



标准

ECMA-357

2nd 版本 / 2005 年 12 月

**ECMAScript for XML（E4X）**

**规范**

Ecma 国际 罗纳街 114 号 CH-1204 日内瓦 T/F: +41 22 849 6000/01 www.ecma-international.org 网站

.

**介紹**

2002 年 6 月 13 日， 由 BEA 系统公司牵头的一组公司提出了一套编程语言扩展， 旨在为 ECMAScript（ECMA-262）添加原生 XML 支持。这些编程语 言扩展旨在提供一种简单、熟悉且通用的 XML 编程模型，通过利用全球最大的开发者社区之一现有的技能和知识， 降低 XML 的学习曲线。这种 XML 编程模型的优势包括降低代码复杂性、缩短修订周期、加快上市时间、减少 XML 占用空间的需求以及降低代码与 XML 数据之间的耦合度。

ECMAScript 小组（Ecma TC39-TG1）一致同意该提案， 并成立了一个子小组来制定一套通用、跨平台、厂商中立的编程语言扩展的语法和语义标准， 这套扩展被称为“ECMAScript for XML（E4X）”。该标准的开发工作于 2002 年 8 月 8 日启动。此标准是作为对 ECMAScript 第 3 版的扩展而开发的， 但也可应用于其他版本的 ECMAScript。

本标准为 ECMAScript语言添加了原生的 XML 数据类型， 扩展了用于操作 XML 数据的常见 ECMAScript运算符的语义， 并添加了一组用于常见 XML 操作（如搜索和筛选）的新运算符。它还增加了对 XML 文本、命名空间、限定名和其他机制的支持， 以方便 XML 处理。

本标准将被整合进未来版本的 ECMA-262（ECMAScript）。ECMAScript 小组正在为 ECMAScript 语言的未来版本进行重大改进， 包括使用 XML 模式 语言定义 XML 类型的机制以及对类的支持。

|  |  |
| --- | --- |
| 以下人员为本规范做出了贡献：  约翰·施奈德，BEA/AgileDelta（主编） 罗克·于， 微软（副 主编） 杰夫·戴尔，Macromedia（副主编） |  |
| 史蒂夫·亚当斯基， 美国在线/网景公司 帕特 里克·比尔德， 美国在线/网景公司 亚当·博 斯沃思，BEA 公司 史蒂夫·布兰德利，BEA 公司 维克拉姆·达内什瓦尔，微软公司 布伦 丹·艾奇 ，Mozilla 基金会 维拉·弗莱舍，  Macromedia 公司 内森尼尔·弗里塔斯，  palmOne 公司 沃尔德马尔·霍瓦特， 美国在 线/网景公司 伊桑·哈格，AgileDelta 公司 马 克·伊格拉，BEA 公司  大卫·雅各布斯， 麻省理工学 院林肯实验室 亚历克斯·赫 辛， | BEA 特里·卢卡斯，BEA  Milen Nankov, AgileDelta Brent Noorda,  Openwave Richard Rollman, AgileDelta Markus Scherer, IBM Werner Sharp, Macromedia Michael Shenfield, RIM Edwin Smith, Macromedia Dan  Suciu， 华盛顿大学Peter Torr，微软  埃里克·瓦西利克，BEA 公司  赫尔曼·文特尔，微软公司；韦恩 · 维克内尔，IBM 公司  罗杰·韦伯，BEA |

此 Ecma 标准已于 2005 年 12 月的大会上获得通过。



**目录**

[**1 范围 1**](#bookmark2)

[**2 一致性 1**](#bookmark3)

[**3 参考文献 1**](#bookmark4)

[3.1 规范性引用文件 1](#bookmark5)

[3.2 参考文献 1](#bookmark6)

[**4 定义 2**](#bookmark7)

[**5 动机 3**](#bookmark8)

[5.1 XML 处理的兴起 3](#bookmark9)

[5.2 当前的 XML 处理方法 3](#bookmark10)

[5.2.1 文档对象模型（DOM） 3](#bookmark11)

[5.2.2 可扩展样式表语言（XSLT） 3](#bookmark12)

[5.2.3 对象映射 3](#bookmark13)

[5.3 E4X 方法 4](#bookmark14)

[**6 设计原则 4**](#bookmark15)

[**7 符号约定 4**](#bookmark16)

[7.1 算法约定 5](#bookmark17)

[7.1.1 缩进风格 5](#bookmark18)

[7.1.2 属性访问 5](#bookmark19)

[7.1.3 迭代 6](#bookmark20)

[7.1.4 条件重复 7](#bookmark21)

[7.1.5 方法调用 7](#bookmark22)

[**8 词汇约定 8**](#bookmark23)

[8.1 上下文关键词 9](#bookmark24)

[8.2 标点符号使用者 10](#bookmark25)

[8.3 XML 初始化器输入元素 10](#bookmark26)

[**9 类型 12**](#bookmark27)

[9.1 XML 类型 12](#bookmark28)

[9.1.1 内部属性和方法 12](#bookmark29)

[9.2 XMLList 类型 22](#bookmark30)

[9.2.1 内部属性和方法 22](#bookmark31)

[9.3 属性名称类型 28](#bookmark32)

*- i-*

[9.3.1 内部属性 28](#bookmark33)

[9.4 任意名称类型 29](#bookmark34)

[**10 类型转换 29**](#bookmark35)

[10.1 ToString 29](#bookmark36)

[10.1.1 应用于 XML 类型的 ToString 方法 29](#bookmark37)

[10.1.2 应用于 XMLList 类型的 ToString 方法 30](#bookmark38)

[10.2 ToXMLString（输入参数， [祖先命名空间]， [缩进级别]） 30](#bookmark39)

[10.2.1 应用于 XML 类型的 ToXMLString 方法 31](#bookmark40)

[10.2.2 应用于 XMLList 类型的 ToXMLString 方法 33](#bookmark41)

[10.3 ToXML 33](#bookmark42)

[10.3.1 应用于字符串类型的 ToXML 34](#bookmark43)

[10.3.2 应用于 W3C XML 信息项的 ToXML 35](#bookmark44)

[10.4 ToXMLList 37](#bookmark45)

[10.4.1 应用于字符串类型的 ToXMLList 38](#bookmark46)

[10.5 属性名称 39](#bookmark47)

[10.5.1 应用于字符串类型的“ToAttributeName” 39](#bookmark48)

[10.6 ToXMLName 39](#bookmark49)

[10.6.1 应用于字符串类型的 ToXMLName 40](#bookmark50)

[**11 表达式 40**](#bookmark51)

[11.1 主要表达式 40](#bookmark52)

[11.1.1 属性标识符 41](#bookmark53)

[11.1.2 合格标识符 42](#bookmark54)

[11.1.3 通配符标识符 43](#bookmark55)

[11.1.4 XML 初始化器 43](#bookmark56)

[11.1.5 XMLList 初始化器 46](#bookmark57)

[11.2 左值表达式 47](#bookmark58)

[11.2.1 属性访问器 47](#bookmark59)

[11.2.2 函数调用 49](#bookmark60)

[11.2.3 XML 后代访问器 51](#bookmark61)

[11.2.4 XML 过滤谓词运算符 52](#bookmark62)

[11.3 一元运算符 53](#bookmark63)

[11.3.1 删除操作符 53](#bookmark64)

[11.3.2 typeof 操作符 54](#bookmark65)

[11.4 加法运算符 55](#bookmark66)

[11.4.1 加法运算符（+） 55](#bookmark67)

[11.5 相等运算符 56](#bookmark68)

[11.5.1 抽象相等比较算法 56](#bookmark69)

[11.6 赋值运算符 57](#bookmark70)

[11.6.1 XML 赋值运算符 57](#bookmark71)

[11.6.2 XMLList 赋值运算符 58](#bookmark72)

[11.6.3 复合赋值（op=） 59](#bookmark73)

*- ii-*

[**12 陈述 60**](#bookmark74)

[12.1 默认的 XML 命名空间声明 60](#bookmark75)

[12.1.1 获取默认命名空间（ ） 61](#bookmark76)

[12.2 for-in 语句 62](#bookmark77)

[12.3 “for-each-in”语句 63](#bookmark78)

[**13 原生 E4X对象 65**](#bookmark79)

[13.1 全局对象 65](#bookmark80)

[13.1.1 全局对象的内部属性 65](#bookmark81)

[13.1.2 全局对象的函数属性 65](#bookmark82)

[13.1.3 全局对象的构造属性 66](#bookmark83)

[13.2 命名空间对象 66](#bookmark84)

[13.2.1 作为函数调用的命名空间构造函数 66](#bookmark85)

[13.2.2 命名空间构造函数 67](#bookmark86)

[13.2.3 命名空间构造函数的属性 68](#bookmark87)

[13.2.4 命名空间原型对象的属性（内置方法） 68](#bookmark88)

[13.2.5 命名空间实例的属性 68](#bookmark89)

[13.3 QName 对象 69](#bookmark90)

[13.3.1 作为函数调用的 QName 构造函数 69](#bookmark91)

[13.3.2 QName 构造函数 69](#bookmark92)

[13.3.3 QName 构造函数的属性 70](#bookmark93)

[13.3.4 QName 原型对象的属性 71](#bookmark94)

[13.3.5 QName 实例的属性 71](#bookmark95)

[13.4 XML 对象 72](#bookmark96)

[13.4.1 作为函数调用的 XML 构造器 72](#bookmark97)

[13.4.2 XML 构造器 72](#bookmark98)

[13.4.3 XML 构造函数的属性 73](#bookmark99)

[13.4.4 XML 原型对象的属性（内置方法） 76](#bookmark100)

[13.4.5 XML 实例的属性 89](#bookmark101)

[13.5 XMLList 对象 89](#bookmark102)

[13.5.1 作为函数调用的 XMLList 构造函数 89](#bookmark103)

[13.5.2 XMLList 构造函数 89](#bookmark104)

[13.5.3 XMLList 构造函数的属性 90](#bookmark105)

[13.5.4 XMLList 原型对象的属性（内置方法） 90](#bookmark106)

[**14 错误 96**](#bookmark107)

[**附录 A（规范性附录） - 可选功能 97**](#bookmark108)

*- iii-*

*- iv-*

**1 范围**

本标准定义了 ECMAScript for XML（E4X） 的语法和语义， E4X 是一组编程语言扩展， 为 ECMAScript 增加 了对 XML 的原生支持。

**2 一致性**

符合 E4X 规范的实现应当提供并支持本规范中所描述的所有强制类型、值、对象、属性、 函数以及程序语法 和语义。

本标准的符合性实现应符合《ECMAScript 语言规范》（ISO/IEC 16262:2001）。

本标准的符合性实现应按照 Unicode 标准 2.1 版或更高版本以及 ISO/IEC 10646-1 标准（采用 UCS-2 或 UTF- 16 编码形式， 实现级别 3）来解释字符。如果未另行指定所采用的 ISO/IEC 10646-1 子集， 则假定为 BMP 子 集（集合 300）。如果未另行指定所采用的编码形式， 则假定为 UTF-16 编码形式。

符合 E4X 规范的实现可以提供本规范未描述的其他类型、值、对象、属性和函数。特别是， 符合 E4X 规范的 实现可以为本规范中描述的对象提供本规范未描述的属性以及这些属性的值。符合 E4X 规范的实现不得为 XML.prototype 和 XMLList.prototype 提供本规范未描述的方法。

**3 参考文献**

**3.1 规范性引用文件**

Document Object Model (DOM) Level 2 Specificions, W3C Recommendation, 13 November 2000. ECMA-

262, 1999, ECMAScript Language Specification -3 edition.

Extensible Markup Language 1.0 (Second Edition), W3C Recommendation 6 October 2000. Namespaces in XML, W3C Recommendation, 14 January 1999.

ISO/IEC 10646:2003, Information TTehnology -Universal Multiple-Octet Coded Character Set Inc. (1996), The Unicode Standard , Version 2.0. ISBN: 0-201-48345-9, Addison-Wesley Menlo Park, California.

(UCS). Unicode Publishing Co.,

TM

Unicode Inc. (1998), Unicode Technical Report #8: The Unicode Standard , Version 2.1.

Unicode Inc. (1998), Unicode Technical Report #15: Unicode Normalization Forms. XML Information Set, W3C Recommendation 24 October 2001.

XML Path Language (XPath) Version 1.0, W3C Recommendation 16 November 1999. XML Schema Part 1: Structures, W3C Recommendation, 2 May 2001.

XML Schema Part 2: Datatypes, W3C Recommendation, 2 May 2001.

**3.2 参考文献**

XSL 转换（XSLT） ， W3C 推荐标准， 1999 年 11 月 16 日。

*- 1-*

**4 定义**

就本 Ecma 标准而言， 以下定义适用：

**4.1 XML**

可扩展标记语言（XML）是由万维网联盟（W3C）认可的信息编码标准， 用于在万维网上发送、接 收和处理数据。XML 由一系列字符组成， 这些字符不仅包含实质性信息， 即字符数据， 还包含有关 字符数据结构和布局的元信息， 称为标记。

**4.2 标记**

XML 数据的两个基本组成部分之一（另一个是字符数据）。标记是一系列字符， 用于提供有关字符 数据结构或布局的信息。常见的标记形式有起始标签、结束标签、空元素标签、注释、CDATA 标记 定界符和处理指令。

**4.3 字符数据**

XML 数据的两个基本组成部分之一（另一个是标记）。字符数据是一系列代表实质性数据的字符， 这些数据由 XML 标记封装。字符数据被定义为任何不包含标记的字符序列。

**4.4 标签**

一种作为字符数据分隔符的单一标记实体。标签可以是起始标签、结束标签或空元素标签。起始标 签以小于号（<）开始， 以大于号（>）结束。结束标签以小于号和斜杠的组合（</）开始， 以大于 号（>）结束。空元素标签以小于号（<）开始， 以斜杠和大于号的组合（/>）结束。

**4.5 元素**

一种数据结构， 由两个标签（起始标签和结束标签）组成， 用于界定字符数据或嵌套元素。如果给 定元素既无字符数据也无嵌套元素， 则该元素可由一个空元素标签来定义。每个格式良好的 XML 文 档至少包含一个元素， 称为根元素或文档元素。

**4.6 属性**

一个可选的名称 - 值对， 由等号（=）分隔， 可出现在标签内部。属性可以存储有关元素的信息或原 本会作为字符数据存储的实际数据。

**4.7 命名空间**

一组元素和属性的标识符， 它们共同绑定到一个统一资源标识符（URI）， 这样在与不同命名空间中 同名的标识符一起使用时， 不会引起命名冲突。

**4.8 处理指令**

处理指令实体包含有供处理 XML 的应用程序使用的指令或信息。处理指令标签以小于号（<）和问 号（？ ） 的组合（<?）开始， 并以相同字符组合的逆序（？>）结束。

**4.9 类型**

一组数据值。

*- 2-*

**5 动机**

本节包含对 ECMAScript for XML 背后动机的非规范性概述。

**5.1 XML 处理的兴起**

开发用于创建、导航和操作 XML 数据的软件是每个开发人员工作的重要组成部分。开发人员被大量采用可扩 展标记语言（XML）编码的数据所淹没。 网页越来越多地使用 XML 词汇表进行编码， 包括 XHTML 和可缩放 矢量图形（SVG）。在移动设备上， 数据使用无线标记语言（WML）进行编码。Web 服务使用简单对象访问 协议（SOAP）进行交互， 并使用Web 服务描述语言（WSDL）进行描述。部署描述符、项目生成文件和配置 文件现在都采用 XML 编码， 更不用说为垂直行业设计的无数自定义 XML 词汇表了。XML 数据本身甚至使用 XML 模式和 XSL 样式表以 XML 的形式进行描述和处理。

**5.2 当前的 XML 处理方法**

当前的 XML 处理技术要求 ECMAScript程序员学习并掌握一系列复杂的新概念和编程技术。对于 ECMAScript 程序员来说，XML 编程模型常常显得繁重、复杂且陌生。本节简要概述了较为流行的 XML 处理技术。

**5.2.1 文档对象模型（DOM）**

处理 XML 最常见的方法之一是使用实现 W3C XML DOM（文档对象模型）所定义接口的软件包。XML DOM 使用通用树抽象来表示 XML 数据， 并提供基于树的 API 用于导航和操作数据（例如， getParentNode（）、

getChildNodes（） 、removeChild（） 等）。

这种访问和操作数据结构的方法与访问和操作原生 ECMAScript数据结构的方法大不相同。ECMAScript 程序员 必须学会编写树形导航算法， 而不是对象导航算法。此外， 他们还必须学习一个相对复杂的接口层次结构， 以 便与 XML DOM 进行交互。 由此产生的 XML DOM 代码通常比操作原生 ECMAScript数据结构的代码更难阅读、 编写和维护。它更冗长， 而且常常因冗长的树形导航逻辑而掩盖了开发者的意图。 因此，XML DOM 程序的开 发需要更多的时间、知识和资源。

**5.2.2 可扩展样式表语言（XSLT）**

XSLT 是一种用于将 XML 文档转换为其他 XML 文档的语言。与 XML DOM 类似， 它使用基于树的抽象来表示 XML 数据， 但还提供了一种称为 XPath 的表达式语言， 专门用于在树中导航。在此基础上， 它还添加了一种声 明式、基于规则的语言， 用于匹配输入文档的某些部分， 并相应地生成输出文档。

从这段描述中可以清楚地看出，XSLT 访问和操作数据结构的方法与访问和操作 ECMAScript数据结构的方法 完全不同。 因此， 对于 ECMAScript 程序员来说， 学习 XSLT 的曲线相当陡峭。 除了要学习新的数据模型，

ECMAScript 程序员还必须学习声明式编程模型、递归下降处理模型、新的表达式语言、新的 XML 语言语法以 及各种新的编程概念（模板、模式、优先级规则等）。这些差异也使得 XSLT 代码对于 ECMAScript程序员来 说更难阅读、编写和维护。此外， 无法使用熟悉的开发环境、调试器和测试工具来处理 XSLT。

**5.2.3 对象映射**

还有几种方法试图通过将 XML 数据映射到原生的 ECMAScript对象以及从这些对象映射回 XML 来导航和操作 XML 数据。其思路是将 XML 数据映射到一组 ECMAScript对象上， 直接操作这些对象， 然后再将其映射回 XML。这样，ECMAScript 程序员就可以利用他们对 ECMAScript对象的知识来操作 XML 数据。

*- 3-*

这是一个很棒的想法， 但遗憾的是， 它并不适用于广泛的 XML 处理任务。原生的 ECMAScript对象无法保留 原始 XML 数据的顺序， 而顺序对于 XML 来说至关重要。XML 开发人员不仅需要保留 XML 数据的顺序， 还 需要控制和操作 XML 数据的顺序。此外，XML 数据包含一些难以用 ECMAScript对象模型表示的内容， 例 如命名空间、属性、注释、处理指令和混合元素内容。

**5.3 E4X 方法**

ECMAScript for XML 旨在解决这些问题。E4X 通过原生支持 XML 数据扩展了 ECMAScript 对象模型。 它复 用了熟悉的 ECMAScript操作符来创建、导航和操作 XML， 这样任何使用过 ECMAScript 的人都能够几乎无 需额外知识就开始使用 XML。这些扩展包括原生的 XML 数据类型、XML 文本（即初始化器） 以及一组用于 常见 XML 操作（如搜索和筛选） 的小型新操作符。

与类似的 XSLT 或 DOM 应用程序相比， E4X 应用程序对于 ECMAScript开发人员来说更小且更直观。它们更 易于阅读、编写和维护， 需要更少的开发时间、技能和专门知识。最终结果是代码复杂性降低、修订周期缩 短以及互联网应用程序的上市时间缩短。此外， E4X 是一种更轻量级的技术， 能够支持各种移动应用程序。

**6 设计原则**

以下这些非规范性的设计原则用于指导 E4X 的开发， 并鼓励做出一致的设计决策。在此列出这些原则， 旨在 为 E4X 的设计思路提供见解， 并为有关 E4X 期望特性的讨论提供依据。

• **简单：**E4X 最重要的目标之一就是简化常见的编程任务。不应为了那些有趣或独特的但无法解决常见编程 问题的功能而牺牲简洁性。

• **一致：**E4X 的设计应保持内部一致性， 以便开发人员能够预测其行为。

• **熟悉：**用于操作 ECMAScript对象的常见运算符也应可用于操作 XML 数据。这些运算符的语义对于熟悉 ECMAScript 对象的开发人员来说不应感到意外。 已经熟悉 ECMAScript对象的开发人员应当能够以最少的 意外情况开始使用 XML 对象。

• **最小化：**在适当的情况下， E4X 定义了一些用于操作 XML 的新运算符， 这些运算符目前尚不能用于操作 ECMAScript 对象。这一组运算符应保持在最小限度， 以避免不必要的复杂性。E4X 的目标并非提供例如

XPath 的全部功能。

• **松散耦合：**在实际可行的范围内， E4X 操作符将使应用程序能够最大程度地减少对外部数据格式的依赖。 例如， E4X 应用程序应当能够提取嵌套在 XML 结构深处的值， 而无需指定数据的完整路径。 因此， 数据的 包含层次结构发生变化时， 应用程序无需进行修改。

• **互补性：**E4X 应当能与诸如 XPath、XSLT 和 XML 查询等用于处理 XML 的其他语言良好集成。例如， 在 需要更强的表达能力时， E4X 应当能够调用这些互补语言， 同时又不损害 E4X 语言本身的简洁性。

**7 符号约定**

本规范扩展了 ECMAScript第 3 版规范中使用的符号约定。特别是， 它扩展了算法符号， 以提高本规范的清晰 度、可读性和可维护性。新的算法约定在本节中进行了描述。

*- 4-*

**7.1 算法约定**

本节介绍**了**本规范为描述 ECMAScript**第** 3 版语义**所**使用**的**算法约定**之外新增的约定。这些约定并非 E4X 语言的 一部分。它们在本规范中用于描述 E4X 操作的语义。**

**7.1.1 缩进风格**

本规范通过定义一种算法缩进风格扩展了 ECMAScript第 3 版规范中使用的符号。新的算法缩进风格在本规范中用 于将相关的步骤集合分组。这种约定对于表达一组有条件执行或重复执行的步骤很有用。例如， 以下算法片段使 用缩进来描述一组有条件执行的步骤：

1. 如果 *resetParameters* 为**真** 令 *x* = **0**

b. 令 *y* = **0**

c. 令 *deltaX* 等于 **0.5**

**2. 其他情况**

a. 令 *deltaX*= *deltaX*+ *accelerationX*

在上述示例中， 如果步骤 1 中所表达的条件为**真， 则执行步骤 1.**a 至 1.c；否则， 执行步骤 2.a。 标准的编号形式用于标识步骤， 并区分由于分页可能不明显的缩进嵌套级别。

**7.1.2 属性访问**

本规范扩展了 ECMAScript第 3 版规范中使用的符号， 定义了三种属性访问约定。在本规范中， 当用于赋值操作的 左侧时， 这些属性访问约定用于修改指定对象的指定属性的值。在本规范的其他上下文中， 这些属性访问约定用 于指定根据属性名称从指定对象中检索指定属性的值。

有三种属性访问约定的形式， 其中两种用于访问普通属性， 一种用于访问内部属性。用于访问普通属性的第一种 约定形式采用以下表示法：

**对象. 属性名**

当在赋值操作的左侧使用此属性访问约定时， 它等同于调用对象的 *[[Put]]* 方法， 将包含与*propertyName* 相同参 数序列的字符串字面量以及赋值运算符右侧的值作为参数传递。例如， 以下算法片段：

**1. 令 *item.price* = "5.95**

等同于以下算法片段：

1. 调用 item 的 [[Put]] 方法， 传入参数 **“price”** 和 **“5.95”** 。

在其他上下文中使用此属性访问约定时， 其效果等同于调用对象的 [[Get]] 方法， 并将包含与*propertyName* 相同字 符序列的字符串字面量作为参数传递。例如， 以下算法片段：

**1. 令 *currentPrice* = *item.price***

等同于以下算法片段：

1. 令 *currentPrice* 为调用 item 的 [[Get]] 方法并传入参数 **“price”** 所得到的结果。 访问普通属性的第二种约定使用以下符号表示：

*- 5-*

当在赋值操作的左侧使用此属性访问约定时 ， 它等同于以对象作为 **this** 对象调用 Object [[Put]] 方法 ， 并将 propertyName 的 ToString（） 结果和赋值操作符右侧的值作为参数传递。例如， 以下算法片段：

1. 将 *item[1]* 设为 *item2* 的值

等同于以下算法片段：

1. 使用 *item* 作为 this 对象， 调用对象的 [[Put]] 方法， 并将参数设置为 ToString(1) 和 *item2* 。

在其他上下文中使用此属性访问约定时， 其效果等同于以对象作为**this** 对象， 并将参数 ToString（propertyName） 传 递给 *Object* [[Get]] 方法。例如， 以下算法片段：

1. 令 *item2* = *item[1]* 等同于以下算法片段：

1. 令 *item2* 为调用 Object 的 [[Get]] 方法的结果， 其中 **this** 对象为 *item*， 参数为 ToString(1) 。

这是一种方便且熟悉的表示法， 用于指定用作数组索引的数字属性名称。

访问内部属性名称（包括那些指向内部方法的名称）的约定使用以下符号表示：

当在赋值操作的左侧使用此属性访问约定时， 它等同于将指定对象的 [[内部属性名称]] 的值设置为赋值运算符右侧 的值。例如， 以下算法片段：

1. 令 *x* 的 *[[Class]]* 属性值为 **“element”**

等同于以下算法片段：

1. 令 *x* 的 [[Class]] 属性值为 **"element**

在其他上下文中使用时， 此属性访问约定等同于获取对象的 [[internalPropertyName]] 属性的值。例如， 以下算法片段：

1. 令 *class* = *x.[[Class]]*

等同于以下算法片段：

1. 令 *class* 为 *x* 的 [[Class]] 属性的值

**7.1.3 迭代**

本规范通过定义两种迭代约定扩展了用于描述 ECMAScript第 3 版的符号。这些迭代约定在本规范中用于表示对集合 中的每个项或指定范围内的每个整数执行一组步骤。

第一个迭代约定用于表示针对集合中的每个成员都应执行一次的步骤序列。它使用以下“**对于每个**”符号来表示：

对于集合步骤中的每个项目

**对于每个**记号， 这相当于将变量项绑定到集合中的每个成员， 并反复执行给定的步骤。在执行步骤之前，集合的值仅 计算一次， 并且在执行步骤期间不会改变。项绑定到集合成员的顺序由实现决定。当项已绑定到集合中的所有成员， 或者算法通过返回或抛出异常退出时， 重复结束。步骤可以在同一行中用逗号分隔后指定， 也可以在后续行中使用 第 7.1.1 节中描述的缩进风格指定。例如，

*- 6-*

1. 令 *total* = **0**

2. 对于 *groceryList* 中的每一种产品

a. 如果 *product.price* 大于 *maxPrice*， 则抛出异常。

*b.* 让 *total* 等于 *total* 加上 *product.price* 的值

在此示例中， 步骤 2.a 和 2.b 会针对集合 *groceryList* 中的每个成员重复执行一次， 或者直到在步骤 2.a 中抛出异常为止。在每次重复执 行这些步骤之前， 变量*product*都会绑定到 *groceryList*中不同成员的值。

本规范定义的第二个迭代约定用于表示针对指定整数范围内的每个整数重复执行的一系列步骤。它使用以下“**for**”符号表示： 对于变量从第一个到最后一个，每次步长为 *steps*

这种表示法等同于先计算*first*和 *last*， 它们将分别求值**为**整数 *i* 和*j*， 然后按照数值顺序依次将变量 *variable* 绑定到序列 i、*i* + 1……j 中的每个 成员， 并重复执行给定的步骤。在执行步骤之前，*first*和 *last* 的值仅计算一次， 在执行步骤期间不会改变。重复操作在变量 *variable* 已绑定 到该序列中的每个项或算法通过 return 语句退出或抛出异常时结束。如果 *i* 大于*j*， 则不执行步骤。步骤可以在同一行中用逗号分隔后指定， 也 可以在下一行中按照上述缩进风格指定。例如，

1. 对于 *i* 从 **0** 到*priceList* 长度减 *1*， 调用 ToString（priceList[i]） 。

在此示例中， 对于*priceList* 中的每个项目，都将按顺序调用一次 ToString 方法。

存在一种修改过的“for”表示法， 用于以逆序方式遍历整数范围。其表示形式如下：

对于变量从*first* 到 last 以 steps 为步长递减循环

修改后的 **for** 表示法的工作方式与上述描述完全相同， 只是变量 variable 会按数值降序绑定到序列 *i*， i - 1, ..*j* 的每个成员。如果 *i* 小于*j*， 则 不会执行这些步骤。

**7.1.4 条件重复**

本规范通过定义一种表示一组步骤有条件重复的约定， 扩展了 ECMAScript第 3 版规范中使用的符号。该约定通过以下符号来定义：

**当（表达式）为真时执行步骤**

`while` 表示法等同于计算表达式， 该表达式的计算结果要么为**真**， 要么为**假**。如果结果为**真**， 则执行给定的步骤， 并重复此过程， 直到表达 式的计算结果为**假**， 或者算法通过`return` 语句退出或抛出异常。这些步骤可以在同一行中用逗号分隔指定， 也可以在下一行中使用上述缩进 风格指定。例如，

1. 令 *log2* = **0**

2. 当（n > **1）**时：a. 令 *n* = *n* / **2** ； b. 令 *log2* = *log2* + **1** 。

在此示例中， 步骤 2.a 和 2.b 会一直重复执行， 直到表达式 *n* > 1 的计算结果为**假为止。**

**7.1.5 方法调用**

本规范通过定义一种方法调用约定来扩展 ECMAScript第 3 版规范中使用的符号。此规范中使用该方法调用约定来调用给定对象的某个方法， 并传入一组给定的参数， 然后返回结果。该约定通过以下符号来定义：

对象. 方法名(参数

*- 7-*

其中，*arguments* 是一个由零个或多个值以逗号分隔的列表。方法调用表示法等同于创建一个新的引用 *r*， 其基对象设置为 *object*，属性名称设 置为包含与*methodName* 相同字符序列的字符串字面量，创建一个内部列表 *list*， 其中包含 *arguments* 中的值，调用 CallMethod 操作符（第 11.

2.2.1 节）， 将 *r* 和 *list*作为参数传递， 并返回结果。例如， 以下算法片段：

1. 令 *sub*= *s* 的子字符串从索引 *2* 到 4 。

相当于以下算法片段：

1. 令 *r* 为一个新的引用， 其基对象为 *s*，属性名为**“substring”**

2. 令 *list*为一个内部列表， 其中包含值 2 和 5 。

3. 令 *sub* = CallMethod(r, *list*

**8 词汇惯例**

本节介绍 E4X 为 ECMAScript 增添的词法约定。

E4X 对现有的词法语法产生式“*InputElementRegExp*” 和“*Punctuators*” 进行了修改 。 它还引入了 目标符号“*InputElementXMLTag*”和 “*InputElementXMLContent*”， 用于描述 Unicode 字符序列如何转换为 XML 初始化器的各个部分。

`*InputElementDiv*` 符号用于允许除法（/）、除法赋值（/=）、小于（<）、小于等于（<=）、左移（<<）或左移赋值（<<=）运算符的语法上下 文中。`*InputElementXMLTag*` 用于允许 XML 标签的文本内容的语法上下文中。`*InputElementXMLContent*` 用于允许 XML 元素的文本内容的语 法上下文中。`*InputElementRegExp*` 符号用于所有其他语法上下文中。

通过添加生产规则 *InputElementRegExp* :: *XMLMarkup* 以及扩展使用现有的生产规则 *InputElementRegExp* :: *Punctuator* :: < ， 可以识别出 XML 初 始化器的起始位置。

为了更好地理解这些目标符号的适用情况，请看以下示例：

从词法语法返回的输入元素， 以及用于此示例的目标符号和产生式如下：

*- 8-*

|  |  |  |
| --- | --- | --- |
| **输入元素** | **目标** | **产品** |
| 订单 | *InputElementRegExp* | 标识符令牌 |
| = | 输入元素容器 | 标点符号添加器 |
| < | *InputElementRegExp* | 标点符号添加器 |
| { | 输入元素*XML*标签 | **{** |
| x | *InputElementRegExp* | 标识符令牌 |
| } | 输入元素容器 | 标点符号添加器 |
| > | *InputElementXMLTag* | *XML*标签分隔符 |
| { | 输入元素*XML*内容 | **{** |
| 物品 | *InputElementRegExp* | 标识符令牌 |
| } | 输入元素容器 | 标点符号添加器 |
| </ | 输入元素*XML*内容 | **</** |
| { | 输入元素*XML*标签 | **{** |
| x | *InputElementRegExp* | 标识符令牌 |
| } | 输入元素容器 | 标点符号添加器 |
| > | *InputElementXMLTag* | *XML*标签分隔符 |
| ; | *InputElementRegExp* | 标记符：：标点符号 |

**语法**

E4X 扩展了由 ECMAScript 定义的 *InputElementRegExp* 目标符号， 添加了以下生成规则：

E4X 通过添加以下目标符号来扩展 ECMAScript：

**8.1 上下文关键词**

E4X 通过添加一组上下文关键字来扩展 ECMAScript。在不允许使用标识符的特定上下文中使用时， 这些上下文关键字具有特定的含义。然而， 它们与 ECMAScript第 3 版的关键字不同， 因为它们也可以用作标识符。E4X 没有向 ECMAScript添加任何其他关键字。

*- 9-*

**语法**

E4X 通过替换标识符生成规则并添加上下文关键字生成规则来扩展 ECMAScript， 具体如下：

标识符

标识符名称，**但不是**保留字**或**上下文关键字 上下文关键字

上下文关键词

**每个 XML**

**命名空间**

**8.2 标点符号**

E4X 扩展了 ECMAScript 定义的标点符号列表， 添加了后代（ ..）输入元素以支持 XML 后代访问器（第 11.2.3 节）、 属性（@）输入元素以支持 XML 属性查找（第 11.1.1 节） 以及名称限定符（： ：）输入元素以支持限定名称查找 （第 11.1.2 节）。

**语法**

E4X 通过以下产生式扩展了 *Punctuator* 非终结符： 标点符号生成器

**..**

**@**

**::**

**8.3 XML 初始化器输入元素**

目标符号 *InputElementXMLTag* 和 *InputElementXMLContent* 描述了如何将 Unicode 字符转换为输入元素， 这些输入元 素用于描述 XML 初始化器的部分内容。这些输入元素由第 11.1.4 节和第 11.1.5 节中描述的语法语法进行处理。

词法语法允许出现可能无法构成有效 XML 初始化器的字符。而句法语法中所描述的语法和语义则确保最终的初始化 器是格式良好的 XML。

与字符串字面值不同， 在 XML 初始化器中， 反斜杠（\）不会被视为转义序列的起始符。相反， 应使用 XML 1.0 规 范中指定的 XML 实体引用对字符进行转义。例如， 可以使用实体引用 **&apos；** 表示单引号（'）， **&quot；** 表示双 引号（"）， **&lt；** 表示小于号（<）。

左花括号（{）和右花括号（}）用于界定可能嵌入在标签或元素内容中的表达式， 以便动态计算 XML 初始化器的某 些部分。花括号可以在属性值、CDATA、处理指令（PI）或 XML 注释中以字面形式出现。在所有其他情况下， 应 使用字符引用 **&#x7B；** 表示左花括号（{）， 使用字符引用 **&#x7D；** 表示右花括号（}）。

**语法**

*XML* 标记：*XML* 注释 *XML* 文 本 数 据 *XML* 处理指令

*XML* 标签字符

源字符但无**嵌入的***XML* 标签和标点符号

左花括号 **{ 或**单引号 **' 或**双引号 **" 或**正斜杠 **/ 或***XML* 空白字符

*XML* 空白字符：：*<*空格>

*- 10-*

*- 11-*

*XMLTagPunctuator***（XML 标签标点符号）** ：**其中之一 = > />**

**9 类型**

E4X 通过添加两种新的基本数据类型来扩展 ECMAScript， 用于表示 XML 对象和 XML 对象列表。未来的版本还将 提供使用 XML 模式为特定 XML 词汇表派生用户定义类型的功能。

**9.1 XML 类型**

XML 类型是具有名称、一组 XML 属性、一组作用域内的命名空间和父节点的有序属性集合。XML 对象的每个属性 都有一个唯一的数字属性名称 *P*， 使得 ToString（ToUint32(P)） 等于 *P*， 并且其值为 XML 类型， 表示一个子节点。 XML 对象的名称是一个 QName 对象或为 **null**。每个 XML 属性都是 XML 类型的一个实例。每个命名空间都是一个 Namespace 对象。父节点的值为 XML 类型或为 **null**。使用非数字属性名称将方法与 XML 对象相关联。

每个 XML 类型的值都表示一个 XML 元素、属性、注释、处理指令或文本节点。 内部的 [[Class]] 属性会根据具体情 况分别设置为“element”（元素）、“attribute”（属性）、“comment”（注释）、“processing-instruction”（处 理指令）或“text”（文本）。每个表示 XML 属性、注释、处理指令（PI）或文本节点的 XML 对象都没有用户可见 的属性， 并在从 Object类型逻辑继承的 [[Value]] 属性中存储一个字符串值， 该值表示相关属性、注释、PI 或文本节 点的值。

E4X 有意模糊了单个 XML 对象与仅包含该对象的 XMLList之间的区别。为此， 所有适用于 XMLList对象的操作也 适用于 XML 对象。扩展 E4X 的实现应保留此约束条件。

注意：上述描述的内部 XML 数据模型将 XML 子节点表示为具有数字属性名称的属性。这些属性的数字名称表示给定子节点在其 父节点中的序号位置。这些属性的值是具有关联名称（例如元素名称）的 XML 对象。E4X 定义了 XML 的 [[Get]] 和 [[Put]] 操作 符（如下）， 它们基于属性值的名称而非其内部数字属性名称来访问 XML 对象的属性。

**9.1.1 内部属性和方法**

内部属性和方法并非 E4X 语言的一部分。本规范仅出于说明目的而定义它们。E4X 的实现应表现得如同其以本规范 所述的方式生成和操作内部属性一样。本规范沿用了 ECMAScript第 3 版规范中内部属性的表示法， 即内部属性的名 称用双方括号 [[ ]] 括起来。 当算法使用对象的内部属性而该对象未实现所指示的内部属性时， 将抛出 **TypeError** 异 常。

XML 类型在逻辑上源自 Object 类型， 并继承其内部属性。除非另有说明，XML 类型还继承了为 Object类型定义的 类型转换语义（ECMAScript第 3 版第 9 节）。下表总结了 XML 类型在 Object类型所定义的内部属性基础上新增的 内部属性。

*- 12-*

**财产**

[[姓名]] [[父级]] [[属性]]

[[作用域命名空间]] [[长度]]

[[按索引删除]]

[[深度复制]] [[解析值]]

【后裔】

[[等于]]

[[插入]]

[[替换]]

[[在作用域命名空间中添加]]

**参数**

无 无 无 无 无

（属性名称）

() ()

（属性名称）

（价值）

（属性名称，值）

（属性名称，值）

（命名空间）

**描述**

此 XML 对象的名称。

此 XML 对象的父对象。

与此 XML 对象相关联的属性。

此 XML 对象的作用域内的命名空间 此 XML 对象中有序属性的数量。

删除具有数字型属性名称的属性。

索引

返回此 XML 对象的深拷贝。

返回此 XML 对象。此方法在尝试解析空 XMLList 的值时使用。

返回一个 XMLList 对象， 其中包含此 XML 对象中名称与*propertyName* 匹配的所有后代节点。

返回一个布尔值， 该值指示此 XML 对象是否与给定的 XML 值具有相同 的 *XML* 内容。

在名称为 *PropertyName*（数字索引） 的属性之前插入一个或多个新属 性。

用一个或多个新属性替换名称为 *PropertyName*（数字索引） 的属性的 值。

将命名空间添加到此 XML 对象的属性 中。

[[作用域命名空间]]

[[Name]] 属性的值应为 **null** 或包含合法 XML 元素名称、 属性名称或处理指令名称的 QName 对象。 如果 XML 对象表示 XML 注释或文本节点， 则 [[Name]] 属性的值为 **null**。对于表示处理指令的每个 XML 对象， 其 [[Name]] 的 *uri* 属性将被设置为空字符串。

[[Parent]] 属性的值应为 XML 对象或 **null**。 当 XML 对象作为另一个 XML 对象的属性（即子对象） 出现时， [[Parent]] 属性将被设置为包含该 XML 对象的 XML 对象（即父对象）。

[[Attributes]] 属性的值是一个包含零个或多个 XML 对象的集合。 当向 [[Attributes]] 集合添加新对象时， 它会 替换集合中具有相同集合标识的任何现有对象。对于每个 XML 对象 *x* ∈ [[Attributes]]， 其集合标识定义为 *x. [[Name]]*。因此， 不存在两个对象 *x*、*y* ∈ [[Attributes]] 使得 *x.[[Name]] ==y.[[Name]*] 的比较结果为**真**。 如果 XML 对象表示 XML 属性、注释、处理指令或文本节点， 则 [[Attributes]] 属性的值为空集。

注意：尽管在 XML 中使用属性语法声明命名空间， 但它们不会在 [[Attributes]] 属性中表示。

[[InScopeNamespaces]] 属性的值是一个包含零个或多个 Namespace 对象的集合， 这些对象代表此 XML 对象 作用域内的命名空间声明。 [[InScopeNamespaces]] 属性中的所有 Namespace 对象都具有一个前缀属性， 其值 不是**未定义**的。 当向 [[InScopeNamespaces]] 集合添加新对象时， 它会替换集合中具有相同集合标识的任何现 有对象。每个 Namespace 对象 *n* ∈ [[InScopeNamespaces]] 的集合标识定义为 *n.prefix*。因此， 不存在两个对象 *x*、*y* ∈ [[InScopeNamespaces]]， 使得比较 *x.p*refix ==*y.p*refix 的结果为**真。**

*- 13-*

[[Length]] 属性的值是一个非负整数。

除非另有规定， 新创建的 XML 类型的实例的 [[Prototype]] 初始化为 XML 原型对象（第 13.4.4 节）， [[Class]] 初始化为字符串 **“text”**， [[Value]] 初始 化为 **undefined**， [[Name]] 初始化为 **null**， [[Parent]] 初始化为 **null**， [[Attributes]] 初始化为空集 **{ }**， [[InscopeNamespaces]] 初始化为空集 **{ }**， [[Length]] 初始化为整数 0。

[**9.1.1.1**](9.1.1.1) **[[获取]] (P**

**概述**

XML 类型覆盖了由 Object类型定义的内部 [[Get]] 方法。XML 的 [[Get]] 方法用于通过名称检索 XML 属性或通过名称检索一组 XML 元素。输入参数 *P* 可以是 XML 属性的未限定名称（通过前导“@”符号与 XML 元素的名称区分开来）、一组 XML 元素、一组 XML 元素的 QName、一组 XML 属性的 AttributeName、属性通配符“@\*”或元素通配符“\*”。当输入参数 *P* 是未限定的 XML 元素名称时， 它标识默认命名空间中的 XML 元素。 当输入参 数 *P* 是未限定的 XML 属性名称时， 它标识无命名空间的 XML 属性。

此外，输入参数 *P* 可以是数字属性名。如果 *P* 是数字属性名，XML 的 [[Get]] 方法会将此 XML 对象转换为仅包含一个值的 *XMLList*列表， 并使用参数 *P* 调用该列表的 [[Get]] 方法。这种处理有意模糊了单个 XML 对象与仅包含一个值的 XMLList之间的区别。

注意：与内部的 Object [[Get]] 方法不同， 内部的 XML [[Get]] 方法从不用于检索与 XML 对象相关联的方法。E4X 如第 11.2.2 节所述修改了 ECMAScript 中针对 XML 对 象的方法查找语义。

**语义学**

当对 XML 对象 *x* 调用 [[Get]] 方法并传入属性名 *P* 时，将执行以下步骤：

1. 如果 ToString（ToUint32(P)） 等于 *P*

a. 令 *list* = ToXMLList(x

b. 返回调用列表的 [[Get]] 方法并传入参数 *P* 所得到的结果 2. 设*n* = ToXMLName(P)

3. 令 *list* 为一个新的 XMLList，其中 *list.[[TargetObject]]* = *x* 且 *list.[[TargetProperty]]* = *n*

4. 如果 Type(n) 是属性名称

对于 *x.[[Attributes]]* 中的每个 *a*

i. 如果（n.[[Name]].localName 等于 **“\*”** 或者 n.[[Name]].localName 等于 *a.[[Name]].localName*） 并且（（n.[[Name]].uri **为空）**或者（n.[[Name]].uri 等于 *a.[[Name]].uri*）

1. 调用列表的 `Append` 方法， 并将参数设为 `*a*` 。 b. 退货清单

5. 对于 (k 从 **0** 到 *x.[[Length]] - 1* a. 如果 ((n.localName == **"\***

或者 (（x[k].[[Class]] == **"element"）** 并且 (x[k].[[Name]].localName == *n.localName*

并且（（n.uri **为 null）**或者（（x[k].[[Class]] 等于 **“element”）**并且（n.uri 等于 *x[k].[[Name]].uri*）） 调用列表的 [[Append]] 方法， 并将 *x[k]* 作为参数传入。

6. 退货清单

[**9.1.1.2**](9.1.1.2) **[[放入]] (P， V**

**概述**

XML 类型覆盖了由 Object类型定义的内部 [[Put]] 方法。XML 的 [[Put]] 方法用于替换和插入 XML 对象中的属性或 XML 属性。参数 *P* 用于标识 XML 对象中将受影响的部分， 它可以是 XML 属性的未限定名称（通过前导“@”符号与 XML 值属性名称区分开来）或一组 XML 元素、一组 XML 元素的 QName、一组 XML 属性的 AttributeName 或属性通配符“\*”。当参数 *P* 是未限定名称时

*- 14-*

XML 元素名称， 用于标识默认命名空间中的 XML 元素。 当参数 *P* 是未限定的 XML 属性名称时， 它用于标识无命名空间的 XML 属性。参数 *V* 可以是 XML 对象、XMLList 对象或任何可通过 ToString（） 转换为字符串的值。

如果 *P* 是一个数字属性名称， XML 的 [[Put]] 方法将抛出一个**类型错误**异常。此操作保留用于 E4X 的未来版本。

注意：与内部对象的 [[Put]] 方法不同， 内部 XML 的 [[Put]] 方法绝不会用于修改与 XML 对象相关联的方法集。

**语义学**

当对 XML 对象 *x* 调用 [[Put]] 方法， 传入属性名 *P* 和值 *V* 时， 将执行以下步骤：

*- 15-*

10. 设 *`primitiveAssign*` 为 `(Type(c) 庄{XML, XMLList})` 且 `（n.localName 不等于字符串 **“\*”）**`

11. 对于 (k 从 *x* 的长度减 *1* 递减至 **0** a. 如果 ((n.localName == **"\***

或者 (（x[k].[[Class]] == **"element"）** 并且 (x[k].[[Name]].localName == n.localName

并且（（n.uri **为 null）**或者（（x[k].[[Class]] 等于 **“element”）**并且（n.uri 等于 *x[k].[[Name]].uri*）） i. 如果 i 不是**未定义的**， 则调用 *x* 的 [[DeleteByIndex]] 属性， 其参数为 ToString(i) 的值。

2。令 *i* = *k*

**12. 如果 *i* 未定义**

a. 令 *i* = *x* 的长度

b. 如果（primitiveAssign == **true）** i. 如果 (n.uri **为空**

1. 假设通过调用构造函数 `new` 创建了一个新的 QName 对象 `name` 。

QName (GetDefaultNamespace (), *n)*

2。否则

1. 假设通过调用构造函数 new QName(n) 创建了一个新的 QName 对象， 将其命名为 name 。

3。创建一个新的 XML 对象 *y*， 使其具有 *y.[[Name]]* = *name*、*y.[[Class]]* = **"element"** 以及 *y.[[Parent]]* = *x* 的属性。 iv. 令 *ns* 为调用 [[GetNamespace]] 方法获取 *name* 的结果， 且不传入任何参数。

v. 调用 *x* 的 [[Replace]] 方法， 传入参数 ToString(i) 和 *y*

vi. 对 *y* 调用 [[AddInScopeNamespace]] 方法， 传入参数 *ns* 13. 如果（原始赋值为**真）**

a. 删除 XML 对象 *x[i]* 的所有属性

b. 令 *s* = c 的字符串表示形式

c. 如果 *s* 不是空字符串， 则使用参数 **“0”** 和 *s* 调用 *x[i]* 的 [[Replace]] 方法 14. 否则

a. 调用 *x* 的 [[Replace]] 方法， 传入参数 ToString(i) 和 *c*

15. 返回

[**9.1.1.3**](9.1.1.3) **[[删除]] (P**

**概述**

XML 类型覆盖了由 Object 类型定义的内部 [[Delete]] 方法。XML 的 [[Delete]] 方法用于通过名称移除一组 XML 属性或一组 XML 值属性。与内部的 Object [[Delete]] 不同，XML 的 [[Delete]] 方法会将删除属性之后的所有属性上移， 以填补删除操作所造成的空缺。输入参数 *P* 可以是 XML 属性的未 限定名称（通过前导“@”符号与 XML 元素名称区分开来）、一组 XML 元素、一组 XML 元素的 QName、一组 XML 属性的 AttributeName、属性 通配符“\*”或属性通配符“@\*”。 当输入参数 *P* 是未限定的 XML 元素名称时， 它标识默认命名空间中的 XML 元素。 当输入参数 *P* 是未限定的 XML 属性名称时， 它标识无命名空间的 XML 属性。

如果 *P* 是一个数字属性名称，XML 的 [[Delete]] 方法将抛出一个**类型错误**异常。此操作保留用于 E4X 的未来版本。

**语义学**

当对 XML 对象 *x* 调用带有属性名称 *P* 的 *[[Delete]]* 方法时， 将执行以下步骤：

1. 如果 `ToString(ToUint32(P))` 等于 `P`，则抛出一个 `TypeError` 异常。注意，此操 作保留用于 E4X 的未来版本。

*- 16-*

[**9.1.1.4**](9.1.1.4) **[[DeleteByIndex]] (P)**

**概述**

XML 类型向 Object 类型定义的方法中添加了内部的 [[DeleteByIndex]] 方法。XML 的 [[DeleteByIndex]] 方法用于通过 其数字属性名称删除 XML 属性。与 XML 的 [[Delete]] 方法类似，XML 的 [[DeleteByIndex]] 方法会将删除属性之后 的所有属性向下移动， 以填补删除操作所造成的空缺。输入参数 *P* 可以是数字属性名称。

**语义学**

当对 XML 对象 *x* 调用 [[DeleteByIndex]] 方法并传入属性名称 *P* 时， 将执行以下步骤：

3. 否则抛出一个**类型错误**异常

[**9.1.1.5**](9.1.1.5) **[[默认值]]（提示）**

**概述**

XML 类型覆盖了由 Object 类型定义的内部 [[DefaultValue]] 方法。XML 的 [[DefaultValue]] 方法返回一个表示此

XML 对象的基本值。与 Object 类型定义的 [[DefaultValue]] 方法不同，XML 的 [[DefaultValue]] 方法始终返回一个字 符串。提示参数将被忽略。

**语义学**

当对 XML 对象 *x* 调用 [[DefaultValue]] 方法并传入参数 *hint*时， 将执行以下步骤：

1. 返回 ToString(x) 的值

[**9.1.1.6**](9.1.1.6) **[[具有属性]] (P**

**概述**

XML 类型重写了由 Object 类型定义的内部 [[HasProperty]] 方法。XML 的 [[HasProperty]] 方法用于根据名称确定此 XML 对象是否包含 XML 元素或属性。输入参数 *P* 可以是 XML 属性的未限定名称（通过与 XML 元素的名称加以区 分）。

*- 17-*

输入参数 P 可以是 XML 元素名称（带前导“@”符号）或 XML 元素集、XML 元素集的限定名、XML 属性集的属性名、属性通配符“@\*”或元 素通配符“\*”。 当输入参数 *P* 是未限定的 XML 元素名称时， 它标识默认命名空间中的 XML 元素。 当输入参数 *P* 是未限定的 XML 属性名称时， 它标识无命名空间的 XML 属性。

此外， 输入参数 *P* 可以是数字属性名。如果 *P* 是等于**“0”的数字属性名**， 则 XML [[HasProperty]] 方法返回 true。如果 *P* 是除**“0” 以外的数字属性 名**， 则 XML [[HasProperty]] 方法返回 false**。**这种处理有意模糊了单个 XML 对象与仅包含一个值的 XMLList 之间的区别。

**语义学**

当对 XML 对象 *x* 调用 [[HasProperty]] 方法并传入属性名称 *P* 时， 将执行以下步骤：

5. 返回**假**值

[**9.1.1.7**](9.1.1.7) **[[DeepCopy]] ( ) -> 9.1.1.7 [[DeepCopy]] ( )**

**概述**

XML 类型向由 Object 类型定义的内部属性添加了内部的 [[DeepCopy]] 方法。XML 的 [[DeepCopy]] 方法用于创建并返回此对象的深拷贝， 包括其属 性、属性值、命名空间以及所有后代对象的属性、属性值和命名空间。返回值的内部 [[Parent]] 属性被设置**为 null**， 而每个被复制后代对象的内部 [[Parent]] 属性则根据情况被设置为其新复制的父对象。

**语义学**

当调用 XML 对象 x 的 [[DeepCopy]] 方法时， 将执行以下步骤：

1. 令 *y* 为一个新的 XML 对象 ， 其 *[[Prototype]]* 属性值为 *x* 的 *[[Prototype]]* 属性值 ， [[Class]] 属性值为 x 的 [[Class]] 属性值， [[Value]] 属性值为 x 的 [[Value]] 属性值， [[Name]] 属性值为 x 的 [[Name]] 属性值， [[Length]] 属性值为 x 的 [[Length]] 属性值。

2. 对于 *x.[[InScopeNamespaces]]* 中的每个 *ns*

a. 假设通过调用构造函数 new Namespace（ns） 创建了一个新的命名空间 *ns2*。*b.* 令 *y.[[InScopeNamespaces]*] *= y. [[InScopeNamespaces]*] U *{ ns*2 } 。

3. 令 *y.[[Parent]]* = **null**

4. 对于 *x.[[Attributes]]* 中的每个 *a*

a. 设 *b* 为调用 *a* 的 [[DeepCopy]] 方法的结果

b. 令 *b* 的父级= *y*

c. 令 *y.[[Attributes]]* = *y.[[Attributes]]* U { *b* } 5. 对于 *i* 从 **0** 到 *x.[[Length]] - 1*

a. 设 *c* 为调用 *x*[i] 的 [[DeepCopy]] 方法的结果

b. 令 *y[i]* = *c*

c. 令 *c.[[Parent]]* = *y*

*- 18-*

**6. 返回*y***

[**9.1.1.8**](9.1.1.8) **[[后裔]] (P**

**概述**

XML 类型向由 Object 类型定义的内部属性添加了内部的 [[Descendants]] 方法。XML 的 [[Descendants]] 方法用于检索此 XML 对象中所有名称与输入 参数 P 匹配的 XML 值后代（即子节点、孙节点、 曾孙节点等）。输入参数 *P* 可以是 XML 属性的无限定名称（通过前导“@”符号与 XML 元素名 称区分开来）、XML 元素的集合、XML 元素的限定名称、XML 属性的集合、属性通配符“@\*”或属性通配符“\*”。 当输入参数 *P* 是无限定的 XML 元素名称时， 它标识默认命名空间中的 XML 元素。 当输入参数 *P* 是无限定的 XML 属性名称时， 它标识无命名空间的 XML 属性。

**语义学**

当对 XML 对象 x 调用 [[Descendants]] 方法并传入属性名称 *P* 时， 将执行以下步骤：

1. 设*n* = ToXMLName(P)

2. 令 *list* 为一个新的 XMLList，其中 *list.[[TargetObject]]* = **null** 。

3. 如果 Type(n) 是属性名称

对于 *x.[[Attributes]]* 中的每个 *a*

如果（（n. [[Name]].localName 等于 **“\*”）**或者（n. [[Name]].localName 等于 *a. [[Name]].localName*） 并且（（n.[[Name]].uri **为空）**或者（n.[[Name]].uri 等于 *a.[[Name]].uri*）

1. 调用列表的 `Append` 方法， 并将参数设为 *a* 。

4. 对于 (k 从 **0** 到 *x.[[Length]] - 1* a. 如果 ((n.localName == **"\***

或者 （（x[k].[[Class]] == **"element"）** 并且 （x[k].[[Name]].localName == *n.localName*））*)*

并且（（n.uri **为 null）**或者（（x[k].[[Class]] 等于 **“element”）**并且（n.uri 等于 *x[k].[[Name]].uri*）） 调用列表的 [[Append]] 方法， 并将 *x[k]* 作为参数传入。

b. 设 *dq* 为调用 *x[k*] 的 [[Descendants]] 方法并传入参数 *P* 所得到的结果。

c. 如果 *dq.[[Length]]* 大于 0， 则调用 *list* 的 [[Append]] 方法并将 *dq* 作为参数传入。

5. 退货清单

[**9.1.1.9**](9.1.1.9) **[[等于]] (V**

**概述**

XML 类型为 Object 类型所定义的内部属性添加了内部的 [[Equals]] 方法。XML 的 [[Equals]] 方法用于将此 XML 对象与另一个 XML 对象 V 进行 XML 内容相等性比较。如果 *V* 是与该 XML 对象相等的 XML 类型的值， 则 [[Equals]] 运算符返回 **true**， 否则返回 **false。**

**语义学**

当对 XML 对象 *x* 调用 [[Equals]] 方法并传入值 *V* 时， 将执行以下步骤：

1. 如果 V 的类型不是 XML， 则返回 **false** 。

2. 如果 *x.[[Class]]* 不等于 *V.[[Class]]*， 则返回 **false**

3. 如果 *x.[[Name]]* 不为**空**

a. 如果 *V.[[Name]]* 为**空， 则**返回 **false**

b. 如果 *x.[[Name]].localName* 不等于 *V.[[Name]].localName*， 则返回 **false**

c. 如果 *x.[[Name]].uri* 不等于 *V.[[Name]].uri*， 则返回 **false**

4. 否则，如果 *V.[[Name]]* 不为**空， 则**返回 **false**

5. 如果 *x.[[Attributes]]* 中的项数与 *V.[[Attributes]]* 中的项数不同， 则返回 **false** 。

6. 如果 *x.[[Length]]* 不等于 *V.[[Length]]*， 则返回 **false** 。

7. 如果 *x.[[Value]]* 不等于 *y[[Value]]*， 则返回 **false** 。

8. 对于 *x.[[Attributes]]* 中的每个 *a*

*- 19-*

a. 如果 *V.[[Attributes]]* 中不存在属性 *b*， 使得 *b.[[Name]].localName* ==

如果 *a.[[Name]].localName*、*b.[[Name]].uri* 与 *a.[[Name]].uri* 相同且 *b.[[Value]]* 与 *a.[[Value]]* 相同，则返回 **false**。

9. 对于 *i* 从 **0** 到 *x.[[Length]] - 1*

a. 设 *r* 为调用 *x[i]* 的 *[[Equals]]* 方法并将 *V[i]* 作为参数所得到的结果。

b. 如果 *r* **为假**， 则返回**假**

10. 返回**真**

[**9.1.1.10**](9.1.1.10) **[[ResolveValue]] ( )**

**概述**

XML 类型向由 Object 类型定义的内部属性添加了内部的 [[ResolveValue]] 方法。XML 的 [[ResolveValue]] 方法返回 此 XML 对象。它由 XMLList 的 [[ResolveValue]] 方法使用， 以支持对空 XMLList 的 [[Put]] 操作。

**语义学**

当调用 XML 对象 *x* 的 [[ResolveValue]] 方法时， 将执行以下步骤：

1. 返回 *x*

[**9.1.1.11**](9.1.1.11) **[[插入]] (P， V**

**概述**

XML 类型向由 Object 类型定义的内部属性添加了内部的 [[Insert]] 方法。XML 的 [[Insert]] 方法用于在特定位置 P 插 入值 *V*。输入参数 *P* 应为数字属性名称。输入参数 *V* 可以是 XML 类型、XMLList 类型的值， 或者任何可以通过 ToString（） 转换为字符串的值。

**语义学**

当对 XML 对象 *x* 调用 [[Insert]] 方法， 并传入属性名称 *P* 和值 *V* 时， 将执行以下步骤：

[**9.1.1.12**](9.1.1.12) **[[替换]] （P， V） 概述**

XML 类型向由 Object 类型定义的内部属性添加了内部的 [[Replace]] 方法。XML 的 [[Replace]] 方法可用于将特定位 置 *P* 的属性替换为值 V。输入参数 *P* 应为数字属性名称。输入参数 *V* 可以是 XML、XMLList 类型的值， 或者任何可 以通过 ToString（） 转换为字符串的值。

*- 20-*

**语义学**

当对 XML 对象 *x* 调用 [[Replace]] 方法， 并传入属性名 *P* 和值 *V* 时， 将执行以下步骤：

1. 如果 *x.[[Class]]* **属于** {"text"**, "comment", "processing-instruction", "attribute"}，** 则返回

2. 令 *i* = ToUint32(P

3. 如果（i 的字符串表示形式不等于 *P*）， 则抛出一个**类型错误**异常。

4. 如果 *i* 大于或等于 *x* 的长度，

a. 令 *P* = x.[[Length]] 的字符串表示形式

b. 令 *x.[[Length]]* = *x.[[Length]]* + **1**

5. 如果 Type(V) 是 XML 且 V*.[[Class]]* 属于 {"element", **"comment"**, **"processing-instruction"**, **"text"}** a. 如果 V 的 [[Class]] 属性值为“元素”， 且 V 为 *x* 或 *x* 的祖先节点， 则抛出一个**错误**异常。

b. 令 *V.[[Parent]]* = *x*

c. 如果 *x* 具有名称为 *P* 的属性

i. 令 *x[P].[[Parent]]* = **null**

d. 令 *x[P]* = *V*

6. 否则， 如果 V 的类型为 XMLList

a. 调用 *x* 的 [[DeleteByIndex]] 方法， 传入参数 *P*

b. 调用 *x* 的 [[Insert]] 方法， 传入参数 *P* 和 *V* 7. 否则

令 *s* =V 的字符串表示形式

b. 创建一个新的 XML 对象 *t*， 使其 *[[Class]]* 属性值为 **“text”**，*[[Parent]]* 属性值为 *x*，*[[Value]]* 属性值为 *s* 。

c. 如果 *x* 具有名称为 *P* 的属性

i. 令 *x[P].[[Parent]]* = **null**

d. 令 *x* 的属性 *P* 的值为 *t*

8. 返回

[**9.1.1.13**](9.1.1.13) **[[AddInScopeNamespace]] (N) 概述**

XML 类型向由 Object 类型定义的内部属性添加了内部的 [[AddInScopeNamespace]] 方法。XML 的 [[AddInScopeNamespace]] 方法用于向给定 XML 对象的 [[InScopeNamespaces]] 中添加新的命名空间。输入参数 *N* 是要添加到此 XML 对象的 [[InScopeNamespaces]] 属性中的类型为 Namespace 的值。 如果 N.prefix 未定义， 则不会将该命名空间添加到 [[InScopeNamespaces]] 中。如果 N.prefix 与 [[InScopeNamespaces]] 中某个命名空间的前缀匹配，

则匹配项将被 N 替换， 并且此元素中具有相同前缀的名称的前缀将被设置为未定义。

**语义学**

当对 XML 对象 *x* 调用 [[AddInScopeNamespace]] 方法并传入命名空间 *N* 时， 将执行以下步骤：

*- 21-*

**3. 返回**

**9.2 XMLList 类型**

XMLList 类型是属性的有序集合。XMLList 对象的每个属性都有一个唯一的数字属性名 *P*， 使得 ToString （ToUint32(P)） 等于 *P*， 其值的类型为 XML。与 XMLList对象相关的方法使用非数字属性名。

XMLList 类型的值表示一个 XML 文档、XML 片段或任意的 XML 对象集合（例如， 查询结果）。

E4X 有意模糊了单个 XML 对象与仅包含该对象的 XMLList之间的区别。为此， 适用于 XML 对象的所有操作也 适用于 XMLList对象。扩展 E4X 的实现应保留此约束条件。

**9.2.1 内部属性和方法**

XMLList 类型在逻辑上派生自 Object 类型 ， 并继承其内部属性。 除非另有说明 ， XMLList 类型还继承了为

Object 类型定义的类型转换语义（ECMAScript 第 3 版第 9 节）。下表总结了 XMLList 类型添加到 Object 类型所 定义的内部属性中的属性。

**财产**

[[长度]]

[[目标对象]]

[[目标属性]]

[[添加]]

[[深度复制]] 【后裔】

[[等于]]

[[解析值]]

**参数**

无

无

无

（价值）

()

（属性名称）

（价值）

()

**描述**

此 XMLList对象中包含的属性数量。

与此对象相关联的 XML 或 XMLList 对象， 当向此 XMLList 插入项时， 该对象将会受到影响。

当向空的 XMLList 中添加对象时，在 [[TargetObject]] 中可能创建的属性 的名称。

向此 XMLList对象的末尾添加一个新属性。

返回此 XMLList对象的深复制。

返回一个 XMLList， 其中包含此 XMLList 中所有值的后代， 其名称与 *propertyName* 匹配。

返回一个布尔值， 该值指示此 XMLList对象是否与给定的值具有相同的 内容，或者此 XMLList对象是否包含与给定值相等的对象。

解析此 XML 对象的值。如果此 XML 对象不为空，则返回该对象。否则， [[ResolveValue]] 将尝试创建一个合适的值。

[[Length]] 属性的值是一个非负的数字。

除非另有说明， 新创建的 XMLList 类型的实例的 [[Prototype]] 初始化为 XMLList 原型对象， [[Class]] 初始化为 字符串 **“XMLList”** ， [[Value]] 初始化为**未定义**， [[Length]] 初始化为整数 **0**， [[TargetObject]] 初始化为 **null**， [[TargetProperty]] 初始化为 **null。**

[**9.2.1.1**](9.2.1.1) **[[获取]] (P**

**概述**

*- 22-*

XMLList 类型重写了由 Object 类型定义的内部 [[Get]] 方法。XMLList 的 [[Get]] 方法用于通过其数字属性名称获取此 XMLList 对象的特定属性， 或者遍历此 XMLList对象的 XML 值属性， 通过名称获取其 XML 属性或通过名称获取其 XML 值属性。输入参数 *P* 可以是数字属性名称、未限定的 XML 属性名称（通过前导“@ ”符号与 XML 元素名称区 分开来）、一组 XML 元素、一组 XML 元素的 QName、一组 XML 属性的 AttributeName、属性通配符“\* ”或属性 通配符“@\* ”。当输入参数 *P* 是未限定的 XML 元素名称时， 它标识默认命名空间中的 XML 元素。当输入参数 *P* 是 未限定的 XML 属性名称时， 它标识无命名空间的 XML 属性。

注意： 与内部对象的 [[Get]] 方法不同 ， 内部 XMLList 的 [[Get]] 方法从不用于检索与 XMLList 对象相关联的方法。E4X 对 XMLList 对象的 ECMAScript 方法查找语义进行了修改，如第 11.2.2 节所述。

**语义学**

当对 XMLList对象 *x* 调用 [[Get]] 方法并传入属性名 *P* 时， 将执行以下步骤：

1. 如果 ToString（ToUint32(P)） 等于 *P*

a. 返回调用对象的 [[Get]] 方法的结果， 其中 *x* 作为 **this** 对象， 参数为 *P* 。

2. 令 *list* 为一个新的 XMLList，其中 *list.[[TargetObject]]* = *x* 且 *list.[[TargetProperty]]* = *P*

3. 对于 *i* 从 **0** 到 *x.[[Length]] - 1* ，

a. 如果 x[i].[[Class]] 等于 **“element” ，**

i. 令 *gq* 为调用 *x[i*] 的 [[Get]] 方法并传入参数 *P* 所得到的结果。

2。如果 *gq*的长度大于 0， 则调用列表的 [[Append]] 方法并将 *gq*作为参数传入。

4. 退货清单

[**9.2.1.2**](9.2.1.2) **[[放入]] (P， V**

**概述**

XMLList 类型重写了由 Object 类型定义的内部 [[Put]] 方法。XMLList 的 [[Put]] 方法用于修改或替换 XMLList 中的 XML 对象及其父对象的上下文。此外， 当 XMLList包含单个 XML 对象属性时， [[Put]] 方法用于通过名称修改、替 换和插入该值的属性或 XML 属性。输入参数 *P* 用于标识 XMLList及相关 XML 对象中将受影响的部分， 它可以是数 字属性名称、未限定的 XML 属性名称（通过前导“@ ”符号与 XML 值属性名称区分开来）或 XML 元素集、XML 元素集的 QName、XML 属性集的 AttributeName 或属性通配符“\* ”。 当输入参数 *P* 是未限定的 XML 元素名称时， 它标识默认命名空间中的 XML 元素。当输入参数 *P* 是未限定的 XML 属性名称时， 它标识无命名空间的 XML 属性。 输入参数 *V* 可以是 XML、XMLList 类型的值， 或者可以使用 ToString（） 转换为字符串的任何值。

注意：与内部的 Object [[Put]] 方法不同， 内部的 XMLList [[Put]] 方法绝不会用于修改与 XMLList 对象相关联的方法集。

**语义学**

当对 XMLList对象 *x* 调用 [[Put]] 方法， 并传入属性名 *P* 和值 *V* 时， 将执行以下步骤：

1. 令 *i* = ToUint32(P

2. 如果 ToString(i) 等于 *P*

a. 如果 *x.[[TargetObject]]* 不为 **null**

i. 设 *r* 为调用 *x.[[TargetObject]]* 的 *[[Resolve Value]]* 方法的结果。 2。如果 *r* 为 **null， 则**返回

否则令 *r* = **null**

c. 如果 *i* 大于或等于 *x* 的长度。

i. 如果 Type(r) 是 XMLList 类型

1. 如果 *r.[[Length]]* 不等于 **1**， 则返回*- 23-*

2. 否则令 *r* = *r[0]*

2。如果 *r.[[Class]]* 不等于 **“element”**， 则返回

3。创建一个新的 *XML* 对象 *y*， 使其 [[Parent]] 属性等于 *r*，*[[Name]]* 属性等于 *x* 的 *[[TargetProperty]]* 属性，*[[Attributes]]* 属性等于 {}, *y.[[*长度*]]*= **0**

iv. 如果 Type（x.[[TargetProperty]]） 的类型为 AttributeName

1. 令 *attributeExists* 为调用 *r* 的 [[Get]] 方法并传入参数 *y.[[Name]]* 所得到的结果。

2. 如果（属性存在.[[长度]] > 0） ， 则返回

3. 令 *y.[[Class]]* = **"属性**

否则， 如果 *x.[[TargetProperty]]* **为 null** 或者 *x.[[TargetProperty]].localName* 等于

1. 令 *y.[[Name]]* = **null**

2. 令 *y.[[Class]]* = **"text** 否则， 令 *y.[[Class]]* = **"元素** 7。令 *i = x* 的长度

8。如果（y 的 [[Class]] 属性不等于 **“attribute”）**

1. 如果 *r* 不为**空**

如果（i > **0）**

令*j* = **0**

2。 当 （j < *r.[[Length]] - 1*）时

并且（r[j] 与 *x[i - 1]* 不是同一个对象）

1. 令*j* =*j* + **1**

b. 否则

令*j* = *r.[[Length]] - 1*

c. 调用 *r* 的 [[Insert]] 方法， 传入参数 ToString（j + 1） 和 *y*

2. 如果 V 的类型为 XML， 则令 *y.[[Name]]* = *V.[[Name]]*

3. 否则， 如果 V 的类型为 XMLList， 则令 *y.[[Name]]* = *V.[[TargetProperty]]* 9。调用 *x* 的 [[Append]] 方法， 并将 *y* 作为参数传入。

d. 如果 （V 的类型不属于 {XML， XMLList}） 或者 （V 的 [[Class]] 属于 {"text", **"attribute"}）** ， 则令 *V = V* 的字符串表示形式 e. 如果 *x[i].[[Class]]* 等于 **"attribute**

i. **令 *z* =** ToAttributeName(x[i].[[Name

2。调用 *x[i].[[Parent]]* 的 *[[Put]]* 方法，传入参数 ***z*** 和 *V*。

3。令 *attr* 为调用 *x[i].[[Parent]]* 的 *[[Get]]* 方法并传入参数 ***z*** 所得到的结果。 iv. 令 *x[i]* = *attr[0]*

f. 否则， 如果 V 的类型为 XMLList i. 创建 *V* 的浅层副本 *c*

2。令 *parent* = *x[i].[[Parent]]* 3。如果父对象不为**空**

1. 设 *q* 为父对象的属性， 使得*parent[q]* 与 *x[i]* 是同一个对象。

2. 调用父类的 [[Replace]] 方法， 并传入参数 *q* 和 *c* 。

3. 对于*j* 从 **0** 到 *c.[[Length]] - 1*

a. 令 *c[j]* =*parent[ToUint32(q) +j]* iv. 如果 *c.[[Length]]* 等于 0

1. 对于*j* 从 *i* + 1 到 *x.[[Length]]* - 1， 将 *x* 的属性*j* 重命名为 ToString(j - 1) 否则

1. 对于*j* 从 *x* 的长度减 *1* 到 *i + 1*， 将 *x* 的第*j* 个属性重命名为 ToString(j + *c* 的长度 - 1) vi. 对于*j* 从 **0** 到 *c.[[Length]] - 1*， 令 *x[i* +*j]* = *c[j]*

7。令 *x.[[Length]]* = *x.[[Length]]* + *c.[[Length]]* - 1

否则， 如果（V 的类型为 XML）或者（x[i].[[Class]] 属于 {"text", **"comment"**, **"processing-instruction"}）**

g。

i. 令 *parent* = *x[i].[[Parent]]* 2。如果父对象不为**空**

1. 设 *q* 为父对象的属性， 使得*parent[q]* 与 *x[i]* 是同一个对象。

2. 调用父类的 [[Replace]] 方法， 并传入参数 *q* 和 *V*。

3. 令 *V = parent[q]* 3。如果 V 的类型为字符串

1. 创建一个新的 XML 对象 *t*， 使其 *[[Class]]* 属性值为 **“text”**，*[[Parent]]* 属性值为 *x*，*[[Value]]* 属性值为 *V*。

2. 令 *x[i]* = *t* iv. 否则

1. 令 *x[i]* = *V*

*- 24-*

h. 否则

调用 *x[i]* 的 *[[Put]]* 方法，传入参数 **“\*”** 和 *V*。

3. 否则，如果 *x* 的长度小于或等于 **1**

a. 如果 *x.[[Length]]* 等于 **0**

i. 设 *r* 为调用 *x* 的 [[ResolveValue]] 方法的结果

2。如果 （r **为 null）** 或者 （r 的长度不等于 **1）**，则返回 3。调用 *x* 的 [[Append]] 方法，并将 *r* 作为参数传入。

b. 调用 *x[0]* 的 *[[Put]]* 方法，传入参数 *P* 和 *V*

4. 返回

[**9.2.1.3**](9.2.1.3) **[[删除]] (P**

**概述**

XMLList 类型重写了由 Object 类型定义的内部 [[Delete]] 方法。XMLList 的 [[Delete]] 方法用于通过其数字属性名称移除 XMLList 的特定属性，

或者遍历 XMLList 中的 XML 值属性，通过名称移除其 XML 属性或元素。输入参数 *P* 可以是数字属性名称、未限定的 XML 属性名称（通过前 导“@”符号与 XML 元素名称区分开来）、一组 XML 元素、一组 XML 元素的 QName、一组 XML 属性的 AttributeName、属性通配符“\*”或 属性通配符“@\*”。当输入参数 *P* 是未限定的 XML 元素名称时， 它标识默认命名空间中的 XML 元素。当输入参数 *P* 是未限定的 XML 属性名 称时， 它标识无命名空间的 XML 属性。

**语义学**

当对 XMLList对象 *x* 调用 [[Delete]] 方法并指定属性名称 *P* 时，将执行以下步骤：

1. 令 *i* = ToUint32(P

[**9.2.1.4**](9.2.1.4) **[[默认值]]（提示）**

**概述**

XMLList 类型重写了由 Object 类型定义的内部 [[DefaultValue]] 方法。XMLList 的 [[DefaultValue]] 方法返回一个表示此 XMLList 对象的原始值。 与 Object 的 [[DefaultValue]] 方法不同，XMLList 的 [[DefaultValue]] 方法始终返回一个字符串。hint 参数将被忽略。

**语义学**

*- 25-*

当对 XMLList 对象 *list* 调用 [[DefaultValue]] 方法并带有参数 *hint* 时， 将执行以下步骤：

1. 将列表转换为字符串并返回。

[**9.2.1.5**](9.2.1.5) **[[具有属性]] (P**

**概述**

XMLList 类型重写了由 Object 类型定义的内部 [[HasProperty]] 方法。XMLList 的 [[HasProperty]] 方法用于确定此 XMLList 对象是否包含按其序号位 置指定的 XML 元素或属性， 或者此 XMLList对象中包含的任何对象是否包含按其名称指定的 XML 元素或属性。输入参数 *P* 可以是数字属性名称、 未限定的 XML 属性名称（通过前导“@”符号与 XML 元素名称区分开来）、 一组 XML 元素、 一组 XML 元素的 QName、 一组 XML 属性的

AttributeName、属性通配符“\*”或属性通配符“@\*”。 当输入参数 *P* 是未限定的 XML 元素名称时， 它标识默认命名空间中的 XML 元素。 当输入 参数 *P* 是未限定的 XML 属性名称时， 它标识无命名空间的 XML 属性。

**语义学**

当对 XMLList 对象 *x* 调用 [[HasProperty]] 方法并传入属性名 *P* 时， 将执行以下步骤：

1. 如果 ToString（ToUint32(P)） 等于 *P* a. 返回（ToUint32(P) < *x.[[Length]]*）

2. 对于 *i* 从 **0** 到 *x.[[Length]] - 1*

a. 如果 x[i].[[Class]] 的值为 **“element”** 并且调用 *x[i]* 的 *[[HasProperty]]* 方法（传入参数）的结果 如果 *P* 为**真**， 则返回**真**。

3. 返回**假**值

[**9.2.1.6**](9.2.1.6) **[[附加]] (动词**

**概述**

XMLList 类型向由 Object 类型定义的内部属性添加了内部的 [[Append]] 方法。XMLList 的 [[Append]] 方法用于将由 *V* 指定的零个或多个值添加到 XMLList 的末尾。输入参数 *V* 应为 XMLList类型或 XML 类型的值。

**语义学**

当对 XMLList 对象 *x* 调用 [[Append]] 方法并传入值 *V* 时， 将执行以下步骤：

*- 26-*

XMLList 类型向由 Object 类型定义的内部属性添加了内部的 [[DeepCopy]] 方法。XMLList 的 [[DeepCopy]] 方法用于创建并返回此 XMLList 对象的副 本， 其中包含其所有属性的深复制。

**语义学**

当调用 XMLList 对象 *x* 的 [[DeepCopy]] 方法时， 将执行以下步骤：

1. 令 *list* 为一个新的 XMLList 对象

2. 将 *x* 的所有内部属性复制到列表中

3. 对于 *i* 从 **0** 到 *x.[[Length]] - 1*

a. 令 *list[i]* 为调用 *x*[i] 的 [[DeepCopy]] 方法的结果

4. 退货清单

[**9.2.1.8**](9.2.1.8) **《后裔》（P）**

**概述**

XMLList 类型在由 Object 类型定义的内部属性中添加了内部的 [[Descendants]] 方法。XMLList 的 [[Descendants]] 方法可用于检索此 XMLList 中属性的 所有 XML 值后代（即子节点、孙节点、 曾孙节点等）， 其名称与输入参数 P 匹配。输入参数 *P* 可以是数字属性名称、XML 属性的未限定名称（通 过前导“@”符号与 XML 元素名称区分开来）、XML 元素集、XML 元素集的 QName、XML 属性集的 AttributeName、属性通配符“\*”或属性通配 符“@\*”。 当输入参数 *P* 是未限定的 XML 元素名称时， 它标识默认命名空间中的 XML 元素。 当输入参数 *P* 是未限定的 XML 属性名称时， 它标识 无命名空间的 XML 属性。

**语义学**

当对 XML 对象 *x* 调用 [[Descendents]] 方法并传入属性名称 *P* 时， 将执行以下步骤：

1. 令 *list* 为一个新的 XMLList，其中 *list.[[TargetObject]]* = **null** 。

2. 对于 *q* 从 0 到 *x.[[Length]]* - **1**

a. 如果 (x[q].[[Class]] == **"element**

i. 设 *dq* 为调用 *x[q*] 的 [[Descendants]] 方法并传入参数 *P* 所得到的结果。

2。如果 *dq*的长度大于 0， 则调用 *list* 的 [[Append]] 方法并将 *dq*作为参数传入。

3. 退货清单

[**9.2.1.9**](9.2.1.9) **[[等于]] (V**

**概述**

XMLList 类型向由 Object 类型定义的内部属性添加了内部的 [[Equals]] 方法。XMLList 的 [[Equals]] 方法用于将此 XMLList 对象的内容与另一个 XMLList 对象 *V* 进行比较， 以确定其是否相等， 或者确定此 XMLList对象是否仅包含一个与 V 相等的 XML 对象。如果此 XMLList对象被认为与 V 相等， 或者仅包含一个被认为与 *V* 相等的 XML 对象， 则 [[Equals]] 运算符返回 **true**， 否则返回 false**。**空的 XMLList 对象被认为与 **undefined** 相等。 输入参数 *V* 可以是 XMLList 类型、XML 类型、**undefined** 或任何可以使用 ToString（） 转换为字符串的值。

**语义学**

当对 XML 对象 *x* 调用 [[Equals]] 方法并传入值 *V* 时， 将执行以下步骤：

1. 如果 *V***未定义且** *x.[[Length]]* 的值为 **0**， 则返回 **true** 。

2. 如果 Type(V) 是 XMLList 类型

a. 如果 *x.[[Length]]* 不等于 *V.[[Length]]*， 则返回 **false**

b. 对于 *i* 从 **0** 到 *x* 的长度

*- 27-*

如果比较结果 *x[i]* == *V[i]* 为**假**，则返回**假**。

c. 返回 **true**

3. 否则，如果 *x* 的长度为 **1**

返回比较结果 *x[0]* 是否等于 *V*

4. 返回**假**值

[**9.2.1.10**](9.2.1.10) **[[ResolveValue]] ( )**

**概述**

XMLList 类型向由 Object 类型定义的内部属性添加了内部的 [[ResolveValue]] 方法。XMLList 的 [[ResolveValue]] 方法用于解析空 XMLList 的值。如果此 XMLList 对象不为空， 则 [[ResolveValue]] 方法将返回它。如果此 XMLList 为空， [[ResolveValue]] 方法将尝试根据 [[TargetObject]] 和 [[TargetProperty]] 属性创建它。如果无法创建 XMLList， [[ResolveValue]] 将返回 **null。**

**语义学**

当调用 XMLList 对象 *x* 的 [[ResolveValue]] 方法时，将执行以下步骤：

1. 如果 *x.[[Length]]* 大于 **0**，则返回 *x* 。

2. 否则

a. 如果 （x.[[TargetObject]] **为 null）** 或者 （x.[[TargetProperty]] 为 **null）**

或者（`type(x.[[TargetProperty]])` 是 `AttributeName`）或者（`x.[[TargetProperty]].localName` 等于 `**\*`）**，**则返回 `null**` 。

b. 令 *base* 为递归调用 *x.[[TargetObject]]* 的 *[[Resolve Value]]* 方法的结果

c. 如果 *base* **为 null， 则**返回 **null**

d. 令 *target* 为在 *base* 上调用 [[Get]] 并传入参数 *x.[[TargetProperty]]* 所得到的结果。 e. 如果（目标.[[长度]] 等于 **0）**

如果（`base` 的类型为 `XMLList`）并且（`base` 的长度大于 **1）**，**则返回 `null**` 。 2。在基对象上调用 [[Put]] 方法，传入参数 *x.[[TargetProperty]]* 和空字符串

3。令 *target* 为在 *base* 上调用 [[Get]] 方法并传入参数 *x.[[TargetProperty]]* 所得到的结果。 f. 回归目标

**9.3 属性名称类型**

**内部**的“***AttributeName***”**类型并非语言数据**类型**。**本规范仅出于说明目的对其进行定义。E4X 的实现应表现得如同其以此处所述的方式生成和操作 “AttributeName”一样。然而，“**AttributeName**”类型的值仅用作表达式求值的中间结果，不能作为变量或属性的值进行存储。

“AttributeName” 类 型 指 定 XML 属 性 的 名 称 。 可 以 使 用 “AttributeIdentifier” 指 定 “AttributeName” 类 型 的 值。如 果 属 性 名 称 未 指 定 为 “*QualifiedIdentifier*”，则关联的“QName” 的“*uri*”属性将是表示无命名空间的空字符串。

**9.3.1 内部属性**

“属性名称”类型在逻辑上派生自“对象”类型， 并继承其内部属性。除非另有说明，“属性名称”类型还继承了为“对象”类型定义的类型转换语义 （ECMAScript第 3 版第 9 节）。下表总结了“属性名称”类型添加到“对象”类型所定义的内部属性中的属性。

**财产**

[[姓名]]

**参数**

无

**描述**

属性的名称

[[Name]] 属性的值是 QName 类型的值。

*- 28-*

**9.4 任意名称类型**

**内部**的 ***AnyName* 类型并非语言数据**类型**。**本规范仅出于说明目的对其进行定义。E4X 的实现应表现得如同其以此处 所述的方式生成和操作 AnyName 值一样。然而，**AnyName** 类型的值仅用作表达式求值的中间结果， 不能作为变量 或属性的值进行存储。

AnyName 类型是一种标记类型， 用于表明特定属性的名称被指定为通配符标识符（即“\* ”）。AnyName 类型恰好 有一个值，称为 **anyname。**

**10 类型转换**

E4X 扩展了在 ECMAScript 中定义的自动类型转换操作符。 除非本节另有说明 ，XML 类型继承了为 Object 类型 （ECMAScript 第 3 版第 9 节）定义的类型转换语义。

注意：与 ECMAScript第 3 版一样， 这些类型转换函数是内部函数， 用户无法直接访问。它们在 E4X 算法中按需出现， 并在此处 进行描述， 以帮助说明类型转换的语义。此外，ToString 和 ToXMLString 通过第 13.4.4.38、 13.4.4.39、 13.5.4.21 和 13.5.4.22 节中 定义的内置方法 toString（） 和 toXMLString（） 间接提供给 E4X 用户使用。

**10.1 转换为字符串**

E4X 通过指定以下类型的“ToString”运算符的行为来扩展其功能。

**输入类型**

XML

XML 列表 属性名称

**结果**

按照第 10.1.1 节的定义，将 XML 对象作为字符串返回。

按照第 10.1.2 节中的定义，将 XMLList对象作为字符串返回。

给定输入参数 *a*，返回将字符串**“@”**与 a.[[Name]] 的字符串表示形式连接起来的结果。

**10.1.1 应用于 XML 类型的 ToString 方法**

**概述**

给定一个 XML 对象 *x*， ToString 操作符将 *x* 转换为字符串 *s*。如果 XML 类型的值具有简单内容（即不包含任何元 素）， 则它表示一个基本值， ToString 将返回 XML 对象的字符串内容， 省略起始标签、属性、命名空间声明和结束 标签。否则， ToString 将返回表示整个 XML 对象的字符串， 包括起始标签、属性、命名空间声明和结束标签。请注 意， 结果字符串内容的实际格式由实现定义。

结合 `ToString` 对 `XMLList` 的处理方式（见 10.1.2 节）， 这种行为使得 E4X 程序员能够以与访问对象属性值几乎相 同的方式访问 XML 叶节点的值。例如， 假设有变量 `*order*` 被赋值为以下 XML 值：

E4X 程序员可以像这样访问 XML 值的各个值：

*- 29-*

E4X 不需要程序员显式地选择与每个叶元素关联的文本节点，也不需要显式地选择每个 XMLList返回值中的第一个元素。对于不希望出现这种行为的情 况，提供了 ToXMLString 操作符（请参阅第 10.2 节）。

注意：在上述示例中，在执行乘法运算之前， 与 XML 对象 order.item.price 和 order.item.quantity 相关联的字符串值属性会隐式转换为数字类型。

对于 [[Class]] 被设置为“attribute”（属性）或“text”（文本）的 XML 对象，ToString 方法只是将其值作为字符串返回。

**语义学**

给定一个 XML 对象 *x*，ToString 会执行以下步骤：

3. 否则

a. 返回 ToXMLString(x

**10.1.2 应用于 XMLList 类型的 ToString 方法**

**概述**

操作符 ToString 将 XMLList对象列表转换为字符串 *s*。返回值是 XMLList 中每个项的字符串表示形式按顺序连接而成的字符串。

请注意，对只包含一个元素的列表调用 `ToString` 方法的结果与对 XMLList 中的单个元素调用 `ToString` 方法的结果相同。这种处理方式有意模糊了单个 XML 对象与仅包含一个值的 XMLList之间的区别， 以简化程序员的任务。它允许 E4X 程序员以与访问对象属性几乎相同的方式访问仅包含单个基本值 的 XMLList 的值。

**语义学**

对于给定的 *XMLList*对象 *list*，ToString 执行以下步骤：

1. 如果 *list.hasSimpleContent*（）的值为 **true** 设 *s* 为空字符串

b. 对于 i 从 **0** 到 *list.[[Length]] - 1*，

如果 *x[i].[[Class]]* 不属于 {"comment", **"processing-instruction"}**

1. 令 *s* 为 *s* 与 list[i] 的字符串表示形式连接后的结果。 c. 退货

2. 否则

a. 返回 ToXMLString(x

**10.2 ToXMLString（输入参数， [祖先命名空间]， [缩进级别]）**

E4X 为 ECMAScript 增加了转换操作符 ToXMLString。ToXMLString 是 ToString 的一种变体，用于将其参数转换为 XML 编码的字符串。与 ToString 不同， 无论内容如何， 它始终包含与 XML 元素相关联的起始标签、属性、命名空间声明和结束标签。这在以下情况中很有用：

*- 30-*

在不希望使用默认的 `ToString` 行为的情况下。`ToXMLString` 的语义由以下表格指定。

|  |  |
| --- | --- |
| **输入类型** | **结果** |
| 未定义 | 抛出一个**类型错误**异常。 |
| 空值 | 抛出一个**类型错误**异常。 |
| 布尔型 | 将输入参数转换为字符串并返回。 |
| 数字 | 将输入参数转换为字符串并返回。 |
| 字符串 | 返回 EscapeElementValue（输入参数） |
| XML | 根据第 10.2.1 节中指定的 XML 对象的内容创建一个 XML 编码的字符串值。 |
| XML 列表 | 根据第 10.2.2 节中指定的内容，基于 XMLList对象创建一个 XML 编码的字符串值。 |
| 对象 | 请按照以下步骤操作：  1. 设 *p* 为调用 ToPrimitive（输入参数， 提示 String） 的结果。 |

2. 设 *s* 为调用 ToString(p) 的结果。

3. 返回转义元素值（s）

**10.2.1 ToXMLString 在 XML 类型上的应用**

**语义学**

给定一个 XML 对象 *x* 以及可选参数 *AncestorNamespaces* 和可选参数 *IndentLevel*，ToXMLString 通过执行以下步骤将其转换为 XML 编码字符串 *s*：

1. 设 *s* 为空字符串

2. 如果未提供缩进级别，则令缩进级别= **0**

3. 如果（XML.美化打印 == **真）**

对于 *i* 从 **0** 到 *IndentLevel - 1*，令 *s* 为 *s* 与空格 <SP> 字符的连接结果 4. 如果 *x.[[Class]]* 等于 **“text” ，**

a. 如果 (XML.美化打印 == **真**

设 *v* 为从 *x* 的值中去除所有前导和尾随的 *XML* 空白字符后的结果。

2。返回将 *s* 与 EscapeElementValue(v) 的结果连接起来的值。

b. 否则

i. 返回 EscapeElementValue（x.[[Value]]） 的值

5. 如果 *x.[[Class]]* 等于 **“attribute”**，则返回 *s* 与 EscapeAttributeValue（x.[[Value]]） 的连接结果。

6. 如果 *x.[[Class]]* 等于 **“comment”**，则返回将字符串 *s*、字符串 **“<!-- ”、***x.[[Value]]* 以及字符串 **“-->”** 拼接后的结果。

7. 如果 *x.[[Class]]* 等于 **“processing-instruction”**， 则返回将 *s*、字符串**“<?”、***x.[[Name]].localName*、空格 <SP> 字符、*x. [[Value]]* 以及字符串**“？”连接起来的结果。*>*"**

8. 如果未提供 *AncestorNamespaces*，则令 *AncestorNamespaces* = { }

9. 令 *namespaceDeclarations* = { }

10. 对于 *x.[[InScopeNamespaces]]* 中的每个 *ns*

a. 如果不存在任何 *ans* 属于 AncestorNamespaces，使得 *ans.uri* 等于 *ns.uri* 并且 *ans.prefix* 等于 *ns.prefix* i. 让 *ns1* 成为 *ns* 的一份副本

2。令 *`namespaceDeclarations`* 等于 *`namespaceDeclarations`* 与 *`*{*ns1*}*`* 的并集。 注意：实现方式也可以从“命名空间声明” 中排除未使用的命名空间声明。

11. 对于由 x.[[*Name*]] 以及每个属性的名称所组成的名称集合中的每个名称 x. [[属性]]

a. 令 *namespace* 为 调 用 [[GetNamespace]] 方 法 对 *name* 所 得 结 果 的 副 本 ， 其 参 数 为 （AncestorNamespaces U namespaceDeclarations） 。

b. 如果（命名空间前缀**未定义），**

*- 31-*

i. 设 *namespace.prefix* 为任意实现定义的命名空间前缀， 使得不存在 *ns2* ∈ (AncestorNamespaces U namespaceDeclarations) 且 *namespace.prefix == ns2.prefix* 。

2。注意：如果空字符串尚未在集合（*AncestorNamespaces* U namespaceDeclarations）中使用，实现应优先将其作为实现定义的前缀。 3。令 *namespaceDeclarations* = *namespaceDeclarations* U { *namespace* }

12. 令 *s* 为将 *s* 与字符串**“<”连接后**的结果。

13. 如果命名空间前缀不是空字符串，

令 *s* 为将 *s*、命名空间前缀以及字符串**“：”**连接起来的结果。

14. 令 *s* 为 *s* 与 *x.[[Name]].localName* 拼接的结果。

15. 令 *attrAndNamespaces* = *x* 的 *[[Attributes]]* 并上 namespaceDeclarations

16. 对于 *attrAndNamespaces* 中的每个 an

设 *s* 为将 *s* 与空格 <SP> 字符连接后的结果

b. 如果 Type（an） 是 XML 且 *an.[[Class]]* 等于 **"attribute**

i. 令 *ans* 为调用 a.[[*Name]]* 的 *[[GetNamespace]]* 方法并将 *AncestorNamespaces* 作为参数所得到的结果的副本。 2。如果（ans.prefix **未定义），**

1. 令 *ans.prefix* 为任意实现定义的命名空间前缀， 使得不存在 *ns2* ∈ (AncestorNamespaces U namespaceDeclarations) 且 ans. *prefix == ns2.prefix* 。

2. 如果不存在 *ns2* ∈ (AncestorNamespaces U namespaceDeclarations)， 使得 *ns2.uri* ==

*ans.uri* 与 *ns2.prefix* 相等且 *ans.prefix* 相同

a. 令 *namespaceDeclarations* = *namespaceDeclarations* U { *ans* } 3。如果 *ans.prefix* 不是空字符串

1. 令 *s* 为将 *s*、命名空间前缀以及字符串**“：”**连接起来的结果。 iv. 让 *s* 为将 *s* 与 a.*[[Name]].localName* 连接后的结果。

c. 否则

令 *s* 为将字符串 *s* 与字符串 **“xmlns”** 连接后的结果。 2。如果 （an.prefix **未定义），**

1. 令一个前缀为任意实现定义的命名空间前缀， 使得不存在

*ns2* 属于（AncestorNamespaces 并集 namespaceDeclarations），且 *an* 前缀与 *ns2* 前缀相同。 3。如果前缀不是空字符串

1. 令 *s* 为将字符串 *s*、冒号**“ ：”**以及 an*.prefix* 拼接后的结果。

d. 设 *s* 为将字符串 *s*、等号（即**“=”**）和一个双引号字符（即 Unicode 编码）连接起来的结果。 代码点 **\u002**

e. 如果一个 *[[Class]]* 类型为**“属性”**

i. 令 *s* 为 *s* 与 EscapeAttributeValue（an.[[Value]]） 的连接结果 f. 否则

令 *s* 为 *s* 与 EscapeAttributeValue（an.uri） 的连接结果。

g. 令 *s* 为将 *s* 与一个双引号字符（即 Unicode 编码点 **\u0022）**连接后的结果。 17. 如果 *x.[[Length]]* 等于 **0**

令 *s* 为将 *s* 与 **“/>” 连接后**的结果。

b. 退货

18. 令 *s* 为 *s* 与字符串**“>”连接后**的结果。

19. 令 `*indentChildren*` 等于 `（（x.[[Length]] > **1）** 或 （x.[[Length]] == **1** 且 *x[0].[[Class]]* 不等于 **“text”））**`

20. 如果（XML.美化打印 == **真且**缩进子节点 == **真）**

a. 令 *nextIndentLevel* = *IndentLevel* + XML.PrettyIndent 。 21. 否则

a. 令 *nextIndentLevel* = **0**

22. 对于 *i* 从 **0** 到 *x.[[Length]] - 1*

a. 如果（XML.美化打印 == **真且**缩进子节点 == **真）** i. 让 *s* 成为 *s* 与一个行终止符连接后的结果

b. 令 *child* = ToXMLString(x[i], (AncestorNamespaces U namespaceDeclarations), *nextIndentLevel*

c. 让 *s* 为 *s* 与 child 拼接后的结果

23. 如果（XML.美化打印 == **真且**缩进子节点 == **真），**

令 *s* 为将 *s* 与一个行终止符连接后的结果。

b. 对于 *i* 从 **0** 到 *IndentLevel*，令 *s* 为 *s* 与一个空格 <SP> 字符连接的结果 24. 令 *s* 为将 *s* 与字符串 **“</”** 连接后的结果。

*- 32-*

25. 如果命名空间前缀不是空字符串

令 *s* 为将 *s*、*namespace.prefix* 以及字符串**“：”**连接起来的结果。

26. 令 *s* 为将 *s*、*x.[[Name]].localName* 以及字符串**“*>*”连接后**的结果。

27*.* 回报

注意：实现方式也可以在 ToXMLString（） 中保留无意义的空白字符（例如，在元素标签内部和标签之间） 以及属性的引号约定。

[**10.2.1.1**](10.2.1.1) **EscapeElementValue(s)**

**语义学**

给定一个字符串值 *s*， 操作符 EscapeElementValue 执行以下步骤：

1. 令 *r* 为空字符串

2. 对于字符串 *s* 中的每个字符 *c*

a. 如果 （c == **"<"）** ， 则令 *r* 为 *r* 与字符串“&lt； ”连接后的结果。

b. 否则， 如果 （c == **">"）** ， 则将 *r* 与字符串 **“&g**t;” 连接的结果赋值给 *r* 。

c. 否则， 如果 c 等于**“&”**， 则将 *r* 与字符串“&**amp； ”连接起来， 结果赋值**给 *r* 。

d. 否则， 将 c 与 *r* 连接起来， 结果仍记为 *r* 。

3. 回报率 *r*

[**10.2.1.2**](10.2.1.2) **EscapeAttributeValue(s)**

**语义学**

给定一个字符串值 *s*， 操作符 EscapeAttributeValue 执行以下步骤：

1. 设 *r* 为空字符串

2. 对于字符串 *s* 中的每个字符 *c*

a. 如果 c 是双引号字符（即 “） ， 则将字符串 ”**&quot;"** 与 *r* 连接起来， 结果赋值给 r 。

b. 否则， 如果 c 等于**“<”**， 则将 *r* 与字符串“**&lt； ”**连接起来， 结果赋值给 *r* 。

c. 否则， 如果 （c == **"&"）** ， 则将 *r* 与字符串 **“&amp;” 连接起来， 结果赋值**给 *r* 。 d. 否则， 如果 （c == **\u000A）** ， 则令 *r* 为 *r* 与字符串**“&#xA； ”**连接后的结果。

e. 否则， 如果 c 等于 **\u000D**， 则将 *r* 与字符串**“&#xD； ”**连接起来的结果赋值给 *r* 。

f. 否则， 如果 （c ==**\u0009）** ， 则令 *r* 为 *r* 与字符串**“&#x9； ”**连接后的结果。 g. 否则， 将 c 与 *r* 连接起来， 结果仍记为 *r* 。

3. 回报率 *r*

**10.2.2 ToXMLString 在 XMLList 类型中的应用**

**语义学**

给定一个 XMLList 对象 *x*、一个可选参数 *AncestorNamespaces* 以及一个可选参数 *IndentLevel*， ToXMLString 通过执行以下步骤将 *x* 转换为 XML 编码的字符串 *s*：

1. 设 *s* 为空字符串

2. 对于 *i* 从 **0** 到 *x.[[Length]] - 1*

a. 如果（XML.prettyPrinting 为**真**且 *i* 不等于 **0）**

i. 让 *s* 成为 *s* 与一个行终止符连接后的结果

b. 令 *s* 为将 *s* 与 ToXMLString（x[i], *AncestorNamespaces*）连接后的结果。

3. 回报

**10.3 转 XML**

E4X 在 ECMAScript 中添加了ToXML 操作符。ToXML 会根据以下表格将其参数转换为 XML 类型的值：

*- 33-*

**输入类型**

未定义 空值

布尔型

数字

字符串

XML

XML 列表

对象

万 维 XML

网 联 盟 项 目

信息

**结果**

抛出一个**类型错误**异常。 抛出一个**类型错误**异常。

使用 ToString 将输入参数转换为字符串，然后按照第 10.3.1 节中的规定将结果转换为 XML。

使用 `ToString` 方法将输入参数转换为字符串，然后按照第 10.3.1 节中的规定将结果转 换为 XML。

按照下面第 10.3.1 节中的规定，从字符串创建一个 XML 对象。

返回输入参数（不做转换）。

如果 XMLList 中仅包含一个属性且该属性的类型为 XML，则返回该属性。否则，抛出 一个**类型错误**异常。

如果输入参数的 [[Class]] 属性为 **“String”、“Number” 或 “Boolean”**， 则使用 ToString 将输入参数转换为字符串，然后按照第 10.3.1 节的规定将结果转换为 XML。 否则，抛出一个 **TypeError 异常**。

按照下文第 10.3.2 节中所规定的那样，从 W3C XML 信息项创建一个 XML 对象。

**10.3.1 应用于字符串类型的 ToXML 概述**

当 ToXML 应用于字符串类型时， 它会将该字符串解析为 XML 并进行转换。在转换之前， 可以使用字符串连接来构 建 XML 对象的部分内容， 而无需考虑诸如格式良好之类的 XML 约束。例如， 考虑以下情况。

此示例的最后一行将几个单独的字符串连接起来以形成一个单一的字符串值， 然后将生成的字符串传递给 XML 构造 函数。XML 构造函数使用内部的 ToXML 操作符将给定的字符串转换为 XML 对象。

**语义学**

给定一个字符串值 *s*， ToXML 通过以下步骤将该字符串转换为 XML 对象：

1. 令 *defaultNamespace* = 获取默认命名空间

2. 令 *parentString* 为将字符串 **“<parent xmlns=‘ ”**， *defaultNamespace,*“**’*>***”， *s* 和 **“<**/parent>” 连接起来 的结果。

3. 将 *parentString* 解析为 W3C 元素信息项 *e*， 如果解析失败， 则抛出**语法错误**异常。

4. 令 *x* = ToXML(e

5. 如果 *x.[[Length]]* 等于 0

a. 返回一个新的 XML 对象 *t*， 其中 *t.[[Class]]* = **"text"**， *t.[[Parent]]* = **null** 且 *t.[[Value]]* 为空字符串。 6. 否则， 如果 *x* 的长度为 1

令 *x[0].[[Parent]]* 为**空**

b. 返回 *x[0] - 34-*

**7. 否则抛出一个语法错误异常**

注意：此处使用 W3C XML 信息项仅作示例说明。执行此类转换并不需要 W3C XML 信息项， 实现时可以使用任何具有相同语义 的机制。

**10.3.2 将 ToXML 应用于 W3C XML 信息项**

**概述**

当 ToXML 应用于符合 W3C XML 信息集规范的 W3C XML 信息项的实现时， 它会将 E4X 数据模型映射到给定的信 息项上， 从而可以使用E4X 操作符来查询、导航和操作给定的信息项。

本节中提供的 ToXML 定义是有意设计得较为通用的。它允许实现从 W3C DOM 对象、XPath 节点、XQuery 项或任 何其他可映射到 W3C XML 信息项的数据模型的实例创建 XML 对象。E4X 并不要求实现支持这些数据模型， 也不要 求支持这些数据模型的实现提供从它们创建 XML 对象的功能。相反， 它为选择实现这些操作的实现定义了这些操作 的语义。

例如， 一种实现方式可能会部署 E4X 来方便在网页浏览器中操作文档对象。 因此，E4X 用户可能会编写如下 E4X 代 码：

与其编写下面等效的 DOM 代码：

注意：上述第一个示例使用 XML 构造函数在 W3C DOM 对象周围创建一个 E4X 包装对象（第 2 行），然后对生成的 E4X 对象执 行操作以操作底层的 DOM 对象。实现也可以提供在新表达式中使用 XML 构造函数的能力（例如 ，“var doc = new XML

(document)；”）， 以创建一个独立的 XML 对象， 该对象不会操作底层的 DOM 对象。如上所述，E4X 实现不需要支持 W3C DOM 或提供从 W3C DOM 对象构造 XML 对象的能力。

**语义学**

*- 35-*

对于给定的 W3C 信息项 *i*， ToXML 按照如下方式将其映射到 XML 对象*x*：

1. 令 *x* = MapInfoItemToXML(i

2. 如果 x **为 null**， 则返回一个新的 XML 对象， 该对象的 *[[Class]]* 属性值为 **“text”**，*[[Value]]* 属性值为空字符串，*[[Parent]]* 属性值**为 *null*** 。

3. 返回 *x*

[**10.3.2.1**](10.3.2.1) **MapInfoItemToXML ( i ) ->** [**10.3.2.1**](10.3.2.1) **MapInfoItemToXML ( i )**

**概述**

当对 W3C XML 信息项调用 MapInfoItemToXML 时， 它会返回**空**值或 XML 对象。在没有从 W3C XML 信息到 E4X 数据模型的映射， 或者映射生成 空结果的情况下， 会返回**空**值。例如， 当对文档类型声明信息项调用 MapInfoItemToXML 时， 或者当 XML.ignoreComments 设置为**true** 并对注释信 息项调用 MapInfoItemToXML 时， 会返回空值**。**

在 MapInfoItemToXML 中使用的 Map-to 标记表明了其第一个参数和第二个参数之间的关系。具体来说， 它表明第一个参数和第二个参数的行为就好 像它们是同一个对象一样， 这样对第一个参数所做的更改将导致第二个参数发生更改， 反之亦然。例如， 在以下步骤之后：

1. 将 *x.[[Value]]* 映射到 *i* 的 **[content]** 属性

对 *x.[[Value]]* 所做的所有更改都将应用于 *i* 的 **[content]** 属性。 同样， 对 *i* 的 **[content]** 属性所做的所有更改也将应用于 *x*。

同样地， 在步骤之后：

1. 将 *x* 的 *[[Attributes]]* 映射到 i 的 **[attributes] 属性**上

对集合 *x.[[Attributes]]* 中添加或移除的每个属性， 都会导致 *i* 的 **[attributes] 属性**进行相应的添加或删除操作。 同样， 对 *i* 的 **[attributes]** 属性添加或 移除的每个属性， 都会导致 *x.[[Attributes]]* 进行相应的添加或删除操作。

如果“映射到”操作的其中一个参数是常量值， 则两个参数都不能更改。但是， 已被映射到常量值的参数之后可以映射到不同的值。

注意：Map-to 关系的强制执行机制取决于具体实现。只有当用户围绕非 E4X 数据模型（例如 W3C DOM 节点）的实例创建 XML 包装对象时，Map-to 语义才可见。 在其他情况下，Map-to 表示法可能被视为 Let表示法。

**语义学**

对于给定的 W3C 信息项 *i*，MapInfoItemToXML 按照如下方式将其映射到 XML 对象 *x* 或 **null**：

1. 将 *x.[[Parent]]* 映射为 **null**

2. 如果 *i* 是一个字符信息项

将 *x.[[Class]]* 映射到 "text

b. 将 *x.[[Value]]* 映射到从索引 *i* 开始、在文档顺序中连续向前且具有相同父节点的最大字符信息项序列。将字符串 *x.[[Value]]* 中的每 个字符映射到该序列中每个**字符信息项的相应 [**字符**代码]**属性。

c. 如果 （XML.ignoreWhitespace 为 **true）** 并且 （*x.[[Value]]* 中的每个字符都是 *XML* 空白字符）， **则返回 null**

d. 否则返回 *x*

3. 如果 *i* 是一条评论信息项

a. 如果 XML.ignoreComments 的值为 **true**， **则返回 null** 。

b. 将 *x.[[Class]]* 映射为 "comment

c. 将 *x.[[Value]]* 映射到 *i* 的 **[content]** 属性

*- 36-*

d. 返回 *x*

4. 如果 *i* 是一个处理指令信息项

如果 XML.ignoreProcessingInstructions 的值为 **true**，**则返回 null** 。

b. 将 *x.[[Class]]* 映射到 "处理指令

c. 将 *x.[[Name]]* 映射到 *i* 的 **[target]** 属性 d. 将 *x.[[Value]]* 映射到 *i* 的 **[content]** 属性 e. 返回 *x*

5. 如果 *i* 是一个属性信息项

将 *x.[[Class]]* 映射到 "属性

b. 将 *x.[[Name]].localName* 映射到 *i* 的 **[本地名称]** 属性 c. 将 *x.[[Name]].uri* 映射到 *i* 的 **[命名空间名称**] 属性

注意：实现方式也可能将 *x.[[Name]].[[Prefix]]* 映射到 *i* 的 [prefix] 属性。如果 [prefix] 无值，则将 x.[[Name]].[[Prefix]] 映射为空字符串。

d. 将 *x.[[Value]]* 映射到 i 的 **[规范化值]** 属性 e. 返回 *x*

6. 如果 *i* 是一个元素信息项

将 *x.[[Class]]* 映射到 "element

b. 将 *x.[[Name]].localName* 映射到 *i* 的 **[本地名称]** 属性 c. 将 *x.[[Name]].uri* 映射到 *i* 的 **[命名空间名称**] 属性

注释 实现方式也可能将 *x.[[Name]].[[Prefix]]* 映射到 *i* 的 **[prefix]** 属性。如果 [prefix] 如果无值，则 x.[[Name]].[[Prefix]] 映射为空字符串。

d. 将 *x.[[Attributes]]* 映射到 *i* 的 **[attributes] 属性** e. 对于 i 的 **[attributes] 属性**中的每个属性信息项 a

i. 将 *x.[[Attributes]]* 中的成员属性 *attr* 映射到调用 MapInfoItemToXML(a) 的结果上。 2。将属性 *[[Parent]]* 映射到 *x*

f. 对于 I 的 **[作用域内命名空间]**属性中的每个命名空间信息项 *n*， 除了 其 **[前缀]**属性等于 “xml” 的命名空间信息项

i. 按照以下方式将 *x.[[InScopeNamespaces]]* 中的成员 *ns* 映射到 *n*：

1. 将 *ns.prefix* 映射到 *n* 的 **[prefix]** 属性。如果 [prefix] 没有值， 则将 *ns.prefix* 映射为空字符串。

2. 将 *ns.uri* 映射到 *n* 的 **[命名空间名称**] 属性上 g. 令*j* = 0

h. 将 *xmlChild* 设为 0

将 *x* 中的属性集映射到 i 的 **[children]** 属性上

j. 令 *numItemChildren* 为 i 的 **[children]** 属性中信息项的数量 k. 当 （j < *numItemChildren*）时

i. 令 *item* 为 *i* 的 **[children]** 属性中的第 j 个信息项 2。令 *c* = MapInfoItemToXML(item

3。如果 *c* 不为**空**

1. 将 *x[xmlChild]* 映射到 *c*

2. 将 *x[xmlChild].[[Parent]]* 映射到 *x*

3. 如果 *c.[[Class]]* 等于 **"text**

令*j* =*j* + *c.[[Value]].length* - 1

4. 将 *xmlChild* 的值加 1 ， 并赋值给 *xmlChild* 。 iv. 令*j* =*j* + 1

1. 将 *x.[[Length]]* 映射到 *xmlChild* m. 返回 *x*

7. 如果 *i* 是一个文档信息项

a. 返回对 i 的 **[文档元素]** 属性调用MapInfoItemToXML 的结果

8. 如果 *i* 是未展开的实体引用信息项 抛出一个**类型错误**异常

9. 返回**空值**

注意：MapInfoItemToXML 会忽略文档类型声明信息项。

**10.4 转 XML 列表**

E4X 在 ECMAScript 中添加了 ToXMLList 操作符。ToXMLList 操作符会根据下表将其参数转换为 XMLList 类型的值：

*- 37-*

|  |  |
| --- | --- |
| **输入类型** | **结果** |
| 未定义 | 抛出一个**类型错误**异常。 |
| 空值 | 抛出一个**类型错误**异常。 |
| 布尔型 | 按照第 10.4.1 节的规定， 先使用 ToString 将输入参数转换为字符串，然后再将结果转 换为 XMLList。 |
| 数字 | 使用 `ToString` 将输入参数转换为字符串，然后按照第 10.4.1 节中的规定将结果转换为 XMLList。 |
| 字符串 | 按照下面第 10.4.1 节中的说明，从字符串创建一个 XMLList对象。 |
| XML | 给定输入参数 *x*， 返回一个新的 *XMLList* 对象列表 ， 该列表满足以下条件：*list. [[Length]]* = 1，*list[0]* = *x*，*list.[[TargetObject]]* = *x.[[Parent]]* 以及 *list.*  *[[TargetProperty]]* = *x.[[Name]]*。 |
| XML 列表 | 返回输入参数（不做转换）。 |
| 对象 | 如果输入参数的 [[Class]] 属性为 **“String”、“Number” 或 “Boolean”**， 则使用 ToString 将输入参数转换为字符串，然后按照第 10.3.1 节的规定将结果转换为 XML。 否则，抛出一个 **TypeError 异常**。 |

**10.4.1 ToXMLList 应用于字符串类型**

**概述**

当将 ToXMLList应用于字符串类型时， 它会通过将字符串解析为 XML 片段来将其转换为 XMLList类型。在转换之前， 可以使用字符串连接来构建 XMLList 值的部分内容。例如，

此示例的最后一行将两个字符串连接起来形成一个单一的字符串值， 然后将生成的字符串传递给 XMLList构造函数。XMLList 构造函数使用内部的 ToXMLList 运算符将给定的字符串转换为 XML 对象。

**语义学**

给定一个字符串值 *s*， ToXMLList 会按照以下步骤将其转换为 XMLList：

1. 令 *defaultNamespace* = 获取默认命名空间

2. 令 *parentString* 为将字符串 **“<parent xmlns=‘”， *defaultNamespace,*“’*>***”， *s* 和 **“<**/parent>” 连接起来的结果；

3. 将 *parentString* 解析为 W3C 元素信息项 *e*

4. 如果解析失败， 则抛出一个**语法错误**异常。

5. 令 *x* = ToXML(e

6. 令 *list* 为一个新的 XMLList，其中 *list.[[TargetObject]]* = **null** 。

7. 对于 *i* 从 0 到 *x.[[Length]] - 1*

a. 将 *x[i].[[Parent]]* **设为 null**

b. 调用列表的 [[Append]] 方法， 参数为 *x[i]* 8. 退货清单

注意：此处使用 W3C XML 信息项仅作示例说明。执行此类转换并不需要 W3C XML 信息项，实现方式可以采用任何能提供相同语义的机制。

*- 38-*

**10.5 属性名称**

E4X 在 ECMAScript 中添加了 ToAttributeName 操作符。ToAttributeName 操作符会根据下表将其参数转换为类型为 AttributeName 的值：

**输入类型**

未定义 空值

布尔型 数字

字符串

XML

XML 列表

对象

属性名称 AnyName

**结果**

抛出一个**类型错误**异常。 抛出一个**类型错误**异常。 抛出一个**类型错误**异常。 抛出一个**类型错误**异常。

按照第 10.5.1 节中指定的方式从字符串创建一个属性名称。

ToString 将输入参数转换为字符串，然后按照第 10.5.1 节的规定将结果转换为属性名

使用 称。

使用 称。

ToString 将输入参数转换为字符串，然后按照第 10.5.1 节的规定将结果转换为属性名

如果输入参数是一个 QName 对象（即其内部的 [[Class]] 属性为 **“QName”）**，则返回一个 新的 AttributeName， 其 [[Name]] 属性设置为输入参数。否则， 使用 ToString 将输入参数转 换为字符串，然后按照第 10.5.1 节的规定将结果转换为 AttributeName。

返回输入参数（不做转换）。

返回调用 ToAttributeName（"\*"） 的结果。

**10.5.1 应用于字符串类型的 ToAttributeName**

给定一个字符串 *s*， ToAttributeName 转换函数会返回一个属性名称 *a*。a 的 [[Name]] 属性被设置为一个新的 QName *q*， 其本地名称设置为给定的字 符串， 其 URI 设置为空字符串， 表示没有命名空间。

**语义学**

给定一个字符串值 *s*， ToAttributeName 会按照以下步骤将其转换为属性名称：

1. 假设通过调用构造函数 new Namespace（） 创建了一个新的命名空间 *ns* 。

2. 设 *q* 为通过调用构造函数 new QName（ns, *s*, 创建的新 QName 。

3. 返回一个新的属性名 *a*， 其中 *a.[[Name]]* = *q*

**0.61ToXMLName**

E4X 在 ECMAScript 中添加了ToXMLName 操作符。ToXMLName 是一个内部操作符， 它会根据下表将其参数转换为类型为 AttributeName 的值或 QName 对象：

*- 39-*

|  |  |
| --- | --- |
| **输入类型** | **结果** |
| 未定义 | 抛出一个**类型错误**异常。 |
| 空值 | 抛出一个**类型错误**异常。 |
| 布尔型 | 抛出一个**类型错误**异常。 |
| 数字 | 抛出一个**类型错误**异常。 |
| 字符串 | 按照第 10.6.1 节中的规定，从指定的字符串创建一个 QName 对象或属性名称。 |
| XML | 使用 `ToString` 将输入参数转换为字符串， 然后按照第 10.6.1 节中的规定将结果转换为 `  QName` 对象或 `AttributeName` 。 |
| XML 列表 | 使用 `ToString` 将输入参数转换为字符串， 然后按照第 10.6.1 节中的规定将结果转换为 `  QName` 对象或 `AttributeName` 。 |
| 对象 | 如果输入参数是一个 QName 对象（即其 [[Class]] 属性为 **“QName”）**， 则返回该输入参 数。否则，使用 ToString 将输入参数转换为字符串，然后按照第 10.6.1 节的规定将结果转换 为 QName 对象或属性名称。 |
| 属性名称 | 返回输入参数（不做转换）。 |
| AnyName | 返回调用 ToXMLName（"\*"） 的结果。 |

**10.6.1 ToXMLName 应用于字符串类型**

给定一个字符串 *s*， ToXMLName 转换函数会返回一个 QName 对象或属性名称 。 如果 *s* 的第一个字符是 **“@”**， ToXMLName 将使用 ToAttributeName 操作符创建一个属性名称。否则， 它将使用 QName 构造函数创建一个 QName 对象。

**语义学**

给定一个字符串值 *s*， ToXMLName 操作符会按照以下步骤将其转换为 QName 对象或属性名称：

1. 如果 `ToString(ToUint32(s))` 等于 `ToString(s)`， 则抛出一个**类型错误**异常。

2. 如果字符串 *s* 的首字符是**“@”**

a. 令 *name* = *s* 的子字符串从索引 *1* 到 *s* 的长度。

b. 返回 ToAttributeName(name

3. 否则

a. 返回一个如同通过调用构造函数 new QName(s) 创建的 QName 对象

**11 表达式**

**11.1 基本表达式**

**语法**

E4X 通过以下产生式扩展了 ECMAScript定义的主要表达式：

*- 40-*

*XML*列表初始化器

属性标识符：

属性标识符

合格标识符

通配符标识符

**语义学**

生产式“*PrimaryExpression* : *PropertyIdentifier*” 的求值方式如下：

1. 设 *n* 为计算属性标识符的结果。

2. 令 *name* = ToXMLName(n

3. 当（为真）时

a. 如果作用域链上没有更多的对象， 抛出一个**引用异常**

b. 让 *o* 成为作用域链中的下一个对象。

注意， 在第一次迭代中， o 将是作用域链上的第一个对象。

c. 如果 Type(o) 属于 {XML， XMLList}

i. 令 *hasProp* 为调用对象 *o* 的 *[[HasProperty]]* 方法的结果， 传入 *name* 作为属性名。 2。如果 *hasProp* 为**真**

1. 返回一个类型为“引用” 的值， 其基对象为 *o*， 属性名为 *name* 。 属性标识符“*PropertyIdentifier* : *AttributeIdentifier*” 的计算方式如下：

1. 返回对“属性标识符”的求值结果。

除了在步骤 1 中将 *AttributeIdentifier* 分别替换为 *QualifiedIdentifier* 和 *WildcardIdentifier* 之外， 对生产式 *PropertyIdentifier* : *QualifiedIdentifier* 和 *PropertyIdentifier : WildcardIdentifier* 的求值方式完全相同。

**11.1.1 属性标识符**

**语法**

E4X 通过添加属性标识符来扩展 ECMAScript。属性标识符的语法由以下产生式指定：

属性标识符：

@ 属性选择器 @ 合法标识符 @ [ 表达式 ]

属性选择器：标识符 通配符标 识符

**概述**

属性标识符用于标识 XML 属性的名称。其计算结果为类型为“属性名称 ”的值。前面的“@”字符用于将具有相同名称的 XML 属性与 XML 元素区分开来。选择这种属性标识符语法是为了与熟悉的 XPath 语法保持一致。

**语义学**

生产属性标识符：*@*属性选择器 的计算方式如下：

1. 令 *name* 为一个字符串值， 其中包含与 *PropertySelector* 中相同的字符序列。

*- 41-*

**2. 返回属性名称(name**

生产属性标识符：@限定标识符 的求值方式如下：

1. 设 *q* 为对 *QualifiedIdentifier* 进行求值的结果。

2. 返回 ToAttributeName(q

生产属性标识符：@ [ 表达式 ] 的求值方式如下：

1. 设 *e* 为表达式求值的结果。

2. 返回 ToAttributeName（GetValue(e)） 的值

**11.1.2 合法标识符**

**语法**

E4X 通过添加限定标识符来扩展 ECMAScript。 限定标识符的语法由以下产生式指定：

**概述**

限定标识符用于标识特定命名空间内定义的值。它们可用于访问、操作和创建具有命名空间限定的 XML 元素和属性名称。例如，

**语义学**

一个“*QualifiedIdentifier*”会求值为一个“QName”对象。“*QualifiedIdentifier* : *PropertySelector :****:*** *PropertySelector*”这一产生式按如下方式求值：

1. 令 *ns* 为评估第一个属性选择器的结果。

2. 如果 （ns 等于 **anyname）** ， 则令 *ns* = **null**

3. 令 *localName* 为一个字符串值， 其中包含与第二个属性选择器中相同的字符序列。

4. 返回一个新的 QName 对象， 该对象的创建方式就如同调用了构造函数 new QName（GetValue(ns), *localName*, 一样。

合格标识符的生成：属性选择器：**：** [ 表达式 ] 按照以下方式求值：

*- 42-*

1. 设 *ns* 为评估属性选择器的结果。

2. 如果 （ns 等于 **anyname）**， 则令 *ns* = **null**

3. 设 *e* 为表达式求值的结果。

4. 返回一个新的 QName 对象， 其创建方式就如同调用了构造函数 new QName(GetValue（ns), GetValue(e)） 一样。

**11.1.3 通配符标识符**

**语法**

E4X 通过添加通配符标识符来扩展 ECMAScript。通配符标识符的语法由以下产生式指定：

通配符标识符 ：

\*

**概述**

通配符标识符用于标识任何名称。它可用于匹配命名空间、 XML 对象的属性或 XML 属性。通配符标识符的计算结果为 **anyname。**

**语义学**

通配符标识符“\*” 的生成过程如下：

**1. 归还任何物品**

**11.1.4 XML 初始化器**

**概述**

XML 初始化器是一种描述 XML 对象初始化的表达式， 以字面量的形式编写。它可以使用普通的 XML 语法指定 XML 元素、XML 注释、 XML 处理指令或 CDATA 部分。对于 XML 元素， 它提供了 XML 对象的名称、XML 属性和 XML 属性值。

XML 初始化器的句法语法用于查找 XML 初始化器的结束位置并计算嵌入的表达式。它允许出现不符合 XML 语法的字符序列。在计算完所 有嵌入的表达式之后， 由 XML 值组成的字符结果字符串将传递给 XML 构造函数， 该构造函数会根据更严格的 XML 语法对其进行解析。

XML 初始化器的句法语法处理由词法语法目标符号 *InputElementXMLTag* 和 *InputElementXMLContent* 生成的输入元素。这些输入元素在第 8.3 节中有描述。

以下是一些 XML 初始化器的示例。

表达式可用于计算 XML 初始化器的部分内容。表达式由花括号分隔， 可出现在标签内或元素内容中。在标签内， 表达式可用于计算标签名、 属性名或属性值。在元素内， 表达式可用于计算元素内容。例如，

*- 43-*

在解析字面 XML 值之前， 会先对每个表达式进行求值并用其值替换。例如以下表达式，

会将以下 XML 值赋给变量 x 。

**语法**

**语义学**

生产 *XMLInitialiser* : *XMLMarkup* 的评估方式如下：

1. 令 *markup* 为一个字符串字面值， 其中包含与 *XMLMarkup*相同的字符序列。

2. 返回一个新创建的 XML 对象， 其创建方式如同通过调用 XML 构造函数并传入参数*markup* 一样。

*- 44-*

生产 *XMLInitialiser* : *XMLElement* 的评估方式如下：

1. 令 *element* 为对 *XMLElement* 进行求值的结果。

2. 返回一个新创建的 XML 对象，其创建方式如同通过调用 XML 构造函数并传入参数 *element*（第 13.4.2 节）一样。

生产 *XMLElement*： < *XMLTagContent XMLWhitespaceopt* **/>** 的评估方式如下：

1. 将 *XMLTagContent* 的计算结果赋值给 *content* 。

2. 返回由字符串值**“<”、内容以及字符串值“/>”依次连接而成**的结果。

对生产 *XMLElement*： < *XMLTagContent XMLWhitespaceopt* > *XMLElementContentopt* <**/** *XMLTagContent XMLWhitespaceopt* **>** 的评估如下：

1. 令 *startTag* 为对第一个 *XMLTagContent* 进行求值的结果。

2. 让内容成为结果 如果存在，则用于评估 *XMLElementContent*；如果不存在，则使用空字符串

3. 令 *endTag* 为对第二个 *XMLTagContent*进行求值的结果。

4. 返回由字符串值**“<”、*startTag*、**字符串值**“>”、*content*、字符串值“</”、*endTag* 以及字符串值“>”**依次连接而成的结果。

生产 *XML* 标签内容：*XML* 标签名 *XMLAttributesopt* 的评估方式如下：

1. 令 *name* 为对 *XMLTagName* 进行求值的结果。

2. 若不存在属性，则将其视为评估结果；若无评估结果，则使用空字符串 *XMLAttributesopt*;

3. 返回将名称与属性连接后的结果。

生产 *XML* 标签名称：**{** 表达式 **}** 的计算方式如下：

1. 令 *expRef*为表达式求值的结果

2. 令 *expression* = 取值(expRef

3. 返回对表达式执行 ToString（） 操作后的结果。

生产 *XML* 标签名：*XMLName* 的评估方式如下：

1. 返回包含与 *XMLName* 相同字符序列的字符串字面值。

生产 *XMLAttributes* : *XMLWhitespace* **{** 表达式 **}** 的计算方式如下：

1. 令 *expRef*为表达式求值的结果

2. 令 *expression* = 取值(expRef

3. 令属性 = 表达式.ToString

4. 返回由一个包含单个空格字符的字符串和属性连接而成的结果。

生产 *XML* 属性：*XMLAttributeopt XML* 属性的评估方式如下：

1. 若未出现，则将属性设为评估结果；若不存在，则使用空字符串 *XMLAttributeopt*;

2. 让属性成为评估 *XML* 属性的结果。

3. 返回将属性和属性列表连接后的结果。

生产 *XMLAttribute* : *XMLWhitespace XMLName XMLWhitespaceopt* **=** *XMLWhitespaceopt* **{** 表达式 **}** 的计算方式如下：

1. 令 *name* 为包含与 *XMLName* 相同字符序列的字面字符串。

2. 令 *expRef*为表达式求值的结果

3. 令 *expression* = 取值(expRef

4. 令 *value* 为调用 EscapeAttributeValue（ToString(expression)） 所得的结果。

*- 45-*

5. 返回由包含单个空格字符的字符串、名称、包含“=”字符的字符串以及值连接而成的结果。

生产 *XMLAttribute* : *XMLWhitespace XMLName XMLWhitespaceopt* **=** *XMLWhitespaceopt XMLAttribute Value* 的计算方式如下：

1. 令 *name* 为包含与 *XMLName* 相同字符序列的字面字符串。

2. 令 *value* 为一个字符串字面值， 其中包含与 XMLAttributeValue 相同的字符序列。

3. 返回由包含单个空格字符的字符串、名称、包含“*=*”字符的字符串以及值连接而成的结果。

生产 *XMLElementContent* : **{** 表达式 **}** *XMLElementContentopt* 的计算方式如下：

1. 令 *expRef*为表达式求值的结果

2. 令 *expression* = 取值(expRef

3. 如果 Type（expression） 属于 {XML， XMLList} ，

a. 令 *value* 为调用 ToXMLString（expression） 的结果

4. 否则

a. 令 *value* 为调用 EscapeElementValue（ToString(expression)） 所得的结果

5. 如果内容不存在，则评估 *XMLElementContentopt*; ，若无内容则使用空字符串

6. 返回将 value 与 content 拼接后的结果。

生产 *XMLElementContent* : XMLMarkup *XMLElementContentopt* 的评估方式如下：

1. 设标记为评估 *XMLMarkup* 的结果

2. 如果内容不存在，则将其设为评估结果；若无评估结果，则使用空字符串。

3. 返回将标记与内容连接后的结果

生产 *XMLElementContent* : *XMLText XMLElementContentopt* 的评估方式如下：

1. 令 *text* 为一个字符串字面值， 其中包含与 *XMLText*相同的字符序列。

2. 如果内容不存在，则评估 *XMLElementContentopt*; ，若无内容则使用空字符串

3. 返回将文本与内容连接后的结果。

生产 *XMLElementContent* : *XMLElement XMLElementContentopt* 的评估方式如下：

1. 令 *element* 为对 *XMLElement* 进行求值的结果

2. 如果内容不存在，则将其设为评估结果；若无评估结果，则使用空字符串。

3. 返回将元素与内容连接后的结果。

**11.1.5 XMLList 初始化器**

**概述**

XMLList 初始化器是一种用于描述 XMLList对象初始化的表达式， 其形式类似于字面量。它使用匿名 XML 元素语法来描述一个有序的 XML 属性列表。XMLList 初始化器以字符序列“<>”开头， 以字符序列“</>”结尾。

XML 初始化器的句法语法处理由词法语法目标符号 *InputElementXMLTag* 和 *InputElementXMLContent* 生成的输入元素。这些输入元素在第 8.3 节 中有描述。

以下是一些 XMLList初始化器的示例，

*- 46-*

**语法**

**语义学**

生产 *XML* 列表： < **>** *XMLElementContentopt* <**/ >** 的评估方式如下：

1. 如果未指定， 则将内容设为对 *XMLElementContent*进行求值的结果；否则使用空字符串。

2. 返回一个新创建的 XMLList对象， 其创建方式如同通过调用 XMLList构造函数并传入参数 *content*一样。

**11.2 左值表达式**

E4X 通过以下产生式扩展了 ECMAScript 中定义的左值表达式：

成员表达式：

成员表达式 **.** 属性标识符 成员表达式 **..** 标识符 成员表达式 **..** 属性标识符 成员表达式 **. （**表达 式 **）**

调用表达式：

调用表达式**.** 属性标识符 调用表达式**.. 标 识**符 调用表达式**..** 属性标识符 调用表达 式**. （**表达式 **）**

此外，E4X 为应用于 XML 和 XMLList类型值的现有左值表达式定义了新的语义。

**11.2.1 属性访问器**

**语法**

E4X 重用并扩展了 ECMAScript 的属性访问语法， 用于访问类型为 XML 和 XMLList 的值中的属性和 XML 属性。XML 属性可以通过名称访问， 使用点符号表示法：

成员表达式**.** 标识符 成员表达式**.** 属性标识符 调 用表达式**.** 标识符 调用表达式**.** 属性标识符

或者使用方括号表示法：

成员表达式 **[** 表达式 **]** 调用表达式 **[** 表达 式 **]**

**概述**

当成员表达式或调用表达式计算结果为 XML 对象时，属性访问器会使用 XML 的 [[Get]] 方法来确定结果。如果右操作数计算结果为数组索引， 则 XML 的 [[Get]] 方法会将 XML 对象转换为 XMLList， 并从生成的 XMLList 中获取请求的属性。这种处理方式

*- 47-*

故意模糊了单个 XML 对象与仅包含一个值的 XMLList之间的区别。否则，XML 的 [[Get]] 方法会检查左操作 数的 XML 属性和 XML 属性， 并按顺序返回名称与右操作数匹配的 XMLList。例如，

当 *MemberExpression* 或 *CallExpression* 的计算结果为 XMLList 时， 属性访问器会使用 XMLList 的 [[Get]] 方法 来确定结果。如果使用方括号表示法且标识符为数字， 则 XMLList 的 [[Get]] 方法仅返回左操作数中与该数字 标识符匹配的属性。否则，XMLList 的 [[Get]] 方法会对列表中的每个 XML 对象应用属性访问器操作， 并返回 一个新的 XMLList， 其中包含按顺序排列的结果。例如，

在上述第一个属性访问器语句中， 表达式“order.item”会检查绑定到“order” 的 XML 对象的 XML 属性， 并 返回一个包含两个名为“item” 的 XML 对象的 XMLList。然后， 表达式“order.item.description”会检查结果 XMLList 中每个项目的 XML 属性， 并返回一个包含两个名为“description” 的 XML 对象的 XMLList。

当 `*MemberExpression*` 或 `*CallExpression*` 的计算结果不是 `XML` 或 `XMLList` 类型， 且表达式的右侧为标识 符时 ， 属性访问器将按照 ECMAScript 第 3 版中指定的语义执行 。 然而 ， 如果 `*MemberExpression*` 或 `

*CallExpression*` 的计算结果不是 `XML` 或 `XMLList` 类型， 且表达式的右侧为属性标识符， 则属性访问器将抛 出 `**TypeError**` 异常。

**语义学**

*- 48-*

与 ECMAScript 第 3 版一样， 以下生产式的运行情况：

成员表达式：成员表达式 **.** 标识符 的行为与以下产生式的行为相 同：

成员表达式：成员表达式 **[** *<*标识符字符串*>* **]** 同样，生产方面的表现：

调用表达式：调用表达式 **.** 标识符 与生产环境的行为完全相同：

调用表达式：调用表达式 **[** *<*标识符字符串*>* **]**

其中 <标识符字符串*>* 是一个字符串字面值，包含与标识符相同的字符序列。

生产表达式 *MemberExpression* : *MemberExpression* **[** *Expression* **]** 的求值过程如下：

1. 令 *oRef* 为对 *MemberExpression* 求值的结果

2. 令 *o* = ToObject(GetValue(oRef

3. 设 *pRef* 为表达式求值的结果

4. 令 *p* = GetValue(pRef

5. 如果 （p 的类型属于 {属性名称，任意名称}） 或者 (p 的类型为对象且 *p*的 *[[* 类 *]]* 属性**等于 "QName** a. 如果 （Type(o) 属于 {XML， XMLList}）

返回一个类型为“引用” 的值，其基对象为 *o*，其属性名为 ToXMLName(p) 的结果。 b. 否则

抛出一个**类型错误**异常

6. 返回一个类型为“引用” 的值，其基对象为 *o*，其属性名称为 ToString(p) 的结果。

生产调用表达式 *`CallExpression* : *CallExpression* **[** *Expression* **]**` 的求值方式完全相同，只是其中包含的 `*CallExpression`* 在第 1 步进行求值。

生产式 `*MemberExpression* : *MemberExpression* **.** *PropertyIdentifier*` 的行为与生产式 `MemberExpression : *MemberExpression* **[** *Expression* **]` 完全相同**， 只是所包含的 `*PropertyIdentifier*` 在第 3 步进行求值 。 同样 ， 生产式 `*CallExpression* : *CallExpression* **.** *PropertyIdentifier*` 的行为与生产式 ` *CallExpression* : *CallExpression* **[** *Expression* **]` 完全相同**，只是所包含的 `*PropertyIdentifier*` 在第 3 步进行求值。

**11.2.2 函数调用**

**语法**

E4X 重用 ECMAScript 的函数调用语法来调用类型为 XML 和 XMLList 的值的方法。ECMAScript 中函数调用的语法由以下产生式描述：

*- 49-*

参数列表，赋值表达式

**概述**

与 Object 类型的值不同，XML 和 XMLList 类型的值将属性和方法分开存储和检索， 这样 XML 方法名就不会与 XML 属性名冲突。例如，

为实现这一目标，E4X 修改了调用表达式的语义， 以调用操作 CallMethod（参见第 11.2.2.1 节）。

**语义学**

生产调用表达式：成员表达式 参数， 其求值过程如下：

1. 设 *r* 为计算 *MemberExpression* 的结果

2. 令参数 是评估参数的结果， 生成一个内部参数值列表

3. 返回调用操作 CallMethod（r, *args*）的结果。

[**11.2.2.1**](11.2.2.1) **CallMethod ( *r* , *args* )**

**概述**

在本规范中， CallMethod 抽象操作用于调用函数和方法。此操作并非语言的一部分， 而是在此定义以辅助语言规范 的说明。

当调用带有单个参数 *r* 的 *CallMethod* 操作时， 它首先检查 *r* 是否为引用。如果不是， 则尝试将 *r* 作为函数调用。然而， 如果 *r* 是引用， 它会从引用 r 中提取出基对象和属性名。然后， CallMethod 调用对象的 [[Get]] 方法来获取基对象中 具有给定属性名的属性。

注意：在查找方法时，永远不会调用 XML 和 XMLList 的 [[Get]] 方法。

如果不存在这样的属性， 并且 *base* 是大小为 1 的 XMLList， 则 CallMethod 会将方法调用委托给它所包含的单个属性。 这种处理方式有意模糊了 XML 对象和大小为 1 的 XMLList之间的区别。

如果不存在这样的属性， 并且 *base* 是一个不包含 XML 值子节点的 XML 对象（即属性、叶节点或具有简单内容的元 素）， CallMethod 会尝试将方法查找委托给叶节点中包含的字符串值。这种处理方式允许用户直接对叶节点的值执 行操作， 而无需显式选择它。例如，

*- 50-*

**语义学**

给定一个引用 *r* 和一个参数列表 *args*，操作 CallMethod 执行以下步骤：

1. 设*f* = *r*

2. 令 *base* = **null**

3. 如果 Type(r) 是引用类型 a. 令 *base* = GetBase(r

b. 如果 *base* 为**空， 则**抛出一个 **ReferenceException 异常**

c. 设 *P* = GetPropertyName(r

d. 设*f* 为以 *base* 作为 this 对象并以参数 *P* 调用 Object [[Get]] 方法的结果。 e. 如果*f* **未定义且** base 的类型为 XMLList 且 *base.[[Length]]* 等于 **1**

i. 设 *r0* 为一个新的引用， 其基对象为 *base[0]*，属性名为 *P* 。 2。递归调用 CallMethod（r0, *args*）并返回其结果。

f. 如果*f* **未定义且** base 的类型为 XML 并且 base*.hasSimpleContent*（） 的值为 **true**

i. 令 *r0* 为一个新的引用， 其基对象 = ToObject（ToString(base)），属性名称 = *P* 2。递归调用 CallMethod（r0, *args*）并返回其结果。

4. 如果 Type(f) 不是 Object， 则抛出一个 **TypeError** 异常

5. 如果*f*未实现内部的 [[Call]] 方法， 则抛出一个**类型错误**异常。

6. 如果 *base* 是一个活动对象， 则 *base* = **null**

7. 返回调用函数*f* 的 [[Call]] 方法的结果， 将 *base* 作为 **this** 值， 将列表 *args* 作为参数值。

**11.2.3 XML 后代访问器**

**语法**

E4X 通过添加一个后代访问器扩展了ECMAScript。 以下产生式描述了后代访问器的语法：

成员表达式：成员表达式 **..** 标识符 成员表达式 **..** 属性标识符

调用表达式：

调用表达式 *.***.** 标识符 调用表达式 *.***.** 属性标 识符

**概述**

当 *MemberExpression* 或 *CallExpression* 评估为 XML 对象或 XMLList 时， 后代访问器会检查其左操作数的所有后代 XML 属性（即子项、孙项、 曾孙项等）， 并按顺序返回一个包含名称与右操作数匹配的 XMLList。例如，

*- 51-*

**语义学**

生产表达式 `*MemberExpression* : *MemberExpression* **..** *Identifier`* 的求值过程如下：

1. 令 *ref* 为对 *MemberExpression* 求值的结果

2. 令 *x* = GetValue(ref

3. 如果 Type(x) 不属于 {XML， XMLList}， 则抛出一个**类型错误**异常。

4. 设 *P* 为一个字符串值， 其包含的字符序列与标识符相同。

5. 返回调用 *x* 的 [[Descendants]] 方法并传入参数 *P* 所得到的结果。

生产调用表达式 *`CallExpression* : *CallExpression* **..** *Identifier*` 的求值方式完全相同， 只是其中包含的 `*CallExpression`* 在步骤 1 中进行求值。

生产成员表达式 *`MemberExpression* **..** *PropertyIdentifier`* 的求值过程如下：

1. 令 *ref* 为 *MemberExpression* 表达式求值的结果

2. 令 *x* = GetValue(ref

3. 如果 Type(x) 不属于 {XML， XMLList}， 则抛出一个**类型错误**异常。

4. 设 *P* 为计算属性标识符的结果

5. 返回调用 *x* 的 [[Descendants]] 方法并传入参数 *P* 所得到的结果。

生产调用表达式 *`CallExpression* : *CallExpression* **..** *PropertyIdentifier*` 的求值方式完全相同， 只是其中包含的 `*CallExpression`* 在步骤 1 中进行求值。

**11.2.4 XML 过滤谓词运算符**

**语法**

E4X 通过添加一个过滤谓词运算符来扩展 ECMAScript。 以下产生式描述了过滤谓词运算符的语法：

**概述**

当左操作数计算结果为 XML 对象时， 筛选谓词会将左操作数添加到当前执行上下文的作用域链前端， 使用扩展的作用域链计算表达式， 将结果转 换为布尔值， 然后恢复作用域链。如果结果为**真**， 则筛选谓词返回一个包含左操作数的 XMLList；否则返回一个空的 XMLList。

当左操作数为 XMLList 时， 将按顺序对 XMLList 中的每个 XML 对象应用筛选谓词， 使用 XML 对象作为左操作数， 表达式作为右操作数。它会将 结果连接起来， 并以一个包含所有结果为**真的**XML 属性的单个 XMLList形式返回。例如，

过滤谓词的效果类似于 SQL 的 WHERE 子句或 XPath 的过滤谓词。

*- 52-*

例如， 以下陈述：

其结果与以下这组语句相同：

**语义学**

成员表达式 *`MemberExpression* **. （**表达式 **,***`* 的求值过程如下：

1. 令 *objref* 为对 *MemberExpression* 进行求值的结果

2. 令 *x* = GetValue(objref

3. 如果 Type(x) 不属于 {XML， XMLList}， 则抛出一个**类型错误**异常。

4. 令 *list* = ToXMLList(x

5. 令 *r* 为一个新的 XMLList， 其中 *r.[[TargetObject]]* = **null** 。

6. 对于 *i* 从 0 到 *list.[[Length]] - 1*

a. 将 *list[i]* 添加到作用域链的前端

b. 令 *ref*为使用步骤 6a 中的扩展作用域链对表达式求值的结果

c. 令 *match* = ToBoolean（GetValue(ref)） 的值

d. 从作用域链的前端移除 *list[i]*

e. 如果（match 为 **true）**， 则调用 *r* 的 [[Append]] 方法， 并将 *list[i]* 作为参数传入。

7. 回报率

生产调用表达式 *`CallExpression* : *CallExpression* **. *(*** *Expression* ***)***` 的求值方式完全相同， 只是所包含的 `*CallExpression`* 在步骤 1 中进行求值。

**11.3 一元运算符**

**11.3.1 删除操作符**

本节旨在描述 XML [[Delete]] 操作符对删除操作符的影响。E4X 未对 ECMAScript删除操作符的语法或语义进行任何扩展， 除了由 XML 和 XMLList [[Delete]] 操作符所指定的内容之外。

**语法**

E4X 重用 ECMAScript 的 delete 操作符来从 XML 对象和 XML 列表中删除 XML 属性和 XML 特性。delete 操作符的语法由以下产生式描述：

一元表达式：**delete** 一元表达式

**概述**

当一元表达式计算结果为引用 *r* 且其基对象类型为 XML 时， delete 操作符会从基对象中移除由*r* 的属性名称指定的 XML 属性或属性。 当一元表达 式计算结果为引用 *r* 且其基对象类型为 XMLList 时， delete 操作符会从基对象中移除由*r* 的属性名称指定的 XML 对象以及相关联的 XML 对象。例 如，

*- 53-*

**语义学**

E4X 扩展了 delete 操作符的语义， 提供了更复杂的 [[Delete]] 方法， 用于当一元表达式计算结果为类型为 ReferenceValue 的值且其基类型为 XML 或 XMLList 时（分别参见第 9.1.1.3 节和第 9.2.1.3 节）。如果操作数的类型为 XMLList， 则会抛出 TypeError 异常。

**11.3.2 typeof操作符**

**语法**

E4X 重用了 ECMAScript 的 typeof 操作符的语法来确定 XML 和 XMLList 对象的类型。ECMAScript 中 typeof 操作符的语法由以下产生式描述：

一元表达式：

一元表达式**类型**

**概述**

E4X 扩展了 ECMAScript 中 typeof 操作符的语义， 用于确定 XML 和 XMLList 对象的类型。 当一元表达式计算结果为 XML 类型的值时，typeof 操 作符返回字符串 **“xml”。** 当一元表达式计算结果为 XMLList类型的值时，typeof 操作符也返回字符串 **“xml”。**

**语义学**

一元表达式“**typeof** *UnaryExpression*”的求值过程如下：

1. 设 *u* 为对一元表达式求值的结果

2. 如果 Type(u) 是引用类型且 GetBase(u) **为 null， 则**返回**“未定义”**

3. 根据以下表格， 返回由 Type（GetValue(u)） 决定的字符串：

**类型**

未定义 空值

布尔型 数字

字符串 XML

XML 列表

对象无法 调用

（原生和实现）

（“[[Cal l]]”）

对象（原生及实现 [[Call]]）

**结果**

**“未定义” “物体”**

**布尔型**

**“数字”**

**“字符串”**

**XML**

**“ ”**

**XML**

**“ ”**

**“物体”**

**“功能”**

*- 54-*

**11.4 加法运算符**

**语法**

E4X 重用了 ECMAScript加法运算符的语法来连接两个类型为 XML 或 XMLList 的值。ECMAScript 中加法运算符的 语法由以下产生式描述：

加法表达式：加法表达式 **+** 乘法表达式

**11.4.1 加法运算符（+）**

|  |  |
| --- | --- |
| **概述** |  |
| E4X 扩展了 | ECMAScript 加法运算符的语义， 使其根据操作数的不同执行字符串连接、XML 和 XMLList连接或数值 |
| 相加操作。 |  |

当加法表达式和乘法表达式都计算为 XML 对象或 XMLList 时， 加法运算符首先会创建一个新的空 XMLList作为返 回值。 如果左操作数计算为 XML 对象， 则将其添加到返回值中。 如果左操作数计算为 XMLList， 则按顺序将

XMLList 的每个 XML 属性添加到返回值中。 同样， 如果右操作数计算为 XML 对象， 则将其添加到返回值中。否则， 如果它是 XMLList， 则按顺序将 XMLList 的每个 XML 属性添加到返回值中。

例如，

注意：使用加法运算符对类型为 XML 和 XMLList 的操作数进行操作时，结果始终为 XMLList。如果需要对 XML 对象进行数值相 加，则应将操作数显式转换为数字。这可以通过使用一元“+”运算符或 Number 转换函数来实现。例如，

同样， 当需要连接 XML 对象时， 至少有一个操作数应显式地转换为字符串。这可以通过将其与空字符串（“ ”）连 接或使用字符串转换函数来实现。例如，

**语义学**

加法表达式“*AdditiveExpression* : *AdditiveExpression* **+** *MultiplicativeExpression*” 的计算方式如下：

1. 设 *a* 为 *AdditiveExpression* 的计算结果。

*- 55-*

2. 令 *left* = GetValue(a

3. 设 *m* 为计算乘法表达式的结果。

4. 令 *right* = GetValue(m

5. 如果（left 的类型属于 {XML， XMLList}）并且（right 的类型属于 {XML， XMLList}）

a. 让 *list* 成为一个新的 XMLList 对象

b. 调用列表的 [[Append]] 方法，并将 *x* 作为参数传入。

c. 调用列表的 [[Append]] 方法，并将参数设为 *y*

d. 返还清单

6. 令 *pLeft* = ToPrimitive(left

7. 令 *pRight* = ToPrimitive(right

8. 如果 pLeft 的类型为字符串或者 pRight 的类型为字符串

返回将 pLeft 的字符串表示形式与 pRight 的字符串表示形式连接起来的结果。

9. 否则

a. 对 ToNumber（pLeft） 和 ToNumber（pRight） 应用加法运算，并返回结果。详情请参阅《ECMAScript 第 3 版》第 11.6.3 节。

**11.5 等号运算符**

**11.5.1 抽象相等比较算法**

**概述**

E4X 扩展了由 ECMAScript定义的抽象相等比较算法， 以支持涉及 QName 和 Namespace 对象以及 XML 和 XMLList类型的相等比较。

**语义学**

比较表达式 *x* ==*y* 中，*x* 和 *y* 为值，其结果为**真**或**假。**此比较通过以下步骤执行：

1. 如果 Type(x) 是 XMLList，则调用 *x* 的 [[Equals]] 方法并将 *y* 作为参数传入，然后返回结果。

2. 如果 Type(y) 是 XMLList 类型，则调用 *y* 的 [[Equals]] 方法并将 *x* 作为参数传入，然后返回结果。

3. 如果 Type(x) 与 Type(y) 相同

a. 如果 Type(x) 是 XML，

如果（（x 的 [[Class]] 属于 {"text**", "attribute"}** 并且 y 具有简单内容）或者（y 的 [[Class]] 属于 {"text**", "attribute"}** 并且 x 具有简单内容）

1. 返回比较结果 ToString(x) == ToString(y) 的值 2。否则，返回将 *x* 的 [[Equals]] 方法与参数 *y* 调用的结果。

b. 如果 Type(x) 是 Object 且 *x.[[Class]]* 等于 **“QName”** 并且 *y.[[Class]]* 也等于 **"QName**

如果比较结果 *x.u*ri ==*y.u*ri 为**真**且比较结果 *x.localName* ==*y.localName* 也为**真，则返回 true**。否则，返回 **false**。

c. 如果 Type(x) 为 Object 且 *x.[[Class]]* 等于 **“Namespace”** 并且 *y.[[Class]]* 也等于 **“Namespace”**，则返回 *x.u*ri 与 *y.uri* 比较的结果。

d. 如果 Type(x) **未定义**，则返回 **true** e. 如果 Type(x) 为 **null，则**返回 **true** f. 如果 Type(x) 是 Number 类型

如果 *x* 是 **NaN**，则返回 **false**

2。如果 *y* 是 **NaN**，则返回 **false**

3。如果 *x* 与 *y* 的数值相同，则返回 **true** 。 iv. 如果 *x* 是 **+0** 且 *y* 是 **-0**，则返回 **true**

v. 如果 *x* 是 **-0** 而 *y* 是 **+0**，则返回 **true** vi. 返回 **false**

g. 如果 Type(x) 是字符串类型，则当 *x* 和 *y* 是完全相同的字符序列（长度相同且对应位置的字符相同）时返回 **true**，否则返回 **false**。 h. 如果 Type(x) 的类型为布尔值，则当 *x* 和 *y* 都为**真或都为假时返回 true**，否则返回 **false**。

如果 *x* 和 *y* 指向同一个对象， 或者它们指向彼此关联的对象（ECMAScript 第 3 版第 13.1.2 节）， 则返回 **true**。否则， 返回 **false**。

*- 56-*

4. 如果（x 的类型为 XML 且 x 的内容为简单内容**）**或者（y 的类型为 XML 且 *y* 的内容为简单**内容）** 返回比较结果 ToString(x) == ToString(y) 的值

5. 如果 *x* 为 **null** 且 *y* **未定义**， 则返回 **true**

6. 如果 *x* **未定义且** *y* **为 null， 则**返回 **true**

7. 如果 Type(x) 是数字类型且 Type(y) 是字符串类型， 则返回 *x* == ToNumber(y) 的比较结果。

8. 如果 x 的类型为字符串且 y 的类型为数字， 则返回 ToNumber(x) ==*y* 的比较结果。

9. 如果 Type(x) 为布尔值， 则返回 ToNumber(x) ==*y* 的比较结果。

10. 如果 Type(y) 是布尔值， 则返回比较结果 *x* == ToNumber(y) 的值。

11. 如果 Type(x) 为字符串或数字， 且 Type(y) 为对象， 则返回 *x* == ToPrimitive(y) 的比较结果。

12. 如果 Type(x) 为对象且 Type(y) 为字符串或数字， 则返回 ToPrimitive(x) ==*y* 的比较结果。

13. 返回 **false 。**

**11.6 赋值运算符**

**11.6.1 XML 赋值运算符**

本节旨在描述 XML [[Put]] 操作符对赋值操作符的影响。E4X 未对 ECMAScript赋值操作符的语法或语义进行任何扩 展， 除了由 XML 和 XMLList [[Put]] 操作符所规定的那些扩展之外。

**语法**

E4X 重用 ECMAScript赋值运算符来修改、替换和插入 XML 对象中的属性和 XML 属性。ECMAScript 中赋值运算符 的语法由以下产生式描述：

赋值表达式：左值表达式 **=** 赋值表达式

**概述**

赋值运算符首先对左操作数表达式进行求值，该表达式解析为一个由基对象*parent* 和属性名组成的引用 *r*。如果 *parent* 是一个 XML 对象， 则赋值运算符执行第 节中描述的步骤（如果*parent* 是一个 XMLList， 则执行的步骤见第 11.6.2 节）。

如果属性名称以字符“@ ”开头，XML 赋值运算符将在父节点中创建或修改一个 XML 属性。如果指定的 XML 属性 已存在， 则赋值运算符会修改其值， 否则会创建一个具有给定名称和值的新 XML 属性。如果 *AssignmentExpression* 的计算结果为 XMLList， 则指定属性的值将是一个由空格分隔的值列表（即 XML 属性列表）， 通过将 XMLList 中的 每个值转换为字符串并用空格连接结果来构建。如果 *AssignmentExpression* 的计算结果不是 XMLList， 则指定属性的 值将通过计算 *AssignmentExpression* 并对结果调用 ToString 方法来得出。例如，

如果属性名称是一个数组索引，XML 赋值运算符将抛出一个**类型错误**异常。此操作保留用于 E4X 的未来版本。

如果属性名称不以“@ ”开头且不是数组索引， 则 XML 赋值运算符将通过 XML 名称在父对象中替换、修改或追加 一个或多个 XML 对象。如果仅存在一个具有该名称的 XML 对象， 那么

*- 57-*

如果给定名称存在且赋值表达式计算结果为 XML 对象或 XML 列表， 则赋值运算符会用给定值替换所标识的 XML 对象。如果不存在具有给定名称的 XML 属性， 则会将具有给定名称和值的新 XML 属性添加到父级的末尾。如果存 在多个具有给定名称的 XML 对象且赋值表达式计算结果为 XML 对象或 XML 列表， 则赋值运算符会用给定值替换 第一个具有匹配名称的 XML 属性， 并删除具有给定名称的其余 XML 属性， 实质上是用给定值替换所有具有给定名 称的 XML 对象。如果赋值表达式计算结果不是 XML 对象或 XML 列表， 则赋值运算符会调用给定值的 ToString 方 法， 并替换相应 XML 对象的属性（即内容）， 而不是替换 XML 对象本身。这为将具有指定名称的 XML 属性的值 设置为基本值提供了一种简单直观的语法。例如，

**语义学**

E4X 通过提供更复杂的 [[Put]] 方法来扩展赋值运算符的语义， 这些方法在 *MemberExpression* 评估结果为 XML 或 XMLList 类型的值时使用（分别参见第 9.1.1.2 节和第 9.2.1.2 节）。

**11.6.2 XMLList 赋值运算符**

本节旨在说明 XMLList [[Put]] 操作符对赋值操作符的影响。E4X 除了通过 XML 和 XMLList [[Put]] 操作符所提供的 内容外， 未对 ECMAScript赋值操作符的语法或语义进行任何扩展。

**语法**

E4X 重用 ECMAScript赋值运算符来替换或追加 XML 列表及其相关联的 XML 对象的值。ECMAScript 中赋值运算符 的语法由以下产生式描述：

**概述**

赋值运算符首先计算左操作数表达式，该表达式解析为一个由基对象*parent* 和属性名组成的引用 *r*。如果*parent* 是

XMLList 类型， 则赋值运算符执行本节中描述的步骤（有关*parent* 为 XML 对象时执行的步骤， 请参阅第 11.6.1 节）。

如果属性名称不是数组索引，XMLList 赋值运算符会检查此 XMLList对象是否仅包含一个项且该项为 XML 类型。 如果是这样，XMLList 赋值运算符会将其行为委托给所包含 XML 对象的 [[Put]] 方法（请参阅第 11.6.1 节）。这种 处理方式有意模糊了单个 XML 对象与仅包含一个 XML 对象的 XMLList之间的区别。例如，

在上述第一个语句中， 表达式“order.customer”返回一个仅包含一个 XML 项的 XMLList。表达式“order.customer. name”会将此 XMLList 隐式转换为 XML 值， 并将值“Fred Jones”赋给该值。

*- 58-*

如果属性名称是一个数组索引， 赋值运算符会替换 XMLList 中由属性名称标识的属性， 或者如果不存在具有该属性 名称的属性， 则添加一个新的属性。此外， 如果由属性名称标识的属性是一个具有非空父节点的 XML 值， 则该

XML 值也会在其父节点的上下文中被替换。如果赋值表达式计算结果为 XML 值， 则赋值运算符会将由属性名称标 识的属性的值替换为给定 XML 对象的深拷贝。如果赋值表达式计算结果为 XMLList， 则赋值运算符会按顺序将

XMLList 中每个项目的深拷贝替换为由属性名称标识的属性的值， 从而有效地删除原始属性， 并在其位置插入

XMLList 的内容。如果赋值表达式计算结果不是 XML 或 XMLList类型的值， 则赋值运算符会调用给定值的 ToString 方法， 并将给定位置的属性替换为结果。 以下是一些示例：

**语义学**

E4X 通过提供更复杂的 [[Put]] 方法来扩展赋值运算符的语义， XMLList 类型的值时使用（分别参见第 9.1.1.2 节和第 9.2.1.2 节）

这些方法在 *MemberExpression* 评估为 XML 或

。

**11.6.3 复合赋值（op=）**

本节旨在描述 XML 和 XMLList 的 [[Get]]、[[Put]] 以及加法运算符对复合赋值运算符的影响 。E4X 并未对

ECMAScript 复合赋值运算符的语法或语义进行任何扩展， 仅提供了 XML 和 XMLList 的 [[Get]]、[[Put]] 以及加法运 算符的相关内容。

|  |  |
| --- | --- |
| **语法** |  |
| E4X 可以利用复合赋值运算符 | “+= ”， 而无需额外的 ECMAScript扩展。复合赋值运算符“+= ”的语法由以下产生 |
| 式描述： |  |

赋值表达式：左值表达式 **+=** 赋值表达式

**概述**

表达式 *LeftHandSideExpression* += *AssignmentExpression* 是表达式 LeftHandSideExpression = *LeftHandSideExpression* + *AssignmentExpression* 的简写形式 ， 二者行为相同 。 当 *LeftHandSideExpression* 标识一个或多个 XML 对象 ， 且

*AssignmentExpression* 标识一个或多个 XML 对象时，“+= ”运算符会将 *LeftHandSideExpression* 标识的 XML 对象替 换为将 *LeftHandSideExpression* 标识的 *XML* 对象与 *AssignmentExpression* 标识的 XML 对象相连接的结果。这通常会 将 *AssignmentExpression* 指定的 XML 对象插入到 *LeftHandSideExpression* 指定的 XML 对象之后， 位于其父对象的上 下文中。例如， 在执行以下语句之后，

变量“e ”将包含 XML 值：

*- 59-*

同样地， 在评估了以下陈述之后，

变量“e ”将包含 XML 值：

但请注意，“+= ”操作不一定能保持由 *LeftHandSideExpression* 指定的 *XML* 对象的同一性。

**语义学**

E4X 通过提供更复杂的 [[Get]] 和 [[Put]] 方法来扩展复合赋值运算符的语义， 当 *MemberExpression* 评估为 XML 或 XMLList 类型的值时会使用这些方法（分别参见第 9.1.1.1 节、第 9.1.1.2 节、第 9.2.1.1 节和第 9.2.1.2 节）。

**12 项声明**

E4X 通过以下语法扩展了 ECMAScript 中提供的语句：

**12.1 默认的 XML 命名空间声明**

**语法**

E4X 通过添加默认的 XML 命名空间语句来扩展 ECMAScript。 以下产生式描述了默认 XML 命名空间语句的语 法：

**概述**

默认的 XML 命名空间语句会设置与当前执行上下文相关联的变量对象的内部属性 [[DefaultNamespace]] 的值 （参见 ECMAScript第 3 版第 10 节）。全局作用域的默认 XML 命名空间的初始值为空命名空间。如果默认 XML 命名空间语句出现在 *FunctionDeclaration* 内部 ， 则会向激活对象的变量对象添加内部属性

[[DefaultNamespace]]， 并将其初始值设置为空命名空间。 此 [[DefaultNamespace]] 属性会隐藏外部作用域的 [[DefaultNamespace]] 属性。

*- 60-*

当执行默认的 XML 命名空间语句时， 它会计算表达式， 将结果转换为字符串 *s*， 创建一个新的命名空间对象 *n*， 就 好像通过调用构造函数 *n* = new Namespace（"", *s*）一样， 并将当前执行上下文关联的默认 XML 命名空间设置为 n。 在当前作用域中， 位于默认 XML 命名空间声明之后的未限定 XML 元素名称将与由表达式指定的默认 XML 命名空 间相关联。例如，

**语义学**

生产规则“*DefaultXMLNamespaceStatement* : **default xml namespace =** 表达式” 的求值过程如下：

1. 令 *uriRef* 为表达式求值的结果

2. 令 *uri* = 取值(uriRef

3. 令 *namespace* 为一个新的命名空间对象， 其创建方式如同调用构造函数 new Namespace（"", *uri*）一样。

4. 令 *varObj* 为与当前执行上下文相关联的变量对象（参见 ECMAScript 第 3 版第 10.1.3 节）

5. 令 *varObj.[[DefaultNamespace]]* = *namespace*

**12.1.1 GetDefaultNamespace ( )**

**概述**

在本规范中， 使用“GetDefaultNamespace”抽象操作符来获取默认 XML 命名空间的值。此操作符并非语言的一部分， 而是在此定义以辅助语言规范的说明。

**语义学**

当调用内部的 GetDefaultNamespace 方法时， 将执行以下步骤：

1. 当（作用域链上还有更多对象时）

a. 令 *o* 成为作用域链中的下一个对象

注意，在第一次迭代中，*o* 将是作用域链上的第一个对象。

b. 如果对象 *o* 具有内部属性 [[DefaultNamespace]]， 则返回 *o.[[DefaultNamespace]]* 2. 抛出一个**类型错误**异常

注意， 此步骤不应发生， 因为全局对象具有 [[DefaultNamespace]] 属性。

*- 61-*

**12.2 for-in 语句**

**语法**

E4X 扩展了用于遍历对象属性的 ECMAScript for-in 语句的语义。for-in 语句的语法由以下产生式指定：

**概述**

for-in 语句会计算表达式 ， 并遍历结果对象的每个属性 。对于每个属性 ，for-in 语句会将属性名赋值给由 *LeftHandSideExpression* 或 *VariableDeclarationNoLn* 标识的变量，并计算语句。例如，

当表达式计算结果为 XML 对象时，for-in 语句会将该 XML 对象转换为 XMLList， 并对生成的 XMLList进行迭代。这种处理有意模糊了单个 XML 对象与仅包含一个值的 XMLList之间的区别。

**语义学**

生产迭代语句：**for**（ 左值表达式 ***in***表达式 ） 语句 的求值过程如下：

1. 令 *ref*为表达式求值的结果

2. 令 *e* = GetValue(ref

3. 如果 Type(e) 是 XML，则 a. 令 *list* = ToXMLList(e

4. 否则，如果 Type(e) 不是 XMLList 类型， 则令 *list* = ToObject(e)

5. 令 *V* = **空集**

6. 当（列表还有更多属性时）

a. 设 *p* 为列表中的下一个属性（见下文注释）

b. 如果*p* 没有 DontEnum 属性

i. 设 *i* 为对 *LeftHandSideExpression* 进行求值的结果

2。设 *name* 为属性*p* 的名称

3。将值赋给索引 i 的名称 *name* iv. 设 *s* 为对语句进行求值的结果。

v. 如果 *s.value* 不为**空**，则令 *V* = *s.value*

vi. 如果 *s.t*ype 为 **break** 且 *s.t*arget 在当前标签集中，则返回 (正常， *V*， **空**

7。如果（s.type 不是 **continue）**或者（s.target 不在当前标签集中）

1. 如果 *s* 是突然完成的，则返回 *s*

7. 返回（正常， *V*，**空）**

生产迭代语句：**for** ( **var** *VariableDeclarationNoLn* ***in Expression*** ) *Statement* 的求值过程如下：

*- 62-*

1. 令 *varRef* 为对 *VariableDeclarationNoLn* 进行求值的结果

2. 令 *ref*为表达式求值的结果

3. 令 *e* = GetValue(ref

4. 如果 e 的类型为 XML

a. 令 *list* = ToXMLList(e

5. 否则，如果 e 的类型不是 XMLList 类型 a. Let *list* = ToObject(*e*)

6. 令 *V* = **空集**

7. 当（列表还有更多属性时）

a. 设 *p* 为列表中的下一个属性（见下文注释）

b. 如果*p* 没有 DontEnum 属性

i. 假设将 *varRef* 作为标识符进行求值，设 i 为该求值的结果（参见 ECMAScript 第 3 版第 11.1.2 节）

2。设 *name* 为属性*p* 的名称

3。将值赋给索引 i 的名称 *name* iv. 设 *s* 为对语句进行求值的结果。

v. 如果 *s.value* 不为**空**，则令 *V* = *s.value*

vi. 如果 *s.t*ype 为 **break** 且 *s.t*arget 在当前标签集中，则返回 (正常， *V*， **空**

7。如果（s.type 不是 **continue）**或者（s.target 不在当前标签集中）

1. 如果 *s* 是突然完成的，则返回 *s*

8. 返回（正常， *V*，**空）**

枚举的顺序由对象定义（第一个算法中的步骤 6 和 6a 以及第二个算法中的步骤 7 和 7a）。当 *e* 评估为 XML 或 XMLList类型的值时，属性将按照 其数字属性名称的升序顺序进行枚举（即对于 XML 对象为文档顺序）。其他对象的属性则按照实现依赖的顺序进行枚举。

枚举属性的机制（第一个算法中的步骤 6 和 6a 以及第二个算法中的步骤 7 和 7a）取决于具体实现。在枚举过程中，被枚举对象的属性可能会被 删除。如果在枚举过程中尚未访问的属性被删除，则该属性将不会被访问。如果在枚举过程中向被枚举对象添加了新属性，则不能保证在当前枚 举中会访问这些新添加的属性。枚举对象的属性包括递归地枚举其原型以及原型的原型等的属性；但如果原型链中某个之前的对象具有同名属性， 则不会枚举该原型的属性。

**12.3 for-each-in 语句语法**

E4X 通过添加一个用于遍历对象属性值的 for-each-in 语句来扩展 ECMAScript。for-each-in 语句的语法由以下产生式指定：

迭代语句：

**对于**表达式中的每个（**左值**表达式）**执行**语句

**对于每个**（**变量声明（无行尾分号）in**表达式）语句

**概述**

for-each-in 语句会计算表达式的值，并遍历结果对象的每个属性。对于每个属性，for-each-in 语句会将属性的值赋给由 *LeftHandSideExpression* 或 *VariableDeclarationNoLn* 标识的变量，并计算语句。例如，

*- 63-*

在上面的第一个 for-each-in 语句中，表达式 “e..name” 返回一个 XMLList，其中包含 XML 对象 “e” 中所有名为 “name” 的后代 XML 属性。 for-each-in 语句按顺序遍历 XMLList对象中的每个属性。对于列表中的每个 XML 属性， 它将 XML 属性的值赋给变量 “n” 并执行花括号中嵌 套的代码。 同样，在上面的第二个 for-each-in 语句中，表达式 “order.item[0].\*” 返回一个 XMLList， 其中包含名为 “order” 的 XML 对象中第 一个名为 “item” 的 XML 对象的所有子节点。for-each-in 语句按顺序遍历 XMLList 对象中的每个属性， 将 XML 属性的值赋给变量 “child” 并执行花括号中嵌套的代码。

当表达式计算结果为一个 XML 对象时，for-each-in 语句会将该 XML 对象转换为一个 XMLList， 并对生成的 XMLList进行迭代。这种处理有意 模糊了单个 XML 对象与仅包含一个值的 XMLList之间的区别。

注意：for-each-in 语句与 for-in 语句的行为有所不同。特别是， 它将循环变量赋值于对象的范围， 而非对象的域。也就是说， for-each-in 语句将循环变量绑定到 给定对象的属性值， 而非属性名。

**语义学**

生产迭代语句：**for each**（ 左值表达式 ***in***表达式 ） 语句 的求值过程如下：

1. 令 *ref*为表达式求值的结果

2. 令 *e* = GetValue(ref

3. 如果 Type(e) 是 XML，则 a. 令 *list* = ToXMLList(e

4. 否则，如果 Type(e) 不是 XMLList 类型， 则令 *list* = ToObject(e)

5. 令 *V* = **空集**

6. 当（列表还有更多属性时）

a. 设 *p* 为列表中的下一个属性（见下文注释）

b. 如果*p* 没有 DontEnum 属性

i. 设 *i* 为对 *LeftHandSideExpression* 进行求值的结果

2。令 *value* 为属性*p* 的值 3。将值赋给索引 i：value

iv. 设 *s* 为对语句进行求值的结果。

v. 如果 *s.value* 不为**空**，则令 *V* = *s.value*

vi. 如果 *s.t*ype 为 **break** 且 *s.t*arget 在当前标签集中，则返回 (正常， *V*， **空**

7。如果（s.type 不是 **continue）**或者（s.target 不在当前标签集中）

1. 如果 *s* 是突然完成的，则返回 *s*

7. 返回（正常， *V*，**空）**

对于生产环境中的迭代语句：**`for each** ( **var** *VariableDeclarationNoLn* ***in Expression*** ) *Statement*`，其求值过程如下：

1. 令 *varRef* 为对 *VariableDeclarationNoLn* 进行求值的结果

2. 令 *ref*为表达式求值的结果

3. 令 *e* = GetValue(ref

4. 如果 e 的类型为 XML

a. 令 *list* = ToXMLList(e

*- 64-*

5. 否则， 如果 e 的类型不是 XMLList 类型

a. 令 *list* = ToObject(e) 译文：a. 令 list = 对象化(e

6. 令 *V* = **空集**

7. 当（列表还有更多属性时）

a. 设 *p* 为列表中的下一个属性（见下文注释）

b. 如果*p* 没有 DontEnum 属性

i. 假设将 *varRef* 作为标识符进行求值， 其结果记为 i（参见 ECMAScript 第 3 版第 11.1.2 节）

2。令 *value* 为属性*p* 的值 3。将值赋给索引 i：value

iv. 设 *s* 为对语句进行求值的结果。

v. 如果 *s.value* 不为**空**， 则令 *V* = *s.value*

vi. 如果 *s.t*ype 为 **break** 且 *s.t*arget 在当前标签集中， 则返回 (正常， *V*， **空**

7。如果（s.type 不是 **continue）**或者（s.target 不在当前标签集中）

1. 如果 *s* 是突然完成的， 则返回 *s*

8. 返回（正常， *V*， **空）**

枚举的顺序由对象定义（第一个算法中的步骤 6 和 6a 以及第二个算法中的步骤 7 和 7a）。当 *e* 评估为 XML 或 XMLList类型的值时， 属性将按照 其数字属性名称的升序顺序进行枚举（即对于 XML 对象， 按照文档顺序）。

枚举对象属性的机制（第一个算法中的步骤 6 和 6a， 第二个算法中的步骤 7 和 7a）取决于具体实现。在枚举过程中， 被枚举对象的属性可能会被删 除。如果在枚举过程中尚未访问的属性被删除， 则该属性可能不会被访问。如果在枚举过程中向被枚举对象添加了新属性， 则不能保证在当前枚举 中会访问到这些新添加的属性。枚举对象的属性包括递归枚举其原型以及原型的原型等的属性；但如果原型链中某个之前的对象具有同名属性， 则 不会枚举该原型的属性。

**13 原生 E4X对象**

E4X 为 ECMAScript 增加了四个原生对象， 分别是命名空间对象（Namespace 对象）、 限定名对象（QName 对象）、XML 对象和 XML 列表对象 （XMLList对象）。此外， E4X 还向全局对象添加了新的属性。

**13.1 全球目标**

**13.1.1 全局对象的内部属性**

E4X 通过向全局对象添加以下内部属性来扩展 ECMAScript。

[**13.1.1.1**](13.1.1.1) **[[默认命名空间]]**

**概述**

最初， 全局对象具有一个内部属性 [[DefaultNamespace]]， 其值被设置为一个表示无命名空间的命名空间对象， 该对象的创建方式就如同调用未传入 任何参数的 Namespace 构造函数一样。 因此， 除非使用默认 XML 命名空间语句（参见第 12.1 节）另行指定， 否则用于指定 XML 对象属性的未限 定名称将与无命名空间的 XML 属性相匹配。

**13.1.2 全局对象的功能属性**

E4X 通过向全局对象添加以下函数属性来扩展 ECMAScript。

[**13.1.2.1**](13.1.2.1) **isXMLName ( 值**

**概述**

isXMLName 函数会检查给定的值， 并确定其是否为有效的 XML 名称， 该名称可用于 XML 元素或属性名称。如果是， 则返回 **true**， 否则返回 **false。**

*- 65-*

**语义学**

当调用 isXMLName 函数并传入一个参数值时，将执行以下步骤：

1. 令 *q* 为通过调用构造函数 new QName（value） 创建的新 QName，如果抛出 **TypeError** 异常，则返回 **false** 。

2. 如果 *q.localName* 与生产名称 *NCName* 不匹配，则返回 **false** 。

3. 返回**真**

其中的生产 *NCName* 在 XML 名称空间规范的第 2 节中有定义。

**13.1.3 全局对象的构造函数属性**

E4X 通过向 ECMAScript 添加以下构造函数属性来扩展 ECMAScript。

[**13.1.3.1**](13.1.3.1) **命名空间（……）**

请参阅第 13.2.1 节和第 13.2.2 节。

[**13.1.3.2**](13.1.3.2) **限定名（QName）（...）**

请参阅第 13.3.1 节和第 13.3.2 节。

[**13.1.3.3**](13.1.3.3) **XML（……）**

请参阅第 13.4.1 节和第 13.4.2 节。

[**13.1.3.4**](13.1.3.4) **XMLList（……）**

请参阅第 13.5.1 节和第 13.5.2 节。

**13.2 命名空间对象**

命名空间对象表示 XML 命名空间， 并提供命名空间前缀与唯一资源标识符（URI）之间的关联。前缀要么是**未定义**的值， 要么是一个字符串值， 可用于在 XML 值的词法表示中引用该命名空间。 当使用 ToXMLString（） 方法将包含**未定义**前缀的命名空间的 XML 对象编码为 XML 时， 实现将自动生成一个前缀。URI 是 一个字符串值，用于唯一标识命名空间。

**13.2.1 语 法作为函数调用的命名空间构造函数**

命名空间 ( )

命名空间（*uriValue*）命名空间（前缀值，*uriValue*）

**概述**

如果将 `Namespace` 构造函数作为函数调用， 并且恰好传入一个 `Namespace` 对象作为参数，则返回该参数而不作任何更改。否则，将创建一个新的 `Namespace` 对象并返回，就好像将相同的参数传递给对象创建表达式 `**new Namespace(...)` 一**样。请参阅第 13.2.2 节。

**语义学**

当将 Namespace 作为函数调用时，若未传入参数、传入一个参数 *uriValue* 或传入两个参数*prefix Value* 和 *uriValue*，则会执行以下步骤：

1. 如果（未指定前缀值且 uriValue 的类型为对象且 *uriValue* 的 *[[Class]]* 属性等于**“Namespace”）**

a. 返回 *uriValue*

*- 66-*

2. 创建并返回一个新的命名空间对象， 其方式与使用相同的参数调用命名空间构造函数时完全相同（第 13.2.2 节）。

**13.2.2 命名空间构造函数**

**语法**

新的命名空间 ()

Namespace *( uriValue )* new Namespace ( 前 缀 *uriValue*

new

值，

**概述**

当在 new表达式中调用 Namespace 时， 它作为一个构造函数， 会创建一个新的 Namespace 对象。

新构造对象的 [[Prototype]] 属性被设置为原始的命名空间原型对象， 即 Namespace.prototype 的初始值（第 13.2.4 节）。新构造对象的 [[Class]] 属性 被设置为 “Namespace”。

如果未指定任何参数， 则命名空间 *URI* 和前缀将被设置为空字符串。将 *URI* 设置为空字符串的命名空间表示无命名空间。在 XML 对象中， 无命名 空间用于明确指定某个名称不在任何命名空间内， 并且该名称永远不能与除空字符串之外的任何前缀相关联。

当仅指定了 *uriValue* 参数且 *uriValue* 是一个 Namespace 对象时， 将返回 *uriValue* 的副本。 当仅指定了 *uriValue* 且其为空字符串时， 前缀将被设置为 空字符串。在所有其他仅指定了 *uriValue* 的情况下， 命名空间前缀将被设置为**未定义**值。

当指定了前缀值参数且将其设置为空字符串时， 该命名空间被称为默认命名空间。默认命名空间用于 XML 对象中， 以隐式指定未指定限定符的限定 名称的命名空间。

**语义学**

当调用命名空间构造函数时， 若未传入参数、传入一个参数 *uriValue* 或传入两个参数*prefix Value* 和 *uriValue*， 则执行以下步骤：

1. 创建一个新的命名空间对象 *n*

*- 67-*

2。否则抛出一个**类型错误**异常

d. 否则， 如果*prefix Value* **未定义**， 则令 *n.prefix* = **未定义**

e. 否则， 如果 isXMLName（prefixValue） 的值为 **false** i. 让 *n.prefix* = **未定义**

否则，令 *n.p*refix = ToString(prefixValue

5. 返回 *n*

**13.2.3 命名空间构造函数的属性**

Namespace 构造函数的内部 [[Prototype]] 属性的值是 Function 原型对象。

除了内部属性和长度属性（其值为 2）之外，Namespace 构造函数还具有以下属性。

[**13.2.3.1**](13.2.3.1) **Namespace.prototype**

Namespace.prototype 属性的初始值是 Namespace 的原型对象（第 13.2.4 节）。

此属性具有 { 不可枚举、不可删除、只读 } 属性。

**13.2.4 命名空间原型对象的属性（内置方法）**

|  |  |
| --- | --- |
| 命名空间原型对象本身就是一个命名空间对象 | （其 [[Class]] 为“Namespace”）， 其 *uri* 和 *prefix* 属性均被设置为空 |
| 字符串。 |  |

命名空间原型对象的内部 [[Prototype]] 属性的值是对象原型对象（ECMAScript 第 3 版第 15.2.3.1 节）。

[**13.2.4.1**](13.2.4.1) **Namespace.prototype.constructor**

`Namespace.prototype.constructor` 的初始值是内置的 `Namespace` 构造函数。

[**13.2.4.2**](13.2.4.2) **Namespace.prototype.toString()**

**概述**

toString（） 方法返回此命名空间对象的字符串表示形式。

`toString` 函数不是泛型的。如果其 `**this**` 值不是 `Namespace` 对象， 它会抛出一个 `**TypeError**` 异常。 因此， 它不能 被转移到其他类型的对象上用作方法。

**语义学**

当对命名空间对象 *n* 调用toString 方法且未传入任何参数时， 将执行以下步骤：

1. 如果 Type(n) 不是 Object 或者 *n.[[Class]]* 不等于 **“Namespace”** ， 则抛出一个**类型错误**异常。

2. 返回 *n.uri*

**13.2.5 命名空间实例的属性**

命名空间实例从命名空间原型对象继承属性， 并且还具有前缀属性和 *URI* 属性。

[**13.2.5.1**](13.2.5.1) **前缀**

前缀属性的值要么是**未定义**值， 要么是字符串值。当前缀属性的值为空字符串时， 该命名空间被称为默认命名空间。 默认命名空间用于 XML 对象中， 以确定未指定限定符的名称的命名空间。

此属性具有 { 不可删除， 只读 } 属性。

*- 68-*

[**13.2.5.2**](13.2.5.2) **统一资源标识符（URI）**

*uri* 属性的值是一个字符串值。当 *uri*属性的值为空字符串时，命名空间表示无名命名空间。在 XML 对象中， 无名命名空间用于明确指定某个名称不在 任何命名空间内。

此属性具有 { 不可删除，只读 } 属性。

**13.3 QName 对象**

QName 对象用于表示 XML 元素和属性的限定名。每个 QName 对象都有一个类型为字符串的本地名和一个类型为字符串或 null 的命名空间 URI。当命 名空间 URI**为 null 时**， 此限定名与任何命名空间匹配。

实现方式可能包含一个内部的 [[Prefix]] 属性， 该属性对于 E4X 用户来说是不可见的。 当使用 Namespace 对象参数创建 QName 对象时， 内部的 [[Prefix]] 属性可用于保留 Namespace 对象的前缀。如果关联的 Namespace 未指定命名空间前缀， 则 [[Prefix]] 属性可能**未定义。**

QName 类型的值可以使用限定标识符来指定。如果 XML 元素的 QName 未指定命名空间（即作为未限定标识符）， 则关联的 QName 的 *uri*属性将设置 为作用域内的默认命名空间（第 12.1 节）。如果 XML 属性的 QName 未指定命名空间， 则关联的 QName 的 *uri*属性将是表示无命名空间的空字符串。

**13.3.1 作为函数调用的 QName 构造器**

**语法**

QName ()

QName（名称）QName（命名空间，名称）

**概述**

如果将 QName 构造函数作为函数调用， 并且恰好有一个参数为 QName 对象， 则返回该参数而不作任何更改。否则， 将创建一个新的 QName 对象并返 回， 就好像将相同的参数传递给对象关系表达式 **new QName（...） 一**样。请参阅第 13.3.2 节。

**语义学**

当调用 QName 函数时， 将执行以下步骤。

1. 如果未指定命名空间且 Type（Name） 为 Object 且 *Name.[[Class]]* 等于 **“QName”**， 则返回 *Name*。

2. 创建并返回一个新的 QName 对象， 其方式与使用相同的参数调用 QName 构造函数时完全相同（第 13.3.2 节）。

**13.3.2 QName 构造函数**

**语法**

新的 QName ()

新的 QName（名称）

新的 QName（命名空间，名称）

**概述**

当在新表达式中调用 QName 时， 它是一个构造函数， 并会创建一个新的 QName 对象。

*- 69-*

新构造对象的 [[Prototype]] 属性被设置为原始的 QName 原型对象， 即 QName.prototype 的初始值（第 13.3.3.1 节）。新构造对象的 [[Class]] 属性被设置为 “QName”。

如果**未定义**或未指定名称， 则使用空字符串作为名称。

如果 *Name* 是一个带命名空间的名称（QName）， 而 Namespace 未指定， 则 QName 构造函数将返回给定 *Name* 的副本。

当同时指定了命名空间（Namespace）和名称（*Name*）参数时，新创建对象的 *localName* 属性将根据给定的 Name 进行设置， 而新创建对象的 *uri* 属性将根据 *Namespace* 参数进行设置。如果 *Name* 是一个 *QName* 对象，则新创建的 *QName* 的 *localName* 将与 *Name* 的 *localName* 相等。如 果 *Namespace* 参数是一个 Namespace 对象， 则新创建对象的 *uri* 属性将设置为 Namespace 对象的 *uri* 属性。如果 *Namespace* 参数**为 null， 则** 新创建对象的 *uri*属性也将**为 null**， 这意味着它将匹配任何命名空间中的名称。

**语义学**

当使用一个参数 Name 或两个参数 *Namespace* 和 *Name* 调用 *QName* 构造函数时， 将执行以下步骤：

1. 如果（Name 的类型为对象且 *Name.[[Class]]* 等于 **“QName”）** ：a. 如果

（未指定命名空间） ， 则返回 *Nam*e 的副本b. 否则，令 *Name = Na*m*e.*

*localNa*me

2. 如果（名称**未定义**或未指定） a. 将名称设为

3. 否则， 将 *Name* 赋值为 Name 的字符串形式。

4. 如果（命名空间**未定义**或未指定）

a. 如果名称=

i. 令 *Namespace* = **null**

b. 否则

i. 令 *Namespace* = GetDefaultNamespace

5. 设 *q* 为一个新的 QName， 其中 *q.localName* = *Name*

6. 如果命名空间为**空**

a. 将 *q.uri* **设为 null**

注意：保留限定名前缀的实现也可将 *q.[[Prefix]]* 设置为 **undefined** 。

7. 否则

a. 令 *Namespace* 为通过调用构造函数 new Namespace（Namespace） 而创建的新命名空间。

b. 令 *q.u*ri *=* 命名空间的 *uri*

注意：保留限定名前缀的实现也可将 *q.[[Prefix]]* 设置为 *Namespace.prefix* 。

8. 退货 *q*

**13.3.3 QName 构造函数的属性**

QName 构造函数的内部 [[Prototype]] 属性的值是 Function 原型对象。

除了内部属性和长度属性（其值为 2）之外， QName 构造函数还具有以下属性。

[**13.3.3.1**](13.3.3.1) **QName.prototype**

QName.prototype 属性的初始值是 QName 原型对象（第 13.3.4 节）。

此属性具有 { 不可枚举、不可删除、只读 } 属性。

*- 70-*

**13.3.4 QName 原型对象的属性**

QName 原型对象本身就是一个 QName 对象（其 [[Class]] 为 “QName”）， 其 *uri* 和 *localName* 属性均被设置为空字 符串。

QName 原型对象的内部 [[Prototype]] 属性的值是对象原型对象（ECMAScript 第 3 版第 15.2.3.1 节）。

[**13.3.4.1**](13.3.4.1) **QName.prototype.constructor**

QName.prototype.constructor 的初始值是内置的 QName 构造函数。

[**13.3.4.2**](13.3.4.2) **QName.prototype.toString()**

**概述**

toString 方法返回此 QName 对象的字符串表示形式。

toString 函数不是泛型的。如果其 **this** 值不是 QName 对象， 它会抛出一个 **TypeError** 异常。 因此， 它不能被转移到 其他类型的对象上用作方法。

**语义学**

当对 QName 对象 *n* 的toString 方法不带任何参数进行调用时， 将执行以下步骤：

1. 如果 Type(n) 不是 Object 或者 *n.[[Class]]* 不等于 **“QName”** ， 则抛出一个**类型错误**异常。

2. 设 *s* 为空字符串

3. 如果 *n.u*ri 不是空字符串

a. 如果 *n.u***ri 为空， 则**令 *s* 为字符串 **"\***

b. 否则，令 *s* 为 *n.u*ri 与字符串**“： ：”**连接后的结果。

4. 令 *s* 为 *s* 与 *n.localName* 拼接后的结果。

5. 回报

**13.3.5 QName 实例的属性**

QName 实例从 QName 原型对象继承属性， 并且具有 *uri* 属性、*localName* 属性以及一个可选的内部 [[Prefix]] 属性， 该属性可用于在限定名称中保留前缀的实现。

[**13.3.5.1**](13.3.5.1) **localName**

*localName* 属性的值为字符串类型。当 *localName* 属性的值为“\* ”时， 它表示通配符， 可匹配任何名称。

此属性应具有 { 不可删除， 只读 } 属性。

[**13.3.5.2**](13.3.5.2) **统一资源标识符（URI）**

*uri* 属性的值**为 null 或为**标识此 QName 命名空间的字符串类型值。 当 *uri*属性的值为空字符串时， 称此 QName 位于 无命名空间中。在 XML 对象中， 使用无命名空间来明确指定某个名称不在任何命名空间内。当 *uri* 属性的值**为 null 时**， 此 QName 将匹配任何命名空间中的名称。

此属性应具有 { 不可删除， 只读 } 属性。

[**13.3.5.3**](13.3.5.3) **[[前缀]]**

[[Prefix]] 属性是一个可选的内部属性， 用户无法直接看到。 它可能被那些在限定名中保留前缀的实现所使用。 [[Prefix]] 属性的值为字符串类型或**未定义**。如果 [[Prefix]] 属性**未定义**， 则与该 QName 相关联的前缀未知。

*- 71-*

[**13.3.5.4**](13.3.5.4) **[[GetNamespace]]（[InScopeNamespaces]）概述**

[[GetNamespace]] 方法是一个内部方法， 它返回一个具有与此 QName 的 URI 相匹配的 URI 的 Namespace 对象。*InScopeNamespaces* 是一个可 选参数。如果未指定 *InScopeNamespaces*， 则将其设置为空集。如果 *InScopeNamespaces* 中存在一个或多个具有与此 QName 的 URI 相匹配的 URI 的 Namespace， 则返回其中一个匹配的 Namespace。如果 *InScopeNamespaces* 中不存在这样的 Namespace， 则 [[GetNamespace]] 会创建并返 回一个具有与此 QName 相同 URI 的新 Namespace。对于保留 QName 中前缀的实现 ， [[GetNamespace]] 可能会返回一个具有匹配前缀的

Namespace。输入参数 *InScopeNamespaces* 是一个 Namespace 对象的集合。

**语义学**

当对 QName *q* 的 [[GetNamespace]] 方法不带参数或仅带一个参数 *InScopeNamespaces* 调用时， 将执行以下步骤：

1. 如果 *q.uri* 为 **null， 则**抛出一个**类型错误**异常。

请注意， 由于本规范中对 [[GetNamespace]] 的调用方式， 上述异常情况应永远不会发生。

2. 如果未指定 *InScopeNamespaces*，则令 *InScopeNamespaces* = { }

3. 在 *InScopeNamespaces* 中查找一个命名空间 *ns*， 使得 ns.*uri* == *q.u*ri。如果存在多个这样的命名空间 *ns*， 实现可以任意选择其中一个匹 配的命名空间。注：保留限定名中前缀的实现还可以进一步限制 *ns*，使得 ns*.prefix* == *q.[[Prefix]]*

4. 如果不存在这样的命名空间 *ns*

a. 假设通过调用构造函数 new Namespace（q.uri） 创建了一个新的命名空间 *ns* 。

请注意， 保留前缀和限定名的实现可能会像调用构造函数 Namespace（q.[[Prefix]], *q.uri*）一样创建新的命名空间。

5. 返回 *ns*

**13.4 XML 对象**

**13.4.1 作为函数调用的 XML 构造器**

**语法**

XML（[值]）

**概述**

当将 XML 作为函数而非构造函数调用时， 它会执行类型转换。如果未提供参数，XML 函数将返回一个表示空文本节点的 XML 对象。

**语义学**

当调用 XML 函数时，如果没有提供参数或仅提供一个参数值， 则执行以下步骤。

1. 如果值为 **null、未定义**或未提供， 则将值设为空字符串。

2. 将值转换为 XML 格式（ReturnToXML（value））

注意：ToXML 操作符定义了一种机制， 用于从 W3C 信息集的实现（例如 W3C DOM 节点）构建 XML 对象。E4X 实现可以通过 XML 构造函数向 用户公开此功能；但这并非符合 E4X 的必要条件。有关更多信息， 请参阅第 10.3.2 节。

**13.4.2 XML 构造器**

**语法**

新的 XML（[值]）

**概述**

*- 72-*

当在新表达式中调用 XML 时， 它是一个构造函数， 并且可能会创建一个新的 XML 对象。 当 XMLList 构造函数未传 入任何参数时， 它会返回一个表示空文本节点的 XML 对象。

**语义学**

当调用 XML 构造函数时， 如果未传入任何参数或仅传入一个参数值， 则会执行以下步骤：

1. 如果值为 **null、未定义**或未提供， 则将值设为空字符串。

2. 令 *x* = ToXML(value

注意：ToXML 操作符定义了一种机制， 用于从 W3C 信息集的实现（例如 W3C DOM 节点）构建 XML 对象。E4X 实现 可以通过 XML 构造函数向用户公开此功能；但这并非符合 E4X 的必要条件。有关更多信息，请参阅第 10.3.2 节。

3. 如 果 Type（value） 属 于 {XML、XMLList、W3C XML 信 息 项}， 则 a. 返回调用 *x* 的 [[DeepCopy]] 方法的结果。

4. 返回 *x*

**13.4.3 XML 构造函数的属性**

XML 构造函数的内部 [[Prototype]] 属性的值是 Function 原型对象。

除了内部属性和长度属性（其值为 1）之外，XML 构造函数还具有以下属性：

[**13.4.3.1**](13.4.3.1) **XML.prototype**

XML.prototype 属性的初始值是 XML 原型对象（第 13.4.3.1 节）。

此属性具有 { 不可枚举、不可删除、只读 } 属性。

[**13.4.3.2**](13.4.3.2) **XML.ignoreComments**[**13.4.3.2**](13.4.3.2) **XML.忽略注释**

ignoreComments 属性的初始值为 true。**如果 ignoreComments** 为 **true**， 则在构建新的 XML 对象时会忽略 XML 注释。

此属性具有 { 不可枚举， 不可删除 } 属性。

[**13.4.3.3**](13.4.3.3) **XML.ignoreProcessingInstructions**

ignoreProcessingInstructions 属性的初始值为 true。如果 ignoreProcessingInstructions 为 **true**， 则在构建新的 XML 对象 时会忽略 XML 处理指令。

此属性具有 { 不可枚举， 不可删除 } 属性。

[**13.4.3.4**](13.4.3.4) **XML.ignoreWhitespace**

`ignoreWhitespace` 属性的初始值为 `true`。如果 `ignoreWhitespace` 为 `**true**`， 在处理构建新的 XML 对象时， 将忽略 无关紧要的空白字符。 当元素标签和/或嵌入式表达式仅由空白字符分隔时， 这些空白字符被定义为无关紧要的。空 白字符被定义为空格（`\u0020`）、 回车（`\u000D`）、换行（`\u000A`）和制表符（`\u0009`）。

此属性具有 { 不可枚举， 不可删除 } 属性。

[**13.4.3.5**](13.4.3.5) **XML 美化输出**

prettyPrinting 属性的初始值为 true。如果 prettyPrinting 为 **true**， 则 ToString 和 ToXMLString 运算符会将某些标签之间 的空白字符进行规范化处理， 以实现统一且美观的外观。

此属性具有 { 不可枚举， 不可删除 } 属性。

*- 73-*

[**13.4.3.6**](13.4.3.6) **XML.prettyIndent**

prettyIndent 属性的初始值为 2。如果 XML 构造函数的 prettyPrinting 属性为 **true**，则 ToString 和 ToXMLString 运算符会将某些标签之间的空白字 符进行规范化处理， 以实现统一且美观的外观。某些子节点相对于其父节点的缩进量将由 prettyIndent属性指定的空格数决定。

此属性具有 { 不可枚举，不可删除 } 属性。

[**13.4.3.7**](13.4.3.7) **XML 设置 ()**

**概述**

设置方法是一种便利方法，用于管理作为 XML 构造函数属性存储的全局 XML 设置集合（第 13.4.3.2 节至第 13.4.3.6 节）。它返回一个对象，其 中包含用于存储 XML 设置的 XML 构造函数的属性。此对象稍后可作为参数传递给 setSettings 方法以恢复相关设置。例如，

**语义学**

当调用 XML 构造函数的设置方法时，会执行以下步骤：

1. 假设通过调用构造函数 new Object（） 创建了一个新的对象 *s* 。

2. 让我们将 *`ignoreComments`* 设置为 `XML.ignoreComments` 的值。

3. 让我们将 *`ignoreProcessingInstructions`* 设置为 `XML.ignoreProcessingInstructions` 的值。

4. 让我们将 *`ignore Whitespace`* 设置为 `XML.ignoreWhitespace` 的值。

5. 让我们将 *`s.prettyPrinting`* 设置为 `XML.prettyPrinting` 的值。

6. 让我们将 *s.prettyIndent* 设置为 XML.prettyIndent 的值。

7. 回报

[**13.4.3.8**](13.4.3.8) **XML.setSettings ( [ 设置 ]**

`setSettings` 方法是用于管理作为 XML 构造函数属性存储的全局 XML 设置集合的便捷方法（参见第 13.4.3.2 节至 13.4.3.6 节）。它可以用于恢复 之前使用相关 `settings` 方法捕获的 XML 设置集合。当使用单个参数 `settings` 调用时， `setSettings` 方法会从 `settings` 对象复制用于存储 XML 设 置的 XML 构造函数的属性。当不带参数调用时， `setSettings` 方法会恢复默认的 XML 设置。

**语义学**

当 XML 构造函数的 setSettings 方法未传入任何参数或仅传入一个参数 *settings* 时，将执行以下步骤：

1. 如果 settings 为**空、未定义**或未提供

a. 将 XML.ignoreComments 设为 **true**

b. 将 XML.ignoreProcessingInstructions 设为 **true**

c. 将 XML.ignoreWhitespace 设为 **true** d. 将 XML.prettyPrinting 设为 **true**

e. 将 XML.prettyIndent 设为 **2**

*- 74-*

2. 否则， 如果 settings 的类型为对象

a. 如果 Type（settings.ignoreComments） 的类型为布尔值， i. 让 XML.ignoreComments *= settings.ignoreComments*

b. 如果 Type（settings.ignoreProcessingInstructions） 的类型为布尔值

i. 令 XML.ignoreProcessingInstructions *= settings.ignoreProcessingInstructions* c. 如果 Type（settings.ignoreWhitespace） 的类型为布尔值

i. 令 XML.ignoreWhitespace *= settings.ignore Whitespace*

d. 如果 Type（settings.prettyPrinting） 的类型为布尔值 i. 让 XML.prettyPrinting *= settings.prettyPrinting*

e. 如果 Type（settings.prettyIndent） 的类型为数字 i. 令 XML.prettyIndent *= settings.prettyIndent*

3. 返回

[**13.4.3.9**](13.4.3.9) **XML.defaultSettings ( )**

defaultSettings 方法是一个用于管理作为 XML 构造函数属性存储的全局 XML 设置集合的便捷方法（第 13.4.3.2 节至 13.4.3.6 节）。它可用于获取一 个包含默认 XML 设置的对象。可以检查此对象以确定默认设置， 或者将其作为参数传递给 setSettings 方法以恢复默认 XML 设置。

**语义学**

当 XML 构造函数的 defaultSettings 方法未传入任何参数时， 将执行以下步骤：

1. 设 *s* 是一个新创建的对象， 就好像通过调用构造函数 new Object（） 创建的一样。

2. 让我们忽略注释，将其设为 **true** 。

3. 让我们忽略处理指令，将其设为 **true** 。

4. 让我们将忽略空白设置为 **true** 。

5. 将 *s* 的 *prettyPrinting* 属性设置为 **true** 。

6. 将 *s.prettyIndent* 的值设为 **2** 。

7. 回报

[**13.4.3.10**](13.4.3.10) **[[HasInstance]] ( V )**

**概述**

XML 构造函数的 [[HasInstance]] 方法比其他函数对象更为复杂。XML 构造函数的 [[HasInstance]] 方法定义为：对于给定的 XML 对象或仅包含一个 值的 XMLList对象 *x*， 表达式 *x* instance of XML 将返回 true。这种处理有意模糊了单个 XML 对象与仅包含一个值的 XMLList之间的区别。

**语义学**

当对 XML 构造函数对象 *F* 调用 [[HasInstance]] 方法并传入值 *V* 时， 将执行以下步骤：

1. 如果 *V* 不是对象， 则返回 **false**

2. 令 *xmlProto* = XML 的原型

3. 令 *listProto* = XMLList.prototype

4. 如果 `xmlProto` 不是对象或者 `listProto` 不是对象， 则抛出一个 `**TypeError`** 异常。

5. 令 *objProto* = *V.[[Prototype]]*

6. 当 objProto 不为**空**时

a. 如果 *xmlProto* 和 *objProto* 指向同一个对象， 或者它们指向彼此关联的对象（ECMAScript 第 3 版第 13.1.2 节） ， 则返回 **true** 。

b. 如果 *listProto* 和 *objProto* 指向同一个对象， 或者它们指向彼此关联的对象（ECMAScript 第 3 版第 13.1.2 节）， 则返回 **true** 。

c. 令 *objProto* = *objProto.[[Prototype]]*

**7. 返回 false**

*- 75-*

**13.4.4 XML 原型对象的属性（内置方法）**

每种 XML 类型的值都具有一组可用于执行常见操作的内置方法。 各节中进行了描述。

这些内置方法是 XML 原型对象的属性， 并在以下

XML 原型对象本身就是一个 XML 对象（其 [[Class]] 属性为**“text”）**， 其值为空字符串。

XML 构造函数的原型对象的内部 [[Prototype]] 属性的值为对象原型对象（参见 ECMAScript 第 3 版第 15.2.3.1 节）。

XML.prototype 上定义的所有内置函数都不是通用的。如果**this** 值不是 XML 对象， 它们会抛出一个 **TypeError** 异常。 因此， 它们不能转移到其他类型的对象上作为方法使用。

[**13.4.4.1**](13.4.4.1) **XML.prototype.constructor**

XML.prototype.constructor 的初始值是内置的 XML 构造函数。此属性为 E4X 的未来版本保留。 注意：此属性的值无法访问，这实际上使该属性成为只写属性。

[**13.4.4.2**](13.4.4.2) **XML.prototype.addNamespace ( 命名空间**

**概述**

`addNamespace` 方法会将命名空间声明添加到此 XML 对象的作用域命名空间中， 并返回此 XML 对象。如果 XML 对象的作用域命名空间中已包含具有与给定参数的前缀相匹配的命名空间， 则将现有命名空间的前缀设置为**未定义。**

**语义学**

当对 XML 对象 *x* 调用 addNamespace 方法并传入一个参数 *namespace* 时， 将执行以下步骤：

1. 假设通过调用函数 Namespace（namespace） 构造了一个名为 *ns* 的命名空间。

2. 调用 *x* 的 [[AddInScopeNamespace]] 方法， 并传入参数 *ns* 。

3. 返回 *x*

[**13.4.4.3**](13.4.4.3) **XML.prototype.appendChild ( child )**

**概述**

appendChild 方法将给定的子节点添加到此 XML 对象属性的末尾， 并返回此 XML 对象。例如，

**语义学**

当对 XML 对象 *x* 调用 appendChild 方法并传入一个参数 *child* 时， 将执行以下步骤：

1. 让 *children* 成为调用 *x* 的 [[Get]] 方法并传入参数 **“\* ”** 的结果。

2. 调用 children 的 [[Put]] 方法，传入参数 *children.[[Length]]* 和 *child* 。

3. 返回 *x*

[**13.4.4.4**](13.4.4.4) **XML.prototype.attribute（attributeName）**

**概述**

*- 76-*

属性方法返回一个 XMLList， 其中包含与此 XML 对象相关联且具有给定属性名称的零个或一个 *XML* 属性。例如，

**语义学**

当对 XML 对象 *x* 调用属性方法并传入参数 *attributeName* 时， 将执行以下步骤：

1. 令 *name* = ToAttributeName(attributeName

2. 返回调用 *x* 的 [[Get]] 方法并传入参数 *name* 所得到的结果。

[**13.4.4.5**](13.4.4.5) **XML.prototype.attributes （） 方法**

**概述**

“attributes”方法返回一个包含此对象的 XML 属性的 XMLList。例如，

**语义学**

当调用 XML 对象 *x* 的 attributes 方法时， 会执行以下步骤：

1. 返回调用 *x* 的 [[Get]] 方法并传入参数 ToAttributeName（"\*"） 所得到的结果。

[**13.4.4.6**](13.4.4.6) **XML.prototype.child（属性名称）**

**概述**

`child` 方法返回此 XML 对象中与给定的 `*propertyName*` 匹配的子项列表。如果 `*propertyName*` 是一个数字索引， 则 ` child` 方法返回一个包含由 `*propertyName`* 标识的序号位置处的子项的列表。例如，

**语义学**

当调用 XML 对象 *x* 的 child 方法时， 它会执行以下步骤：

1. 如果 ToString（ToUint32(propertyName)） 等于*propertyName*

a. 让 *children* 成为调用 *x* 的 [[Get]] 方法并传入参数 **“\* ”** 的结果

b. 令 *temporary* 为调用 children 的 [[Get]] 方法并传入参数*propertyName* 所得到的结果

c. 如果 *temporary* **未定义**， 则令 *temporary* = new XMLList

d. 暂时返回

2. 令 *temporary* 为调用 *x* 的 [[Get]] 方法并传入参数*propertyName* 所得到的结果。

3. 返回 ToXMLList(临时变量

[**13.4.4.7**](13.4.4.7) **XML.prototype.childIndex ( )**

**概述**

childIndex 方法返回一个数字， 表示此 XML 对象在其父对象中的序号位置。例如，

*- 77-*

**语义学**

当调用 XML 对象 *x* 的 childIndex 方法时， 它会执行以下步骤：

1. 令 *parent* = *x.[[Parent]]*

2. 如果 （parent **为 null）** 或者 （x 的 [[Class]] 属性值为 **“attribute”）**， 则返回 **-1 。**

3. 设 *q* 为父对象的属性， 其中*parent[q]* 与 *x* 为同一对象。

4. 返回 ToNumber(q

[**13.4.4.8**](13.4.4.8) **XML.prototype.children( )**

**概述**

children 方法返回一个 XMLList， 其中包含此 XML 对象的所有属性， 按顺序排列。例如，

**语义学**

当调用 XML 对象 *x* 的 children 方法时， 它会执行以下步骤：

1. 返回调用 *x* 的 [[Get]] 方法并传入参数 **“\*”** 所得到的结果。

[**13.4.4.9**](13.4.4.9) **XML.prototype.comments ( )**

**概述**

comments 方法返回一个 XMLList， 其中包含此 XML 对象中表示 XML 注释的属性。

**语义学**

当调用 XML 对象 *x* 的 comments 方法时， 它会执行以下步骤：

1. 令 *list* 为一个新的 XMLList，其中 *list.[[TargetObject]]* = *x* 且 *list.[[TargetProperty]]* = **null** 。

2. 对于 *i* 从 **0** 到 *x.[[Length]] - 1*

a. 如果 *x[i].[[Class]]* 的值为 **“comment”**， 则调用列表的 [[Append]] 方法， 并将 *x[i]* 作为参数传入。

3. 退货清单

[**13.4.4.10**](13.4.4.10) **XML.prototype.contains ( value )**

**概述**

contains 方法返回将此 XML 对象与给定值进行比较的结果。这种处理有意模糊了单个 XML 对象与仅包含一个值的 XMLList之间的区 别。

**语义学**

当对 XML 对象 *x* 调用 contains 方法并传入参数 *value* 时， 会执行以下步骤：

1. 返回比较结果 *x* 是否等于 value 的值

[**13.4.4.11**](13.4.4.11) **XML.prototype.copy( )**

**概述**

*- 78-*

复制方法返回此 XML 对象的深复制副本， 并将内部的 [[Parent]] 属性设置为 **null。** **语义学**

当对 XML 对象 *x* 调用复制方法时， 将执行以下步骤：

1. 返回调用 *x* 的 [[DeepCopy]] 方法的结果。

[**13.4.4.12**](13.4.4.12) **XML.prototype.descendants ( [ name ] )**

**概述**

descendants 方法返回此 XML 对象具有给定名称的所有 XML 值的后代（子代、孙代、 曾孙代等）。如果省略名称参 数， 则返回此 XML 对象的所有后代。

**语义学**

当在 XML 对象 *x* 上调用 descendants 方法并带有可选参数 *name* 时， 将执行以下步骤：

1. 如果未指定名称，则令名称=

2. 返回调用 *x* 的 [[Descendants]] 方法并传入参数 *name* 所得到的结果。

[**13.4.4.13**](13.4.4.13) **XML.prototype.elements( [ name ] )**

**概述**

当使用一个参数 *name* 调用 *elements* 方法时， 它会返回一个 XMLList， 其中包含此 XML 对象中所有名称为给定名称 的 XML 元素的子节点。 当不带参数调用 elements 方法时， 它会返回一个 XMLList， 其中包含此 XML 对象中所有 XML 元素的子节点， 不论其名称如何。

**语义学**

当在 XML 对象 *x* 上调用 elements 方法并带有可选参数 *name* 时， 将执行以下步骤：

1. 如果未指定名称，则令名称=

2. 令 *name* = ToXMLName(name

3. 令 *list* 为一个新的 XMLList，其中 *list.[[TargetObject]]* = *x* 且 *list.[[TargetProperty]]* = *name*

4. 对于 *i* 从 **0** 到 *x.[[Length]] - 1*

a. 如果 *x[i].[[Class]]* 等于 **"element**

如 果 （name.localName 等 于 **“\* ”** 或 者 name.*localName* 等 于 *x[i].[[Name]]. localName*）并且（name.uri **为空**或者 *n.u*ri 等于 *x[i].[[Name]].uri*）

5. 退货清单 1. 调用列表的 `Append` 方法， 并将参数设为 `*x[i]`* 。

[**13.4.4.14**](13.4.4.14) **XML.prototype.hasOwnProperty(P)**

**概述**

`hasOwnProperty` 方法返回一个布尔值， 该值指示此对象是否具有由 *P* 指定的属性。对于除 XML 原型对象之外的所 有 XML 对象， 此结果与内部方法 `[[HasProperty]]` 返回的结果相同。对于 XML 原型对象， `hasOwnProperty` 还会检 查本地属性列表， 以确定是否存在具有给定名称的方法属性。

**语义学**

*- 79-*

当对 XML 对象 *x* 调用 hasOwnProperty 方法并传入参数 *P* 时，将执行以下步骤：

1. 如果调用此对象的 [[HasProperty]] 方法并传入参数 *P* 的结果为**真**，则返回**真**。

2. 如果 *x* 具有名为 ToString(P) 的属性，则返回 **true**

3. 返回**假**值

[**13.4.4.15**](13.4.4.15) **XML.prototype.hasComplexContent( )**

**概述**

`hasComplexContent` 方法返回一个布尔值， 表示此 XML 对象是否包含复杂内容。如果 XML 对象表示具有子元素的 XML 元素， 则认为其包含 复杂内容。表示属性、注释、处理指令和文本节点的 XML 对象不包含复杂内容。XML 对象中属性、注释、处理指令和文本节点的存在对于确 定其是否包含复杂内容并不重要。

**语义学**

当对 XML 对象 *x* 调用 *hasComplexContent*方法时，将执行以下步骤：

1. 如果 *x.[[Class]]* **属于** {"attribute"**, "comment", "processing-instruction", "text"}，** 则返回 **false**

2. 对于 *x* 中的每个属性*p*

a. 如果*p.[[Class]]* 等于 **“element”**，则返回 **true**

3. 返回**假**值

[**13.4.4.16**](13.4.4.16) **XML.prototype.hasSimpleContent( )**

**概述**

`hasSimpleContent` 方法返回一个布尔值，表示此 XML 对象是否包含简单内容。如果 XML 对象表示文本节点、属性节点，或者表示没有子元素 的 XML 元素，则认为其包含简单内容。表示注释和处理指令的 XML 对象不包含简单内容。XML 对象中是否存在属性、注释、处理指令和文本 节点，对于确定其是否包含简单内容并不重要。

**语义学**

当在 XML 对象 *x* 上调用 *hasSimpleContent*方法时，将执行以下步骤：

1. 如果 *x.[[Class]]* 属于 {"comment", **"processing-instruction"}**，则返回 **false**。

2. 对于 *x* 中的每个属性*p*

a. 如果*p.[[Class]]* 等于 **“element”**，则返回 **false**

3. 返回**真**

[**13.4.4.17**](13.4.4.17) **XML.prototype.inScopeNamespaces( )**

**概述**

`inScopeNamespaces` 方法返回一个 `Namespace` 对象的数组，这些对象代表此 XML 对象在其父对象上下文中处于作用域内的命名空间。如果此 XML 对象的父对象被修改，相关的命名空间声明可能会发生变化。此方法返回的命名空间集合可能是此值所使用的命名空间的超集。

**语义学**

当在 XML 对象 *x* 上调用 *inScopeNamespaces* 方法时，将执行以下步骤：

1. 令 *y* = *x*

2. 令 *inScopeNS* = { }

3. 当（y 不为**空）**时

*- 80-*

a. 对于 *y.[[InScopeNamespaces]]* 中的每个 *ns*

i. 如果不存在任何 *n* 属于 inScopeNS，使得 n 的前缀与 ns 的前缀相等

1. 令 *inScopeNS* = *inScopeNS* U { *ns* } b. 令 *y* = *y.[[Parent]]*

4. 令 a 为通过调用构造函数 new Array（） 创建的新数组。

5. 令 *i* = **0**

6. 对于 *inScopeNS* 中的每个 *ns*

a. 调用 a 的 [[Put]] 方法，传入参数 ToString(i) 和 *ns*

b. 令 *i* = *i* + **1**

7. 返回一个

[**13.4.4.18**](13.4.4.18) **XML.prototype.insertChildAfter ( 子节点 1 ， 子节点 2**

**概述**

`insertChildAfter` 方法将给定的 `*child2*` 插入到此 XML 对象中给定的 `*child1*` 之后 ， 并返回此 XML 对象 。如果 `*child1*` 为 `**null`， 则** `

insertChildAfter` 方法将 `*child2*` 插入到此 XML 对象的所有子节点之前（即在所有子节点之前）。如果 `*child1*` 不存在于此 XML 对象中，则该方 法不修改此 XML 对象而直接返回。

**语义学**

当在 XML 对象 *x* 上调用 insertChildAfter 方法，并传入参数 *child1* 和 *child2* 时，将执行以下步骤：

1. 如果 *x.[[Class]]* **属于** {"text"**, "comment", "processing-instruction", "attribute"}，** 则返回

2. 如果（child1 为**空）**

a. 调用 *x* 的 [[Insert]] 方法，传入参数 **“0”** 和 *child2*

b. 返回 *x*

3. 否则，如果 child1 的类型为 XML

a. 对于 *i* 从 **0** 到 *x.[[Length]] - 1*

i. 如果 *x[i]* 与 *child1* 是同一个对象

1. 调用 *x* 的 [[Insert]] 方法，传入参数 ToString（i + **1）** 和 *child2* 。

2. 返回 *x*

4. 返回

[**13.4.4.19**](13.4.4.19) **XML.prototype.insertChildBefore ( child1 , child2 ) -> 1XML.prototype.insertChildBefore （child1 , child2 ） 方法**

**概述**

`insertChildBefore` 方法在当前 XML 对象中将给定的 `*child2*` 插入到给定的 `*child1*` 之前， 并返回当前 XML 对象。如果 `*child1*` 为 `**null`， 则** ` insertChildBefore` 方法将 `*child2*` 插入到当前 XML 对象中所有子节点之后（即在所有子节点之前）。如果 `*child1*` 不存在于当前 XML 对象中，则 该方法不修改当前 XML 对象而直接返回。

**语义学**

当在 XML 对象 *x* 上调用 insertChildBefore 方法，并传入参数 *child1* 和 *child2* 时，将执行以下步骤：

1. 如果 *x.[[Class]]* **属于** {"text"**, "comment", "processing-instruction", "attribute"}，** 则返回

2. 如果（child1 为**空）**

a. 调用 *x* 的 [[Insert]] 方法，传入参数为 ToString（x.[[Length]]） 和 *child2*b. 返回 x 3. 否则，如果 child1 的类型为 XML

a. 对于 *i* 从 **0** 到 *x.[[Length]] - 1*

i. 如果 *x[i]* 与 *child1* 是同一个对象

1. 调用 *x* 的 [[Insert]] 方法，传入参数 ToString(i) 和 *child2* 。

2. 返回 *x*

4. 返回

*- 81-*

[**13.4.4.20**](13.4.4.20) **XML.prototype.length ( )**

**概述**

对于 XML 对象，length 方法始终返回整数 **1**。这种处理方式有意模糊了单个 XML 对象与仅包含一个值的 XMLList之间的区别。

**语义学**

当对 XML 对象 *x* 调用 *length* 方法时， 将执行以下步骤：

1. 返回 **1**

[**13.4.4.21**](13.4.4.21) **XML.prototype.localName ( )**

**概述**

localName 方法返回此 XML 对象的限定名中的本地名部分。

**语义学**

当在 XML 对象 *x* 上调用 *localName* 方法时， 将执行以下步骤：

1. 如果 *x.[[Name]]* 为 **null， 则**返回 **null** 。

2. 返回 *x.[[Name]].localName* 的值

[**13.4.4.22**](13.4.4.22) **XML.prototype.name( )**

**概述**

name 方法返回与此 XML 对象相关联的限定名称。

**语义学**

当对 XML 对象 *x* 调用 *name* 方法时， 将执行以下步骤：

1. 返回 *x* 的 *[[Name]]* 属性值

[**13.4.4.23**](13.4.4.23) **XML.prototype.namespace（[ 前缀 ]）**

**概述**

如果未指定前缀， 则命名空间方法将返回与此 XML 对象的限定名称相关联的命名空间。

如果指定了前缀， `namespace` 方法会在该 XML 对象的作用域内查找具有给定前缀的命名空间，如果找到则返回该命名空间。如果未找到这样 的命名空间， 则 `namespace` 方法返回 `**undefined`。**

**语义学**

当在 XML 对象 *x* 上调用命名空间方法且未传入参数或仅传入一个参数*prefix* 时， 将执行以下步骤：

1. 令 *y* = *x*

2. 令 *inScopeNS* = { }

3. 当（y 不为**空）**时

a. 对于 *y.[[InScopeNamespaces]]* 中的每个 *ns*

i. 如果不存在任何 *n* 属于 inScopeNS， 使得 n 的前缀与 ns 的前缀相等

1. 令 *inScopeNS* = *inScopeNS* U { *ns* } b. 令 *y* = *y.[[Parent]]*

4. 如果未指定前缀

*- 82-*

a. 如果 *x.[[Class]]* 属于 {"text", **"comment"**, **"processing-instruction"}**，**则返回 null**

b. 返回调用 *x.[[Name*]] 的 [[GetNamespace]] 方法并将 *inScopeNS* 作为参数传入的结果。 5. 否则

a. 令 *prefix* = ToString(prefix

b. 查找一个作用域内命名空间 *ns* ∈ inScopeNS，使得 ns.*prefix* =*prefix*。如果不存在这样的 *ns*，则令 *ns* = **undefined。**

c. 返回 *ns*

[**13.4.4.24**](13.4.4.24) **XML.prototype.namespaceDeclarations ( )**

**概述**

`namespaceDeclarations` 方法返回一个 `Namespace` 对象数组， 这些对象代表了此 XML 对象在其父对象上下文中关联的命名空间声明。如果此 XML 对象的父对象被修改，相关的命名空间声明可能会发生变化。

**语义学**

当对 XML 对象 *x* 调用 *namespaceDeclarations* 方法时，将执行以下步骤：

1. 令 a 为通过调用构造函数 new Array（） 创建的新数组。

2. 如果 *x.[[Class]]* **属于** {"text"**, "comment", "processing-instruction", "attribute"}，**则返回 ***a***

3. 令 *y* = *x* 的父级。

4. 令 ancestorNS = { }

5. 当（y 不为**空）**时

a. 对于 *y.[[InScopeNamespaces]]* 中的每个 *ns*

i. 如果不存在任何 *n* 属于祖先命名空间集合 ancestorNS，使得 *n* 的前缀与 *ns* 的前缀相同

1. 令 *ancestorNS* 等于 *ancestorNS* 与 *{ ns* } 的并集 b. 令 *y* = *y.[[Parent]]*

6. 令 *declaredNS* = { }

7. 对于 *x.[[InScopeNamespaces]]* 中的每个 *ns*

a. 若不存在 *n* ∈ ancestorNS，使得 *n.p*refix == ns.prefix 且 *n.u*ri == ns.*uri*，则i. 令 *declaredN*S *= declaredN*S U *{ n*s }

8. 令 *i* = **0**

9. 对于 *declaredNS* 中的每个 *ns*

a. 调用 a 的 [[Put]] 方法，传入参数 ToString(i) 和 *ns*b. 令 i *=* i **+** 1

10. 返回一个

[**13.4.4.25**](13.4.4.25) **XML.prototype.nodeKind ( )**

**概述**

`nodeKind` 方法返回一个字符串，表示此 XML 对象的 [[Class]] 类型。

**语义学**

当对 XML 对象 *x* 调用 *nodeKind* 方法时，将执行以下步骤：

1. 返回 *x* 的 *[[Class]]* 属性值

[**13.4.4.26**](13.4.4.26) **XML.prototype.normalize( )**

**概述**

normalize 方法会将此 XML 对象及其所有后代 XML 对象中的所有文本节点合并相邻的文本节点并删除空文本节点，从而转换为规范形式。该方 法返回此 XML 对象。

**语义学**

*- 83-*

当对 XML 对象 *x* 调用 *normalize* 方法时， 将执行以下步骤：

[**13.4.4.27**](13.4.4.27) **XML.prototype.parent( )**

**概述**

父方法返回此 XML 对象的父对象。例如，

**语义学**

当在 XML 对象 *x* 上调用父方法时， 将执行以下步骤：

1. 返回 *x* 的父级元素。

[**13.4.4.28**](13.4.4.28) **XML.prototype.processingInstructions（[name]）**

**概述**

当使用一个参数名称调用 *processingInstructions* 方法时， 它会返回一个 XMLList， 其中包含此 XML 对象中所有名称 与给定名称相同的处理指令子节点。 当不带任何参数调用 processingInstructions 方法时， 它会返回一个 XMLList， 其 中包含此 XML 对象中所有处理指令子节点， 不论其名称如何。

**语义学**

当对 XML 对象 *x* 调用 processingInstructions 方法并带有可选参数 *name* 时， 将执行以下步骤：

[**13.4.4.29**](13.4.4.29) **XML.prototype.prependChild（value）**

**概述**

*- 84-*

prependChild 方法将给定的子节点插入到此对象中，位于其现有的 XML 属性之前， 并返回此 XML 对象。例如，

**语义学**

当在 XML 对象 *x* 上调用 prependChild 方法并传入参数 *value* 时， 将执行以下步骤：

1. 使用参数**“0”**和 *value* 调用此对象的 [[Insert]] 方法

2. 返回 *x*

[**13.4.4.30**](13.4.4.30) **XML.prototype.propertyIsEnumerable ( P )**

**概述**

`propertyIsEnumerable` 方法返回一个布尔值， 该值指示当此 XML 对象用于 `for-in` 语句时， 属性 *P* 是否会被包含在迭代的属性集中。 当 ` ToString(P)` 的值为 **“0” 时， 此方法返回`true**` ；否则返回 `false`**。**这种处理方式有意模糊了单个 XML 对象与仅包含一个值的 XML 列表之间 的区别。

**语义学**

当对 XML 对象 *x* 调用 propertyIsEnumerable 方法并传入参数 *P* 时， 将执行以下步骤：

1. 返回比较结果 ToString(P) == **"0"** 的值

[**13.4.4.31**](13.4.4.31) **XML.prototype.removeNamespace ( 命名空间**

**概述**

`removeNamespace` 方法会从当前对象及其所有后代对象的作用域命名空间中移除给定的命名空间 ， 然后返回此 XML 对象的副本。` removeNamespaces` 方法不会移除在对象的 `QName` 或该对象属性的 `QName` 中被引用的命名空间。

**语义学**

当对 XML 对象 *x* 调用 removeNamespace 方法并传入参数 *namespace* 时， 将执行以下步骤：

1. 如果 *x.[[Class]]* **属于** {"text"**, "comment", "processing-instruction", "attribute"}，** 则返回 *x*

2. 假设通过调用函数 Namespace（*namespace*）创建了一个名为 *ns* 的命名空间对象。

3. 令此 *NS* 为调用 [[GetNamespace]]（以 *x.[[InScopeNamespaces]]* 作为参数）作用于 *x.[[Name*]] 所得到的结 果。

4. 如果（thisNS 等于 *ns*），则返回 *x*

5. 对于 *x.[[Attributes]]* 中的每个 *a*

a. 令 *aNS* 为调用 *a.[[Name]]* 的 *[[*GetNamespace]] 方法并传入参数 *x.[[InScopeNamespaces]]* 所得到的结 果。

b. 如果 （aNS 等于 *ns*），则返回 *x*

6. 如果 *ns.prefix* **未定义**

a. 如果存在一个命名空间 *n* 属于 x.[[InScopeNamespaces]]， 且 *n.u*ri 等于 ns.*uri*，则从 *x.*[[InScopeNamespaces]] 中移除该命名空 间 *n* 。

7. 否则

如 果 存 在 一 个 命 名 空 间 *n* 属 于 x.[[InScopeNamespaces]]， 使 得 *n.uri* 等 于 *ns.uri* 并 且 *n.p*refix 等 于 ns.*prefix*，则 从 *x.* [[InScopeNamespaces]] 中移除该命名空间 *n* 。

8. 对于 *x* 的每个属性*p*

a. 如果*p* 的 *[[Class]]* 属性值为 **“element”**，则调用*p* 的 removeNamespace 方法， 并传入参数 *ns* 。

*- 85-*

9. 返回 *x*

[**13.4.4.32**](13.4.4.32) **XML.prototype.replace ( 属性名称 ， 值**

**概述**

replace 方法会用 *value* 替换此 XML 对象中由*propertyName* 指定的 XML 属性， 并返回此 XML 对象。如果此 XML 对象中不存在与*propertyName* 匹配的 属性，则 replace 方法将不作修改直接返回。*propertyName* 参数可以是数字属性名、一组 XML 元素的无限定名称、一组 XML 元素的限定名称或属性通 配符“\*”。 当 *propertyName* 参数为无限定名称时， 它标识默认命名空间中的 XML 元素。*value* 参数可以是 XML 对象、XMLList 对象或任何可通过 ToString（） 转换为字符串的值。例如，

**语义学**

当对 XML 对象 *x* 调用 replace 方法并传入参数*propertyName* 和 *value* 时，将执行以下步骤：

1. 如果 *x.[[Class]]* **属于** {"text"**, "comment", "processing-instruction", "attribute"}，** 则返回 *x*

2. 如果 Type（value） 不属于 {XML， XMLList}，则令 *c* = ToString(value

3. 否则，令 *c* 为调用 value 的 [[DeepCopy]] 方法的结果

4. 如果 ToString（ToUint32(P)） 等于 *P*

a. 调用 *x* 的 [[Replace]] 方法，传入参数 *P* 和 *c*，并返回 *x*

5. 设 *n* 为通过调用函数 QName(P) 创建的 QName 对象。

6. 令 i **未定义**

7. 对于 *k* 从 *x.[[Length]] - 1* 递减至 **0** a. 如果 ((n.localName == **"\***

或者 (（x[k].[[Class]] == **"element"）** 并且 (x[k].[[Name]].localName == n.localName

并且（（n.uri **为 null）**或者（（x[k].[[Class]] 等于 **“element”）**并且（n.uri 等于 *x[k].[[Name]].uri*）） i. 如果 i 不是**未定义的**，则调用 *x* 的 [[DeleteByIndex]] 方法，其参数为 ToString(i) 的值。

2。令 *i* = *k*

8. 如果 *i* **未定义**，则返回 *x*

9. 调用 *x* 的 [[Replace]] 方法，传入参数 ToString(i) 和 *c* 。

10. 返回 *x*

[**13.4.4.33**](13.4.4.33) **XML.prototype.setChildren （value ） 概述**

setChildren 方法会用来自 *value* 的一组新的 XML 属性替换此 XML 对象的 XML 属性。*value* 可以是一个单独的 XML 对象或一个 XMLList。setChildren 方 法返回此 XML 对象。例如，

**语义学**

当对 XML 对象 *x* 调用 setChildren 方法并传入参数值时，将执行以下步骤：

1. 调用 *x* 的 [[Put]] 方法，传入参数 **“\*”** 和 *value* 。

2. 返回 *x*

*- 86-*

[**13.4.4.34**](13.4.4.34) **XML.prototype.setLocalName (name)**

**概述**

setLocalName 方法用由给定名称构造的字符串替换此 XML 对象的本地名称。

**语义学**

当在 XML 对象 *x* 上调用 setLocalName 方法并传入参数 *name* 时， 将执行以下步骤：

1. 如果 *x.[[Class]]* 属于 {"text", **"comment"}**， 则返回

2. 如果（Type（name） 是 对象）并且（name.[[Class]] 等于 **“QName”）** a. 令 *name* = *name.localName*

3. 否则

a. 令 *name = name* 的字符串表示形式

4. 令 *x.[[Name]].localName* = *name*

[**13.4.4.35**](13.4.4.35) **XML.prototype.setName( name )**

**概述**

`setName` 方法会用由给定名称构建的 `QName` 或 `AttributeName` 替换此 XML 对象的名称。

**语义学**

当对 XML 对象 *x* 调用 setName 方法并传入参数 *name* 时， 将执行以下步骤：

1. 如果 *x.[[Class]]* 属于 {"text", **"comment"}**， 则返回

2. 如果（`Type(name)` 是 `Object`）并且（`name.[[Class]]` 等于 `**"QName"`）**并且（`name.uri` 为 `**null`）**， 则令 `*name = name*.*localName`* 。

3. 令 *n* 为通过调用构造函数 new QName（name） 创建的新 QName 。

4. 如果 *x.[[Class]]* 等于 **“processing-instruction”**， 则令 *n.u*ri 为空字符串

5. 令 *x.[[Name]]* = *n*

6. 令 ns 为通过调用构造函数 new Namespace（n.prefix, n.uri） 创建的新命名空间。

7. 如果 *x.[[Class]]* 等于 **"attribute**

a. 如果 x.[[Parent]] 为空， 则返回

b. 调用 x.[[Parent]].[[AddInScopeNamespace]](ns 8. 如果 *x.[[Class]]* 等于 **"element**

a. 调用 x.[[AddInScopeNamespace]](ns

[**13.4.4.36**](13.4.4.36) **XML.prototype.setNamespace (ns)**

**概述**

setNamespace 方法会将与此 XML 对象名称相关联的命名空间替换为给定的命名空间。

**语义学**

当对 XML 对象 *x* 调用 setNamespace 方法并传入参数 *ns* 时， 会执行以下步骤：

1. 如果 *x.[[Class]]* **属于** {"text"**, "comment", "processing-instruction"}， 则**返回

2. 令 *ns2* 为通过调用构造函数 new Namespace（ns） 而创建的新命名空间。

3. 令 *x.[[Name]]* 为通过调用构造函数 new QName（ns2, *x.[[Name]]*）而创建的新 *QName* 。

4. 如果 *x.[[Class]]* 等于 **"attribute**

a. 如果 x.[[Parent]] 为空， 则返回

b. 调用 x.[[Parent]].[[AddInScopeNamespace]](ns2 5. 如果 *x.[[Class]]* 等于 **"element**

*- 87-*

a. 调用 x.[[AddInScopeNamespace]](ns2

[**13.4.4.37**](13.4.4.37) **XML.prototype.text ( )**

**概述**

text 方法返回一个 XMLList， 其中包含此 XML 对象中表示 XML 文本节点的所有 XML 属性。

**语义学**

当调用 XML 对象 *x* 的 text 方法时， 会执行以下步骤：

1. 令 *list* 为一个新的 XMLList，其中 *list.[[TargetObject]]* = *x* 且 *list.[[TargetProperty]]* = **null** 。

2. 对于 *i* 从 **0** 到 *x.[[Length]] - 1*

a. 如果 *x[i].[[Class]]* 的值为 **“text”**， 则调用列表的 [[Append]] 方法， 并将 *x[i]* 作为参数传入。

3. 退货清单

[**13.4.4.38**](13.4.4.38) **XML.prototype.toString( )**

**概述**

toString 方法会根据第 10. 1 节中所述的 ToString 转换运算符返回此 XML 对象的字符串表示形式。

**语义学**

当调用 XML 对象 *x* 的toString 方法时， 会执行以下步骤：

1. 返回 ToString(x) 的值

[**13.4.4.39**](13.4.4.39) **XML.prototype.toXMLString( )**

**概述**

`toXMLString()` 方法会根据第 10.2 节中描述的 `ToXMLString` 转换运算符， 返回此 XML 对象的 XML 编码字符串表示形式。与 `toString` 方法 不同， `toXMLString` 对仅包含 XML 文本节点（即原始值） 的 XML 对象不进行特殊处理。无论 XML 对象的内容如何， `toXMLString` 方法始 终会包含 XML 对象的起始标签、属性和结束标签。它用于在不希望使用默认的 XML 到字符串转换规则时的情况。例如，

**语义学**

当调用 XML 对象 *x* 的toXMLString 方法时， 会执行以下步骤：

1. 将 x 转换为 XML 字符串并返回

*- 88-*

[**13.4.4.40**](13.4.4.40) **XML.prototype.valueOf ( )**

**概述**

valueOf 方法返回此 XML 对象。

**语义学**

当调用 XML 对象 *x* 的 valueOf 方法时， 将执行以下步骤：

1. 返回 *x*

**13.4.5 XML 实例的属性**

XML 实例除了从 XML 原型对象继承的属性外， 没有其他特殊属性。

**13.5 XMLList 对象**

**13.5.1 作为函数调用的 XMLList 构造函数**

**语法**

XMLList（值）

**概述**

当将 XMLList作为函数而非构造函数调用时， 它会将参数转换为 XMLList对象。如果其参数本身就是一个 XMLList， 则会原样返回该输入参数。

**语义学**

当将 XMLList作为函数调用并带有参数 *value* 时， 将执行以下步骤：

1. 如果值为 **null、未定义**或未提供， 则将值设为空字符串。

2. 将值转换为 XML 列表并返回：Return ToXMLList(value

**13.5.2 XMLList 构造函数**

**语法**

新的 XMLList（[值]）

**概述**

当 XMLList作为新表达式的一部分被调用时， 它是一个构造函数， 并创建一个新的 XMLList对象。 当 XMLList构造函数未传入任何参数时， 它会返 回一个空的 XMLList。 当 XMLList构造函数传入一个 XMLList 类型的值时，XMLList 构造函数会返回该值的浅拷贝。 当 XMLList 构造函数传入一个 非 XMLList类型的值时， 它会将输入参数转换为 XMLList对象。

**语义学**

当调用 XMLList构造函数并带有可选参数值时， 将执行以下步骤：

1. 如果值为 **null、未定义**或未提供， 则将值设为空字符串。

2. 如果 Type（value） 是 XMLList 类型

a. 令 *list* 为一个新的 XMLList 对象， 且 *list.[[TargetObject]]* = **null**

b. 调用列表的 [[Append]] 方法， 并传入参数 *value*

c. 退货清单

3. 否则

a. 将 value 转换为 XML 列表并返回

*- 89-*

**13.5.3 XMLList 构造函数的属性**

XMLList 构造函数的内部 [[Prototype]] 属性的值是 Function 原型对象。

除了内部属性和长度属性（其值为 1）之外，XMLList 构造函数还具有以下属性：

[**13.5.3.1**](13.5.3.1) **XMLList.prototype**

XMLList.prototype 属性的初始值是 XMLList 的原型对象（第 13.5.4 节）。

此属性具有 { 不可枚举、不可删除、只读 } 属性。

**13.5.4 XMLList 原型对象的属性（内置方法）**

每种 XMLList类型的值都具有一组可用于执行常见操作的内置方法。这些内置方法将在以下各节中进行描述。

XMLList 原型对象本身就是一个 XMLList 对象（其 [[Class]] 属性为**“XMLList”）** ， 其值为空的 XMLList。

XMLList 构造函数的原型对象的内部 [[Prototype]] 属性的值为 Object 原型对象。（ECMAScript 第 3 版第 15.2.3.1

节）

注意：为了简化程序员的工作， E4X 有意模糊了单个 XML 对象与仅包含一个值的 XMLList之间的区别。为此， E4X 扩展了 ECMAScript 函数调用语义， 使得适用于 XML 类型值的所有方法也适用于大小为 1 的 XMLList。有关更多信息， 请参阅第 11.2. 2 节。

XMLList.prototype 上定义的所有内置函数都不是泛型函数。 如果 **this** 值不是 XMLList 对象， 它们会抛出一个 **TypeError** 异常。 因此， 它们不能转移到其他类型的对象上作为方法使用。

[**13.5.4.1**](13.5.4.1) **XMLList.prototype.constructor**

XMLList 原型构造函数的初始值是内置的 XMLList构造函数。

[**13.5.4.2**](13.5.4.2) **XMLList.prototype.attribute ( 属性名称**

**概述**

此 XMLList对象会调用其中每个 XML 对象的属性方法， 并将 *attributeName* 作为参数传递， 然后按顺序返回一个 包含结果的 XMLList对象。

**语义学**

当在 XMLList 对象 *list* 上调用属性方法并传入参数 *attributeName* 时， 将执行以下步骤：

1. 令 *name* = ToAttributeName(attributeName

2. 返回调用列表的 [[Get]] 方法并传入参数 *name* 所得到的结果。

[**13.5.4.3**](13.5.4.3) **XMLList.prototype.attributes( )**

**概述**

此 XMLList 对象中的每个 XML 对象的 attributes（） 方法都会被调用， 然后返回一个 XMLList， 其中包含按顺序 排列的结果。

**语义学**

*- 90-*

当在 XMLList 对象 *list* 上调用 *attributes* 方法时， 会执行以下步骤：

1. 返回调用列表的 [[Get]] 方法并将参数设为 ToAttributeName（"\*"） 所得到的结果。

[**13.5.4.4**](13.5.4.4) **XMLList.prototype.child（属性名称）**

**概述**

此 XMLList对象中的 child 方法会调用每个 XML 对象的 child（） 方法， 并按顺序返回包含结果的 XMLList对象。

**语义学**

当在 XMLList 对象 *list* 上调用 child 方法并传入参数 *propertyName* 时， 将执行以下步骤：

1. 令 *m* 为一个新的 XMLList， 其中 *m.[[TargetObject]]* = *list* 。

2. 对于 *i* 从 **0** 到 *list.[[Length]] - 1*

a. 令 *r* = *list[i].child(propertyName*

b. 如果 *r.[[Length]]* 大于 0， 则调用 *m* 的 [[Append]] 方法，传入参数 *r* 。

3. 返回 *m*

[**13.5.4.5**](13.5.4.5) **XMLList.prototype.children( )**

**概述**

children 方法会调用此 XMLList 对象中每个 XML 对象的 children（） 方法， 并返回一个 XMLList， 其中包含按顺序连接的结果。例如，

**语义学**

当在 XMLList 对象 *list* 上调用 *children* 方法时， 会执行以下步骤：

1. 返回调用列表的 [[Get]] 方法并传入参数**“\*”**的结果。

[**13.5.4.6**](13.5.4.6) **XMLList.prototype.comments ( )**

**概述**

此 XMLList对象中的 comments 方法会调用该对象中每个 XML 对象的 comments 方法， 并返回一个 XMLList， 其中包含按顺序连接的结果。

**语义学**

当在 XMLList 对象 *list* 上调用 *comments* 方法时， 将执行以下步骤：

3. 返回 *m*

*- 91-*

[**13.5.4.7**](13.5.4.7) **XMLList.prototype.constructor**

XMLList.prototype.constructor 的初始值是内置的 XMLList 构造函数。此属性为 E4X 的未来版本保留。

注意：此属性的值无法访问，这实际上使该属性成为只写属性。

[**13.5.4.8**](13.5.4.8) **XMLList.prototype.contains(value)**

**概述**

contains 方法返回一个布尔值， 该值指示此 XMLList对象是否包含与给定值相等的 *XML* 对象。

**语义学**

当对 XMLList 对象 *list* 调用 contains 方法并传入参数 *value* 时， 将执行以下步骤：

2. 返回**假**值

[**13.5.4.9**](13.5.4.9) **XMLList.prototype.copy ( )**

**概述**

copy 方法返回此 XMLList对象的深复制。

**语义学**

当对 XMLList 对象 *list*调用 *copy* 方法时， 会执行以下步骤：

1. 返回调用列表的 [[DeepCopy]] 方法的结果。

[**13.5.4.10**](13.5.4.10) **XMLList.prototype.descendants([名称**

此 XMLList 对象中的每个 XML 对象的 descendants 方法都会被调用， 传入可选参数 *name*（如果省略 *name*， 则传入 字符串 **“\* ”**）， 然后返回一个 XMLList， 其中包含按顺序连接的结果。

**语义学**

当在 XMLList 对象 *list* 上调用 descendants 方法并带有可选参数 *name* 时， 将执行以下步骤：

1. 如果未指定名称，则名称=

2. 返回调用列表的 [[Descendants]] 方法并传入参数 *name* 所得到的结果。

[**13.5.4.11**](13.5.4.11) **XMLList.prototype.elements ( [ name ] )**

**概述**

此 XMLList对象中的每个 XML 对象的 elements 方法都会被调用， 传递可选参数名称（如果省略则为**“\* ”**）， 并按 顺序返回包含结果的 XMLList。

**语义学**

当在 XMLList 对象 *list* 上调用 elements 方法并带有可选参数 *name* 时， 将执行以下步骤：

1. 如果未指定名称，则令名称=

*- 92-*

2. 令 *name* = ToXMLName(name

3. 令 *m* = 一个新的 XMLList 对象，其中 *m.[[TargetObject]]* = *list* 且 *m.[[TargetProperty]]* = *name*

4. 对于 *i* 从 **0** 到 *list.[[Length]] - 1*

a. 如果 *list[i].[[Class]]* 等于 **"element** i. 令 *r* = *list[i].elements(name*

2。如果 *r* 的 *[[Length]]* 属性大于 0，则调用 *m* 的 [[Append]] 方法，并将 *r* 作为参数传入。

5. 返回 *m*

[**13.5.4.12**](13.5.4.12) **XMLList.prototype.hasOwnProperty(P)**

**概述**

`hasOwnProperty` 方法返回一个布尔值，该值指示此对象是否具有由 `*P*` 指定的属性。对于除 `XMLList` 原型对象之外的所有 `XMLList` 对象，此结果与内 部方法 `[[HasProperty]]` 返回的结果相同。对于 `XMLList` 原型对象， `hasOwnProperty` 还会检查本地属性列表， 以确定是否存在具有给定名称的方法属性。

**语义学**

当对 XMLList 对象 *x* 调用 hasOwnProperty 方法并传入参数 *P* 时，将执行以下步骤：

1. 如果调用此对象的 [[HasProperty]] 方法并传入参数 *P* 的结果为**真**，则返回**真**。

2. 如果 *x* 具有名为 ToString(P) 的属性，则返回 **true**

3. 返回**假**值

[**13.5.4.13**](13.5.4.13) **XMLList.prototype.hasComplexContent( ) -> 1XMLList.prototype.hasComplexContent（） 方法**

**概述**

`hasComplexContent` 方法返回一个布尔值，该值指示此 `XMLList` 对象是否包含复杂内容。如果 `XMLList` 对象不为空、包含具有复杂内容的单个 `XML` 项或包含元素，则认为该对象包含复杂内容。

**语义学**

当在 XMLList 对象 *x* 上调用 *hasComplexContent*方法时，将执行以下步骤：

1. 如果 *x.[[Length]]* 等于 **0**，则返回 **false** 。

2. 如果 *x.[[Length]]* 等于 **1**，则返回 *x[0].hasComplexContent*（）的值。

3. 对于 *x* 中的每个属性 *p*

a. 如果 *p.[[Class]]* 等于 **“element”**，则返回 **true**

4. 返回**假**值

[**13.5.4.14**](13.5.4.14) **XMLList.prototype.hasSimpleContent( )**

**概述**

hasSimpleContent 方法返回一个布尔值，该值指示此 XMLList是否包含简单内容。如果 XMLList对象为空、包含一个具有简单内容的 XML 项或不包含任 何元素，则认为其包含简单内容。

**语义学**

当对 XMLList 对象 *x* 调用 *hasSimpleContent*方法时，将执行以下步骤：

2. 返回**真**

*- 93-*

[**13.5.4.15**](13.5.4.15) **XMLList.prototype.length ( )**

**概述**

length 方法返回此 XMLList 对象中的属性数量。例如，

**语义学**

当调用 XMLList 对象 *list* 的 length 方法时， 会执行以下步骤：

1. 返回列表的长度。

[**13.5.4.16**](13.5.4.16) **XMLList.prototype.normalize ( )**

**概述**

normalize 方法会将此 XMLList 中的所有文本节点、它所包含的所有 XML 对象以及这些 XML 对象的所有后代节点合 并相邻的文本节点并删除空文本节点， 从而将它们转换为规范形式。该方法返回此 XMLList对象。

**语义学**

当在 XMLList 对象 *list* 上调用 *normalize* 方法时， 将执行以下步骤：

[**13.5.4.17**](13.5.4.17) **XMLList.prototype.parent( )**

**概述**

如果此 XMLList对象中的所有项具有相同的父项， 则返回该父项。否则，parent 方法返回**未定义。**

**语义学**

当在 XMLList对象 *list* 上调用父方法时， 将执行以下步骤：

1. 如果 *list.[[Length]]* 等于 **0**， 则返回**未定义**。

2. 令 *parent* = *list[0].[[Parent]]*

3. 对于 *i* 从 **1** 到 *list.[[Length]] - 1*， 如果 *list[i].[[Parent]]* 不等于 *parent*， 则返回**未定义**。

4. 返回父级

*- 94-*

ecma INTERNA

TIONAL

[**13.5.4.18**](13.5.4.18) **XMLList.prototype.processingInstructions ([name]) 概述**

此 XMLList 对象中的 processingInstructions 方法会调用该对象中每个 XML 对象的 processingInstructions 方法， 并传递可选参 数 name（如果省略则为“\*”） ， 然后按顺序返回包含结果的 XMLList。

**语义学**

当在 XMLList 对象 list 上调用 processingInstructions 方法并带有可选参数 name 时， 将执行以下步骤：

6. 如果未指定名称， 则令名称=

7. 令 name = ToXMLName(name

4. 让 m 等于一个新的 XMLList， 其中 m.[TargetObjecf] = list

9. 对于 i 从 0 到 list.[[Length]] - 1

a. 如果 listf[Class] 等于 "element

i. 令 r = list].processingInstructions(name

2。如果 r 的长度大于 0， 则调用 m 的 [Appen] 方法并将 r 作为参数传入。

10. 返回 m

[**13.5.4.19**](13.5.4.19) **XMLList.prototype.propertyIsEnumerable (P)**

概述

`propertyIsEnumerable` 方法返回一个布尔值， 该值指示当此 `XMLList` 对象在 `for-in` 语句中使用时， 属性 P 是否会被包含 在迭代的属性集中。

**语义学**

当对 XMLList 对象 x 调用 propertyIsEnumerable 方法并传入参数 P 时， 将执行以下步骤：

1. 如果 ToNumber(P) 大于或等于 0 且 ToNumber(P) 小于 x.[Length]l， 则返回 true

2. 返回假值

[**13.5.4.20**](13.5.4.20) **XMLList.prototype.text() 概述**

此 XMLList对象调用其中每个 XML 对象的 text 方法， 并返回一个 XMLList， 其中包含按顺序连接的结果。

**语义学**

当在 XMLList 对象 list 上调用 text 方法时， 将执行以下步骤：

1. 令 m 为一个新的 XMLList， 其中 m.[TargetObjecf] = list

2. 对于 i 从 0 到 list.[Length]-1 a. 如果列表-[类] == "元素

i. 设 r = listjl.text

2。如果 r [Lengthil 2 0， 则调用 m 的 FAppend 方法并将 r 作为参数传入。

3. 返回 m

- 95-

ecma INTERNATI ONAL

[**13.5.4.21**](13.5.4.21) **XMLList.prototype.toString() 概述**

toString 方法会根据第 10. 1 节中所述的 ToString 转换运算符返回此 XMLList 对象的字符串表示形式。

**语义学**

当调用 XMLList 对象 list 的 toString（） 方法时， 会执行以下步骤：

1. 返回 ToString（ist） 的值

[**13.5.4.22**](13.5.4.22) **XMLList.prototype.toXMLString() 概述**

`toXMLString()` 方法会按照第 10.2 节中所述的 `ToXMLString` 转换运算符，返回此 `XMLList` 对象的 XML 编码字符串表示形式。 与 `toString` 方法不同， `toXMLString` 对仅包含 XML 文本节点（即原始值） 的 XML 对象不作特殊处理。`toXMLString` 方法总 是对包含在此 `XMLList` 对象中的每个属性调用 `toXMLString`，按顺序连接结果， 并返回一个字符串。

**语义学**

当调用 XMLList 对象 list 的 toXMLString（） 方法时， 会执行以下步骤

1. 将列表转换回 XML 字符串

[**13.5.4.23**](13.5.4.23) **XMLList.prototype.valueof() 概述**

valueOf 方法返回此 XMLList 对象。

**语义学**

当调用 XMLList 对象 list 的 valueOf 方法时， 将执行以下步骤：

1. 退货清单

**14 个错误**

E4X 扩展了错误列表。实现时无需按照以下规定报告错误：

对于将类型为 Object 的参数传递给 ToXML 函数和 ToXMLList 函数的情况， 实现可以定义除抛出 TypeError 异常之外 的行为。

- 96-

ecma INTERNAT IONAL

**附 录 A （规范性）**

**可选功能**

本附件描述了一些选项。使用这些特性的 E4X 实现应符合以下相关规范。

**A.1 XML 内置方法**

E4X 实现可能会向 XML 对象添加以下方法。

**A.1.1 domNode() 概述**

domNode 方法返回此 XML 对象的 W3C DOM 节点表示形式。

**语义学**

domNode 方法的语义取决于具体实现。

**A.1.2 domNodeList() 概述**

`domNodeList` 方法返回一个 W3C DOM `NodeList`， 其中包含此 XML 对象的单个 W3C DOM `Node` 表示形 式。

**语义学**

domNodeList 方法的语义取决于具体实现。

**A.1.3 XPath（XPath表达式） 概述**

`xpath` 方法根据 W3C XPath 推荐标准使用此 XML 对象作为上下文节点来评估 `XPathExpression`。 在评估 `

XPathExpression` 之前， `xpath` 方法会按如下方式设置 XPath 上下文：上下文节点设置为此 XML 对象；上下文位置设置 为 1；上下文大小设置为 1；变量绑定集设置为空集； 函数库设置为空集；命名空间集根据此 XML 对象的作用域内命名 空间集进行设置。如果 `XPathExpression` 评估结果为节点列表， `xpath` 方法将结果作为 `XMList` 返回；否则， `xpath` 方 法将抛出 `TypeError` 异常。例如，

**语义学**

当对 XML 对象 x 调用 xpath 方法并传入参数 XPathExpression 时， 它会执行以下步骤：

1. 设 e 为 XPathExpression 表达式求值的结果。

2. 令 s = GetValue(XpathExpression).ToString

3. 创建一个表示 XML 对象 x 的 XPath 上下文对象。此步骤的语义取决于具体实现。

4. 将 XPath 上下文位置设为 1

5. 让 XPath 上下文大小为 1

6. 令 XPath 变量绑定为 {}

7. 令 XPath 函数库为空。

8. 让 XPath 命名空间为 x.[/nScopeNamespaces]]

9. 设 r 为依照 W3C XPath 推荐标准对 s 进行求值的结果。

10. 如果 ris 一个 XPath 节点集， 则以实现相关的方式将其转换为 XMLList 并返回。

- 97-

ecma INTERNATI

ONAL

**11.抛出一个类型错误异常**

**A.2 XMLList 内置方法**

E4X 实现可能会向 XMLList对象添加以下方法。

**A.2.1 domNode() 概述**

如果此 XMLList对象仅包含一个 XML 值 x， 则 domNode 方法返回 x 的 W3C DOM 节点表示形式。否则， domNode 方法返回未定义。

**语义学**

domNode 方法的语义取决于具体实现。

**A.2.2 domNodeList() 概述**

domNodeList 方法返回此 XMLList 对象的 W3C DOM NodelList 表示形式。

**语义学**

domNodeList 方法的语义取决于具体实现。

**A.2.3 XPath（XPathExpression） 概述**

此 XMLList对象中包含的每个 XML 属性的 XPathExpression 都会由 xpath 方法进行求值， 并将结果按顺序连接起来， 生成一个包含连接结果 的 XMLList。

**语义学**

当在 XMLList 对象 list 上调用 xpath 方法并传入参数 XPathExpression 时， 将执行以下步骤：

1. 令 m = 一个新的 XMLList， 其中 list.[TagetObject)] = null

2. 对于 i 从 0 到 list.Length - 1

a. 如果 Type（list） 是 XML 且 list-[Class] 等于 “element” i. 让 r = listj.xpath(XPathExpression

2。如果 r[[Lengthi]] 大于 0， 则调用 m 的 lAppend 方法并将 r 作为参数传入。

3. 返回 m

- 98-