

# 教 育 信 息 化 技 术 标 准

CELTS-4.2

## 基于规则的 XML 绑定技术

Specification for Rule-Based XML Binding Techniques

CELTS-4.2 CD1.6

(送审稿)

(本稿完成日期：2002 年 8 月 28 日)

---

教育部教育信息化技术标准委员会 发布

# 目 次

基于规则的 XML 绑定技术 .....	4
1 概述 .....	4
1.1 范围 .....	4
1.2 目的 .....	4
2 规范性引用文件 .....	4
3 定义 .....	4
3.1 通过正式引用集成的定义 .....	4
3.2 集合（数据类型，值） .....	4
3.3 绑定 .....	5
3.4 编码 .....	5
3.5 条件数据元素 .....	5
3.6 消费（数据） .....	5
3.7 数据实例 .....	7
3.8 数据对象 .....	7
3.9 数据集 .....	7
3.10 数据结构 .....	7
3.11 扩展数据元素 .....	8
3.12 生成（数据） .....	8
3.13 实现行为 .....	8
3.14 实现定义的行为/值 .....	8
3.15 实现值 .....	8
3.16 解释（数据） .....	8
3.17 本地特定行为 .....	9
3.18 生存期（数据元素） .....	9
3.19 必须数据元素 .....	9
3.20 漫游（访问，系统） .....	9
3.21 约束性（数据元素） .....	9
3.22 过时数据元素 .....	10
3.23 可选数据元素 .....	10
3.24 产生（数据） .....	10
3.25 信息库 .....	10
3.26 保留数据元素 .....	10
3.27 未定义的行为/值 .....	10
3.28 未指定的行为/值 .....	11
3.29 缩写 .....	11
4 XML 编码绑定模板 .....	11
4.1 生成和产生 XML .....	11
4.2 消费和解释数据 .....	15
4.3 基本数据类型的表示 .....	16

4.4 字符表示的编码 .....	19
4.5 对例外和扩展的处理 .....	19

# 基于规则的 XML 绑定技术

## 1 概述

### 1.1 范围

本规范描述了数据模型基于规则的 XML 编码绑定的技术。本规范的措词为相应标准的陈述提供了一种范例，因而可以集成到相应的标准中，用于支持 XML 编码绑定。

### 1.2 目的

在基于规则的绑定方法可行的情况下，本规范为常用的绑定提供通用的标准用词。由于本规范使用基于规则的方法，所以它适用于很多标准。

## 2 规范性引用文件

- IEEE 1484.14.2/D1: “学习技术的指导草案——基于规则的 XML 绑定技术”
- CELTS-2: “术语规范”
- RFC 822
- IETF RFC 2068: “超文本传输协议”
- W3C XML: “扩展标记语言”
- ISO/IEC 11404 (1996): “语言无关的数据类型”
- ANSI X3.30 (1998): “信息交换中日期和时间的表示”
- ANSI X3.42(1990): “信息交换中字符串的表示”
- ANSI X.3.285 (1998): “数据表示的元数据模型”
- ISO/IEC 11179
- ISO/IEC 8601
- 

## 3 定义

以下术语的定义是和“CELTS-2 术语规范”相一致的。

### 3.1 通过正式引用集成的定义

下面的术语和它们的定义是通过正式引用集成进来的：

- IEEE 1484.14.2/D1: “学习技术的指导草案——基于规则的 XML 绑定技术”
- CELTS-2: “术语规范”

### 3.2 集合（数据类型，值）

从根本上来讲，集合的数据类型和值是由组成元素的数据类型和值组成的。该数据类型或值是通过把组成元素的数据类型或值通过一定的算法流程加以组合得到的。组成元素的值可以通过特定的操作加以访问。集合的属性和它的组成元素的属性无关。

例 1：一个数组集合包含相同类型的组成元素。对集合的操作通过索引值（一个数字）

来获取单个的组成元素。

```
my_array:
    array (0..9) of (integer), // 整数数组
my_array(4) // 访问第四个元素
```

例 2: 一个记录的集合包含独立的组成元素, 每个组成元素有自己的类型和标识。对集合的操作通过元素名 (一个标识符) 来访问单个组成元素。

```
A: record
(
    B: integer,
    C: void,
    D: characterstring(iso-10646-1),
),
A.B // 访问标号为 B 的元素
```

注: 该定义来自 ISO/IEC 11404。

### 3.3 绑定

从一个框架或规范到另一个框架或规范的应用或映射。

### 3.4 编码

- (1) 在信息交换中, 信息的规范化或结构化的表示。
- (2) 在某一结构中表示信息的过程。

### 3.5 条件数据元素

在某些上下文中, 如果某些条件被满足, 那么在数据结构的实例中, 某个元素被定义和要求。

一个数据元素的“条件”属性是一个约束属性。

另见: 扩展数据元素, 必须数据元素, 约束性 (数据元素), 可选数据元素。

### 3.6 消费 (数据)

读数据并进行如下处理: 找到词法或编码的边界。

其它形式有: 消费数据, 数据消费者, 数据消费。

另见: 解释 (数据), 产生 (数据)。

注: 数据在解释前被消费。

例 1: 在下列字符流中:

```
<R>
  <A>123.45</A>
  <B>PQR</B>
  <C X="Y">Z</C>
</R>
<R>
```

```

<D>JKL</D>
<E>
  <F>XXX</F>
  <G>YYY</G>
</E>
</R>

```

一个数据消费者能够识别:

- 有两个记录，都用标号“R”标识。
- 第一个“R”记录包含三个记录，分别用标号“A”，“B”，“C”标识。
- 第二个“R”记录包含两个记录，分别用标号“D”，“E”标识。

但是，数据消费者:

- 不知道标号的意义: <B>. . .</B>有什么意义?
- 不能确定标号的合法性: “<C>”可以有属性“X”吗?
- 不能确定记录内容的合法性: 在记录“A”中，“123.45”是合法值吗?
- 可能限制消费的深度: “R”只用往下分析一层就发现标号“D”和“E”，但对“E”的内容只是进行有限的分析(即发现对等的标号)，因此标号“F”和“G”不会被分析和发现。因此，一个数据消费者对一个信息结构可能只进行部分的理解。

例 2: 下面是一个 API 的例子，它表明了扩展数据在实现时被间接使用的情况下(这种实现仍是严格一致的)，数据消费和数据解释之间的区别。

```

//// 这个例子包含两个文件: 头文件"std_data.h"和一个包含该头文件的严
//// 格一致的应用程序
////////////////////////////////////
//// 下面是所包含的头文件 "std_data.h"

struct std_data
{
    int std_element_1;    // 必需元素
    void *std_element_2; // 可选元素
    int ext_element_3;   // 扩展元素.
};

////////////////////////////////////
//// 严格一致的应用程序

// 引用标准头文件 (内容在上面列出)
#include "std_data.h"

struct std_data x; // 声明 x 是 std_data.

my_code()

```

```

{
    struct std_data y,z; // 声明 "y" 和 "z".

    // 严格一致的代码, 但是扩展元素"ext_element_3"被复制。
    memcpy(&y,&x,sizeof x);

    // 把字符串赋给"std_element_2".
    // 把长度赋给 "std_element_1".
    y.std_element_2 = "hello there";
    y.std_element_1 = strlen(y.std_element_2);

    //严格一致的代码, 但是扩展元素"ext_element_3"被复制。
    memcpy(&z,&y,sizeof y);
}
////////////////////////////////////

```

该例子是严格一致的, 因为实现时只解释和生成标准集合中的元素 (即 std\_element\_1 和 std\_element\_2)。memcpy (在内存中复制对象) 操作在这个假设的 API 绑定中相当于消费和产生操作。而直接的元素访问 (如 y.std\_element\_1) 在该假设的 API 绑定中是解释和生成操作。

### 3.7 数据实例

把数据集用某种绑定加以表示。

### 3.8 数据对象

在访问数据的概念模型中数据处理的单元。

注1: 一个数据对象可以是一个数据元素或实现时定义的对象。严格一致的实现只能使用和访问是数据元素的数据对象。

注2: 一个数据对象进一步地进行语义上的定义和限制, 就是一个数据结构。一个数据结构的实例是一个数据集。一个数据集, 在某种绑定中进一步定义, 限制和表示就是一个数据实例。

另见: 数据元素, 数据实例, 数据集, 数据结构

### 3.9 数据集

数据结构的第二条定义。

注: 数据集是独立于绑定的 (和绑定无关)。

### 3.10 数据结构

零个或多个数据元素集合而成的数据类型。

零个或多个数据元素集合而成的实例。

注 1: 在不同的上下文中, 一个数据结构可能被认为是一个完整的不可分割的单元。在这种情况下, 一个数据结构是一个具有更高层数据结构的数据元素。

注 2: 术语“集合”是在 ISO/IEC 11404 中定义的。

例如：一条记录，一个集合，一个序列，一个列表，一个数组。

### 3.11 扩展数据元素

在某些上下文中，数据结构的某个数据元素在标准之外定义，但可能在数据结构的某个实例中使用。

数据元素的“扩展”属性是一种约束属性。

数据元素的“扩展”属性是一个一致性程度的特征。（例如：严格一致性实现与一致性实现）

例如：必须扩展数据元素，可选扩展数据元素，条件扩展数据元素。

另见：条件数据元素，必须数据元素，约束性（数据元素），可选数据元素。

### 3.12 生成（数据）

把数据的含义用某种适合数据交换的形式加以表示。

例如：根据某种概念模型把数据结构序列化，在此过程中不必把数据用某种具体的编码方式加以表示。

另见：解释（数据），产生（数据）。

### 3.13 实现行为

外部的观察，外貌或行动。

另见：实现定义的行为，实现值，未定义的行为，未指定的行为。

### 3.14 实现定义的行为/值

未指定的行为或未指定的值，这些行为或值需要在实现时做出选择。

另见：实现行为，未定义的行为/值，未指定的行为/值。

例如：某种编码所允许的的最大值，用字节表示。

### 3.15 实现值

和实现相关的值。

另见：实现行为，实现定义的行为/值，未定义的行为/值，未指定的行为/值。

### 3.16 解释（数据）

处理数据，直到发现它在所需程度上的含义。

其它形式：解释数据，数据解释器，数据解释。

另见：生成（数据），消费（数据）。

注：数据在解释前被消费。

例 1：在下列字符流中：

```
<R>
  <A>123.45</A>
  <B>PQR</B>
  <C X="Y">Z</C>
</R>
```

```
<R>
  <D>JKL</D>
  <E>
    <F>XXX</F>
    <G>YYY</G>
  </E>
</R>
```

一个数据消费者能够识别：

- 有两个记录，都用标号“R”标识。
- 第一个“R”记录包含三个记录，分别用标号“A”，“B”，“C”标识。
- 第二个“R”记录包含两个记录，分别用标号“D”，“E”标识。

因为只识别了这些标号，所以只有它们可以进行数据解释。假设标号“E”代表一个扩展数据元素，那么一个数据解释器可能只能识别标准标号“A”，“B”，“C”和“D”。

由于 1) “消费”和“解释”的分离，2) 某种特定的标准绑定（如 XML），一个应用程序可能只解释标准化的特征 A, B, C 和 D。

正如上面所说的，一个把数据消费和数据解释组合起来，但仅仅解释标准化数据元素的应用程序可以被称为严格一致的数据阅读器。

### 3.17 本地特定行为

依赖于民族，文化，语言，制度等地方习俗的行为，在具体实现时加以说明。

### 3.18 生存期（数据元素）

数据元素的一个属性，用于表示数据元素同标准的过去，现在或未来版本的关系。

另见：约束性（数据元素），过时数据元素，保留数据元素。

注：生存期属性和约束属性无关。

例 1：一个过时数据元素也许包含在某规范的过去版本中，但不会包含在该规范的将来版本中。

例 2：一个保留数据元素可能并没有包含在某规范的过去版本中，但可能会包含在该规范的将来版本中。

### 3.19 必须数据元素

在某些上下文中，在数据结构实例中定义和要求的的数据元素。“必须”属性是一个数据元素的一个约束属性。

另见：条件数据元素，扩展数据元素，约束性（数据元素），可选数据元素。

### 3.20 漫游（访问，系统）

在不同的通讯时期和地理位置呈现出持续性的服务。

不间断地断开网络，该网络用于子系统或相关系统的通讯。

注：也称为“时断时续的连接”或“时断时续的漫游”。

### 3.21 约束性（数据元素）

对数据元素的要求，用于确定一个数据结构的合法性。

另见：生存期（数据元素），条件数据元素，扩展数据元素，必须数据元素，可选数据元素。

例：一个数据元素 X，有四个元素：A 和 B 是必须的，C 是可选的，D 是条件的（如果 B 的值为真，则 D 存在）。下面是合法和不合法的数据结构的例子：

```
( A=123 ) // 不合法，缺少必须数据元素 B
( A=123, B=false ) // 合法
( A=123, B=true ) // 不合法，缺少可选数据元素 D
( A=123, B=true, D=17 ) // 合法
( A=123, B=false, D=17 ) // 合法
( A=123, B=nil, C=345 ) // 合法
```

### 3.22 过时数据元素

在某些上下文中，某一元素在一个数据结构中定义，但它不能在数据结构的实例中使用。“过时”属性是一个数据元素的生存期属性。

另见：生存期（数据元素），保留数据元素。

注：不提倡使用过时数据元素，因为关于它们的规定也许会在标准的将来版本中被删除。

### 3.23 可选数据元素

在某些上下文中，一个数据结构的某个元素被定义，是允许存在的，但并不一定要求在数据结构的实例中出现。“可选”属性是一个数据元素的约束属性。

另见：条件数据元素，扩展数据元素，必须数据元素，约束性（数据元素）。

### 3.24 产生（数据）

处理数据直到词法或编码边界被定义，然后写下结果数据。

其它形式：产生数据，数据产生器，数据产生。

另见：生成（数据），消费（数据）。

注：数据在产生前被生成。

### 3.25 信息库

数据集的集合以及对信息进行存储，索引，查找和提取的数据访问方法。

### 3.26 保留数据元素

在某些上下文中，在数据结构中没有定义，也不能在数据结构的实例中使用的数据元素。“保留”属性是数据元素的生存期属性。

另见：生存期（数据元素），过时数据元素。

### 3.27 未定义的行为/值

某个标准没有做任何要求的实现行为和实现值

另见：实现行为，实现值，实现定义的行为/值，未指定的行为/值。

例 1：可能的未定义的行为包括，但不局限于：  
完全忽略某种情形。

不可预知的结果。

例 2：可能的未定义的值包括无穷，空等。

### 3.28 未指定的行为/值

某个标准提供两个或更多的可能性，但不做进一步的要求的实现行为或实现值。

另见：实现行为，实现值，实现定义的行为/值，未定义的行为/值。

例 1：一个应用程序对算法的选择，该算法用于创建对象的标识符。

例 2：过程调用参数入栈的次序。

### 3.29 缩写

- API：应用程序接口。
- HTTP：超文本传输协议
- ICS：实现一致性声明
- IETF：Internet Engineering Task Force
- L10N：本地化
- LID：语言无关数据类型，即 ISO/IEC 11404
- MDAS API：元数据访问服务 API
- RFC：Request for Comments
- SPM：最低峰值
- W3C：World Wide Web Consortium
- XML：扩展标记语言

## 4 XML 编码绑定模板

注1：下面的用词也许会集成在某标准的标准化（或条件标准化）部分。“XXX标准”应该用集成下面的用词的标准的名称来代替。

注2：实现的变体在“扩展技术指导规范”中定义。

如果 XXX 标准的应用程序在其实现的一致性陈述中包含“XXX 标准[实现变体]的 XML 编码绑定”，那么该应用程序应该遵循本部分的要求。

例：“XXX 标准严格遵循数据实例的 XML 编码绑定”，“XXX 标准遵循数据阅读器的 XML 编码绑定”。

### 4.1 生成和产生 XML

下面的规则描述了 XXX 标准的数据元素到 XML 记录的转化。

- 规则 1：对于 ISO/IEC 11404 符号中的任何数据元素，除规则 2 中说明的以外，把标识符映射到 XML 标号。匹配的 XML 标号界定了数据元素的值的边界。XML 标号的嵌套代表了数据元素的结构。对于数组和顺序集合，（1）取集合标识符的名称为 XML 标号的名称，来表示该集合，（2）用重复的 XML 标号来表示其中的单个数据元素，这些 XML 标号通过集合的标识符减去后缀“\_list”或“\_bucket”产生，它不是元素的索引值。

- 规则 2: 把所有的多语言字符串这种数据类型映射到:
  - 规则 2A: 多语言字符串数据类型的“语种”元素在父 XML 元素中设置 xml:lang 属性。
  - 规则 2B: “字符串”元素设置父标识的内容。
- 规则 3: 把下面的 XML 标号 (通配符号)

**xxx 标准\_\***

转化为下列 XML 标号 (通配符号):

**CELT5\_xxx 标准\_\***

产生的所有数据都应该是结构良好的 XML。

### 依据

下面是对 XXX 标准的数据类型进行以上三个规则的特定转化的依据。

注: 这个 XML 绑定 (XXX 标准 → XML) 需要 3 个转换规则。其他标准和不同的 XML 绑定可能需要更多的或更少的规则或不同的规则。

### 规则 1 的依据

规则 1 是从 ISO/IEC 11404 的数据类型到 XML 标号进行转换的主要规则。下面的例子说明这种转换:

```

A: record
(
  B: integer,
  C: record
  (
    D: integer,
    E: characterstring(iso-10646-1),
  ),
  F_list: array (0..limit) of (integer),
  G: sample_mlstring_list_type,
)

```

第一句,“对于 ISO/IEC 11404 符号中的任何数据元素,把标识符映射到 XML 标号”,转换标识符,例,“x:” ⇒ “<x>”。

第二句,“匹配的 XML 标号界定了数据元素的值的边界”,要求 (1) 标号是匹配的,并且 (2) 数据元素的值在标号之间,例,“x : 17” ⇒ “<x>17</x>”。

第三句,“XML 标号的嵌套代表了数据元素的结构”,要求集合 (记录,数组,序列/列表) 所隐含的嵌套和 XML 标号的嵌套有相似的结构。利用上面对记录 A 的定义,元素 B, C, D 和 E 采取下面的嵌套:

```

<A>
  <B>...</B>
  <C>
    <D>...</D>
    <E>...</E>
  </C>
  ...
</A>

```

第四句，“对于数组和序列，数据元素用重复的 XML 标号来表示，这些标号基于集合的标识符，不是元素的索引值”，要求数组和序列（列表）用具有相同名称的多个标号来表示——这是典型的 XML 风格。例如，数据元素 F 可以如下表示：

```

<!--对 F_list 的正确的 XML 绑定 -->
<A>
  ...
  <F_list>
    <F>...</F>
    <F>...</F>
    <F>...</F>
  </F_list>
  ...
</A>

```

但不能如下表示：

```

<!--对 F_list 的不正确的 XML 绑定-->
<A>
  ...
  <F_list>
    <0>...</0>
    <1>...</1>
    <2>...</2>
  </F_list>
  ...
</A>

```

### 规则 2 的依据

XXX 标准的记录可能使用几种特殊的数据类型，例如多语言数据类型用于描述数据类型为字符串的特定的数据元素，这些数据元素要在多语言和多文化的环境中加以表示——通常称为国际化（I18N）和本地化（L10N）特征。下面是一个多语言数据类型的一个例子。在这个例子中，数据元素 `sample_mlstring_type` 代表一个对：一个本地化的字符串和一个本地的规范（L10N 映射）。数据类型 `sample_mlstring_array_type` 表示这些字符串对的一个数

组。数组 `example_remarks` 包含三个元素，每个元素是一个字符串对。可以如下假设，一个应用程序根据运行该程序的不同国家（地方）从 `example_remarks` 中选择恰当的字符串。下面是该例子的类型定义和值定义：

```
type sample_mlstring_type =
record
(
  L10N_string: characterstring(iso-10646-1), //字符串元
素
  L10N_locale: string_type, //语言元素
),

type sample_mlstring_array_type =
array (0..limit) of (sample_mlstring_type),

value example_remarks:
sample_mlstring_array_type =
(
(
  L10N_string: "abc abc abc",
  L10N_locale: "en-US",
),
(
  L10N_string: "def def def",
  L10N_map: "fr-CA",
),
(
  L10N_string: "ghi ghi ghi",
  L10N_map: "de-DE",
),
),
```

根据规则 1 和规则 2，把这些数据元素转化为 XML：

```
<example_remarks xml:lang="en-US">abc abc
abc</example_remarks>
<example_remarks xml:lang="fr-CA">def def
def</example_remarks>
<example_remarks xml:lang="de-DE">ghi ghi
ghi</example_remarks>
```

规则 2 包括“vcard”映射，它把 XXX 标准的个人信息转换到“vcard”结构。

注：vcard 标准用于规定如何表示个人或团体的联系方法和相关信息。参见：

<http://www.imc.org/pdi/>。

### 规则 3 的依据

该规则用于重写标号，以此使用某些名称空间说明符（namespace conventions）。该规则可以通过选择一个不同的名称空间说明符（前缀）来指定 XML 的名称空间。

实现时要求保证转化的结果是结构良好的 XML。

## 4.2 消费和解释数据

下面的规则描述了 XML 记录到 XXX 标准数据元素的转换。

所有进行消费的数据都应该是结构良好的 XML。

- 规则 1: 把下面的 XML 标号（通配符号）：

**CELTSC\_XXX 标准\_\***

转化为下列 XML 标号（通配符号）：

**XXX 标准\_\***

- 规则 2: 作如下的转化：
  - 规则 2A: 用 XML 元素的 `xml:lang` 属性设置相应多语言字符串数据类型的数据元素的“语种”元素。
  - 规则 2B: 用标号元素的内容设置相应多语言字符串数据类型的数据元素的“字符串”元素。
- 规则 3: 对于每一个 XML 标号，如果它和 XXX 标准的数据元素的标识符有关系，那么对它的相应的开闭标号进行匹配。对于每一个 XML 标号，除了规则 2 中所说明的，把该标号映射到相应的标识符。XML 标号的嵌套结构表示了数据元素的嵌套关系，即 4.1 中规则 1 的逆操作。每个标号元素的内容被转化成相应的数据元素的值。

### 依据

#### 规则 1 的依据

在处理前，要保证实现进行消费和解释的内容是结构良好的 XML。

该规则在需要的时候去掉了 XML 名称空间的前缀和后缀。在这里并没有使用 XML 的名称空间，而是使用了一个名称空间的前缀（“CELTSC\_”）来减少名称空间冲突的可能性。

#### 规则 2 的依据

该规则做了一个从 `xml:lang` 属性到多语言字符串数据类型的逆映射。该规则只是转化已知的多语言字符串类型的数据元素，因为所有其它的 XML `xml:lang` 属性跟相关标准中的多语言字符串类型的数据元素没有对应关系。

在下一个草案中，规则 2 将包含“vCard”的映射。它描述 XXX 标准的个人信息同“vCard”结构之间的转化。

### 规则 3 的依据

该规则处理 XML 标记及其内容到数据元素的主要转变方法。

第一句，“对于每一个 XML 标号，如果它和 XXX 标准的数据元素的标识符有关系，那么它的相应的开闭标号要进行匹配”，（1）忽略掉所有该标准不认识的标识符，并且（2）正确地匹配它们。

第二句，“对于每一个 XML 标号，除了规则 2 中所说明的，把该标号映射到相应的标识符”，创建和数据元素的联系，但并没有给数据元素赋值。

第三句，“XML 标号的嵌套结构表示了数据元素的嵌套关系，即 4.1 中规则 1 的逆操作”，保证了 XML 标号的内部结构在相关标准所要求的范围内同数据元素的内部结构一致。

第四句，“每个标号元素的内容被转化成相应的数据元素的值”，把 XML 标号内的内容转化为数据元素的值，即填充数据元素。

## 4.3 基本数据类型的表示

下面各部分描述了数据元素的值同字符表示之间的转化关系，用于 XML 绑定中的信息交换。

### 4.3.1 字符和字符串

Character 类型的数据元素应该根据 XML 规范加以表示。

注1：特殊字符，如“&”“<”“>”“;”，需要一定的转化方法，而且可能需要无损的转化。

注2：一些编码，如ISO-8859-1和UTF-8，允许诸如“©”（版权符号）等字符的直接编码。其它的编码，如ASCII，需要扩展编码，如“&#169;”，来表示这些符号。

### 4.3.2 整数

整型的数据元素根据“ISO/IEC 9899:1999, C 编程语言, 6.4.4.1, 整型常量”来表示。包括“U”, “L”, “LL” 后缀及它们的小写形式，也可能包含可选的前缀符号，加号或减号，但不能同时有。

例：

```
0      // 零
23     // 二十三
0x17   // 16 进制的 23
027    // 8 进制的 23
-34    // 负的三十四
+34    // 正的三十四
```

### 4.3.3 实数

实型的数据元素：

- 如果是整数，参照 4.3.2。
- 如果不是整数或不用整数表示，应该根据“ISO/IEC 9899:1999, C 编程语言, 6.4.4.2, 实型常量”加以表示，包括“F”和“L”后缀以及它们的小写形式，也可能包括一个可选的前导符号，加号或减号，但不能同时有。

例：

0 // 零  
0.0 // 零  
130.0 // 一百三十  
1.3E2 //一百三十  
+1.3E2 //一百三十

#### 4.3.4 日期和时间值

time 类型的数据元素应该根据“ISO 8601, 数据元素和交换格式——信息交换——日期和时间的表示”来加以表示。

注1: ISO 8601表示的时间和日期范围为阳历的0001年的1月1号到9999年的12月31号。很明显它有一些局限性。

ISO 8601 如下运用:

- 只使用基本格式。例如:“19990102”和“030405”是合法的,但“1999-01-02”和“03:04:05”是不合法的。依据:对扩展形式的处理会增加错误的可能性和降低互操作性。扩展格式附加的词汇元素可能会和其它词汇,内嵌的或环绕的信息处理环境相干扰或冲突。
- 十进制的小数使用完全停止符(“.”),即句号。例如:“199901020304.1”表示03:04:06 AM,1999年1月2号。注:ISO 8601允许使用逗号(“,”)和完全停止符,并规定尽量使用逗号。依据:逗号会和其它词汇,内嵌的或环绕的信息处理环境相干扰或冲突。

对于时间点,ISO 8601 如下运用:

- 日期只使用日历日期格式。注:日历日期用年,月,日加以表示。ISO 8601 所允许的其它格式在本标准中禁止使用。包括序数日期和日历周日号等。依据:对其它日期格式的处理会增加错误的可能性和降低互操作性。
- 日期和时间应该使用 ISO 8601 的完全表示法,不能使用“T”这个时间指示符。例如:“19990102030405”表示03:04:05 AM,1999年1月2号。依据:“T”指示符被省略,因为它并不是必要的。ISO 8601 允许在没有歧义的情况下省略该指示符。本标准中,歧义通过如下方法避免(1)只允许 ISO 8601 的基本格式。(2)要求对缺少的日期和时间成分进行标识。
- 使用下划符号(“\_”)来表示缺少的成分。一个下划符号表示一个缺少的数字。如:“\_\_0102”表示当年的1月2号。注:缺少组成成分的日期会引起很大的互操作问题,如“2000年问题”。建议在实现时避免产生缺少组成成分的数据,但不可能解决所有的情况。依据:ISO 8601 使用连字符代替缺少的成分,但连字符会和其它词汇,内嵌的或环绕的信息处理环境相干扰或冲突。ISO 8601 使用连字符来表示缺少的成分(如“-0102”),本标准用一个下划符号代替一个数字(如“\_\_0102”),这种方法能减少信息处理的错误量。
- 时间应该用本地时间或格林尼治时间(UTC)加以表示。用本地时间表示的时间应该没有后缀并且不指示本地时间和UTC之间的时间差别。用UTC表示的时间应该使用“Z”后缀,正如ISO 8601规定的。例如:“19990102030405”表示纽约的本地

时间，“19990102030905Z”表示相应的UTC时间。依据：除UTC外，使用时区的信息会增加数据解释的负担，从而更容易出错。禁止使用时间差（例如，“19990102030405+0700”）排除了加号（+）和减号（“-”）的使用，这两个符号会和其它词汇，内嵌的或环绕的信息处理环境相干扰或冲突。

对于时间段，ISO 8601 如下运用：

- 时间段用前缀符“P”开始。如“P2Y”表示两年。注：对于时间段来讲并不是所有的日期和时间成分都是必须的。
- 时间段使用大小写敏感的指示符，“Y”表示年，“M”表示月，“D”表示日，“W”表示周，“h”表示小时，“m”表示分，“s”表示秒。例：“P2Y10M25h2m5s”表示2年，10个月，25小时，2分5秒。对于时间段，日期和时间成分可以超过相应单位的最大值。如，月可以超过12，小时可以超过24，分可以超过60。依据：大小写敏感的指示符能简化时间和日期信息的处理过程。

注2：ISO 8601中描述的字符只规定了字符的名称，没有规定它们的编码。这一点在“ISO 8601 4.4 本规范使用的字符”中给予强调：“在本国际规范中规定的表示法使用数字，字母符号和在ISO 646中规定的特殊符号....注2：字符的编码在本标准之外。”

#### 4.3.5 空类型

空类型应该没有表示和编码。

例：下列记录

```
A: record
(
  B: integer,
  C: void,
  D: characterstring(iso-10646-1),
)
```

用XML表示为：

```
<!--正确的 XML 表示 -->
<A>
  <B>17</B>
  <D>hello</D>
</A>
```

不应该是：

```
<!--不正确的 XML 表示 -->
<A>
  <B>17</B>
```

```
<C></C>  
<D>hello</D>  
</A>
```

#### 4.4 字符表示的编码

如何把字符表示映射到 8 位字节的编码方法由“XML 编码技术”加以规定。

遵循 XML 编码绑定的 XXX 标准的数据实例应该用以下方法编码：ASCII，ISO/IEC 8859-1，ISO/IEC 10646-1 UTF-8，或 ISO/IEC 10646-1 UTF-16。

遵循 XML 编码绑定的 XXX 标准的应用应该支持以下编码：ASCII，ISO/IEC 8859-1，ISO/IEC 10646-1 UTF-8，和 ISO/IEC 10646-1 UTF-16。

#### 4.5 对例外和扩展的处理

##### 4.5.1 实现定义的行为

下面是实现定义的行为，在相关标准其它地方描述的除外。

下面是在 XML 编码的消费和产生中的实现定义的行为：

- 用 XML 编码的，能够被成功处理的，严格一致的 XXX 标准的数据实例的最大长度，用字节表示。
- XML 记录的最大嵌套深度。
- 未规定时区的时间类型的数据元素的时区信息。

##### 4.5.2 未指定的行为

下面是未指定的行为，在相关标准其它地方描述的除外。

下面是在 XML 编码的生成和解释中的未指定的行为。

- 数据元素处理的次序。

下面是在 XML 编码的产生和消费中的未指定的行为。

- 除数据元素值中的空格和 XML 规范中要求的空格以外，附加空格的使用。

##### 4.5.3 未定义的行为

下面是未定义的行为，在相关标准其它地方描述的除外。

下面是在 XML 编码的产生和消费中的未定义的行为。

- 和扩展数据元素对应的 XML 标号的使用。
- 和保留数据元素对应的 XML 标号的使用。
- 使用在相关 XML 编码绑定中未定义的 XML 标号或属性。
- 使用相关标准规定的字符集以外的字符。