DOM / Template Parts API
TPAC 2023 - 13 Sep 2023 Meeting Day 3

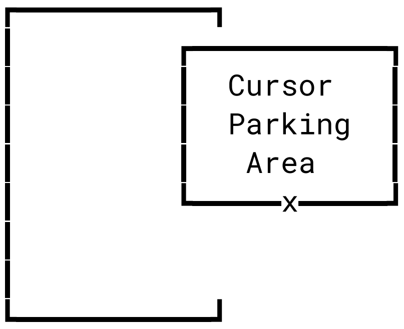
Participants

(Please add your name and affiliation!)

- Peter Burns (Google Web Components Team)
- Mason Freed (Google Chrome DOM Team)
- Tom Wilkinson (Google Web Frameworks Team)
- Tom Nguyen (Google Web Frameworks Team)
- Keith Cirkel (GitHub)
- Olli Pettay (Mozilla)
- Justin Fagnani (Google)
- Jesse Jurman (Orthogonal Networks)
- Rob Eisenberg (Blue Spire)
- Ryosuke Niwa (Apple)
- Henri Sivonen (Mozilla)
- Jocelyn Tran (Google Chrome Accessibility Team)
- Jared White (Whitefusion)
- Rob Flack (Google)
- Chris Lorenzo (Comcast)
- Sean Feng (Mozilla)
- Bruce Anderson

🅿️ Cursor parking

1.



Links

W3C Event: https://www.w3.org/events/meetings/27a383b4-c8ae-4246-8641-1e757488d4d6/
Issue: https://github.com/w3c/tpac2023-breakouts/issues/13
Explainer: https://github.com/WICG/webcomponents/blob/gh-pages/proposals/DOM-Parts.md

Minutes

Ryosuke: We can start by going through what this API is about. We started with this as a way for custom elements to stamp out its own template content. Typically that we have today is that each element has a template and that frameworks stamp out this template, find all the parts that need to be replaced, and then render the HTML. Seems better and potentially faster for the browser to handle this.

Ryosuke: As a part of template instantiation proposal, which is the ability for to declare a template with basically mustache syntax and pass data into it. Feedback was this is too complex, need to extract core piece of this infrastructure and standardize first. DOM Parts is the first part, allows creating a part of HTML and write to it. The dynamic places in a template (e.g. the mustache parts) would be a DOM Part and they can be written to to update the HTML. There are two kinds of part, AttributePart which is responsible for updating an attribute, and ChildNodePart which is responsible for writing nodes into child node position in HTML.

Ryosuke: The proposal is that we'd use curly braces, and the browser would automatically find these tokens in the text and create DOM parts, which JS could then manipulate to update the DOM. Any questions?

[..]: I have one question. I am [name] and I'm from the CSS working group. So a child node part represents a range, and it can write to a range? Why use a ChildNodePart over a range?

Ryosuke: A range has two boundary points. It starts at one point and ends at another. In between there's no guarantee that the start and end boundary points are anchored in the same place, so updating a range becomes very complicated because you have these partial trees. ChildNodePart simplifies this greatly.

Tom N: There's also a NodePart that lets JS just get a reference to a node.

Ryosuke: But why do we need the part API for this?

Tom N: Because we want to be able to get a reference to this element.

Justin: This is for a clonable reference to the element

Anne: How is this difference than an id?

Tom N: Maybe you don't want to have an id element here?

Ryosuke: You could just have a reference to an h1 element?

Tom N: But when you clone a template, you lose the reference to the h1 in the clone

Justin: It's basically an attribute part without the attribute

Tom N: If you have a bag of attributes e.g. you can't use an attribute part, because it's not part of any one attribute.

Justin F: Lit uses this e.g. to add behaviors to elements. Previously to adding this feature people used made up attribute names to get this behavior.

Ryosuke: how do you figure out which element needs the reference?

Tom N: Put the double curly brace inside the element start tag but not as part of any attribute

Ryosuke: Ok, interesting, worth considering. We could bike shed the syntax, but that's interesting to consider. Any other feedback about this proposal? I'm interested in what Olli has to say

Olli: Thinking about the old template instantiation work, and the performance cost, and whether this will help at all for performance. I think at this point only the cloning part will help, and there's a lot of moving between JS and browser bindings. Could batch writes perhaps.

Mason Freed: We've been prototyping in Chrome, and we're very curious about exactly that question. So far it depends a lot on exactly how you implement it, but we have found that it will save times, particularly if you have lots of parts and you do a lot of cloning, you save all of the time you would have spent walking the tree after you clone. So you can get a performance benefit.

Olli: But only the cloning part is improved

Mason: With the declarative syntax you also save time on the initial walk, because you don't have to walk the template on the initial template.

Tom N: Template systems will spend ~30% of the time just walking the tree.

Noam: I'm Noam from Google Chrome. Currently a lot of code in frameworks is concerned with virtual dom, and all of this code could potentially be greatly simplified.

Olli: With earlier proposal, it would only help with cloning time.

Justin: I was going to second Noam's comment. In addition to the runtime perf of prepare and clone. With the addition of the template syntax, this could greatly reduce the code size and complexity of template systems, and could potentially also improve interoperability between templates.

Olli: Yes, could help where templates could be standardized.

Tom N: We've been trying to prove this out by taking different template systems and try to make them use DOM parts internally.

Justin: The thing we look forward to in Lit which basically has a polyfill of DOM parts, this reduces the most complicated part of our template system, the template prep phase down into a single line. This could likewise be a target for other template systems too, including JSX, Angular…

Tom N: Yeah, Angular uses comments as anchors for what would be ChildNodeParts

Olli: A question to Mason, have you been implementing the declarative bits too? Has it been showing up in parsing performance?

Mason: I don't have the answer for that yet, the implementation landed just yesterday so I don't have parse performance info yet. I've implemented the NodePart and ChildNodePart syntax. Would be good to get agreement on the syntax. The syntax we think needs to be an opt in syntax. So you could add an attribute like "parseparts" (needs bikeshed), and the parser will see this and opt into parsing DOM Parts

R: We did discuss opt in attribute last face to face, and when opting in it would parse DOM parts.

Mason: I was a bit afraid of the implementation but it turns out it wasn't too bad.

R: Would it be possible to provide numbers to the performance gains?

Mason: It turns out, you have to try this with a real framework on a real site, depends a lot on the specifics of your benchmark.

Olli: Do we have any interest or discussion with framework devs?

Tom N: With Angular, it's a process. There's interest. [something about Ivy?]

Justin: Solid, and [..] showed interest.

Tom W: The most common response that we get is that they want it to do N+1 things.

Justin: To be perfectly honest and clear, for people who have VDOM systems, this doesn't naturally fit the VDOM approach where you make a copy of the tree and diff against the tree. But frameworks that are investing heavily in signals where you bind a signal to a specific part of the HTML tree does have a natural fit for this API.

R: What do people think about the curly brace syntax? Do people hate it?

[...]: It's familiar, which I like. The more we maintain similarity with existing stuff in userland the better.

Keith: [...] from GitHub. Template systems that use this existing syntax will not be interoperable. Like we'd need to do escaping to work with Mustache

Justin: It's opt in. Though we might also want to have an opt out when nesting.

Keith: When mixing say Jinja with DOM Parts there'd have to be escaping.

Justin: We've seen this with Polymer too, where both have {} and there's escaping needed. There's another issue, we don't want to overlap with syntax that overlaps with JS syntax ${

Bruce Anderson: What about double, triple braces? Is there a thought behind that here?

Justin: It's possible that the expression syntax inside might have object literals, or put one of these expressions inside a script or a style.

Henri: From Mozilla. Where does this parsing happen? If you create an element inside a template, innerHTML etc

Mason: We implemented this inside the HTML parser

Henri: If JS goes and constructs them, if you create them with create text node, or innerHTML, does that create parts?

Mason: It doesn't, similar to how declarative shadow dom behaves.

Keith: I remember there was a discussion around this with innerHTML, and there's concern with security.

Justin: We create templates with innerHTML in <template>s, we absolutely need to support this.

Mason: There was a discussion with this around declarative shadow dom, and it was a backcompat issue.

Ryosuke: A dom part doesn't do anything on its own, it's just an object, so it doesn't create a security concern.

Mason: agreed, it should be safe

Justin: Could be an issue with other APIs that build on top, like declarative custom elements

Olli: If you have some random element that's not part of any document fragment, and it uses this syntax, what happens?

Justin: Tom W has been doing an updated version with a new version. Ryosuke's version was focused on making a template and using it. Tom has been considering SSR, so having these in the main document. When these frameworks hydrate, this way they wouldn't need to walk the tree to find parts, they could call an API on the document to get what's called a PartRoot and the nested parts inside.

Mason: I can talk a bit to Olli's question. I think this is a question for the group. These are performance sensitive APIs, and it matters how much bookkeeping you want to do. In Chrome's prototype, if you have a detached tree that contains a part, and you insert it into a tree, the new parts are then tracked in that tree.

R: Any objection to double curly braces?

Tom N: My understanding for child node part, for default values, how would curly braces work? Would you have like, an opening brace and ending brace?

Justin: Mustache and similar have this. For an empty node, with no default content, the existing sSyntax is fine. For ones with default content, {{#}} and {{/}} could be used:

```
<h1 parseparts>
  {{#}} <span> default </span> {{/}}
</h1>
```

Bruce: It makes sense to me that for CSS we need double braces, but we don't need that for attributes. Could we get away with just one brace in attributes?

Ryosuke: One syntax used everywhere would be less confusing

Mason: Nice for by able to visually see, double curlies means DOM parts

Ryosuke: We also have the imperative API. You have a Part base class that has a value and commit. Each child node part knows the parent and the previous and next sibling. Attribute part has the element, the attribute name, and the namespace.

Mason: in the chromium prototype we also have a part root, to handle nesting. So the constructor takes a parent part. So the document contains a part root, and then a child node part is also a part root.

Tom W: This may be in our PR to the explainer. Anyone can come take a look and review it.

Tom W: I'd love if we could meet on a somewhat regular cadence. Maybe a quarterly face to face. Internally in Google we're meeting regularly, but would love to meet with implementors and libraries.

Ryosuke: DOM Parts seems like it could be valuable to have as a separate meeting

Mason: Having feedback from implementors has been super helpful

Tom N: Yeah, it's been helpful for us to give that feedback

Bruce: is there a way for us to try these features?

Mason: in chromium it's available in [Chromium Canary?] experiment web platform features. It's very new, so please provide feedback if you find issues.

Justin: How would the syntax work with expressions / identifier / metadata? Some separator for the expression, and then metadata.

[Referring to Metadata section here Web Components Proposals - DOM-Parts-Declarative-Template.md]

Mason: The question is, like, what happens to "email" in:

```
  <a href="mailto:{{# email}}john@doe.org{{/}}">
```

Tom N: Could use space separation

Justin: That would preclude expressions from having spaces

Ryosuke: We want to eventually get to an expression processor in the browser itself, so we want to leave space for that in our syntax today

Henri: So how does it work in Chromium?

Mason: Currently the tree builder will convert the child node syntax into empty comments that it can use to anchor the child node part

Henri: does this introduce a new node type, or will we use comments for this indefinitely?

Mason: yeah, comments seems like it works fine and is kinda nice

Henri: why not just use comment nodes for the syntax?

Mason: it's very easy to understand the curly brace syntax, it gets very messy otherwise.

Justin: curly braces work everywhere, including in element position and attribute position

Mason: we haven’t implemented anything for metadata because we’re not sure what the semantics will be

Ryosuke: five minutes left, let’s plan out our next meeting. Does once a quarter sound good to everyone?

Tom N, Peter B, Mason F: sounds good

Ryosuke: What time zones? There are some in Europe. Anyone in Asia Pacific? None? Then we propose early morning west coast US time

Mason: open an issue for scheduling?

[Issue Created for Quarterly Meeting #1027]

[...]: Would be good to have clear code examples

[...]: And performance numbers, and code size reduction numbers

Mason: google has two action items: update the explainer with latest design, and bring performance numbers (runtime and code size)