IBM 软件学院
IBM Software Institute

XPath

本章目标

- 通过本章学习，您应该能够：
  - 了解什么是 XML 转换。
  - 了解 XML 转换技术：CSS 及 XSL，以及它们的区别。
  - 理解 XPath 表达式及数据类型。
  - 理解 XPath 中的节点类型。
  - 理解用来定位 XML 数据的 XPath 定位路径。
  - 掌握常用 XPath 核心函数。

什么是 XML 转换？

- XML 转换就是把 XML 文档转换为其他格式或结构的文档，包括其他结构模型的 XML 文档、HTML 文档，以及甚至是 PDF 文档等。
- 现有的 XML 转换技术可以把 XML 文档转换为：
  - 一般文本文档。
  - 另外一种数据结构的 XML 文档。
  - XHTML，一种格式比 HTML 严格，但功能跟 HTML 差不多。
  - 其他格式的文档，例如 PDF 文档。

XML 转换的几种技术

- 现在 XML 转换技术主要有：
  - Cascading Style Sheets（CSS2）：
    - CSS2 是 CSS1 的一个升级版，CSS 本来主要用在 HTML。
  - eXtension Stylesheet Language（XSL）：
    - XSL 是一种完全用来转换 XML 的语言。

级联样式表（CSS2）

- 级联样式表（Cascade StyleSheet，CSS） 也是 W3C 的一个推荐标准，它最早是为 HTML 定义的，它能为 HTML 中的元素定义其内容的显示效果，例如字体、大小、颜色等等。
- CSS2 是在 CSS1 的基础上制定的，且在原来的基础上新增了一些功能。
- CSS2 现在也可以用在 XML 上了，即可以为 XML 文档中的元素定义它们的显示位置、字体、大小、颜色、边框等等。CSS2 可以为 XML 中的某一个元素、某一类元素、某一批元素定义它们的显示效果。

CSS 的缺陷

- CSS 不能执行计算、重新整理或排序数据、组合多个源码中的数据或根据用户或会话的特征个性化显示的内容。
  (1) CSS不能重新排序文档中的元素；
  (2) CSS不能判断和控制哪个元素被显示，哪个不被显示；
  (3) CSS不能统计计算元素中的数据；

可扩展样式表语言（XSL）（1 of 2）

- XSL 本身也是 W3C 标准组织推荐的一种标准。
- XSL 可以执行的任务可以分成两个阶段：
  - 转换（Transformation）
    - 转换是将一个 XML 文档（或其内存中的表示法）转换成另一个 XML 文档的过程。
  - 格式化 （Formatting）
    - 格式是将已转换的树状结构转换成二维图形表示法或可能是一维字节或音频流的过程。

可扩展样式表语言（XSL）（2 of 2）

- 为了完成转换及格式化任务，XSL 包括了 3 个子语言：
  - **XPath**：一种用来定位 XML 文档节点及内容的语言。
  - **XSLT**：一种转换 XML 的语言。
  - **XSL 格式对象（XSL Formatting Object，XSL FO）**：一种定义 XML 显示方式的语言。
- 1999年11月，XSLT 和 XPath 被作为两个单独的 W3C 推荐标准发布。

XPath 介绍

- XPath 提供了一个通用的句法和语义来查询定位 XML 文档中的节点。
- XPath 把 XML 文档模型化为一棵树，我们可以称之为“节点树”。

- XPath 完全支持 XML 命名空间，因此节点的名称必须是一个扩展名称（Expanded-name），即该名称包括两个部分：本地部分及命名空间 URI，如果节点不属于任何命名空间，命名空间 URI 部分可以是空。

节点相关的一些概念

- 上下文节点（Context Node）
  - 上下文节点就是在表达式中使用相对路径时的相对基路径，非常类似于URL 中的相对基路径。
- 当前节点（Current Node）
  - 在一个表达式运算期间，每一个当前正在被运算的节点称之为当前节点。
- 上下文位置（Context Position）及上下文大小（Context Size）
  - 上下文位置就是 XPath 表达式中的任一个节点相对于上下文节点的位置。而上下文大小就是表达式中的任一个节点处正在被运算的节点的个数。
  - 上下文位置总是小于或等于上下文的大小。

XPath 中的节点类型

- XPath 点树中包含的节点的类型包括：
  - 根结点（Root Node）
  - 元素节点（Element Node）
  - 属性节点（Attribute Node）
  - 文本节点（Text Node）
  - 命名空间节点（Namespace Node），可以认为是一种特殊的属性节点。
  - 处理指令（PI）节点
  - 注释节点（Comment Node）
- 对于任何一种节点类型，XPath 都有对应的一种计算其字符值的方法。

文档顺序（Document Order）

- 在 XPath 的 XML 节点树中，包含的节点都是按照一定的顺序排列的，这就是文档顺序。
- 在 XPath 中，文档顺序是按照节点在 XML 文档中出现的顺序排列的，即从上到下，从左到右。
- 属性节点及命名空间节点出现在其子元素节点之后，而命名空间节点又出现在属性结点之前。

- 每一个元素节点及根结点都有一个排好序的**子节点（children）**。每一个节点，除了根结点，都会有唯一的一个**父节点（parent）**，而且这个父节点只能是元素节点或根结点。**子孙节点（descendant）**是该节点的子节点及这些子节点的子孙节点。

根节点（Root Node）

- 根结点是节点树的根，即文档中的所有节点（包括处理指令节点、根元素节点等）都是根结点的子孙节点。
- 文档根元素节点是根节点的子节点。

- 注意：根结点与文档根元素节点是不同的，文档根元素节点是 XML 文档的第一个顶层元素（Top Element）对应的元素节点。

元素节点

- XML 文档中的每一个元素都对应一个元素节点，而且每一个元素节点都有一个扩展名。
- 元素节点可以包含的子节点包括：元素节点、属性节点、处理指令节点及对应其文本内容的文本节点。
- 实体引用不管是内部实体还是外部实体，都需要扩展为实体内容，而且字符引用也会被解析。

- 一个元素的字符值是其本身包含的文本内容，或者是所有其子孙元素节点的文本值的串联。

属性节点

- XML 中的每一个元素的属性都对应一个属性节点，包含它的元素节点是该属性节点的父节点，但是它不是其父元素节点的子节点。
- 元素从来不会共享它的属性节点，也就是说如果一个元素节点与另外一个元素节点不相同，那么它们的属性节点肯定不一相同。

- 一个属性节点有一个扩展名及字符值。

文本节点

- XML 文档中的字符数据被组织为文本节点。文本节点只有一个父节点，而不会有任何的子节点及兄弟节点。
- 文本节点的字符值就是字符数据本身。
- 注意：在注释、处理指令及属性值中的字符数据不会生成文本节点，也就是说注释节点、处理指令结点不会是文本节点的父节点。

XML 实例——一年的 XML 表达

```
<year>
   <planting>
            <season  period="spring" >
                     <month>March</month>
                     <month>April</month>
                     <month>May</month>
            </season>
            <season  period="summer">
                     <month>June</month>
                     <month>July</month>
                     <month>August</month>
            </season>
   </planting>
   <harvest>
            <season  period="autumn">
                     <month>Setptember</month>
                     <month>October</month>
                     <month>November</month>
            </season>
            <season  period="winter">
                     <month>December</month>
                     <month>January</month>
                     <month>February</month>
            </season>
   </harvest>
</year>
```

XPath 表达式介绍

- XPath 表达式是 XSLT 处理器用来计算结果的一种表述方法，结果是一个专门类型的对象。
- XPath 的表达式的计算结果对象有下面 4 种：
  - **节点集（node-set）**，一个为排序的没用重复的节点集合。
  - **布尔值（boolean）**，值可以是“true”或者“false”。
  - **数字（number）**，一个浮点数值。
  - **字符串（string）**，字符串数据。
- 定位路径是一种特殊的表达式，也是 XPath 中最重要的表达式，它用来从一个 XML 文档中获取节点。

节点集（Node Set）对象

- 一个节点集是一组节点的无序组合，它是 XPath 表达式运算的直接结果。
- 节点集能够包含来自任意七种不同类型的节点。

- 节点集中每一个节点都被认为是集合中其他节点的兄弟节点。
- 如果节点集中的节点包含子节点，这些子节点并不是节点集的一部分。

布尔对象

- 一个 XPath 表达式计算并返回一个 true 或 false 的布尔类型对象。
- XPath 中包括下列运算符：
  - or
  - and
  - > （大于）
  - < （小于）
  - = （等于）
  - != （不等于）
  - <= （小于或等于）
  - >= （大于或等于）
  - |

数字对象

- XPath 表达式运算时可以产生一个数字对象。一个数字对象就是一个数字或一组数字。
- 通过“计算”节点的方法运算一个表达式常常会产生数字对象。

字符串对象

- 最显而易见的东西也是最不明显的，当用字符串去解释或定义某事时，一个字符串就仅仅是一个字母序列，不管它是特定元素的内容、属性值、属性名称、元素类型名称还是表达式中的符号。

XPath 中的定位路径（Location Paths）

- 定位路径是一种特殊的表达式，也是 XPath 中最重要的表达式，它用来从一个 XML 文档中获取节点。
- 一个 XPath 定位路径包括一系列的（一个或多个）定位步长（Location Step），通过“/”分隔符分离。

- 每一个定位步长从左到右，根据之前的定位步长（如果有的话）在文档的节点树中定位它的位置。
  - 绝对定位步长除了能出现在定位路径的最前面外，不能出现在其他任何地方。
  - 相对定位步长可以出现在定位路径的任何地方。

绝对定位步长及绝对定位路径

- XPath 中的绝对定位步长表示要定位的内容从 XML 目标文档的根开始查。
  - 语法上讲，绝对定位步长以“/”分隔符开始，“/”表示从文档的根结点开始。
- 以绝对定位步长开始的定位路径就是绝对定位路径。类似于文件系统或 URL 中的绝对路径或绝对 URL 路径。

相对定位步长及相对定位路径

- 相对定位步长就是一条与上下文节点相关的路径，上下文节点可以是绝对步长指定，也可以是由相对步长指定。
- 相对定位路径就是以相对定位步长开始的路径，所以相对定位路径就是一个或多个用“/”分隔开的定位步长的序列。

构造定位步长（Location Step）

- 在 XML 层次结构中从一层到另一层的移动在一条定位路径中称为一个定位步长。在一个定位路径中，每一个定位步长由“/”分隔开。
- 一个定位步长由一个轴（Axis）、一个节点测试（Node-Test）及一个或多个可选的判断语（Predicate）三部分组成，如下所示：

  axis::nodetest[predicate]
- 轴是定位步长和上下文节点之间的关系。
- 节点测试定义了节点扩展名，而且该节点的类型由轴来指定。
- 判断语是一个表达式。

构造定位步长的轴（Axis）部分

- 在一个*定位路径*中轴是一种相互关系，是*定位步长*本身与*上下文节点*之间的关系。
- 在 XPath 中轴的类型或关系有：

1. parent::
2. child::
3. ancestor::
4. descendant::
5. ancestor-or-self::
6. descendant-or-self::
7. preceding::
8. following::
9. preceding-sibling::
10. following-sibling::
11. self::
12. attribute::
13. namespace::

parent 轴

- parent 轴包含上下文节点的父节点。
- 对于任意给定的节点，只可能有一个父节点。

- 从文档顺序的角度考虑时，parent 轴是一个逆向轴。一个逆向轴表示了与正常文档顺序相反的方向。

ancestor 轴

- ancestor 轴包含所有在上下文节点以上或更多层的节点。
- ancestor 轴总是包含根结点，除非上下文节点就是根结点。
- 从文档顺序角度看，ancestor 轴也是一个逆向轴。

descendant 轴

- descendant 轴包含在当前上下文节点以下的一层或更多层的所有节点。
- descendant 轴可以使用缩写“//”表示。
- 从文档顺序上看，descendant 轴是向前轴，向前轴是文档顺序的通常方向。

following-sibling 轴

- following-sibling 轴包含 在文档顺序中作为上下文节点同一层次向后的节点并共享相同的父节点。
- 如果上下文节点是一个属性或命名空间节点，那么following-sibling 轴返回的是一个空结点集。这是因为属性节点和命名空间节点都没有任何的顺序。
- 从文档顺序上看，following-sibling 轴是向前轴。

preceding-sibling 轴

- preceding-sibling 轴可以认为是 following-sibling 轴意义相反的轴，即preceding-sibling 轴包含上下文节点向前的同胞节点。
- 如果上下文节点是一个属性或命名空间节点，那么preceding-sibling 轴返回的是一个空结点集。
- 从文档顺序上看，preceding-sibling 轴是逆向轴。

following 轴

- following 轴包含在文档顺序中上下文节点后面的节点，但是，上下文节点的后代节点不包括在 following 轴中；而所有向下的同胞节点及它们的所有后代节点都包含在 following 轴中。
- following 轴可以用来从一个节点分支跳到另外一个分支。
- 从文档顺序上看，following 轴是向前轴。

preceding 轴

- preceding 轴包含上下文节点在文档顺序之前的所有节点，但是这些不能是上下文节点的祖先。
- 从文档顺序上看，preceding 轴是逆向轴。

attribute 轴

- attribute 轴包含上下文节点的所有属性节点。
- 只有元素节点才会有 attribute 节点。
- attribute 轴可以表示为 attribute::，但是很少有人会这样表示它，而是一般使用它的缩写形式来表示，它的缩写形式是“@”。
- 从文档顺序上看，属性轴是一个无序轴。

namespace 轴

- namespace 轴包含上下文节点的名称空间节点，除非这个上下文节点是一个元素，否则 namespace 轴将是空的，即只有元素节点才会有 namespace 节点。
- 命名空间跟属性一样，并没有特定的文档顺序，它是一个无序轴。

self 轴

- self 轴返回上下文节点本身。例如：

  self::season

  – 此时当前仅当上下文节点是 season 时，self 轴返回的节点才不会空

descendant-or-self 轴

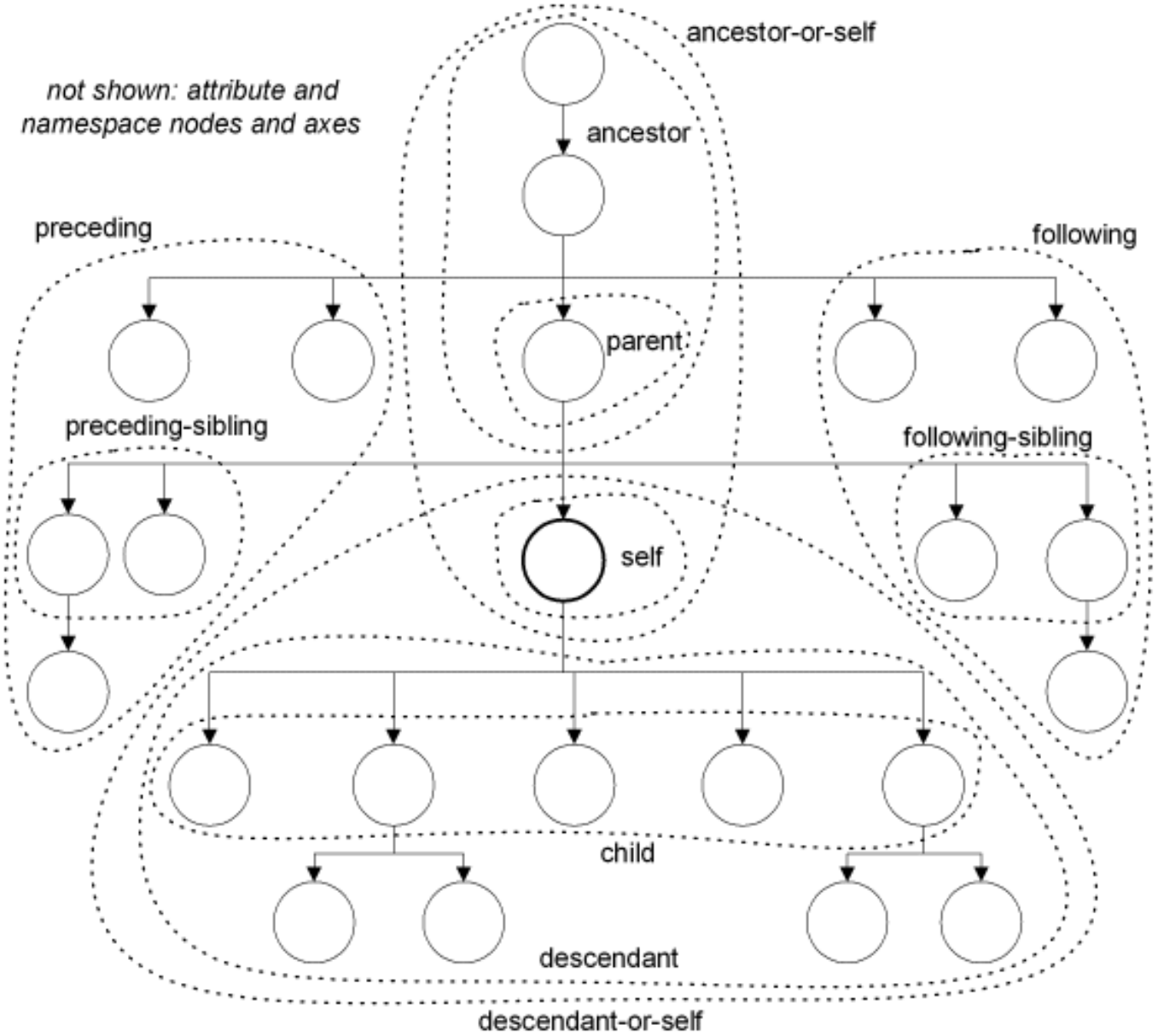
- descendant-or-self 轴包含了上下文节点和它的所有后代节点，即descendant-or-self 轴包含的节点比 descendant 轴包含的多了上下文节点本身。

ancestor-or-self 轴

- ancestor-or-self 轴包含了上下文节点和它的所有后代节点，即 ancestor -or-self 轴包含的节点比 ancestor 轴包含的多了上下文节点本身。

child 轴

- child 轴是一个最常用的轴，它表示上下文节点的子节点。child 轴可以表示为 child::，但很少这样使用，因为定位步长的缺省轴就是 child。
- 任何节点都会是某个节点的子节点，除了根结点不会是任何节点的子节点，所以 child 轴是缺省轴。
- 从文档顺序上看，child 轴是一个向前轴。



构造定位步长的节点测试（Node-Test）部分

- 每一个定位步长都包含一个节点测试，这个节点测试包含处理器必须测试的节点名称。
- 节点测试的作用：如果轴中包含的节点的名称匹配节点测试的名称，那么这些节点将被选中。
- 节点测试基于轴来决定它所选择的节点的类型。
- 节点测试出现在轴之后，用“::”分开。
- 节点测试中可以使用通配符，如“*”。

构造定位步长的判断语（Predicate）部分

- 判断语为一个定位步长增加一个额外的过滤层次。
- 它们在定位步长的上下文节点范围内过虑由轴及节点测试所选择的节点集。
- 判断语用一对方括弧来表示，即“[]”记号，用额外的表达式来完成额外层次的节点过虑。
- 判断语的基本形式是：

  [something operator somevalue]

  其中：

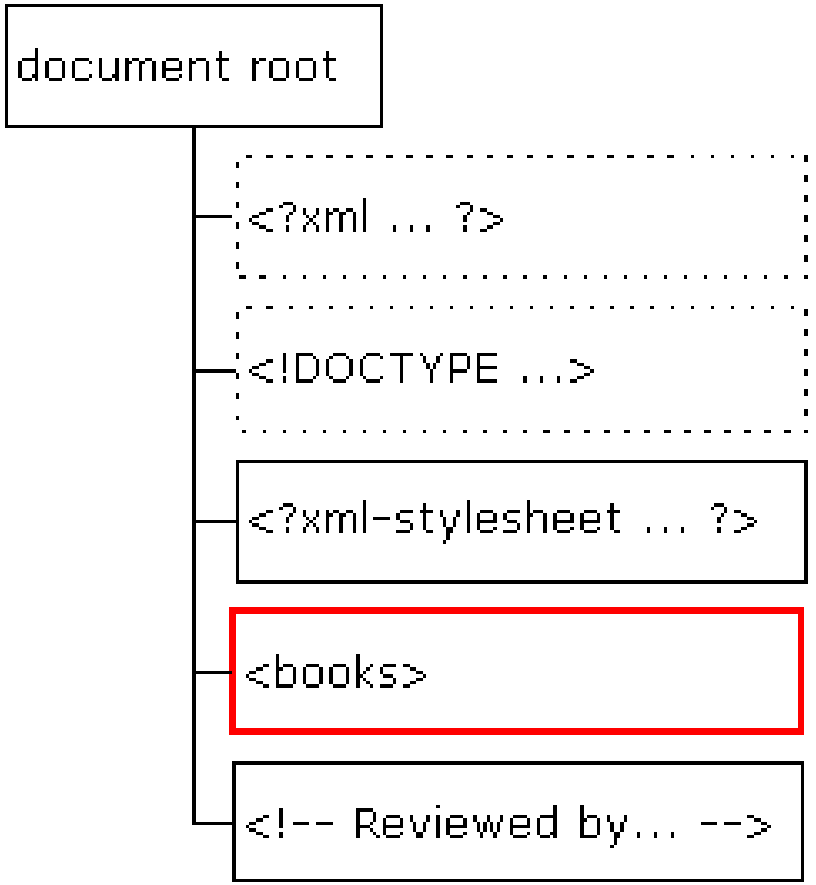
  - 方括号“[”和“]”是必须的。
  - something 可以是另外一个定位步长，这个步长使用的上下文节点为通过轴及节点测试得到的节点，或者也可以是内置 XPath 核心函数。
  - 操作符就是其中一个布尔运算符，例如等于测试“=”。
  - somevalue 就是您想比较的值。

定位路径实例讲解——XML 实例文档

```
<?xml version="1.0" ?>
<!DOCTYPE books [
<!ENTITY oumlaut "&#38;#246;">
]>
<?xml-stylesheet type="text/xsl" href="catalog.xsl" ?>
<books catdate="2000-12-31">
   <book catnum="id2345">
      <title>Jambing on the Trixles</title>
      <!-- Are we sure this guy's name is spelled right??? -->
      <author>Randall, Tristan</author>
   </book>
   <book catnum="id7823">
      <title>For Love of a Toothpick</title>
      <author>Frey, J&oumlaut;rg</author>
   </book>
</books>
<!-- Reviewed by Li Shuigen 2003-12-30 09:11:16 -->
```

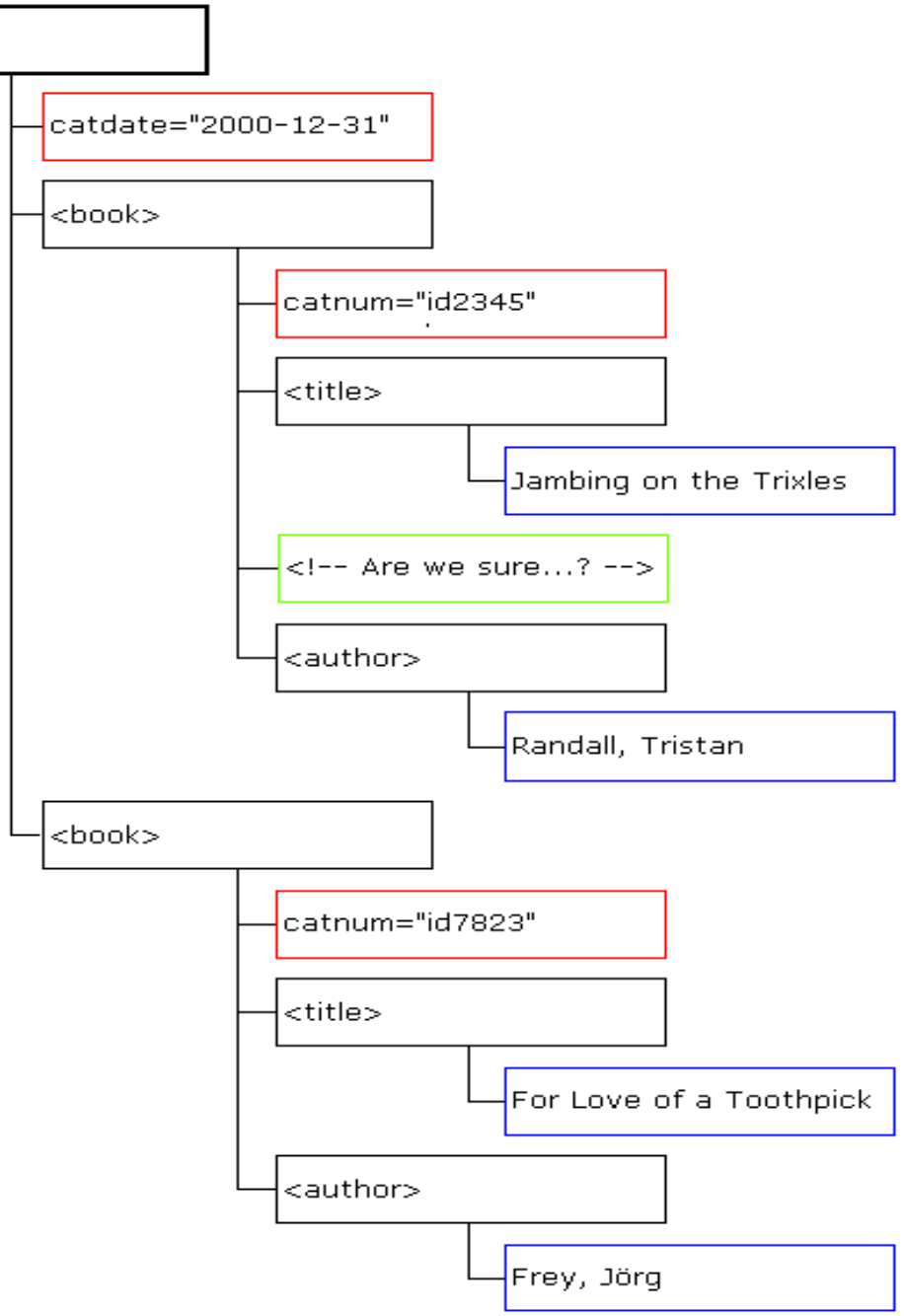
定位路径实例讲解——文档节点树

- 右图包含六个节点：
  - 根结点
  - <?xml … ?>处理指令节点
  - 文档类型定义，也是一种处理指令节点
  - <?xml-stylesheet?> 处理指令节点
  - <books> 元素节点（该 XML 文档的根元素节点）
  - <books> 元素后的注释节点



→ <books>节点树

- 右图中包含的节点包括元素结点（黑色矩形框）、属性节点（红色矩形框）、注释节点（绿色矩形框）及文本节点（篮色矩形框）。



表达式及路径定位实例

- / ： 定位根路径
- **/books**：定位根路径下的所有名为“books”的子元素节点
- **/descendant::title** ：定位根路径下所有的<title>元素节点
- **/comment() | /text()** ：定位根路径下的所有注释节点及文本节点
- **/processing-instruction("xml-stylesheet")**：定位任何目标名称为“xml-stylesheet”的处理指令
- **/books/child::*[position()=1] 或/books/*[position()=1]** ：定位<books> ：元素节点下第一个子元素节点，即属性 catnum 的值为“id2345”的<book>元素节点
- **ancestor-or-self::books[@catdate=“2000-12-31”]** ：定位上下文节点的名称为“books”的祖先节点或它自己，且它有一个值为“2000-12-31”的“catdate”属性。

XPath 核心函数介绍

- 函数是 XPath 中内置的程序，用来处理一些操作及返回一些值。返回的值将替换函数调用部分，就像一般编程语言中的函数或方法。
- XPath 中函数调用的格式如下：

  functionname(arg1...)

  其中：

  – “functionname”为要调用函数的名称。

  – “arg1..”是一组用逗号分隔的函数自变量。

XPath 核心函数类型

- XPath 函数库包括4组（共27个函数）：
  - 节点集函数组
  - 布尔函数组
  - 数字函数组
  - 字符串函数组
- 注意：函数库的分类不是根据函数的返回值分的，而是根据传入参数。

节点集函数组介绍

- 节点集函数组就是那些传入的参数是节点或节点集的函数，它们的返回值可以是任何类型。 包括：
  - **last()**：返回当前选中节点集的最后一个节点的位置，是一个数字
  - **position()** ：返回当前正在处理的节点处于选中的节点集的位置，是一个数字。经常用在判断式，例如 [position()=3]，返回节点集中的第三个节点。
  - **count(ns)**：返回当前选中的节点集的节点数量。
  - **local-name(ns?)**：返回传入节点的本地部分。
  - **namespace-uri(ns?)**：返回传入节点的命名空间 URI 部分
  - **name(ns?)**：返回传入节点的完整扩展名称，包括命名空间 URI 部分（如果有的话）。

布尔函数组介绍

- 布尔函数组中的函数要求传入的参数是布尔变量，返回值一般也是布尔值。 包括：
  - **boolean(obj):** 主要用来测试指定的“obj”是否“存在”。如果 obj 指定的是一个节点集，当且仅当指定节点集不为空时返回的结果才为真。如果是一个字符串，当且仅当字符串的长度大于 0 时返回值才为真。如果是一个数字，当且仅当数字大于 0 才返回真。其他一切情况返回的结果都是假。
  - **not(boolean):** 返回传入值的相反，即传入真时返回假，传入假时返回真。
  - **true()**：简单地返回 true
  - **false()**：简单地返回 false
  - **lang(str)**：返回值为布尔值，根据上下文节点是否有 xml:lang 属性，且它的值是否等于“str”指定的。如果有且相等，则返回 true。

数字函数组介绍

- 数字函数组包括：
  - number(obj)：将一个对象转换为数字。
  - sum(ns?)：把传入的节点集中每一个节点转换为数字并求和。
  - floor(num)：提供一个向后的舍入功能。
  - ceiling(num)：寻找最近的一个整数，大于或等于其参数提供的数字。
  - round(num) ：返回四舍五入的数字结果。

字符串函数组介绍

- 字符串函数组包括：
  - string()：以 XPath 中的四种类型对象为参数，并将其转换为一个字符串。

    原型：`string string(object?)`
  - concat()：传入参数为字符串或模式表达式，返回由两个或更多字符串组成的字符串。

    原型：`string concat(string, string, ...)`
  - substring()：返回字符串指定位置的字符串。

    原型：`string substring(string,number,number?)`
  - contains()：用于检查原始的字符串中是否有一个子字符串存在。

    原型：`boolean contains(string,string)`

本章小结

- 我们本章介绍了：
  - 什么是 XML 转换。
  - XML 转换技术：CSS 及 XSL，以及它们的区别。
  - XPath 表达式及数据类型。
  - XPath 中的节点类型。
  - 用来定位 XML 数据的 XPath 定位路径。
  - 常用 XPath 核心函数。