基于 Object-Z 的 XPath 形式化语义

杨红丽[1,2] 郝克刚[1] 韩俊刚[2]

(西北大学计算机科学系 西安710069)[1] (西安邮电学院计算机系 西安710061)[2]

摘 要 本文描述了 XPath 语言的形式化语义。一个统一的面向对象的语义视角用于建模所有 XPath 语言构造。语义的表示采用形式化规范语言 Object-Z 的符号系统。这种高度结构化的语义模型具有简洁、可组合性和可复用性的特点。

关键词 可标记语言(XML), XPath, XML Schema, Object-Z, 形式化语义

The Formal Semantics of XPath Based on Object-Z

YANG Hong-Li[1,2] HAO Ke-Gang[1] HAN Jun-Gang[2]

(Department of Computer, Northwest University, Xi'an 710069)

(Department of Computer, Xi'an Post and Telecommunication College, Xi'an 710061)

Abstract This paper describes the formal semantics of XPath language. A unifying Objected-Oriented semantic view has been used to model all language constructs of XPath. The presentation of semantics uses formal specification language Object-Z notation. This highly structured semantic model is concise, composable and extensible.

Keywords eXtensible Markup Language(XML), XPath, XML schema, Object-Z, Formal semantics

1 简介

可扩展标记语言(eXtensible Markup Language)XML[14]已成为万维网(Internet)交换信息的格式。随着其重要性的增加,出现了一系列相关的标准,如:XPath[2]语言用于从 XML 文档中选择元素; XSLT(eXtensible Stylesheet Language Transformations,可扩展样式表转换语言)[3]用于转换一个 XML 文档为另一个文档; XQuery 语言用于查询 XML 文档。所有这些标准都是由万维网协会(World Wide Web Consortium,简称 W3C)创建的。目前,XSLT、XPath 以及 XQuery 都是工作草稿,标准化问题将是主要的挑战。XPath 是 XML 核心技术之一,XSLT 利用 Xpath 表达式作为子语言;XQuery 定义为 XPath 的超集。标准化 XPath 规范将有助于其它规范的标准化。

程序语言的形式化模型支持语言的理解和标准化、程序推理和软件可靠性。目前,已经有一些关于 XML 相关语言的语义研究,如:文[7]中利用判断和推理规则描述路径表达式(path expressions)的语义。文[15]利用传统指称语义来描述 XSLT 中模式(patters)的语义。我们试图给 XPath 语言一个完整和高度结构化的语义描述,该语义基于程序语言语义的面向对象角度[4,5]。Object-Z[6,8]作为元语言(meta language)来描述 XPath 语言语义。所有 XPath 语言构造都建模为 Object-Z 对象类。这种高度结构化的语义模型具有简洁(concise)、可组合性(composable)和可复用性(reusable)的特点。

2 XML Schema 数据类型

XML Schema[13,10]现在已经被 W3C 推荐作为标准,用于描述和限制 XML 文档内容。XML Schema 已经广泛用作 XML 相关语言的类型系统。XML Schema 类型系统提供了可扩展的类型定义机制,用户可根据已有的原语类型和获取类型建立自己的新数据类型。

我们建模数据类型 Datatype 为原子类型、列表类型或联合类型。原子类型,其值不可再分;列表数据类型,其值包含有穷长度的(也可能为空)原子类型值序列;联合数据类型的值域是一个或多个数据类型值的并集。

$Datatype \triangleq atomicType \cup listType \cup unionType$

XML Schema 中总共有19种原子类型,如:字符串类型 String,布尔类型 Bool,名称类型 QName,整数类型 Integer 等。

$atomicType == String \mid QName \mid Bool \mid Integer \mid \cdots$

所有数据类型的公有属性建模为类 baseType,该类具有属性:值域 val;集合 constrain 表示值域应满足的限制条件。例如:类型 String 在 XML 中表示字符串,其值域是有穷长度的字符序列,字符串类型的限制有:串长度(length),最大最小串长度(minLength,maxLength),串模式(pattern)等。

consFacets == {length, minLength, maxLength, pattern, enumeration,
whiteSpace, maxInclusive, maxExclusive, minExclusive,
minInclusive, totalDigits, fractionDigits }

例如: maxExclusive 限制值域要有严格的上限; maxinclusive 限制值域要有包括界限的上限; *pattern* 限制数据值必须匹配特定的模式。

类型 *Value* 表示所有值的集合。

Value

函数 *has_type* 用于返回参数值的数据类型。

$has\_type : Value \rightarrow Datatype$

函数 *satisfy* 有两个参数：数据集和限制条件集，检查是否数据集满足每个限制条件。

$satisfy: \mathbb{P}\, Value \times consFacets \rightarrow \mathbb{B}$

baseType

$val: \mathbb{P}\, Value$
$cons: \mathbb{P}\, consFacets$

$\exists t: Datatype \bullet$
$\forall v: value \bullet v \in val \Rightarrow has\_type(v) = t$
$satisfy(val, cons)$

列表类型 *listType* 继承类 *baseType*，因此包含该类的所有属性。另外还包含属性：列表项序列 *items*，其项类型必须是原子类型或联合类型，其值是有穷长度的（也可能为空）项类型值序列。

listType
baseType

$items: seq\ atomicType \cup unionType$

$val = ran\ items$
$\exists n: Nat \bullet \#items < n$

联合类型 *unionType* 继承类 *baseType*，另外还包含属性：成员集合 *members*，其值域是成员类型值域的并集。

unionType
baseType

$members: \mathbb{P}\, DataType$

$val = \cup \{\forall m: members \bullet m.val\}$

3 XML 文档数据模型

本节建模 XML 文档的数据模型，该模型基于 XML 信息集[9]和 XML 文档数据模型[1]，XML 文档数据模型支持 XML Schema 类型系统，并已经用于 XPath[2]、XSLT[3]、XQuery[11]和其它与之索引的 XML 标准。

结点类型 *Node* 是最基本的类型。一个结点可能是下面七种类型结点之一：文档结点（document）、元素结点（element）、属性结点（attribute）、文本结点（text）、注释结点（comment）、处理指令结点（processing instruction）、名字空间结点（namespace）。下面定义结点类型 *nodeType* 为 Z 自由类型。

nodeType::=document|element|attribute
|text|comment|processing-instruction|namespace

一个结点对象包括：类型 type、名称 name、结点值 value、双亲 parent、孩子结点序列 children、属性结点序列 attribute、名字空间结点序列 namespace。此外还包括辅助属性：文档结点 doc、子孙结点序列 descend、祖先结点序列 ancestor、结点标识 id，其中：类型 ID 表示所有结点标识的集合，empty 表示空字符串。有一些规则用于限制上述属性：文档结点（根结点）、文本结点以及注释结点的名称是空字符串；只有文档结点和元素结点可以有孩子结点，只有元素结点、文本结点、注释结点和处理指令结点可以作为孩子结点；只有元素结点可以有自己的属性结点和名字空间结点。

类型 ID 表示所有结点标识的集合。

ID

Node

$type: nodeType$
$name: QName$
$value: String$
$parent: Node$
$children, attribute, namespace: seq\ Node$
$id: ID$
Δ
$doc: Node$
$descend, ancestor: seq\ Node$

$\forall x, y: Node \bullet$
$x.type \in \{docType, textType, commType\} \Rightarrow x.name = empty$
$y \in ran\ x.children \Rightarrow y.type \in \{emType, textType, commType, piType\}$
$y = x.parent \Rightarrow y.type \in \{elemType, docType\}$
$y \in ran\ x.attribute \Rightarrow y.type = attrType \wedge x.Type = elemType$
$y \in ran\ x.namespace \Rightarrow y.type = nmType \wedge x.Type = elemType$
$y = x.doc \Rightarrow y.Type = docType$
$y \in ran\ x.descend \Rightarrow y \in ran\ x.children \vee$
$\exists z: Node \bullet z \in ran\ x.descend \wedge y \in ran\ z.descend$
$y \in ran\ x.ancestor \Rightarrow y = x.parent \vee$
$\exists z: Node \bullet z \in ran\ x.ancestor \wedge y \in ran\ z.ancestor$

4 XPath 表达式

这部分描述 *XPath* 表达式的语义。*XPath* 表达式主要用于访问 XML 文档内容，表达式处理的对象是 XML 文档的抽象逻辑结构，该结构已在第3节数据模型中给出。*XPath* 中最重要的类型是表达式，一个表达式可以是基本表达式、路径表达式、序列表达式、算术表达式、比较表达式、逻辑表达式、条件表达式、量词表达式、序列类型表达式或者 for 表达式。

表达式类 Exp 定义为联合类（*class union*）。

Exp ≙primExp∪pathExp∪seqExp∪arithExp∪compExp∪logicExp
∪condExp∪quanExp∪seqtypeExp∪forExp

表达式的值是一个序列值（sequence），其中每一项是一个原子值或者结点。项类型 *itemType* 定义为原子类型或结点类型。类型 *seqType* 定义为项类型序列。

ItemType::=*atomicType*|*Node*
seqType==*seq itemType*

表达式的类型定义为标量类型 scalType。

scalType::=itemType|seqType

事实上，单个项和长度为一的序列没有区别，为了便于后面的描述，我们分开处理这两种情况。

表达式的标量值定义为 Z 自由类型值构造（*free-value construct*）。

scalVal::=atomicVal《atomicType》|nodeVal《Node》|seqVal《seqType》

函数 type_of 返回参数值的类型。

$type\_of: scalVal \rightarrow scalType$

$\forall a: atomicType; n: Node; s: seqType \bullet$
$type\_of(atomicVal(a)) = atomicType \wedge$
$type\_of(nodeVal(n)) = Node \wedge$
$type\_of(seqVal(s)) = seqType$

所有表达式的用属性建模为类 baseExp。表达式的上下文 context 是一个结点对象。

baseExp

$type: scalType$
Δ
$context: Node$
$val: scalVal$

$type\_of(val) = type$

outVal

$val!: scalVal$

$val! = val$

4.1 基本表达式

一个基本表达式可以是一个字(literal)、变量索引(variable reference)或者函数调用(function call)。类 *primExp* 定义为联合类(*class union*)。

$PrimExp \triangleq Literal \cup VaRef \cup callFunc$

一个字表示一个原子值。下面是类 Literal 的定义。

Literal

baseExp

content: atomicVal

val = content

一个变量是一个名称,该名称绑定到一个值。变量索引类 *VaRef* 需要定义变量存储位置类 *VarLoc*。一个变量存储位置对象包含属性:变量类型(type)和变量内容(cont),以及赋值操作 *assign*。一个变量索引对象包含一个变量名(nm)和一个索引(vl),类 VaRef 的定义继承类 *baseExp*。操作 *OutLoc* 和 *ReLoc* 的定义是为了便于后面函数调用中描述索引参数替换。

VarLoc

type: scalType
cont: scalVal

type_of(cont) = type

INIT

type = atomicType⇒
cont = atomicVal(0)
type = Node⇒
cont = nodeVal(nil)
type = seqType⇒cont =<>

Assign

△(cont)
val?: scalVal

cont' = val?

Varef

baseExp

nm: QName
vl: VarLoc

type = vl.type∧ val = vl.cont

ReLoc

△(vl)
vl?: VarLoc

vl' = vl?

OutLoc

vl!: VarLoc

vl! = vl

Assign ≙ vl.Assign

Module

vdecs: QName ⤖ VarLoc©
lvars: seq VaRef Ⓢ
△
vscope: QName ⤖ VarLoc
context: Node

 # {v:(VaRef ∩ scontain) • v.nm} = # (VaRef ∩ scontain) [pred1]
{vr: (VaRef ∩ scontain) • vr.nm ↦ vr.vl} ⊆ vscope [pred2]

其中,"0"是原子类型变量的初始值,"nil"是结点类型变量的初始值。

函数调用类 *funcCall* 需要定义两个类:函数定义类 *funcDef* 和函数索引类 *funcRef*。

首先,定义模块类 *Module*。该类包含属性:变量声明 *vdecs*、局部变量索引列表 *lvars*,以及辅助属性:变量作用范围 *vscope* 和上下文 *context*。

符号'©'表示每个变量的局部作用范围。符号'Ⓢ'表示直接包含并可共享的对象。

属性 *vdecs* 表示从变量名到变量值的一一映射。辅助属性 *vscope* 表示模块的整个作用范围。该变量的精确含义取决于继承该模块的构造,因此类 *Module* 中只给出签名(signature)。不变性 *pred*1说明:模块中没有两个不同的变量有相同的名字。不变性 *pred*2说明:变量的发生绑定在其作用范围内。

类 *funcDef* 的定义通过继承模块类 *Module*。此外该类还有属性:函数名 *nm*、形式索引参数列表 *frpl*,以及返回值类型 *return*。

funcDef

Module

nm: QName
frpl: seq VeRef
return: scalType
△
val: scalVal

has_type(val) = return
vscope = vdecs ∪ {vr: ran frpl • vr.nm ↦ vr.vl } [pred1]
 # {vr: ran(lvars ∪ frpl) • vr.nm } = # lvars + #frpl [pred2]

谓词 *pred*1描述:函数的整个作用范围是其局部变量及其参数声明的范围。谓词 *pred*2描述:任何两个不同的局部参数或变量有不同的标识名。

一个函数索引对象包含属性:函数名 *nm*、函数定义 *func*。

funcRef

nm: QName
func: funcDef

funcCall

baseExp

fr: funcRef
arpl: seq VeRef

val = fr.func.val
#arpl = # fr.func.frpl [pred1]
∀ j: dom arpl • ap(j).type = fr.func.frpl(j).type [pred2]

RefSubs ≙ ∧ i: dom arpl • arpl(i).OutLoc ‖ fr.func.frpl(i).ReLoc

函数调用类 *funcCall* 包含属性:函数索引 *fr*、实际索引参数列表 *arpl*。状态不变性 *pred*1描述:一个函数调用中实参列表的长度等于被调用函数中形参列表的长度。不变性 *pred*2描述:实参的类型匹配相应的形参类型。函数调用返回被索引函数的值。操作 *RefSubs* 表示索引参数位置(vl)的重新定位(relocation),即赋予新值。

4.2 路径(Path)表达式

路径表达式用于定位并访问 XML 文档树中的结点。一个路径表达式包括一个 *step* 序列,每步间用字符("/")分开。每步的结果是一个结点序列。路径表达式的值是路径中最后一步返回的结果。

一个案例 step 表达式包含:一个轴(axis),表示当前结点和被选择结点之间的关系;结点测试(nodetest),进一步描述被选择的结点。谓词(predicates)用于过滤结点测试选择的结点。

路径表达式有13个轴用于表示结点之间的关系。例如 *attr* 检查是否被选择结点是当前结点的属性结点。类型 *Axis* 定义为 Z 自由类型。

$Axis ::= child \mid desc \mid attr \mid self \mid desOrself \mid followSib \mid follow \mid nmsp \mid parent \mid ance \mid preSib \mid preced \mid ancOrself$

一个结点测试可以是:结点名字测试或者结点类型测试。例如 *pi* 检查是否一个被选择的结点是处理指令结点。类型 *Test* 定义为 Z 自由类型。

$Test ::= text \mid comm \mid pi \mid nd \mid qnm\langle\langle QName\rangle\rangle$

类型 *Pred* 定义为表达式类型 *Exp*。

$Pred == Exp$

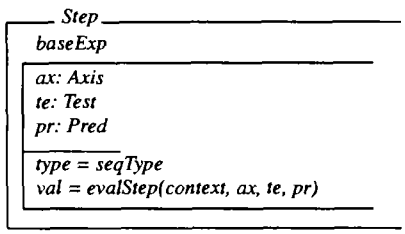
函数 *evalStep* 计算一个 *step* 表达式。

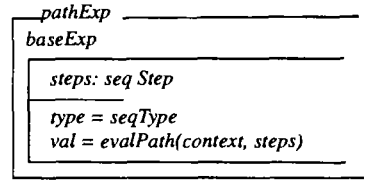
|*evalStep*:*Node*×*Axis*×*Test*×*Pred*→*seq Node*

函数 *evalPath* 基于当前上下文结点，计算一个路径表达式。

|*evalPath*:*Node*×*seq Step*→*seq Node*

下面是类 *Step* 和 *pathExp* 的定义。





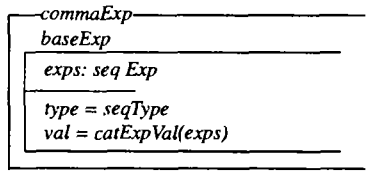
路径表达式(path)中每步(step)的计算都是根据确定的上下文结点(context)。当计算一个路径表达式时，每一步选择的结果结点作为下一步的上下文结点，如果一步有多个上下文结点，依次对每个上下文结点计算该步表达式，其结果结点序列合并作为该步的结果值。

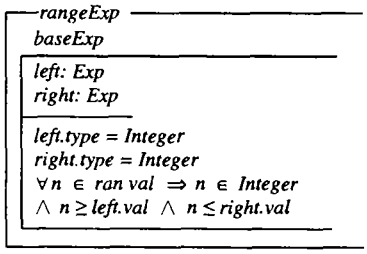
4.3 序列(Sequence)表达式

XPath 支持序列的构造和合并操作。一个序列表达式可以是：逗号表达式(comma expression)、范围表达式(range expression)、结合表达式(combine expression)。类 *seqExp* 定义为联合类(class union)。

$seqExp \triangleq commaExp \cup rangeExp \cup combExp$

逗号表达式类 *commaExp* 包含属性：表达式序列 *exps*。





一个逗号表达式的计算是对每个操作数表达式计算值，然后连接多个结果序列值为单个序列。

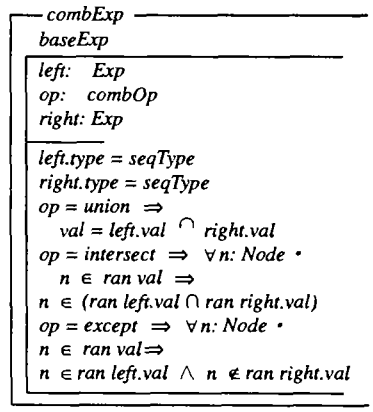
类 *rangeExp* 包含属性：左操作数 *left* 和右操作数 *right*。结果序列包含两个操作数及其之间的所有整数。

一个结合(combine)表达式可以是：表达式的并(union)、交(intersect)和除(except)。类 combExp 包含两个结点序列操作数 *left* 和 *right*，以及结合操作符 *op*。结合操作符类型 *combop* 定义如下：

combOp::=union|intersect|except

函数 catExpVal 用于连接表达式序列的值，形成单个序列值。

|catExpVal:*seq* Exp→seqType



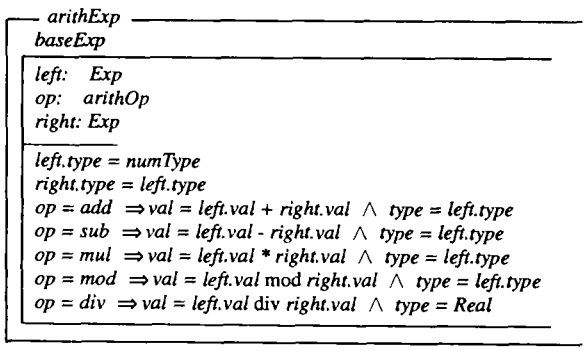
4.4 算术(Arithmetic)表达式

XPath 支持常用的算术操作如+，−，*，div 和 mod。算术操作定义在数字类型 *numType* 上。算术操作类型 *arithOp* 定义为 Z 自由类型。

numType::=*Integer*|*Real*|…

arithOp::=*add*|*sub*|*mul*|*mod*|*div*

类 *arithExp* 包含两个操作数 *left* 和 *right*，以及算术操作符 *op*。



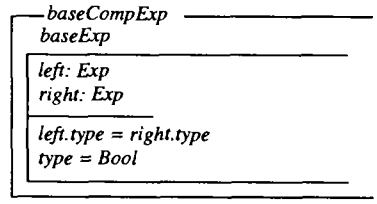
4.5 比较(Comparison)表达式

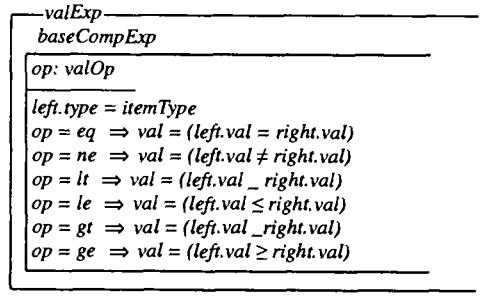
比较表达式容许两个值进行比较。XPath 支持四种类型的比较表达式：值(value)比较，一般(general)比较，结点比较和序(order)比较。比较表达式的结果是布尔值。

类 *compExp* 定义为联合类(class union)。

$compExp \triangleq valExp \cup genExp \cup nodeExp \cup orderExp$

所有比较表达式的公有属性建模为类 *baseCompExp*。其它类可通过继承该类来定义。





值(value)比较操作符定义为Z自由类型。

genExp

baseCompExp

op: genOp

left.type = seqType
op = eq'⇒ val = (∃x, y: itemType •
x ∈ ran left.val ∧ y ∈ ran right.val ∧ left.val = right.val)
op = ne'⇒ val = (∃x, y: itemType •
x ∈ ran left.val ∧ y ∈ ran right.val ∧ left.val ≠ right.val)
op = lt'⇒ val = (∃x, y: itemType •
x ∈ ran left.val ∧ y ∈ ran right.val ∧ left.val < right.val)
op =le'⇒ val = (∃x, y: itemType •
x ∈ ran left.val ∧ y ∈ ran right.val ∧ left.val≤ right.val)
op =gt'⇒ val = (∃x, y: itemType •
x ∈ ran left.val ∧ y ∈ ran right.val ∧ left.val> right.val)
op =ge'⇒ val = (∃x, y: itemType •
x ∈ ran left.val ∧ y ∈ ran right.val ∧ left.val≥ right.val)

valOp::=*eq*|*ne*|*lt*|*le*|*gt*|*ge*

一般比较操作符定义为Z自由类型。

genOp::=*eq'*|*ne'*|*lt'*|*le'*|*gt'*|*ge'*

结点比较表达式检查是否两个操作数结点有相同的结点标识。序(order)比较表达式用操作符'≪'和'≫'来比较两个结点的位置。例如,'≪'操作数返回真值如果在文档序(documentorder)中第一个操作数结点在第二个操作数结点的前面。

ndOp::=*is*|*isnot*
orderOp::=*less*|*great*

函数docOrder定义结点在文档中的序(*document order*)。

|docOrder:Node→numval

4.6 逻辑(Logic)表达式

nodeExp

baseCompExp

op: ndOp

left.type = Node
op = is⇒ val =
(left.val.id = right.val.id)
op = isnot⇒ val =
(left.val.id ≠ right.val.id)

orderExp

baseCompExp

op: orderOp

left.type = Node
op = less⇒ val =
(docOrder(left.val) < docOrder(right.val))
op = great⇒ val =
(docOrder(left.val) > docOrder(right.val))

逻辑表达式可以是:与(and)表达式、或(or)表达式。逻辑操作符类型*logicOp*定义为:

logicOp::=and|or

逻辑表达式类logicExp定义如下:

logicExp

baseExp

left: Exp
op: logicOp
right: Exp

left.type = Bool ∧ right.type = Bool
type = Bool
op = and⇒ val = (left.val ∧ right.val)
op = or ⇒ val = (left.val ∨ right.val)

4.7 条件(Conditional)表达式

一个条件表达式包含三个子表达式:测试表达式test、then表达式、else表达式。条件表达式的结果值定义如下:如果测试表达式的值为真,则结果值为表达式*then*的值,否则,结果值为表达式*else*的值。类*condExp*定义如下。

condExp

baseExp

test: Exp
then: Exp
else: Exp

test.type = Bool ∧ type = Bool
test.val ⇒ type = then.type ∧ val = then.val
¬test.val ⇒ type = else.type ∧val = else.val

4.8 量词(Quantified)表达式

XPath支持存在(existential)和全称(universal)量词。

量词操作类型quanOp定义如下:

quanOp::=*some*|*all*

一个量词表达式包含属性:操作符*op*、变量名*nm*、表达式*ep*、测试表达式*ts*。此外还包括辅助属性:变量索引*ref*。

如果对一些变量绑定值,测试表达式的值为真,则存在量词表达式返回真值;如果对所有变量绑定值,测试表达式的值为真,则全称量词表达式返回真值。

4.9 序列类型(Sequence Type)的表达式

序列类型表达式包括:实例*instanceOf*表达式、cast表达式、treat表达式。类*seqtypeExp*定义为联合类(class union)。

$seqtypeExp \triangleq instExp \cup castExp \cup treatExp$

quanExp

baseExp

op: quanOp
nm: String
ep: Exp
ts: Exp
△
ref: VaRef

type = Bool ∧ ts.type = Bool ∧ ep.type
= seqType∧ref.nm = nm
op = all ⇒ val = ∀ i: 1..#ep.val • ref.val
= ep(i).val ∧ ts.val = true
op = some⇒ val =∃ i: 1 ..#ep.val • ref.val
= ep(i).val ∧ ts.val =true

一个实例*instanceOf*表达式包含属性:表达式*ep*、类型*tp*。如果表达式*ep*的类型匹配类型*tp*,则该表达式返回真,否则返回假。

一个*cast*表达式包含属性:输入表达式*ep*,目标原子类型*tp*。该表达式基于现有的值创建给定类型的新值。

函数*castable*定义是否一个输入类型的值可以投射到给定的目标类型。

|*castable*:*scalType*×*scalType*→*Bool*

一个*treat*表达式包含属性:表达式*ep*、类型*tp*。不同于*cast*表达式,*treat*表达式不改变其操作数的动态类型和值。

函数*subType*定义是否第一个参数类型是第二个参数类型的子类型。

|*subtype*:*scalType* ×*scalType*→*Bool*

4.10 For表达式

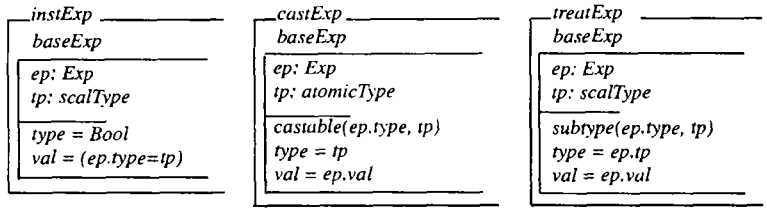
XPath提供*for*表达式来表示循环。一个单变量*for*表达式包含属性:范围变量名*nm*、输入序列(sequence)表达式*input*、返回表达式*return*。此外还包含辅助属性:变量索引

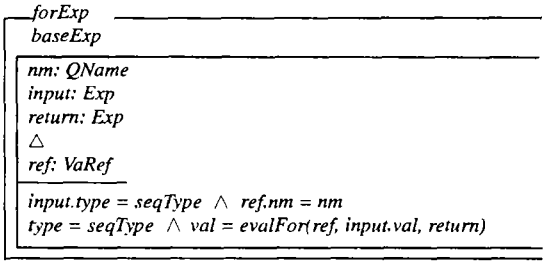
ref。

for 表达式的计算：对输入序列 *input* 中每项，计算返回表达式 *return*；对每次计算所得的结果序列进行连接，形成的单个序列作为表达式的结果值。

函数 *evalFor* 用于计算单变量 *for* 表达式的值。

$$evalFor : VaRef \times seqType \times Exp \rightarrow seqType$$





结论 本文利用形式化描述语言 Object-Z 的符号系统建模 XPath 语言构造。这种面向对象的语义表示不仅具有简洁性，而且便于扩展和复用。采用 Object-Z 的一个好处是：它提供了一个 XML 环境[12]，该环境可自动扩展 Object-Z 类的继承，并生成 UML 例图。利用该工具将有助于语义模型的可视化和易理解性。

我们已经看到 XML 家族语言之间的相互依赖性，将来的一个研究方向是：建模 XML 家族语言的通用语义构造，并复用这些语义构造描述 XML 家族语言的语义。这种方法将有助于说明各个规范之间的相互一致性和协调性。

致谢 感谢联合国大学软件技术研究所对该研究的资助。

参 考 文 献

1 XQuery 1.0 and XPath2.0 Data Model. 2002. http://www.w3.org/TR/querydatamodel/.

2 Boag S, et al. Anders Berglund. XML Path Language (XPath) 2.0, 2002. http://www.w3.org/TR/xpath20/.

3 Clark J. XSL Transformations (XSLT) Version 2.0. 2002. http://www.w3.org/TR/xslt20/.

4 Dong J S. Formal Object Modelling Techniques and Denotational Semantics Studies: [PhD thesis]. University of Queensland, 1995

5 Dong J S, Duke R, Rose G. An object-roiented approach to the semantics of programming languages. Australian Computer Science Communications, 1994, 16

6 Duke R, Rose G. Formal Object Oriented Specification Using Object-Z. Macmillan, 2000

7 Draper D, et al. XQuery 1.0 and XPath 2.0 Formal Semantics, 2002. http://www.w3.org/TR/query-semantics/.

8 Hinchey M. The Object-Z Specification Language. Kluwer Academic Publishers, 1999

9 Cowan R T J. XML Information Set, 2001. http://www.w3.org/TR/xmlinfoset/.

10 Biron A M P V. XML Schema Part 2: Datatypes, 2001. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.

11 Fernandez M F, et al. Scott Boag, Don Chamberlin. XQuery 1.0: An XML Query Language, 15 Nov. 2002. http://www.w3.org/TR/2002/WD-wquery-20021115/.

12 Sun J, Dong J S, Liu J, Wang H. Obiect-Z Web Environment and Porjections to UML. In: WWW-10: 10th Intl. World Wide Web Conf. refereed papers track, ACM Press, May 2001. 725～734

13 Thompson H S, Beech D, Maloney M, Mendelsohn N. XML schema Part 1: Structures, 2001. http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.

14 Sperberg-McQueen C M, Bray T, Paoli J. Extensible Markup Language (XML) 1.0 (Second Editon), 2001. http://www.w3.org/TR/REC-xml/.

15 Wadler P. A formal semantics of patterns in XSLT, MIT Press, JUNE 2001

(上接第159页)

5 Curbera F, et al. Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI IEEE Internet Computing, 2002, 6(2): 86～93

6 Vinoski S. Where is is middleware. IEEE Internet Computing, 2002, 6(2): 83～85

7 Clark D. Next-generation web services. IEEE Internet Computing, 2002, 6(2): 12～14

8 Sahai A, Machiraju V, Ouyang J, Wurster K. Message tracking in SOAP-based Web services. Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP, 2002. 33～47

9 Preuner G, Schrefl M. Integration of Web services into workflows through a multi-level schema architecture. Advanced Issues of E-Commerce and Web-Based Information Systems, 2002. (WECWIS 2002). In: Proc. Fourth IEEE Intl. Workshop on, 2002. 51～60

10 Curbera F, et al. Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing, 2002, 6(2): 86～93

11 Foster I, Kesselman C, Nick J M, Tuecke S. The Physiology of the Grid. Computer, 2002, 35(6)

12 Foster I, Kesselman C, Tuecke S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications

13 Johnston W, et al. 关于计算网格的论述. http://www.itg.lbl.gov/～johnston/Grids/DOE-Science-Grid-PImeeting-Jan, 2002. 1.ppt. http://www.itg.lbl.gov/～johnston/Grids/DOEScienceGrid.and.ComputationalScience.ppt

14 Foster I. Internet Computing and the Emerging Grid. NatureWeb Matters, Dec. 2000. http://www.nature.com/nature/webmatters/grid/grid.html

15 Foster I, Kesselman C, Nick J M, Tuecke S. The Physiology of the Grid. Computer, 2002, 35(6)

16 Loyall JL, Gossett JM, Gill CD, et al. Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications. In: Proc. of the 21st IEEE Intl. Conf. on Distributed Computing Systems (ICDCS-21), Phoenix, Arizona, 2001

17 Narain S, Vaidyanathan R, Moyer S, et al. Middle-ware For Building Adaptive Systems via Configuration. ACM Optimization of Middleware and Distributed Systems (OM 2001) Workshop, Snowbird, Utah, June, 2001

18 Schantz R E, Schmidt D C. Research Advances in Middleware for Distributed Systems: State of the Art, 2002. www.ifip.tu-graz.ac.at/TC6/events/ WCC/WCC2002/papers/Schmidt.pdf

19 Blair G S, Costa F, Coulson G, Duran H, et al. The Design of a Resource-Aware Reflective Middleware Architecture. In: Proc. of the 2nd Intl. Conf. on Meta-Level Architectures and Reflection, St.-Malo, France, Springer-Verlag, LNCS, 1999. 1616