XJSLT

ERIK HETZNER

NOVEMBER 7, 2024

Created: 2024-10-16 Wed 20:39

TABLE OF CONTENTS

- Brief introduction
- What's missing?
- What's next?
- Compiling
- Optimization
- Some interesting aspects of XSLT compilation
- Thank you

BRIEF INTRODUCTION

- XJSLT doesn’t stand for anything, it’s just XSLT + JS
- XJSLT can compile XSLT to JavaScript that can be executed **directly** in the browser or in NodeJS

WHY?

- I thought it would be possible, and interesting to see how far it could go
- The world needs another XSLT 2 implementation

WHAT'S MISSING?

- XJSLT is incomplete, somewhat slow, and only passes about half of the XSLT test suite.
- However, it is substantially more complete than other open source XSLT 2 implementations besides Saxon

- `xsl:number`
- `xsl:attribute-set`
- `xsl:analyze-string`
- Good error reporting
- Lots of edge cases (whitespace is especially tricky!)

WHAT'S NEXT?

- Speedups
- Fixing edge cases
- Whatever others can help with - I am pretty busy!
- https://github.com/egh/xjslt / https://gitlab.com/egh/xjslt

COMPILING

- It's not **faster**, but it's reasonably easy to do in JavaScript, and it means we can create simple redistributable files

EXAMPLE

- After compiling an XSLT stylesheet to `transform.js`, we can use this in the browser like this:

```
<head>
  <script src="transform.js" />
  <script>
    const xmlString = "…"
    const input = new DOMParser()
      .parseFromString(xmlString, "application/xml");
    const output = document.getElementById('main');
    transform(input,
      {outputDocument: document, outputNode: output});
  </script>
</head>
<body>
  <div id="main"/>
</body>
```

nodejs

```
import transform from "./transform";
import * as slimdom from "slimdom";

export default {
  async fetch(request, env, ctx): Promise<Response> {
    …
    const dom = slimdom.parseXmlDocument(
      await response.text())
    );
    const transformed = transform(dom).get("#default").document;
    return new Response(serializer.serializeToString(transformed)
      { headers: new Headers({ "Content-Type": "text/html" })
    });
  },
}
```

CONTROL FLOW

1. `transform` starts the process on the root node
2. Determine template to use for node
3. Pass to compiled template
4. Compiled template calls XJSLT helper functions (e.g. `literalElement`)
5. Compiled template returns flow, either by `return` from a function or by calling `apply-templates` or similar

```
<xsl:template match="header">
  <h1><xsl:apply-templates></h1>
<xsl:template>
```

```
{
    "match": "header",
    "modes": ["#default"],
    "allowedParams": [],
    "apply": context => {
      xjslt.literalElement(context, {
        "name": "h1",
        "attributes": [],
      }, context => {
        xjslt.applyTemplates(context, {
          "select": "child::node()",
          "mode": "#default",
        }, context => {});
      });
    },
}
```

OPTIMIZATION

- XJSLT uses a naive method that checks each possible template in turn looking for matching ones

SOME SPEEDUPS

- XJSLT caches xpath results
- Fast failure/fast success which bypass full XPath processing
  - e.g., the xpath `text()` only matches a text node, and `foo` will only match an element with the name `foo`

FUTURE STEPS

- Move to a decision tree, as Saxon uses, to determine which template to use for “simple” patterns (e.g. `foo`).

SOME INTERESTING ASPECTS OF XSLT COMPILATION

```
<xsl:if test="node()">
  <xsl:call-template name="my-template">
    <xsl:with-param name="prefix">hello</xsl:with-param>
    <xsl:with-param name="node" select="."/>
  </xsl:call-template>
</xsl:if>
```

WHY CAN'T WE JUST?

```
if node():
    my-template(prefix="hello", node=this)
```

BUT...

- XSLT is already in Abstract Syntax Tree format, ready to be transformed using itself

```
<xsl:template match="xsl:with-param[@name='prefix']">
  <xsl:element name="with-param"
               namespace="http://www.w3.org/1999/XSL/Transform">
    <xsl:attribute name="name">prefix</xsl:attribute>
    <xsl:text>hello world</xsl:text>
  </xsl:element>
</xsl:template>
```

```
<xsl:if test="node()">
  <xsl:call-template name="my-template">
    <xsl:with-param name="prefix">hello world</xsl:with-param>
    <xsl:with-param name="node" select="."/>
  </xsl:call-template>
</xsl:if>
```

XSLT → XSLT → XSLT

AN EXAMPLE

- For instance, the first rule of stripping whitespace from an XSLT document is to remove all comments and processing instructions. This can easily be accomplished with XSLT:

```
<xsl:template match="comment() | processing-instruction()"/>
```

- Another example is static error analysis.

```
<xsl:template match="xsl:variable | xsl:with-param">
  <xsl:if test="@select and (node() or text()) ">
    <xsl:message terminate="yes">XTSE0620: Variable or parameter
    <xsl:value-of select="@name"/> has both @select and
    children.</xsl:message>
  </xsl:if>
  <xsl:next-match/>
</xsl:template>
```

- This technique is currently also used to handle `include`, `import`, and more
- I would love to implement even more functionality using this

THANK YOU

- To the authors of fontoxpath, Stef Busking, Martin Middel and others, the TypeScript XPath 3.0 implementation without which this would not be possible.
- To the authors of the XSLT test suite, Michael Kay and Debbie Lockett. Without this extensive test suite I would never have figured out what I was doing right and what I was doing wrong.