——属性值应该尽量简短，应符合 Name Token 类型的要求。

——属性不宜表达与应用相关的信息。

——复杂的信息应表达为元素。

——如果要表达的信息可能需要进一步细分或扩展，则应该使用元素。

9.3 使 XML 文档尽量自包含

为了达到最大的可移植性和健壮性，XML 实例文档应该尽可能包含所有的信息，即做到自包含，而不应过多地依赖缺省属性值、记法、类型、实体引用，以及其他必须要通过 schema 才能获得的信息。虽然在 XML 文档生成过程中能很方便地使用这些内容，但在文档发布之后，它们会对文档的互操作带来不利影响。很多 XML 解析器并不能很好地分析文档外定义的这些内容，有些应用甚至根本不去访问 schema。例如：

```
〈p〉
The Greek word for father is &pi;&alpha;&tau;&rho;&omicron;&sigmaf;.
〈/p〉
```

要理解这一段内容，解析器必须要找到字符实体的定义，同样的内容用字符数据表示则不会有这个问题：

```
〈p〉
The Greek word for father is &#x3C0;&#x3B1;&#x3C4;&#x3C1;&#x3BF;&#x3C2;
〈/p〉
```

反过来，一个 XML 文档的使用者不应假定该文档是自包含的，应该采用验证型解析器来获得所有正确的信息。

9.4 正确使用全局和局部类型

如果某个元素的类型需要重用，则应该将其定义为全局类型，另外，需要支持类型替换和需要隐藏命名空间的复杂性的时候，也要使用全局类型。如果某个类型不会被重用，且文档大小和耦合程度为考虑的重点时，则应该定义为局部类型。

9.5 尽量定义局部属性

尽量在元素的内部定义属性，即将属性的作用域定义为在元素范围内有效，避免使用全局范围的属性。如果确实需要重用属性定义，最好通过定义新的数据类型或属性组来实现。

9.6 避免用属性修饰属性

属性之间不应该有修饰的关系，否则无法判别属性是在修饰元素还是在修饰一组其他的属性。换句话说，属性之间应该是独立的。

9.7 [适用于 Schema] 复杂类型层次不超过 3 层

过深的层次会给 schema 的理解带来难度，出于简便和灵活性的考虑，建议 schema 中的复杂类型层次不超过 3 层。

9.8 子元素或属性的名称中不要包含父元素的名称

例如："〈Customer〉"的子元素"〈Name〉"，不要命名为"〈CustomerName〉"。这样可以减小名称的长度。

9.9 不要用元素是否出现来指示开关条件

应该用属性值来表示开关条件，例如用〈Space allowed="yes"〉来表示允许出现空格，而不是以空元素"〈Space/〉"是否出现来表示。这样有助于理解。

9.10 [适用于 Schema] 尽量少用"derive -by-restriction"

在面向对象的程序设计中，人们已经广为接受了类的继承的概念。但是在设计 Schema 的时候，应该慎重处理类型的继承。在继承复杂类型时，提倡使用"derive-by-extension"，而不是"derive-by-restriction"。因为如果使用"derive-by-restriction"，不但要罗列所有被继承的元素，更重要的是，被继承元素的任何变化，都可能对所有继承而来的元素产生副作用。

9.11 [适用于 Schema] 正确处理派生

采用 derive 派生类型比较适合 sequence 内容模型，对于 all 或 choice 模型类型的派生，可以考虑使用 substitutionGroup 机制。

9.12 不要使用混合元素内容

混合(mixed)元素可以同时包含文本数据和子元素。为使文档有清楚的结构，在定义 schema 时应该尽量避免使用。例如，应将

```
〈Paragraph〉This is an example of a 〈Emphasize〉mixed〈/Emphasize〉 content model in a text-centric document, and is acceptable. 〈/Paragraph〉
```

改成：

```
〈Paragraph〉
  〈Regular〉This is an example of a 〈/Regular〉
  〈Emphasize〉mixed〈/Emphasize〉
  〈Regular〉content model in a text-centric document, and is acceptable. 〈/Regular〉
〈/Paragraph〉
```

9.13 谨慎使用 CDATA 节

CDATA 节的主要用途是给人类读者增加可读性。不应该使用 CDATA 把存在结构和语义的元素

内容替换成莫名其妙的东西。例如：

```
〈Vehicle〉
  〈Price〉300000〈/Price〉
  〈InStock〉4〈/InStock〉
  〈Color〉black〈/Color〉
  〈! [CDATA[
  〈html〉
  〈title〉The G2 SUV
  〈body〉
  〈img src=g2suv.jpg height=100 width=100〉
  The G2 SUV is one of the …
  …
  〈/body〉
  〈/html〉
  ]]〉
〈/Vehicle〉
```

上面的例子是想把一段 HTML 文本加入到 XML 文档之中。然而对于这种方式，任何 XML 工具都难以处理这种结构——无法进行验证，也无法进行 XSL 式样转换。可以很简单地解决这个问题，即将〈html〉作为〈Vehicle〉的一个合法元素。以后即使需要将 HTML 这一段内容导出也很容易。

9.14 ［适合于 DTD］尽量用 URL 来替代非分析实体和记法

非分析实体(unparsed entity)和记法(notation)对于大多数 XML 用户来说是难懂的，在许多 API 中也得不到很好的支持。其实可以利用 URL 属性来达到同样的效果，而不是非使用它们不可。例如，用非分析实体的方法向文档中加入几种图片，在 DTD 中要这样定义：

```
〈! NOTATION SVG SYSTEM "image/scg+xml"〉
〈! ENTITY LOGO SYSTEM "http://www.some.com/images/cup.svg" NDATA SVG〉
〈! ATTLIST Image source ENTITY〉
```

在实例中，再把图片写成：

```
〈Image source="LOGO"〉
```

这很麻烦也不好懂，不如直接写成：

```
〈Image source="http://www.some.com/images/cup.svg"〉
```

况且读取 URL 属性比起从 DTD 中获得实体也要容易一些。更好的做法可以采用 XLink：

```
〈Image xmlns:xlink="http://www.w3.org/1999/xlink"
       xlink:type="simple" xlink:actuate="onload"
       xlink:show="embed" xlink:href="http://www.some.com/images/cup.sv"
```

对于用记法来标识元素类型的用法，W3C 正致力于通过命名空间加上预定义属性来达到同样的目的，而尽量不再使用记法。

9.15 正确设计容纳可变内容的容器元素

Schema 设计中常常需要把不同资源的分散内容集中到一起，而作为这些内容的“容器”元素还要适应这样的情况，即所包含的内容可能会随时间而增长。一般来说，有四种方法来构造容器元素。

方法 1：通过抽象元素及其元素替换来构造容器元素。例如，如果需要设计一个容器元素 Cata-

logue 来容纳可变的内容——Book 或 Magazine，可以这样设计：

```
〈xsd:element name="Publication" abstract="true" type="PublicationType"/〉
〈xsd:element name="Catalogue"〉
  〈xsd:complexType〉
    〈xsd:sequence〉
      〈xsd:element ref="Publication" maxOccurs="unbounded"/〉
    〈/xsd:sequence〉
  〈/xsd:complexType〉
〈/xsd:element〉
```

再将 Publication 替换为 Book 或 Magazine(注意，要能够进行上述替换，BookType 和 MagazineType 需要从 PublicationType 派生而来)：

```
〈xsd:element name="Book" substitutionGroup="Publication" type="BookType"/〉
〈xsd:element name="Magazine" substitutionGroup="Publication" type="MagazineType"/〉
```

一个实例文档便可以按如下方式使用 Catalogue 容器：

```
〈Catalogue〉
  〈Book〉…〈/Book〉
  〈Magazine〉…〈/Magazine〉
  〈Book〉…〈/Book〉
〈/Catalogue〉
```

这种方法适用于下列情况：所有的内容元素都从一个公共类型派生出来；对容器内容的扩展尽量不用修改 schema；在实例文档中所有的内容元素都是全局的(必须暴露命名空间)。

方法 2：通过抽象类型及其类型替换来构造容器元素。为了改进方法 1 的某些不足，可以这样来设计容器：

```
〈xsd:element name="Catalogue"〉
  〈xsd:complexType〉
    〈xsd:sequence〉
      〈xsd:element name="Publication" type="PublicationType" maxOccurs="unbounded"/〉
    〈/xsd:sequence〉
  〈/xsd:complexType〉
〈/xsd:element〉
```

这样，实例文档将是如下形式：

```
〈? xml version="1.0"?〉
  〈Catalogue xmlns="http://www.catalogue.org"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.catalogue.org Catalogue.xsd"〉
    〈Publication xsi:type="BookType"〉
    ……
    〈/Publication〉
    〈Publication xsi:type="MagazineType"〉
    ……
    〈/Publication〉
〈/Catalogue〉
```

这种方法适用于下列情况：所有的内容元素都具有相同的类型或从同一类型派生出来；允许所有的内容元素有相同的名称；内容元素的增加不受容器元素 schema 的控制；希望隐藏内容元素的命名空间。

方法 3：采用 Choice 元素来构造容器。如：

```
〈element name=“Catalogue"〉
  〈complexType〉
    〈choice maxOccurs="unbounded"〉
      〈element name="Book" type="BookType"/〉
      〈element name="Magazine" type="MagazineType"/〉
    〈/choice〉
  〈/complexType〉
〈/element〉
```

这种方法适用于下列情况：容器需包含无关的内容；需具体指定允许哪些元素放入容器；允许修改 schema 反映内容元素的变化。

方法 4：通过在实例文档中指定命名空间的 schema 来实现容器的功能。例如：

```
〈? xml version="1.0"?〉
  〈xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace=……
  xmlns=……
  xmlns:s="http://www.namespace.org"
  elementFormDefault="qualified"〉
  〈xsd:import namespace="http:// www.namespace.org "/〉
  〈xsd:element name="MyElement"〉
    〈xsd:complexType〉
      〈xsd:sequence〉
        〈xsd:element name="ele" type="s:MyType" maxOccurs="unbounded"/〉
      〈/xsd:sequence〉
    〈/xsd:complexType〉
  〈/xsd:element〉
〈/xsd:schema〉
```

注意，在上述〈 xsd:import…〉中没有按一般方式指定 schemaLocation。这样可以在实例文档中指定命名空间“http://www.namespace.org”的具体实现。例如：

```
〈? xml version="1.0"?〉
  〈MyElement xmlns=……
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="…… http://www.namespace.org my_namespace.xsd"〉
    〈ele〉……〈/ele〉
  〈/MyElement〉
〈/xml〉
```

这种方法适用于下列情况：需要简单类型的容器；容器包含大量动态变化的、需要定制的内容；应用允许容器与内容使用不同的命名空间。

9.16 谨慎使用空元素

可选元素的内容不应该允许为空，它们应该或者以非空的形式出现，或者不出现。这样有助于保持Schema的简洁，并节省资源。

对于预设带内容的强制出现的元素，Schema也应该限制其不允许为空，可通过限制字符串的最小长度保证其中至少有一个字符。

10 版本与注释的使用原则

10.1 为schema添加注释

[适用于Schema]对于文档中的注释部分，应使用〈annotation〉的〈documentation〉子元素添加用户可阅读的说明，而使用〈annotation〉的〈appinfo〉子元素添加机器可阅读的说明，不宜使用XML注释来添加这些注释。这样可以使用XSLT技术将元素的说明很容易地进行处理成用户相关的文档数据。而XML注释中的内容则无法处理。

当然注释的使用会给XML的处理增加额外的开销，因此对于广为使用的schema，可以准备两种形式，一种带注释，一种不带注释。

对于DTD，只能用XML注释的方式来添加注释。

10.2 使用版本声明

schema和式样单的设计应该尽量做到向后兼容。应通过注释或内置属性提供版本信息给应用系统，以便正确处理和向后兼容。对于schema，可以通过标识根元素的版本属性反映版本信息，或将版本信息放在头部的注释中。例如：

```
〈? xml version='1.0' encoding='UTF-8' ?〉
〈! ELEMENT root EMPTY〉
〈! ATTLIST root schemaVersion CDATA #FIXED '1.0'〉
```

对于Schema，应该使用〈xsd:schema〉元素的version属性标识Schema的版本。例如：

```
〈? xml version="1.0" encoding="UTF-8" ?〉
〈xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified" version="1.0" 〉
...
〈/xsd:schema〉
```

对于式样单，它必须通过内置的version属性说明式样单自己的版本，另外还需要说明被转换实例所对应的Schema的名字和版本。

schema的版本必须在注册库中注册。在声明版本时，要注意下列问题：

a) 应该保证能从schema中获得版本声明，版本的变更也能在schema中反映出来；

b) 在实例文档中要标明所支持的schema版本；

c) 先前版本的schema要保留下来；

d) 如果schema仅在原来的基础上进行了扩充，应该尽量避免需要重新验证实例文档的有效性；

e) 如果schema发生了大的变化，应增加主版本号，如从1.0变为2.0；如果仅是细微的扩展，增加次版本号即可。

另外，当新的schema改变了原来某些元素的意义，使得原先有效的实例文档在新的schema下变为无效的时候，应该改变目标命名空间。这时候，实例文档相应要做以下改变：更改实例以适应新的目标命名空间，确认与新的schema没有冲突；更新那些标识schema版本的属性；如果需要，还要更新schema名称和位置(location)。

按照惯例，正式版本的版本号采用“m. n”的形式（如“1. 2”）；而非正式版本用“m. na”的形式（如“1. 2b”，表示正式版本 1. 2 之前的第二个测试版）。

10.3 使用说明性的文件头

文件头可作为 XML 注释放在 schema 中，可以规定统一的文件头格式。例如：

——schema 名称；

——注册了的命名空间；

——对口单位；

——指向当前版本的 URL；

——用到的其他 Schema 的名称、命名空间、版本、位置[适用于 Schema]；

——关于 PUBLIC、SYSTEM、记法等的说明[适用于 DTD]；

——外部实体引用及其说明[适用于 DTD]；

——schema 作用描述；

——创建和管理 schema 的应用程序名称和版本；

——对应用接口的简单描述（可通过 URL 指向其详细说明）；

——作者的联系信息；

——版本变更历史（编号、版本、日期和变更内容等）。

使用说明性文件头的规则在其他 XML 文档中也同样适用。例如 XSL 式样单的说明性文件头应该包括：

——式样单名称；

——经过测试的 XSLT 处理程序的名称及版本；

——经过测试的 Schema 名称及版本；

——注册了的命名空间；

——对口单位；

——指向当前版本的 URL；

——式样单作用描述；

——开发和维护式样单的应用程序名称和版本；

——作者的联系信息；

——版本变更历史。

对于 XML 实例，应该在头部说明可用于验证该实例的 Schema 的名字和 URL，如果存在式样单，还应包括可对之正确处理的式样单的名字和 URL。

一个 Schema 使用的说明性文件头的实例如下：

```
〈xs:annotation〉
  〈xs:documentation〉
    Schema Name: SPAWARVPO$2-1_FolderData$1-1.xsd
    Federal XML Registry Information: TBD
    Functional Data Area: Administration
    Current version available at (URL):
    https://www.spawar.navy.mil/vpo/schemas/SPAWARVPO$2-1_FolderData$1-1.xsd
    Other Schemas Imported (XML Schema only):
    * * * * Namespace Prefix: PER
    "http://diides.ncr.disa.mil/xmlreg/user/namespace_list.cfm"
    * * * * Schema File Name: BUPERSBUPERSOnLine$3-0_Document$2-2.xsd
```

```
    * * * * Available at URL: www. bupers. navy. mil/bupersOnLine/schemas/
    Other Schemas Included (XML Schema only): None
    External DTDs Referenced (DTD only): n/a
    * * * * Name: n/a
    * * * * Available at (URL): n/a
    Description: Provides information regarding the content of VPO folders such as
    content file names, file sizes, file owner, file status, and file access information.
    Application: Virtual Program Office
    Application Version: 2. 1
    Application Interface:
    XML data is available from the VPO application via HTTP at
    https://www. spawar. navy. mil/vpo/GetFolderInfo. asp. Input queries via HTTP
    GET with query string format, "…? dir=directoryName". A complete interface
    description document is available at
    https://www. spawar. navy. mil/vpo/interfaces/GetFolderInfo. txt
    Associated Stylesheet:
    * * * * Name: SPAWARVPO $ 2-1_ViewFolderContents $ 1-0. xsl
    * * * * Available at (URL): https://www. spawar. navy. mil/vpo/stylesheets/
    Developed by (Gov't Activity): SPAWAR 08
    Point of Contact Name: Joe Smith
    Point of Contact Email: jsmith@spawar. navy. mil
    Change History:
    CHANGE # Version DATE DESCRIPTION OF CHANGE
    0 1. 0 15 Sep 2001 Initial release
    1 1. 1 30 Sep 2001 Updated to include file size information
    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  </xs:documentation>
</xs:annotation>
```

一个 XML 实例使用的说明性文件头的实例如下：

```
<? xml version="1. 0" encoding="UTF-8"?>
<! -
  Schema and Stylesheet Reference Data:
  stylesheet type = xslt
  url = http://spawar. navy. mil/stylesheets/SPAWARVPO $ 2-1_ViewFolderData $ 1-1. xsl
  version = 1. 1
  schema type = XML Schema (W3C)
  url = http://spawar. navy. mil/schemas/SPAWARVPOV2-1FolderDataV1-1. xsd
  version = 1. 1
  ->
<root/>
```

10.4 在注释中声明元数据

元数据包括 Schema 中定义的各种名称的出处(例如是注册库中的哪个条目)和参考资源的地址。应在注释中声明元数据。例如:

```
〈xs:annotation〉
  〈xs:documentation source="…" xml:lang="…"〉
    〈ebXML〉
      〈CoreComponent UID="core000152"〉Text. Type〈/CoreComponent〉
    〈/ebXML〉
  〈/xs:documentation〉
〈xs:annotation〉
```

10.5 在使用数据集的时候应加辅助信息

采用是国际上通用的,或是正式定义的数据集(如货币代码)有助于互操作。使用这些数据集时,应该用属性注明其来源或通过链接指向其发布机构。例如:

```
〈Currency〉
  ValueSet="ISO 4217"
  xlink:type="simple"
  xlink: href = " http://www. iso. ch/iso/en/CatalogueDetailPage. CatalogueDetail? CSNUMBER =
34749"〉GBP
〈/Currency〉
```

10.6 不要在命名空间 URI 中附加版本信息

不要在命名空间 URI 中附加版本信息,在 schema 版本更新时也尽量不要改变命名空间。因为很多应用高度依靠命名空间,改变命名空间会影响其正常运行,如果不改变命名空间,这些应用大多都能很好地适应新版本的 schema。除非新的版本有了本质的变化,带来了整个词汇表的改变,一般情况下不要轻易改变命名空间。

10.7 编写正式的词汇表使用说明

schema 的开发者必须通过注释,或者专门的使用手册,或者形式化的数据字典,来描述各种 schema 对象的含义和使用方法。

11 实例的编写原则

11.1 所有的 XML 文档应该都有 schema

虽然有大量的应用仅要求格式良好的(well-formed)XML 文档,不需要 schema,但是为了达到更高的可靠性和健壮性,最好把所有的 XML 文档都加上 schema,因为 schema 能很方便地帮助解析器测试 XML 文档,而测试读入的文档是否符合要求是保证程序可靠性的前提。因此,即使 XML 文档没有 schema,也不妨为它设计一个。

11.2 用 SchemaVersion 标识元素的版本

如果需要的话,Schema 应该要求实例文档的元素带有版本标识。这样可以知道实例是由哪一个 schema 生成的。接收 XML 数据的应用也可以甄别哪些是可以正确处理的数据。当一个文档的元素来自多个版本的不同 Schema 时,更加需要版本标识。这时,应该用属性 schemaVersion 来标识实例文档元素的版本,例如:

```
〈xsd:element name="AccountingBook"
  〈xsd:complexType〉
    〈xsd:sequence〉
    〈! – element content goes here –〉
    〈/xsd:sequence〉
    〈xsd:attribute
    name="SchemaVersion"
    type="xsd:NMTOKEN"
    use="required"
    fixed="1.0"/〉
  〈/xsd:complexType〉
〈/xsd:element〉
```

11.3 不推荐在 XML 实例文档中添加注释

XML 实例通常是自动产生的，并会被自动处理，通常不由人来阅读。必需的注释信息最好写在相应的 schema 或者 XSL 式样单中。另外，必须要表达的信息应该出现在 XML 文档内容中，而不是注释中，因为注释的内容通常是不能自动识别的。一个例外是，应该在 XML 注释中写明 DTD、Schema 或式样单的名称、版本和 URI。

11.4 用 Base64 编码图形等二进制数据

对于图形等二进制文件的使用，如果数据量不大，建议将二进制数据进行 Base64 编码转换为文本后直接嵌入 XML 元素中。但对于数据量很大的情形，因为编码和解码要付出较大代价，使用 Base64 编码可能会不适合。这种情况下可以考虑使用 MIME 等特定解释器能理解的起始和终止标记，把图形描述代码加入到 XML 文件中；或者通过链接，将数据作为外部文件处理。

二进制数据的编码应该采用 Base64，而不要采用 UUEncode、HexBinary、Quoted Printable 等编码方法。

应该尽量使用独立的文件存储非 XML 内容的数据块，并使用处理指令、XInclude、XLink 或相似的机制在文档中包含这些数据块。

11.5 适度的文件大小

对于 XML 文件的大小，没有具体的规定，但在实际的设计中应考虑过大的文件对于系统性能可能带来的影响。特别是采用 DOM 解析器时，可能造成时间和内存的大量消耗，而过小的文件不能充分利用 XML 的丰富结构来表示信息模型中的关系。因此，文档编写者需要根据具体情况决定 XML 文件的大小。可采用实体引用或〈include〉、〈import〉机制，将太大的文件拆分成多个。

XML 使用实例参见附录 A。

12 XML 解析器及其选择

12.1 XML 解析器的符合性测试和中文支持

XML 解析器的作用主要是读入 XML 文件，分析、验证其有效性或存取其各结构成分，解析器是 XML 应用的核心。XML 解析器有两类：有效性验证型的和非有效性验证型的。有的解析器同时支持两者，通过开关决定是否需要进行有效性验证。一个 XML 解析器应该能够通过 W3C 发布的 XML 符合性测试套件(XML Conformance Test Suites)的测试。XML 符合性测试套件包含了 2000 多个测试文档和相关的测试报告，测试报告陈述测试背景并描述每个测试文档。

XML 应用应该尽量采用验证型解析器，应该能够完全涵盖 XML 标准所规定的内容。需要能够完全读入 schema，并检测文档中的错误，而不应该有任何假定，例如：

——所处理的文档具有符合 W3C Schema 的 Schema，或其他某种类型的 schema；

——在处理文档之前就知道文档肯定会有某种结构(如表结构);或认为文档内容是可预测的;

——不会存在混合元素内容;

——其他。

因此要格外小心那些只能处理"某种 XML 文档"的专用解析器,由于输入的文档未必都符合它们的要求,这些工具可能会显得不可靠或不稳定。专门的文档处理应该是客户端应用程序的事,而其前端的解析器应该是通用的。

在中文处理方面,一个 XML 解析器应该能够支持 GB18030、GB 13000 和 GB 2312 字符集,并最好能够支持台湾地区的 Big5 字符集。能够正确处理字符引用,通过 XML 声明正确核验字符集,允许所有的 XML 名称使用中文字符。另外,能够正确读入和输出中文 XML 文档。

12.2 三类 XML 解析器

所有 XML 数据都可以表示成树形结构,用各种遍历算法来访问 XML 文档的节点、搜索内容或进行编辑。这种按树形结构处理 XML 文档的解析器一般基于 W3C 的文档对象模型(DOM)。DOM 提供了丰富的应用程序接口(API)来访问 DOM。在文档进行其他操作之前,必须先在内存中生成 DOM 树,所以对于大型的 XML 文档必须有虚拟内存的支持。这类解析器适合在速度要求不高的情况下,构建和解析结构性强的中小型 XML 文档。

处理 XML 数据的另一种方法是使用事件驱动型解析器。这种处理器在处理 XML 文档的时候,随着遇见每类 XML 数据(元素及其属性、字符数据、处理指令、符号或注释),不断触发事件,调用应用程序(事件处理程序)处理。这类解析器一般基于 XML 简单 API(SAX)。SAX 解析器解析 XML 数据后并不维护树形结构,所以和文档大小无关。这类解析器处理速度快,适合处理已构建好的 XML 文档。

另一类 XML 解析器通过绑定来实现 XML 文档和编程对象之间的映射。典型的例子是 JAXB (Java XML Binding),它是除 SAX 和 DOM 之外访问 XML 数据的另一种方法。将编程对象绑定到 XML 文档,可以使一种格式容易地转换为另一种格式,反之亦然。这种访问方式特别适合在 XML 文档中存储配置信息,它非常便于直接访问所需的参数,而无须使用复杂的树形结构。虽然这种解析器对于文档转换或消息传送没什么用处,但对于简单的数据处理是极其方便的。

12.3 XML 解析器的选择

下列情况下可以选择 DOM 解析器:

——内存资源充足;

——需要随机处理大范围文档树的不同部分;

——程序需要前后遍历整棵树;

——程序需要处理文档的所有数据;

——程序内部数据结构与文档树有良好的对应;

——开发者觉得树形结构处理起来比处理事件方便;

——程序需移植到不同语言。

下列情况下可以选择 SAX 解析器:

——处理速度很关键;

——内存资源有限;

——每次处理的都是短小连续的文档片断;

——处理程序内部数据结构与 XML 文档差异很大;

——只需分析文档,不需要将文档写回。

下列情况下可以选择绑定形解析器:

——编程语言提供了可供与 XML 绑定的数据结构;

——XML 文档变化频繁;

——规格化数据组成的 XML 文档(如表格)。

选择 XML 处理器(包括解析器)时应特别注意,XML 处理器应该不折不扣地执行 XML 标准(包括

API 标准)的全部规定,处理器厂商不能随意加进、减少或改变任何内容。要警惕 XML 处理器是否有附加的知识产权内容,以避免额外的代价。一个 XML 应用不应仅仅依赖一个特定的处理器,所有符合标准的处理器应均可作为候选。

12.4 XML 应用程序应该向后兼容先前的 schema

应用程序应该尽量做到能向后兼容先前的 schema,这样能最大程度地保留以前的成果。应用程序同时应该检查 XML 文档的 schema 版本信息以便知道当前正在使用的 schema 的版本,以便使各个版本的 XML 文档都能得到正确处理。如果应用程序无法向后兼容先前的 schema,应该提供相应的版本迁移工具以便进行文档的转换。

12.5 XML 应用程序应该忽略不相关的元素或属性

XML 应用程序应该只处理所关心的元素或属性,对于其他的元素或属性不必作解析或处理,更不应报错,这样可以使应用程序灵活适应更多的变化。

12.6 XML 应用程序不应过度依赖文档结构

一个能适应灵活变化的文档结构的应用程序不应过分依赖一种特定的文档结构,例如 XPath 的指定应尽量用相对路径表示文档片段,而避免使用固定的绝对路径。

12.7 XML 应用程序应该仅利用可靠的解析器结果

XML 的应用程序应该建立在可靠的解析器解析结果之上。例如,几乎所有合格的 XML 解析器均会正确报告元素边界、文本内容、属性值和处理指令等内容,但可能不会准确报告像 CDATA 节、实体引用、字符引用、属性顺序、注释、属性值是缺省的还是实例赋予的等等内容。因为这些东西大多数在解析器分析之前就会被解析掉。虽然有的解析器能报告这些东西,但是 XML 应用程序不应仅依赖一个特定的解析器。

13 应用开发过程

13.1 以工程化的方法来设计 schema

应以软件工程的观念设计 schema,例如:

——对 schema 进行模块化分解,在一个 schema 中只定义一种类型的信息;

——公共信息由专门的 schema 来定义,以提高重用性和效率;

——schema 之间不要相互依赖和影响;

——对于复杂的 schema 可以采用自上而下和自下而上结合的开发和集成方法。

为了保证各种基于 XML 的文档和标准的一致性、互操作性、复用性,必须制定完善的 XML 设计过程标准,覆盖规划、设计、制作、提案、发布、版本更新的全过程。具体而言,这样的过程应包括以下步骤:

a) (特定主题的)XML 标准化工作的发起程序。发起原因可以是:
   1) 由电子政务/电子商务或其他标准化组织计划的 XML 设计工作;
   2) 通过征求意见(RFP)方式收到的提案。发起程序中应该包括发起审议、工作组机构建设、工作规程、信息发布渠道等内容。

b) XML 标准的开发过程。进行需求分析、模型设计、XML schema 开发、相关审议工作。开发过程必须规定各项成果的内容和格式。需要特别注意的是:必须寻找和使用已有的商业模型和 XML 文档标准,不要与之冲突,尽量避免重复开发。

c) 为了设计一个合理的、可重用的 schema,应该将信息建模过程与 schema 的构建过程分开,建模完成后,再运用规则将模型映射到 schema。schema 的开发应是一个团队行为,团队应包括各种相关的专家、管理人员和技术人员,避免仅由信息技术人员来承担。比较适于 schema 开发的是图形化建模语言,可以考虑 UMM(Unified Modeling Methodology)或 UML(Unified Modeling Language),而一些关系建模语言则不太适合。

d) XML 标准的批准和发布。完成的 schema 必须通过质量控制组的审查,包括形式上的完整性、是否与已有商业模型和 XML 文档标准冲突等。为此,XML 设计规范应该明确对 schema

的要求，然后发布 schema 以公开征求意见(RFC)，最后完成的 schema 应在正式渠道发布，并加入到统一的组件注册和版本管理系统中。

13.2 不宜用 XML 取代关系型数据库

虽然 XML 可以沟通异构数据库平台的数据，但 XML 本身并不适于作为数据库使用。考虑到成熟性、方便性和检索效率等方面，目前的 XML 数据库也不能取代关系型数据库的地位。所以，所有适合关系型数据库的应用不应盲目向 XML 迁移。作为 XML 数据，目前可以考虑通过数据库的 XML 接口存入关系型数据库中，或选用一个真正的 XML 数据库来管理。

14 注册规程

14.1 共享信息方面需要考虑的问题

信息共享是一种资源共享。信息共享需要通过资源描述(元数据)来实现。元数据的包含如下三个方面：

——语义(semantics)，即词汇(数据元)的含义；

——结构(structure)，即词汇的组织，如章、节、段的顺序；

——语法(syntax)，即描述的约定(主要针对结构)。

信息共享需要共享各方就元数据的上述方面达成一致。

元数据主要通过置标语言来描述。XML 提供了定义和使用特定置标语言的规则。它以标准的形式和机器可读的方式为特定的置标语言提供了形式化描述的手段，这样使置标语言的描述可以被共享。但是，XML 并不能直接支持元数据的共享。首先，XML 定义了表达结构数据的语法(syntax)，主要用于在程序、应用和系统之间表示和交换数据，然而仅采用公共语法并不能解决信息共享的所有问题；其次，schema 没有说明置标的含义。schema 必须通过附加文档提供该信息给用户。这种附加文档常常以使用指南的形式提供。

信息共享在数据元方面面临两个问题：一、两个文档使用相同的术语(数据元名称)表达不同的概念。XML 可以通过命名空间来避免这种混淆。二、两个文档使用不同的术语表达相同的概念。这种情况在不同的社团之间的 XML 文档中经常出现。XML 本身不能解决这个问题。要解决这个问题，一方面需要使用公共词汇表和公共数据元(数据词典或标记库)；另一方面，为了能够找到这些词汇表和数据元，需要借助注册机制和资源发现服务。在元数据的语义方面，通用元数据如都柏林核心元素集(Dublin Core)及其扩展，主体信息服务的资源组织和发现(ROADS)、RFC1807 等，有助于解决数据元的一致性。信息共享还面临第三个问题，即使用不同的结构习惯表达相同的文档结构。解决这个问题一方面要尽量使用公共词汇表，另一方面要借助资源描述框架(RDF)。RDF 是 W3C 发布的元数据描述的“宏标准”，定义了由资源、属性与声明组成的元数据的基本描述模式，并能集成多个元数据格式，提供了一种统一的元数据置标和交换机制。

一个开放式元数据开发体系一般包括以下几个过程：

a) 按照开放标准和计算机可识别形式对有关信息内容进行定义和描述，形成 XML 表示的元数据格式。

b) 元数据格式可通过一定的标准引用机制，复用或继承其他元数据格式中的元素，来定义或解释自己的元素或处理方式。

c) 通过一定的公开机制对元数据格式进行认证。

d) 经过认证的元数据格式以唯一标识符标识。

e) 唯一标识的元数据格式在开放注册系统注册。注册系统通过开放平台提供元数据格式及相应元数据体系的公开查询和调用，因此保障任何用户和代理系统能够查询到并利用标准方法识别元数据的结构和语义。

14.2 XML 作品注册中心

XML 注册中心通过数据管理系统，提供对 XML 作品(包括 schema、式样单以及 XML 实例文档

等)的管理和服务。通过 XML 注册中心,可以自动获取 XML 作品,搜索或浏览 XML 作品,保存 XML 作品或者对其进行注册。XML 注册中心作为一个枢纽,任何人都可以搜索和访问其中的 XML 文档。除了 XML 作品本身,XML 注册中心还维护了每个文档的基本信息,如作品的主人、作品的类型、创建日期和最后修改日期、作品描述和关键词,以及用于检索的分类号。对于有多个语言版本的 XML 作品,XML 注册中心还会提供各个语言版本(主版本和副版本)的版本信息,指出它们的关联。

为便于作品的管理,注册中心按层次结构组织文档,并对每个作品冠以所属目录里的唯一的名称。XML 注册中心为每个作品赋予一个 URI,URI 应能反映出作品的名称以及从层次结构的根到包含作品的目录构成的路径。可以直接使用这个 URI 访问或浏览 XML 作品。至于 XML 注册中心如何在物理上组织起层次结构则视业务需要而定。

根据实际情况,可能需要构造分层的注册机制,并对服务进行细致的划分和有效的管理。物理上,可以建立由对等网络连接的多个注册站点,但从国家的角度已经建立了权威的 XML 注册中心。该注册中心由指定机构负责其注册等日常工作。指定机构的基本信息参见附录 B。

14.3 XML 作品的注册过程

XML 作品的注册机构的设置和处理流程如下:

a) XML 注册中心下设秘书处和专家组。其中秘书处的主要任务是受理用户或相关组织提出的各种 XML 作品维护请求;根据所受理请求的情况,不定期地提出适当的维护建议提交专家组审查;根据注册中心的决定对所维护的 XML 作品进行更新;以多种形式宣传和推广所维护的 XML 作品的最新版。专家组的主要任务是根据所维护的 XML 作品的技术特征,制定实施动态维护工作所需的工作程序和实施技术评审所遵循的技术评审规则;按照技术评审规则对 XML 作品维护请求实施技术评审;形成书面形式的评审结论并报维护处理结构批准。

b) XML 作品的用户或相关组织可根据需求实时地向注册中心秘书处提出有关申请或获取 XML 作品的最新版。

c) 需求驱动机制由两方面组成,一是通过注册中心为 XML 用户设立的“XML 作品维护请求入口”(由注册中心秘书处承担);二是注册中心的“XML 作品维护请求处理机构”。有关组织和用户可根据实际应用需要及时地通过“入口”向注册中心秘书处提出 XML 作品维护请求。

d) 通过注册中心的 XML 作品维护请求处理机构,向 XML 作品维护请求提出者通报评审结论,并不定期向相关用户公布新一版的 XML 作品,以形成对市场的快速应答机制。

一个 XML 作品的注册过程见图 1。

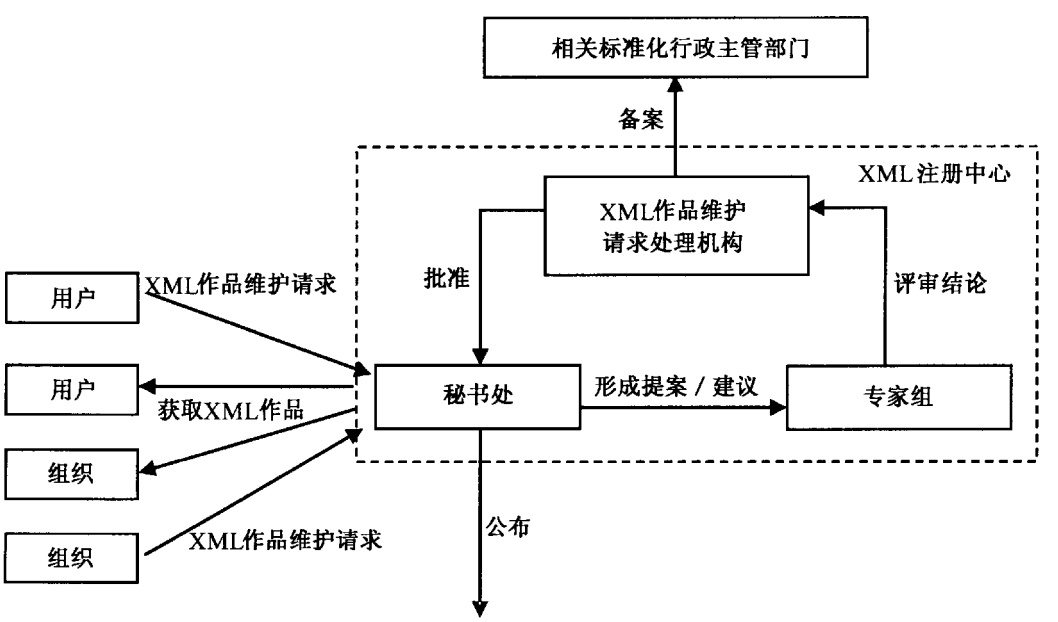


图 1 XML 作品的注册过程

附 录 A
（资料性附录）
使用实例

A.1 电子邮件

这里通过电子邮件词汇表来阐述多语种之间 XML 词汇表的转换。

电子邮件 Schema 采用中文作为主版本。国外机构可以将电子邮件 Schema 翻译成自己需要的语种，并据此生成相应语种的文档实例，在这里对如何翻译和转换可不做规定，仅要求要在实例中保留所有语种相关的标识(loc:locID 和 loc:attrList 属性值)。

为了能够实现国际化的需要，该电子邮件文档中文置标内容的使用范围限定为：元素名、属性名和类型名，其他内容(例如枚举类型值)尽量采用英文表达。另外，避免使用来自不同命名空间的相同属性名。实例中元素名的中外文转换采用标识符对应的方式，通过对相同语义和名称的元素指定唯一的标识符，达到中外文版本相同元素的严格对应。这样，可以通过 XSL 式样单或程序进行简单的转换，使任何语种的电子邮件文档都能通过单一式样单或程序转换回到中文版本。

首先，在本地化命名空间 loc 中，增加两个全局属性：locID 和 attriList，分别用作元素语种标识符和“属性列表”属性的名称。这两个属性专门用于多语种转换。

〈xsd:attribute name="locID" type="xsd:string"/〉

〈xsd:attribute name="attrList" type="xsd:NMTOKENS"/〉

例如：

〈ns:收件人 loc:locID="001" …〉

外文置标的文档，电子邮件实例元素应该有与中文版本一致的 loc:locID。例如：

〈ns:Receiver loc:locID="001" …〉

当收到外文置标的文档时，由于相同 loc:locID 的元素具有同样的语义和名称，因此可以确定该元素的中文名称为“收件人”。

基于前面的约定，加之属性名的翻译仅限于当前命名空间的范围，属性名的翻译将采用如下方式：对每个带有属性的元素，将强制出现一个特殊的 Name Tokens 类型的属性 loc:attriList。这个属性的值列出了该元素允许出现的所有属性名的一个有序列表。例如：

〈信:收件人 … loc:attriList="发送 抄送 密送" 信:发送="true"〉

外文置标的文档，对于每个指定属性的元素应该有相应的属性 loc:attriList，其属性值中属性列表的顺序和每个属性的语义必须与相应的中文属性列表一致。

当收到外文置标的电子邮件文档时，对于带有属性的元素，需要与实际出现的属性的属性名进行匹配(忽略命名空间前缀)，以确定实际出现的属性名应该对应的中文名称。例如，对应上例的情况中，如果相应的外文置标的元素是：

〈ns:Receiver … loc:attriList="sc cc bc" ns:sc="true"〉

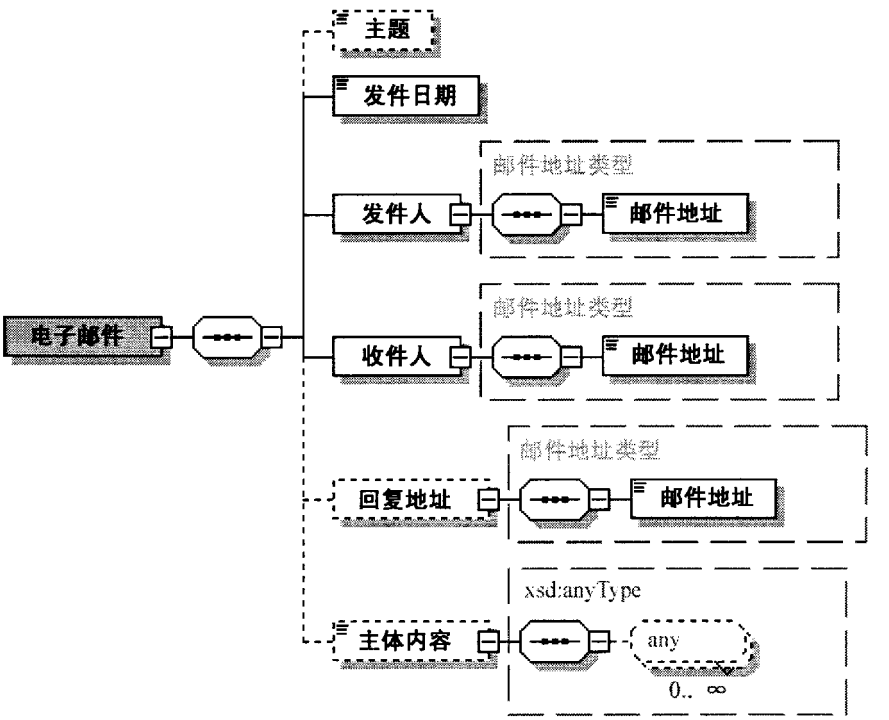
对于实际出现的属性的属性名“ns:sc”，将 sc 与属性列表“sc cc bc”中的每个名称匹配，确定实际出现的元素处于属性列表的第一个位置，从而确定中文元素名应为“发送”。

一个完整的电子邮件的例子如下。

loc. xsd：

```
〈? xml version="1.0" encoding="UTF-8"?〉
〈xsd:schema targetNamespace="http://www.cnegov.com.cn/schemas/loc"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="qualified"〉
  〈xsd:attribute name="locID" type="xsd:string"/〉
  〈xsd:attribute name="attrList" type="xsd:NMTOKENS"/〉
〈/xsd:schema〉
```

中文 schema(cnSchema. xsd)：




```
〈? xml version="1.0"?〉
〈xsd:schema targetNamespace="http://www.cnegov.com.cn/schemas/email/cn" xmlns:loc=
"http://www.cnegov.com.cn/schemas/loc" xmlns:邮件="http://www.cnegov.com.cn/sche-
mas/email/cn" xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault=
"qualified" attributeFormDefault="qualified"〉
  〈xsd:annotation〉
    〈xsd:documentation xml:lang="en"〉Email schema example〈/xsd:documentation〉
  〈/xsd:annotation〉
  〈xsd:import namespace="http://www.cnegov.com.cn/schemas/loc" schemaLocation="../
loc.xsd"/〉
  〈xsd:complexType name="邮件地址类型" id="t01"〉
    〈xsd:sequence〉
      〈xsd:element name="邮件地址" id="e01"〉
        〈xsd:complexType〉
```

```
        〈xsd:simpleContent〉
          〈xsd:extension base="xsd:string"〉
            〈xsd:attribute ref="loc:locID"/〉
          〈/xsd:extension〉
        〈/xsd:simpleContent〉
      〈/xsd:complexType〉
    〈/xsd:element〉
  〈/xsd:sequence〉
〈/xsd:complexType〉
〈xsd:element name="电子邮件" id="e1"〉
  〈xsd:complexType〉
    〈xsd:sequence〉
      〈xsd:element name="主题" id="e1-1" minOccurs="0"〉
        〈xsd:complexType〉
          〈xsd:simpleContent〉
            〈xsd:extension base="xsd:string"〉
              〈xsd:attribute ref="loc:locID" fixed="e1-1"/〉
            〈/xsd:extension〉
          〈/xsd:simpleContent〉
        〈/xsd:complexType〉
      〈/xsd:element〉
      〈xsd:element name="发件日期" id="e1-2"〉
        〈xsd:complexType〉
          〈xsd:simpleContent〉
            〈xsd:extension base="xsd:dateTime"〉
              〈xsd:attribute ref="loc:locID" use="required" fixed="e1-2"/〉
            〈/xsd:extension〉
          〈/xsd:simpleContent〉
        〈/xsd:complexType〉
      〈/xsd:element〉
      〈xsd:element name="发件人" id="e1-3"〉
        〈xsd:complexType〉
          〈xsd:complexContent〉
            〈xsd:extension base="邮件:邮件地址类型"〉
              〈xsd:attribute ref="loc:locID" use="required" fixed="e1-3"/〉
            〈/xsd:extension〉
          〈/xsd:complexContent〉
        〈/xsd:complexType〉
      〈/xsd:element〉
      〈xsd:element name="收件人" id="e1-4"〉
        〈xsd:complexType〉
          〈xsd:complexContent〉
```

```
            〈xsd:extension base="邮件:邮件地址类型"〉
              〈xsd:attribute ref="loc:locID"/〉
              〈xsd:attribute ref="loc:attrList" use="required" fixed="发送 抄送 密送"/〉
              〈xsd:attribute name="发送" type="xsd:boolean" id="a01"/〉
              〈xsd:attribute name="抄送" type="xsd:boolean" id="a02"/〉
              〈xsd:attribute name="密送" type="xsd:boolean" id="a03"/〉
            〈/xsd:extension〉
          〈/xsd:complexContent〉
        〈/xsd:complexType〉
      〈/xsd:element〉
      〈xsd:element name="回复地址" id="e1-5" minOccurs="0"〉
        〈xsd:complexType〉
          〈xsd:complexContent〉
            〈xsd:extension base="邮件:邮件地址类型"〉
              〈xsd:attribute ref="loc:locID" fixed="e1-5"/〉
            〈/xsd:extension〉
          〈/xsd:complexContent〉
        〈/xsd:complexType〉
      〈/xsd:element〉
      〈xsd:element name="主体内容" id="e1-6" minOccurs="0"〉
        〈xsd:complexType〉
          〈xsd:complexContent〉
            〈xsd:extension base="xsd:anyType"〉
              〈xsd:attribute ref="loc:locID" use="required" fixed="e1-6"/〉
            〈/xsd:extension〉
          〈/xsd:complexContent〉
        〈/xsd:complexType〉
      〈/xsd:element〉
    〈/xsd:sequence〉
    〈xsd:attribute ref="loc:locID" use="required" fixed="e1"/〉
  〈/xsd:complexType〉
〈/xsd:element〉
〈/xsd:schema〉
```

英文 Schema(enSchema. xsd):

```
〈? xml version="1. 0"?〉
〈! -- edited with XMLSPY v5 rel. 3 U (http://www. xmlspy. com) by TEAM (RENEGADE) --〉
〈xsd:schema targetNamespace="http://www. cnegov. com. cn/schemas/email/en"
xmlns:loc="http://www. cnegov. com. cn/schemas/loc"
xmlns:mail="http://www. cnegov. com. cn/schemas/email/en"
xmlns:xsd="http://www. w3. org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="qualified"〉
```

```
<xsd:annotation>
  <xsd:documentation xml:lang="en">Email schema example</xsd:documentation>
</xsd:annotation>
<xsd:import namespace="http://www.cnegov.com.cn/schemas/loc" schemaLocation="../loc.xsd"/>
<xsd:complexType name="MailAddressType" id="t01">
  <xsd:sequence>
    <xsd:element name="Address" id="e01">
      <xsd:complexType>
        <xsd:simpleContent>
          <xsd:extension base="xsd:string">
            <xsd:attribute ref="loc:locID"/>
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="Email" id="e1">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Subject" id="e1-1" minOccurs="0">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:string">
              <xsd:attribute ref="loc:locID" fixed="e1-1"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="SentDate" id="e1-2">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:dateTime">
              <xsd:attribute ref="loc:locID" use="required" fixed="e1-2"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="Sender" id="e1-3">
        <xsd:complexType>
          <xsd:complexContent>
            <xsd:extension base="mail:MailAddressType">
```

```
            〈xsd:attribute ref="loc:locID" use="required" fixed="e1-3"/〉
          〈/xsd:extension〉
        〈/xsd:complexContent〉
      〈/xsd:complexType〉
    〈/xsd:element〉
    〈xsd:element name="Receiver" id="e1-4"〉
      〈xsd:complexType〉
        〈xsd:complexContent〉
          〈xsd:extension base="mail:MailAddressType"〉
            〈xsd:attribute ref="loc:locID" use="required" fixed="e1-4"/〉
            〈xsd:attribute ref="loc:attrList" use="required" fixed="sc cc bc"/〉
            〈xsd:attribute name="sc" type="xsd:boolean" id="a01"/〉
            〈xsd:attribute name="cc" type="xsd:boolean" id="a02"/〉
            〈xsd:attribute name="bc" type="xsd:boolean" id="a03"/〉
          〈/xsd:extension〉
        〈/xsd:complexContent〉
      〈/xsd:complexType〉
    〈/xsd:element〉
    〈xsd:element name="ReplyTo" id="e1-5" minOccurs="0"〉
      〈xsd:complexType〉
        〈xsd:complexContent〉
          〈xsd:extension base="mail:MailAddressType"〉
            〈xsd:attribute ref="loc:locID" fixed="e1-5"/〉
          〈/xsd:extension〉
        〈/xsd:complexContent〉
      〈/xsd:complexType〉
    〈/xsd:element〉
    〈xsd:element name="Body" id="e1-6" minOccurs="0"〉
      〈xsd:complexType〉
        〈xsd:complexContent〉
          〈xsd:extension base="xsd:anyType"〉
            〈xsd:attribute ref="loc:locID" use="required" fixed="e1-6"/〉
          〈/xsd:extension〉
        〈/xsd:complexContent〉
      〈/xsd:complexType〉
    〈/xsd:element〉
  〈/xsd:sequence〉
  〈xsd:attribute ref="loc:locID" use="required" fixed="e1"/〉
〈/xsd:complexType〉
〈/xsd:element〉
〈/xsd:schema〉
```

英文实例(Email. xml):

```
〈? xml version="1. 0" encoding="UTF-8"?〉
〈mail:Email loc:locID="e1" xmlns:mail="http://www. cnegov. com. cn/schemas/email/en"
xmlns:loc="http://www. cnegov. com. cn/schemas/loc" xmlns:xsi="http://www. w3. org/
2001/XMLSchema-instance" xsi:schemaLocation="http://www. cnegov. com. cn/schemas/
email/en
enSchema. xsd"〉
  〈mail:Subject loc:locID="e1-1"〉Care Yourself〈/mail:Subject〉
  〈mail:SentDate loc:locID="e1-2"〉1999-05-31T13:20:00. 000-05:00〈/mail:SentDate〉
  〈mail:Sender loc:locID="e1-3"〉
    〈mail:Address loc:locID="e01"〉mickey@disney. com〈/mail:Address〉
  〈/mail:Sender〉
  〈mail:Receiver loc:locID="e1-4" loc:attrList="sc cc bc" mail:sc="true" mail:bc="true"〉
    〈mail:Address loc:locID="e01"〉donald@disney. com〈/mail:Address〉
  〈/mail:Receiver〉
  〈mail:ReplyTo loc:locID="e1-5"〉
    〈mail:Address loc:locID="e01"〉mickey@disney. com〈/mail:Address〉
  〈/mail:ReplyTo〉
  〈mail:Body loc:locID="e1-6"〉Dear Mr Donald: Never go to the roast duck restaurant. Wish you
be good health. 〈/mail:Body〉
〈/mail:Email〉
```

转换后的中文实例:

```
〈? xml version="1. 0" encoding="UTF-8"?〉
〈邮件:电子邮件 xmlns:邮件="http://www. cnegov. com. cn/schemas/email/cn"
xmlns:loc="http://www. cnegov. com. cn/schemas/loc" loc:locID="e1"
xmlns:xsi="http://www. w3. org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www. cnegov. com. cn/schemas/email/cn ../Chinese/cnSchema.
xsd"〉
  〈邮件:主题 loc:locID="e1-1"〉Care Yourself〈/邮件:主题〉
  〈邮件:发件日期 loc:locID="e1-2"〉1999-05-31T13:20:00. 000-05:00〈/邮件:发件日期〉
  〈邮件:发件人 loc:locID="e1-3"〉
    〈邮件:邮件地址 loc:locID="e01"〉mickey@disney. com〈/邮件:邮件地址〉
  〈/邮件:发件人〉
  〈邮件:收件人 loc:attrList="发送 抄送 密送" loc:locID="e1-4" 邮件:发送="true" 邮件:密送
="true"〉
    〈邮件:邮件地址 loc:locID="e01"〉donald@disney. com〈/邮件:邮件地址〉
  〈/邮件:收件人〉
  〈邮件:回复地址 loc:locID="e1-5"〉
    〈邮件:邮件地址 loc:locID="e01"〉mickey@disney. com〈/邮件:邮件地址〉
  〈/邮件:回复地址〉
  〈邮件:主体内容 loc:locID="e1-6"〉Dear Mr Donald: Never go to the roast duck restaurant.
Wish you be good health. 〈/邮件:主体内容〉
〈/邮件:电子邮件〉
```

转换式样单(Email. xslt)

```
〈? xml version="1. 0" encoding="UTF-8"?〉
〈! -- edited with XMLSPY v2004 rel. 2 U (http://www. xmlspy. com) by yang (biti) --〉
〈xsl:stylesheet version="1. 0" xmlns:xsl="http://www. w3. org/1999/XSL/Transform" xmlns:
loc="http://www. cnegov. com. cn/schemas/loc" xmlns:邮件="http://www. cnegov. com. cn/
schemas/email/cn" xmlns:xsi="http://www. w3. org/2001/XMLSchema-instance"〉
  〈xsl:output method="xml" version="1. 0" encoding="UTF-8" indent="yes"/〉
  〈xsl:template match="/"〉
    〈xsl:apply-templates/〉
  〈/xsl:template〉
  〈xsl:template match="node()"〉
    〈xsl:choose〉
      〈xsl:when test="@loc:locID = 'e1'"〉
        〈xsl:element name="邮件:电子邮件"
          namespace="http://www. cnegov. com. cn/schemas/email/cn"〉
          〈xsl:attribute name="loc:locID"〉e1〈/xsl:attribute〉
          〈xsl:attribute name="xsi:schemaLocation"
          namespace="http://www. w3. org/2001/XMLSchema-instance"〉http://www. cnegov.
com. cn/schemas/email/cn . . /Chinese/cnSchema. xsd〈/xsl:attribute〉
          〈xsl:apply-templates/〉
        〈/xsl:element〉
      〈/xsl:when〉
      〈xsl:when test="@loc:locID = 'e1-1'"〉
        〈xsl:element name="邮件:主题"〉
          〈xsl:for-each select="@ * "〉
            〈xsl:call-template name="nsAttr"/〉
          〈/xsl:for-each〉
          〈xsl:apply-templates/〉
        〈/xsl:element〉
      〈/xsl:when〉
      〈xsl:when test="@loc:locID = 'e1-2'"〉
        〈xsl:element name="邮件:发件日期"〉
          〈xsl:for-each select="@ * "〉
            〈xsl:call-template name="nsAttr"/〉
          〈/xsl:for-each〉
          〈xsl:apply-templates/〉
        〈/xsl:element〉
      〈/xsl:when〉
      〈xsl:when test="@loc:locID = 'e1-3'"〉
        〈xsl:element name="邮件:发件人"〉
          〈xsl:for-each select="@ * "〉
            〈xsl:call-template name="nsAttr"/〉
```

```
            </xsl:for-each>
            <xsl:apply-templates/>
          </xsl:element>
        </xsl:when>
        <xsl:when test="@loc:locID = 'e1-4'">
          <xsl:element name="邮件:收件人">
            <xsl:variable name="ens" select="namespace-uri(.)"/>
            <xsl:for-each select="@*">
              <xsl:choose>
                <xsl:when test="local-name()='attrList'and namespace-uri()='http://www.
cnegov.com.cn/schemas/loc'">
                  <xsl:attribute name="loc:attrList" namespace="http://www.cnegov.com.cn/
schemas/loc">发送 抄送 密送</xsl:attribute>
                </xsl:when>
                <xsl:otherwise>
                  <xsl:call-template name="Attr">
                    <xsl:with-param name="AttrStr" select="concat(//@*[local-name() =
'attrList' and namespace-uri()='http://www.cnegov.com.cn/schemas/loc'], ' ')"/>
                    <xsl:with-param name="pos" select="1"/>
                  </xsl:call-template>
                </xsl:otherwise>
              </xsl:choose>
            </xsl:for-each>
            <xsl:apply-templates/>
          </xsl:element>
        </xsl:when>
        <xsl:when test="@loc:locID = 'e01'">
          <xsl:element name="邮件:邮件地址">
            <xsl:for-each select="@*">
              <xsl:call-template name="nsAttr"/>
            </xsl:for-each>
            <xsl:apply-templates/>
          </xsl:element>
        </xsl:when>
        <xsl:when test="@loc:locID = 'e1-5' ">
          <xsl:element name="邮件:回复地址">
            <xsl:for-each select="@*">
              <xsl:call-template name="nsAttr"/>
            </xsl:for-each>
            <xsl:apply-templates/>
          </xsl:element>
        </xsl:when>
```

```
    〈xsl:when test="@loc:locID = 'e1-6' "〉
      〈xsl:element name="邮件:主体内容"〉
        〈xsl:for-each select="@ * "〉
          〈xsl:call-template name="nsAttr"/〉
        〈/xsl:for-each〉
        〈xsl:apply-templates/〉
      〈/xsl:element〉
    〈/xsl:when〉
    〈xsl:otherwise〉
      〈xsl:copy-of select="."/〉
    〈/xsl:otherwise〉
  〈/xsl:choose〉
〈/xsl:template〉
〈xsl:template name="Attr"〉
  〈xsl:param name="AttrStr"/〉
  〈xsl:param name="pos"/〉
  〈xsl:choose〉
    〈xsl:when test="local-name()=substring-before( $ AttrStr,' ')"〉
      〈xsl:choose〉
        〈xsl:when test=" $ pos = '1' "〉
          〈xsl:attribute name="邮件:发送"〉〈xsl:value-of select="."/〉〈/xsl:attribute〉
        〈/xsl:when〉
        〈xsl:when test=" $ pos = '2' "〉
          〈xsl:attribute name="邮件:抄送"〉〈xsl:value-of select="."/〉〈/xsl:attribute〉
        〈/xsl:when〉
        〈xsl:when test=" $ pos = '3' "〉
          〈xsl:attribute name="邮件:密送"〉〈xsl:value-of select="."/〉〈/xsl:attribute〉
        〈/xsl:when〉
      〈/xsl:choose〉
    〈/xsl:when〉
    〈xsl:otherwise〉
      〈xsl:choose〉
        〈xsl:when test="substring-after( $ AttrStr,' ')"〉
          〈xsl:call-template name="Attr"〉
            〈xsl:with-param name="AttrStr" select="substring-after( $ AttrStr,' ' )"/〉
            〈xsl:with-param name="pos" select=" $ pos + 1"/〉
          〈/xsl:call-template〉
        〈/xsl:when〉
        〈xsl:otherwise〉
          〈xsl:call-template name="nsAttr"/〉
        〈/xsl:otherwise〉
      〈/xsl:choose〉
```

```
      〈/xsl:otherwise〉
    〈/xsl:choose〉
  〈/xsl:template〉
  〈xsl:template name="nsAttr"〉
    〈xsl:choose〉
      〈xsl:when test="namespace-uri(.)='http://www.cnegov.com.cn/schemas/loc'"〉
        〈xsl:attribute name="loc:{local-name(.)}"〉〈xsl:value-of select="."/〉〈/xsl:attribute〉
      〈/xsl:when〉
      〈xsl:otherwise〉
        〈xsl:attribute name="{local-name(.)}" namespace="{namespace-uri(.)}"〉〈xsl:value-
of select="."/〉〈/xsl:attribute〉
      〈/xsl:otherwise〉
    〈/xsl:choose〉
  〈/xsl:template〉
〈/xsl:stylesheet〉
```

A.2 电子商务领域船运订单

本章描述了一个完整的有关船运订单的 XML 应用实例。

船运订单的 Schema 文件如下：

```
〈? xml version="1.0" encoding="GB 2312"?〉
〈xsd:schema xmlns:="http://www.cnegov.com.cn/schemas/sample/shiporder" xmlns:xsd="
http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.cnegov.com.cn/sche-
mas/sample/shiporder" elementFormDefault="qualified" attributeFormDefault="unqualified"〉
  〈xsd:annotation〉
    〈xsd:documentation xml:lang="en"〉 International purchase order schema for XML design
guideline Lanague is English. 〈/xsd:documentation〉
  〈/xsd:annotation〉
  〈xsd:element name="ShipOrder"〉
    〈xsd:complexType〉
      〈xsd:sequence〉
        〈xsd:element name="OrderPerson" type="xsd:string"/〉
        〈xsd:element name="ShipTo"〉
          〈xsd:complexType〉
            〈xsd:sequence〉
              〈xsd:element name="Name" type="xsd:string"/〉
              〈xsd:element name="Address" type="xsd:string"/〉
              〈xsd:element name="City" type="xsd:string"/〉
              〈xsd:element name="Country" type="xsd:string"/〉
            〈/xsd:sequence〉
          〈/xsd:complexType〉
        〈/xsd:element〉
```

```
        〈xsd:element name="Item" maxOccurs="unbounded"〉
          〈xsd:complexType〉
            〈xsd:sequence〉
              〈xsd:element name="Title" type="xsd:string"/〉
              〈xsd:element name="Note" type="xsd:string" minOccurs="0"/〉
              〈xsd:element name="Quantity" type="xsd:positiveInteger"/〉
              〈xsd:element name="Price" type="xsd:decimal"/〉
            〈/xsd:sequence〉
          〈/xsd:complexType〉
        〈/xsd:element〉
      〈/xsd:sequence〉
      〈xsd:attribute name="orderId" type="xsd:string" use="required"/〉
    〈/xsd:complexType〉
  〈/xsd:element〉
〈/xsd:schema〉
```

以下是一份符合该船运订单 Schema 的 XML 实例文档：

```
〈? xml version="1.0" encoding="GB 2312"?〉
〈ShipOrder orderId="889923" xmlns:="http://www.cnegov.com.cn/schemas/sample" xmlns:
xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=" http://www.
cnegov.com.cn/schemas/ShipOrder.xsd"〉
  〈OrderPerson〉John Smith〈/OrderPerson〉
  〈ShipTo〉
    〈Name〉Ola Nordmann〈/Name〉
    〈Address〉Langgt 23〈/Address〉
    〈City〉4000 Stavanger〈/City〉
    〈Country〉Norway〈/Country〉
  〈/ShipTo〉
  〈Item〉
    〈Title〉Empire Burlesque〈/Title〉
    〈Note〉Special Edition〈/Note〉
    〈Quantity〉1〈/Quantity〉
    〈Price〉10.90〈/Price〉
  〈/Item〉
  〈Item〉
    〈Title〉Hide your heart〈/Title〉
    〈Quantity〉1〈/Quantity〉
    〈Price〉9.90〈/Price〉
  〈/Item〉
〈/ShipOrder〉
```

A.3 软构件入库管理

本章描述一个构件管理系统的实际 XML 使用的例子。构件管理是通过构件提供者填写所有相关的构件信息(构件入库单),然后以 XML 文件的形式存入 Native XML 数据库,用户可以通过标准的 XML 查询方式来搜索所需的构件。

以下是构件入库单的 Schema 文件:

```
〈? xml version="1.0" encoding="GB 2312"?〉
〈xsd:schema xmlns:="http://www.cnegov.com.cn/schemas/CBSEsample" xmlns:xsd=" http://www.w3.org/2001/XMLSchema" targetNamespace=" http://www.cnegov.com.cn/schemas/CBSEsample" elementFormDefault="qualified" attributeFormDefault="unqualified"〉
  〈xsd:annotation〉
    〈xsd:documentation xml:lang="en"〉Component Management System File:Component Check in file Lanague is English. This Schema file is used for the Component management system.〈/xsd:documentation〉
  〈/xsd:annotation〉
  〈xsd:complexType〉
    〈xsd:sequence〉
      〈xsd:element name="Component" minOccurs="0" maxOccurs="unbounded"〉
        〈xsd:complexType〉
          〈xsd:sequence〉
            〈xsd:element name="Introduction"〉
              〈xsd:annotation〉
                〈xsd:documentation〉构件入库单的简介应提供整个文档的概述。它应包括此测试评估摘要的目的、范围、定义、首字母缩写词、缩略语、参考资料和概述。〈/xsd:documentation〉
              〈/xsd:annotation〉
              〈xsd:complexType〉
                〈xsd:sequence〉
                  〈xsd:element name="Purpose" type="xsd:string"〉
                    〈xsd:annotation〉
                      〈xsd:documentation〉阐明此构件入库单的目的〈/xsd:documentation〉
                    〈/xsd:annotation〉
                  〈/xsd:element〉
                  〈xsd:element name="Scope" type="xsd:string"〉
                    〈xsd:annotation〉
                      〈xsd:documentation〉简要说明此构件入库单的范围:它的相关项目,以及受到此文档影响的任何其他事物。〈/xsd:documentation〉
                    〈/xsd:annotation〉
                  〈/xsd:element〉
                  〈xsd:element name="Abbreviation" type="xsd:string"〉
                    〈xsd:annotation〉
                      〈xsd:documentation〉本小节应提供正确理解此构件入库单所需的全部术语、首字母缩写词和缩略语的定义〈/xsd:documentation〉
```

```
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="Refrence" type="xsd:string">
                    <xsd:annotation>
                        <xsd:documentation>本小节应完整列出此构件入库单中其他部分所引用的任何文档。每个文档应标有标题、报告号(如果适用)、日期和出版单位。列出可从中获取这些参考资料的来源。这些信息可以通过引用附录或其他文档来提供。</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="Summary" type="xsd:string">
                    <xsd:annotation>
                        <xsd:documentation>本小节应说明此构件入库单中其他部分所包含的内容,并解释此文档的组织方式</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="Roles" type="xsd:string">
                    <xsd:annotation>
                        <xsd:documentation>此小节应说明文档中所涉及到的角色以及相应责任</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Information">
        <xsd:annotation>
            <xsd:documentation>列举各个构件的入库信息,每个构件使用一个条目</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Name" type="xsd:string"/>
                <xsd:element name="Version" type="xsd:string"/>
                <xsd:element name="ComCategory">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="Function" type="xsd:string"/>
                            <xsd:element name="Bussiness">
                                <xsd:complexType>
                                    <xsd:sequence>
                                        <xsd:element name="CategoryMaxName" type="xsd:string"/>
                                        <xsd:element name="CategoryMidName" type="xsd:string"/>
```

```
                        <xsd:element name="CategoryMinName" type="xsd:string"/>
                      </xsd:sequence>
                    </xsd:complexType>
                  </xsd:element>
                  <xsd:element name="Technic">
                    <xsd:complexType>
                      <xsd:sequence>
                        <xsd:element name="TechnicPlatform" type="xsd:string"/>
                        <xsd:element name="DevPlatform" type="xsd:string"/>
                        <xsd:element name="RuntimePlatform" type="xsd:string"/>
                      </xsd:sequence>
                    </xsd:complexType>
                  </xsd:element>
                  <xsd:element name="CommonInfo">
                    <xsd:complexType>
                      <xsd:sequence>
                        <xsd:element name="VersionInfo" type="xsd:string"/>
                        <xsd:element name="ReleaseDate" type="xsd:string"/>
                        <xsd:element name="Manufacturer" type="xsd:string"/>
                      </xsd:sequence>
                    </xsd:complexType>
                  </xsd:element>
                  <xsd:element name="Userdefined" type="xsd:string"/>
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="About" type="xsd:string"/>
            <xsd:element name="SeeAlso" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" use="required"/>
  </xsd:complexType>
 </xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

以下是该构件入库单的 XML 实例，该实例是有关一个劳动和社会保障领域中职业介绍的求职者管理构件的入库单。

```
〈? xml version="1.0" encoding="GB2312"?〉
〈CheckIn xmlns:="http://www.wondersgroup.com.cn/schemas/CBSE"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=" http://www.wondersgroup.com.cn/schemas/CBSE "〉
  〈Component id="001"〉
    〈Introduction〉
      〈Purpose〉该文档为"求职者管理"构件进入构件库提供了入库的必要说明。〈/Purpose〉
      〈Scope〉求职者管理构件的有关信息，包括功能接口、类别、发行等〈/Scope〉
      〈Abbreviation〉参见项目词汇表〈/Abbreviation〉
      〈Refrence〉略〈/Refrence〉
      〈Summary〉略〈/Summary〉
      〈Roles〉略〈/Roles〉
    〈/Introduction〉
    〈Information〉
      〈Name〉求职者管理〈/Name〉
      〈Version〉 1.0a 〈/Version〉
      〈ComCategory〉
        〈Function〉为求职者提供基本信息的维护，包括新增求职者的个人基本信息的登记、查询(包括按主键查询、扩展查询)、修改、删除、归档等。〈/Function〉
        〈Bussiness〉
          〈CategoryMaxName〉劳动和社会保障〈/CategoryMaxName〉
          〈CategoryMidName〉劳动就业〈/CategoryMidName〉
          〈CategoryMinName〉职业介绍〈/CategoryMinName〉
        〈/Bussiness〉
        〈Technic〉
          〈TechnicPlatform〉 IBM Txseries 〈/TechnicPlatform〉
          〈DevPlatform〉 Visual Studio 6.0 IBM CICS Client for windows 2000 〈/DevPlatform〉
          〈RuntimePlatform〉 Windows 2000 〈/RuntimePlatform〉
        〈/Technic〉
        〈CommonInfo〉
          〈VersionInfo〉 1.0a〈/VersionInfo〉
          〈ReleaseDate〉 2003/5/1〈/ReleaseDate〉
          〈Manufacturer〉万达信息〈/Manufacturer〉
        〈/CommonInfo〉
        〈Userdefined〉略〈/Userdefined〉
      〈/ComCategory〉
      〈About〉入库时间:2002/12/30;入库状态:测试;〈/About〉
      〈SeeAlso〉"求职者管理"构件设计模型，"求职者管理"构件设计文档，"求职者管理"构件测试用例〈/SeeAlso〉
    〈/Information〉
  〈/Component〉
〈/CheckIn〉
```

附 录 B
（资料性附录）
指定机构的有关信息

指定机构的有关信息如下：

单位：中国电子技术标准化研究所(全国信息技术标准化技术委员会秘书处)；

电话：010-84029573、010-84029795；

传真：010-64007681；

地址：北京市安定门东大街 1 号；

邮箱：北京 1101 信箱；

邮编：100007

网址：www. cesi. ac. cn

参 考 文 献

[1] IANA-CHARSETS 字符集的正式名称
[2] IANA-LANGCODES 语种标记注册
[3] IETF RFC 2141:1997 URN 句法
[4] IETF RFC 2279:1998 UTF-8,GB 13000 的一种编码
[5] IETF RFC 2376:1998 XML 媒体类型
[6] IETF RFC 2396:1998 统一资源标识符(URI):通用语法
[7] IETF RFC 2781:2000 UTF-16,GB 13000 的一种编码
[8] Unified Modeling Language (UML), Version 1.5, 统一建模语言,
http://www.omg.org/technology/documents/formal/uml.htm
[9] The Unicode Standard, Version 4.0, Unicode 标准, Addison-Wesley Developers Press, 2003, ISBN 0-321-18578-1
[10] W3C REC-xml11-20040204 Extensible Markup Language (XML) 1.1.
http://www.w3.org/TR/2004/REC-xml11-20040204/
[11] W3C REC-xmlschema-0-20010502 XML Schema 第 0 部分:简介
http://www.w3.org/TR/xmlschema-0/
[12] W3C REC-xmlschema-1-20010502 XML Schema 第 1 部分:结构
http://www.w3.org/TR/xmlschema-1/
[13] W3C REC-xmlschema-2-20010502 XML Schema 第 2 部分:数据类型
http://www.w3.org/TR/xmlschema-2/
[14] W3C REC-xml-names-19990114 XML 命名空间
http://www.w3.org/TR/REC-xml-names/
[15] W3C REC-xslt-19991116 可扩展式样语言转换 1.0. http://www.w3.org/TR/xslt

GB/Z 21025-2007

*
书号:155066·1-30020
定价: 34.00 元