1. **Pocessing**
2. **XPath**
3. **XPath：**用在XPoint、XSL、XML Schema、XQuery中在XML树中定位节点和片段的语言。
4. **XLink：**用XPointer定位资源，HTML中link的衍生，更有表现力。
5. **XPointer：**XPath的延申。
6. **XPath**是W3C标准，本身不用在XML中，但是在XSLT中用很多。
7. /library/book/chapter/section其中library是根节点，索引到section

**.** 表示当前节点，**..**表示父节点。

/library/book/chapter/\* 表示在/library/book/chapter下的所有元素。

如果**开头不是/**，贼表示是从当前节点开始，eg. book/chapter

**开头为//**表示可以从任何节点开始，但是花销很大，因为要搜索整个文件。

1. **取子节点时从1开始。**

Eg. /library/book[1]/chapter[1]/section[2]

Last( )函数。 eg. book/chapter[last()-1]

1. //\* 取出文件中所有的元素。**不取属性**

/\*/\*/\*/paragraph，取出有3个祖先的paragraph。

1. **取属性，使用@**

**Eg. //chapter[@num]** 取出包含num属性的所有chapter。

**//chapter[@num=‘3’]** 取出包含num属性值为3的所有chapter。

**//chapter[not(@num)]**选出不包含属性num的chapter。

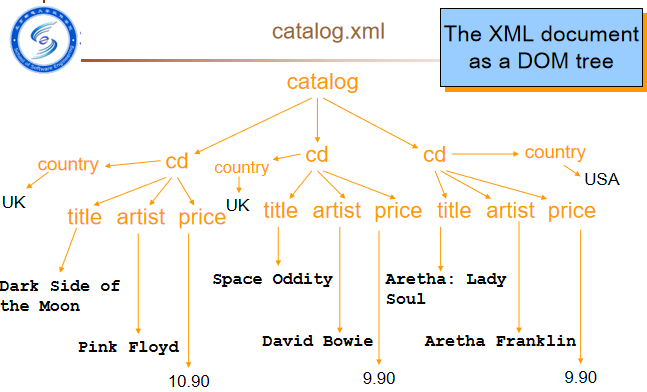
**//chapter[@\*]**选出有任意属性的chapter。

**//chapter[not(@\*)]**选出不包含属性的chapter。

1. **Normalize-space( )函数**可以在比较前去掉值里的空格，将其标准化。

**//chapter[normalize-space(@num)=”3”]**

1. **DOM tree的画法，注意属性和文本也是节点**



**/catalog/cd[price<10] 条件**

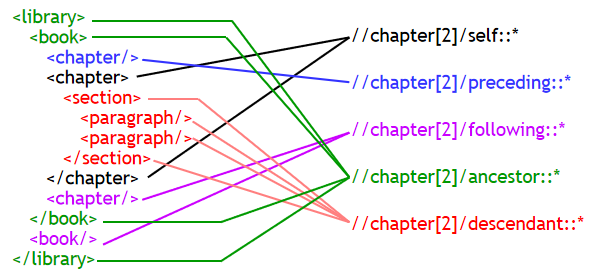
**//title| //price** 选出所有的title和price

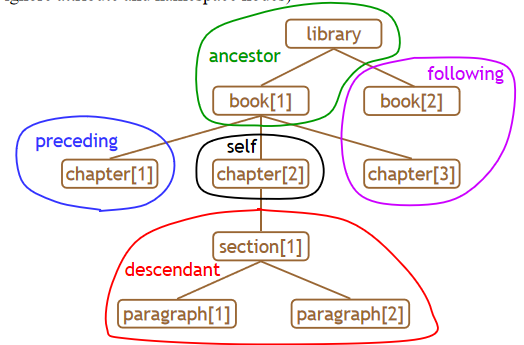
**//\*** 选出所有节点

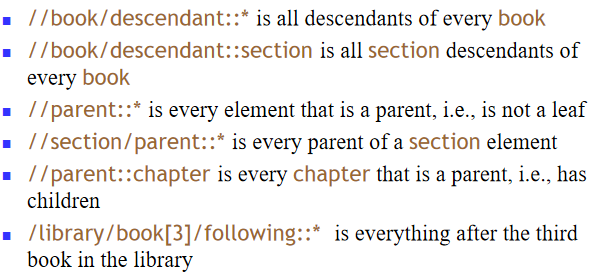
**//\* [price==9.90]** 选出所有price为9.90的cd

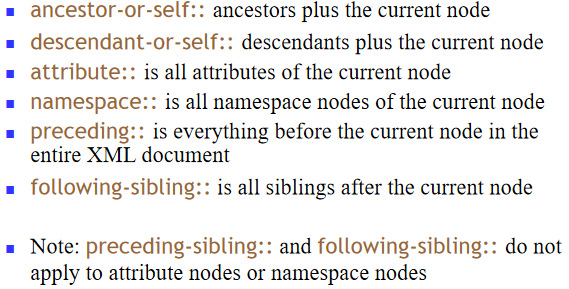
**/catalog/cd[@country=”UK”]** 选出所有属性country为UK的cd

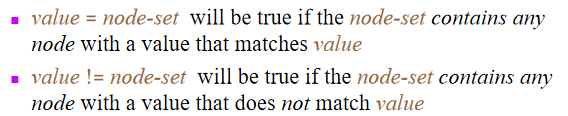
**/catalog/cd/@country** 选出所有country属性











1. **XSLT**
2. 属性中不能包含”<””>”,下图是不对的



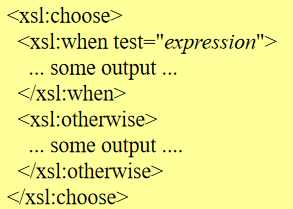
1. **<xsl：value-of** select=” ”/> home是单引号

**concat()**





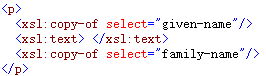
1. **<xsl:choose>**<xsl:when><xsl:otherwise>



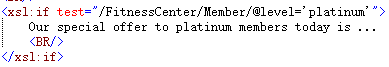
1. **<xsl:copy>**拷贝当前节点，拥有相同的名字、名字空间、类型**，**但是属性、子节点和其他后代不拷贝



1. **<xsl:for-each** select="catalog/cd">
2. <xsl:copy-of>



1. <xsl:if test=””>

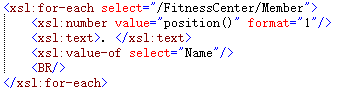


1. **A标签**，链接，id是ID类型，唯一的

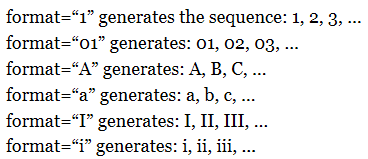




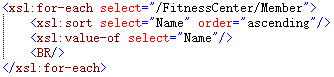
1. **<xsl:number>**  <xsl:text>



**position()从1开始，format的值如下图**



1. **<xsl:sort>**

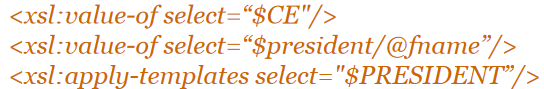


1. **<xsl:variable>**

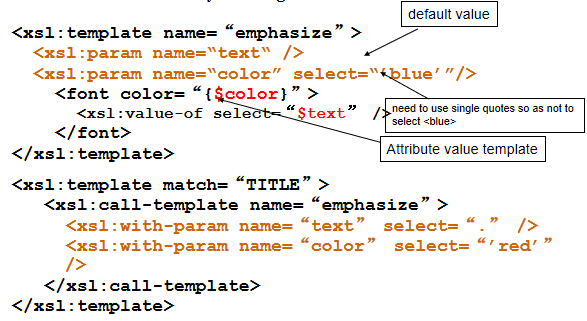
在template里定义的变量在template外不可见



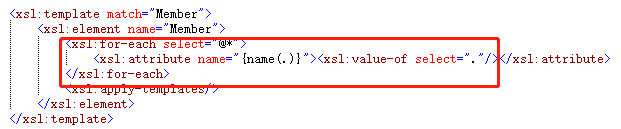
**引用使用$**



1. **<xsl:param>**

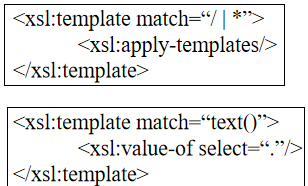


1. **添加属性**



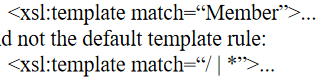
1. **Template默认规则**

**匹配文件或者任何元素**



匹配文本节点

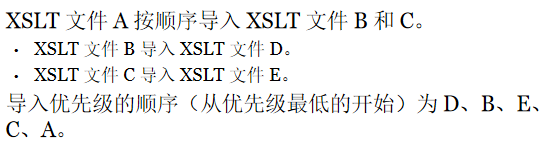
**使用更具体的规则**



1. **<xsl:include>**

include进来的xsl会替换这个元素

1. **<xsl:import>**



1. **DOM\_SAX**

**SAX（simple API for XML）DOM（document object model）**

1. DOM是将XML转化为**树**，**随机访问，结构化文档，在内存中建数据，**适合于交互型应用**，需要很大的CPU和内存，**适合于**树状结构**的数据**。**

SAX是**顺序访问**，非常快，**事件驱动**的解析，将xml翻译成事件流，在解析时调用事件处理器，**不在内存中建数据**，适合不许需要数据模型的或者数据模型是list或matrix的。

1. **DOM和SAX的区别**

**DOM会形成DOM树**

**SAX需要重写handler**

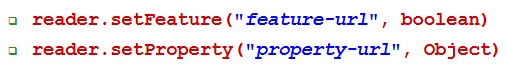
|  |  |  |
| --- | --- | --- |
|  | SAX | DOM |
| 使用标准 | ad-hoc | W3C |
| 是否将xml读入内存 | 否，不一次性读入 | 是，一次性读入 |
| 访问元素 | 顺序访问 | 随机访问 |
| 访问速度 | 快 | 慢 |
| 修改文件 | 不能 | 能 |
| 适合的文件类型 | 较大的文件 | 较小的文件 |

1. Fatal errors会时解析停止。

EntityResolver解析外部实体

DTDHandler

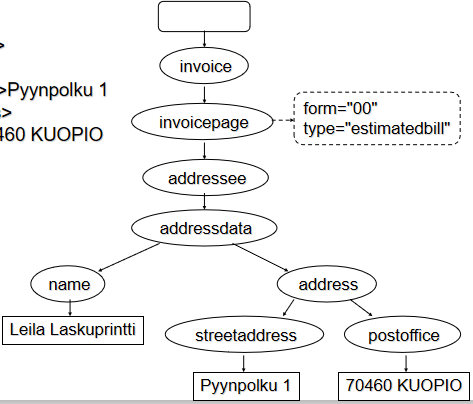
1. SAX可以设置特性和属性。



**Features**：namespaces、validation

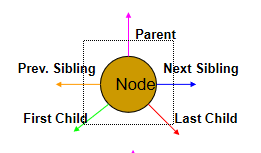
**Properties**：xml-string、lexical-handler

1. DefaultHandler实现了ContentHandler、DTDHandler、EntityResolver、ErrorHandler。
2. **DOM tree，经过parser后会生成tree**



1. **DOM的Node**

**集合为list或者map**



**public void** startElement(String namespaceURI, String localName, String qualifiedName, Attributes attributes)

**public void** characters(**char** ch[], **int** start, **int** length)

**public void** endElement(String namespaceURI, String localName, String qualifiedName)

1. **DOM 函数**

DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();

DocumentBuilder builder = factory.newDocumentBuilder();

Document document = builder.parse("src/lab1/ipo.xml");

Element getElementById(String id);

NodeList getElementsByTagName(String name);

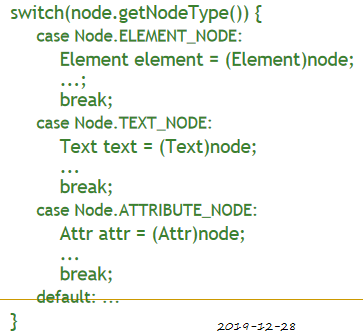
NodeList getElementsByTagNameNS(String nsURI, String local);

Element createElement(String name);

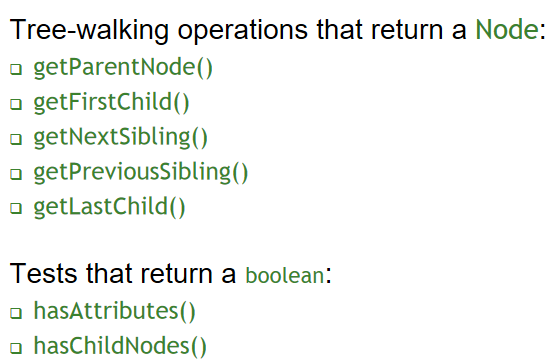
Attr createAttribute(String name);

Text createTextNode(String data);

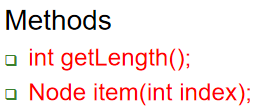




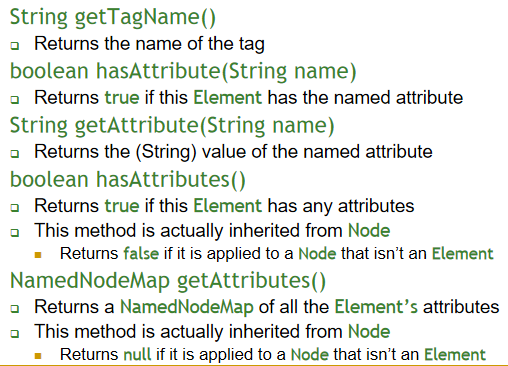
1. **DOM tree遍历的函数**



1. **NodeList函数**



1. **元素的函数**



1. **属性的函数**

