PIMON 2019

XSLT в PI

XSLT basics: concepts and processors

Origin: As a part of XSL standard designed for creating presentation of XML data (data -> XSLT+XPath -> XSL-FO -> presentation)

XSLT main features:

- XML language
- Declarative
- Functional (no side effects, except for calling external code)
- Extendable

Stages of processing:

- Parsing
- Transformation
- Serialization

Major XSLT Processors:

Processor	*Language support*	*XSLT versions support*	*Open source*	*Comments*
libxslt	C	1.0	yes	Unix world & browser standard
xalan	Java, C++	1.0	yes	JDK default XSLT processor
saxon	Java, JS, .NET	1.0, 2.0, 3.0	partially	Most feature-rich processor

XSLT basics: Imperative vs. Declarative

Imperative style:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <ResultRoot>
      <xsl:for-each select="/SourceRoot/SourceNode1">
        <ResultNode1>
          <xsl:value-of select="translate(.,'A','B')"/>
        </ResultNode1>
      </xsl:for-each>
      <ResultNode2>
        <xsl:value-of select="substring(/SourceRoot/SourceNode2,1,3)"/>
      </ResultNode2>
      <xsl:for-each select="/SourceRoot/SourceNode3">
        <ResultNode3>
          <xsl:value-of select=". + 1"/>
        </ResultNode3>
      </xsl:for-each>
    </ResultRoot>
  </xsl:template>
</xsl:stylesheet>
```

Declarative style:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="SourceRoot">
    <ResultRoot>
     <xsl:apply-templates/>
    </ResultRoot>
  </xsl:template>
  <xsl:template match="SourceNode1">
    <ResultNode1>
      <xsl:value-of select="translate(.,'A','B')"/>
    </ResultNode1>
  </xsl:template>
  <xsl:template match="SourceNode2">
    <ResultNode2>
      <xsl:value-of select="substring(.,1,3)"/>
    </ResultNode2>
  </xsl:template>
  <xsl:template match="SourceNode3">
    <ResultNode3>
      <xsl:value-of select=". + 1"/>
    </ResultNode3>
  </xsl:template>
</xsl:stylesheet>
```

XSLT History

Version	*Recommendation year*	*XPath version*	*Editor*	*Usage*	*Major processors*
1.0	1999	1.0	James Clarke	wide	xalan, saxon, libxslt
2.0	2007	2.0	Michael Kay	limited	saxon
3.0	2017	3.0, 3.1	Michael Kay	very limited	saxon

Some XSLT 2.0 new features:

- Numerous XPath 2.0 functions (math, date, string etc.)
- Regular expressions (xsl:analyze-string)
- Grouping (xsl:for-each-group)
- No RTF (node-set extension no longer required)
- Type validation by schema (xsl:import-schema)

Some XSLT 3.0 new features:

- Streaming (xsl:mode, xsl:source-document) – complex rules, will be reviewed
- Reducing recursion (xsl:iterate)
- Error generation and recovery (xsl:assert, xsl:try/xsl:catch)
- Dynamic XPath (xsl:evaluate, w/o extension)
- JSON support

XSLT mappings in PI: Basics

XSLT advantages:

- Easy to make data structure transformations
- High potential for reuse
- External debugging
- Open standard

XSLT drawbacks:

- Performance
- Clumsy handling of binary data (only via Java enhancements)
- Slow language development and adoption

Universal mapping example (JDBC sender case):

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:sp="http://sap.com/xi/XI/SplitAndMerge">

  <xsl:template match="/">

    <sp:Messages>

      <sp:Message1>

        <xsl:apply-templates/>

      </sp:Message1>

    </sp:Messages>

  </xsl:template>

  <xsl:template match="row">

    <ESP>

      <xsl:apply-templates/>

    </ESP>

  </xsl:template>

  <xsl:template match="row/*">

    <xsl:if test="string-length(.) > 0">

      <xsl:copy-of select="."/>

    </xsl:if>

  </xsl:template>

</xsl:stylesheet>
```

XSLT in PI: Java enhancements & access to mapping objects

Access to some runtime parameters: via global mapping parameter

Access to objects, defined as StreamTransformationConstants: via “inputparam”

Access to everything: via “TransformationInput”

Example (access to custom mapping parameter):

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:ti="com.sap.aii.mapping.api.TransformationInput"
    xmlns:ip="com.sap.aii.mapping.api.InputParameters"
    xmlns:xsltc="http://xml.apache.org/xalan/xsltc"
    exclude-result-prefixes="ti ip xsltc" version="1.0">
    <xsl:param name="TransformationInput"/>
    <xsl:template match="/">
        <root>
            <paramValue>
                <xsl:value-of
select="ip:getString(ti:getInputParameters(xsltc:cast('com.sap.aii.mapping.api.TransformationInput',$Tra
nsformationInput)),'extParam')"/>
            </paramValue>
        </root>
    </xsl:template>
</xsl:stylesheet>
```

XSLT in PI: Xalan extensions

Xalan internals: interpretive processor (ex-LotusXSL) and compiling processor (XSLTC).

XSLTC is the default processor in JDK and PI (since 7.1).

If Xalan is used as a library, interpretive processor is the default.

EXSLT module support:

Module	*Useful functions*	*Supported by interpretive*	*Supported by XSLTC*
common	node-set	Yes	Yes
math	max, min, random	Yes	Yes
sets	intersection, distinct, difference	Yes	Yes
dates and times	date, date-time, format-date, parse-date, difference	Yes	Yes
dynamic	evaluate	Yes	No
strings	concat, split	Yes	Yes
regular expressions	test, replace	No	No

Interpretive processor also has SQL extension.

XSLT in PI: External processors

Summary:

Processor	*XSLT version*	*Open source*	*Works in PI (7.5 SP09)*
Xalan Interpretive	1.0	Yes	Yes
Saxon 6.5.5	1.0	Yes	Yes
SaxonB	2.0	Yes	No
SaxonHE 9.5	2.0 (basic 3.0)	Yes	Yes (via Note)
SaxonHE 9.9	2.0 (basic 3.0)	Yes	No
Saxon EE 9.9	3.0	No	Yes

Local XSLT 3.0 Streaming Tests (-Xmx256m):

Size	*Rows*	*Test*	*Result (Streaming)*	*Time (Streaming)*	*Result (No streaming)*	*Time (No streaming)*
33Mb	1 million	count	ok	2.5s	ok	2s
339Mb	10 million	count	ok	9s	oom	-
1739Mb	50 million	count	ok	41s	oom	-
33Mb	1 million	copy (50% rows)	ok	4.5s	ok	3s
339Mb	10 million	copy (50% rows)	ok	25s	oom	-
1739Mb	50 million	copy (50% rows)	ok	113s	oom	-

PI mappings: XSLT vs. Java vs. MM

Mapping type selection:

Message size/Transformation complexity	*Small*	*Medium*	*Large*
Low	XSLT	XSLT	MM
Medium	XSLT/Java	Java/XSLT	Java
High	Java/XSLT	Java	Java

Mapping lifecycle:

- Mapping logic tends to become more complicated over time
- Detailed documentation for medium/high complexity mappings is a must
- Code reuse via Import Archives in Underlying SWCV is recommended