Alternative to customized builtins (custom elements with is=) - TPAC 2023 - 13 Sep 2023 Meeting Day 3

Present:

- Ryosuke Niwa (Apple)
- Brian kardell (Igalia)
- Anne van Kesteren (Apple)
- Olli Pettay (Mozilla)
- Henri Sivonen (Mozilla)
- Keith Cirkel (GitHub)
- Peter Burns (Google)
- Jared White (Whitefusion)
- Cole Peters (Begin)
- Simon MacDonald (Begin)
- Peter Van der Beken (Mozilla)
- Chris Lorenzo (Comcast)
- Rob Eisenberg (Blue Spire)
- Jesse Jurman (Orthogonal Networks)
- Xiaoqian Wu (W3C)
- Nikki Massaro Kauffman (Red Hat)
- Christian Liebel (Thinktecture)
- Edgar Chen (Mozilla)
- Thomas Nguyen (Google)
- Joey Arhar (Google)
- Kagami Rosylight (Mozilla)
- Justin Fagnani (Google)
-

Links

- Alternative to customized builtins (custom elements with is=) · Issue #44 · w3c/tpac2023-breakouts (github.com)
- Customized built-in elements · Issue #97 · WebKit/standards-positions (github.com)
- lume/element-behaviors: An entity-component system for HTML elements. (github.com)

- Alternative to customized built-in - Element custom enhancements · Issue #1000 · WICG/webcomponents (github.com)
- lume/custom-attributes: Define custom attributes that provide mixins for HTML elements (github.com)

Minutes

Rniwa: We're looking to discuss alternatives to custom built-ins - there are a few possible options in the issue. Should we go over them?

(yes)

Given the historical objections from WebKit - this session's goal is to come up with an agreement to solve the same use cases but which is acceptable to WebKit.

Olli: Is there a link to the objection/summary?

Annevk: https://github.com/WebKit/standards-positions/issues/97 (in particular my and rniwa's comments)

Rniwa: The main issue I think, is the fact that we support underlying use cases, but the proposed solution that we objected to - there are limitations in terms of what the browser can do with regard to the accessibility tree. The whole reason custom elements are customized are because they don't want just the functionality (or DOM) that exists.. Input is an often cited use case, you cannot even touch the shadow dom there

Annevk: More bluntly, it's a hacky solution that doesn't truly allow for subclassing.

Justin: Can we phrase that in a way that the alternatives - we can see why it's clear they don't have the same problem.

Annevk: Because they don't actually exactly pretend to be other elements. These alternative solutions allow for composition. E.g., custom global attributes can probably reasonably compose with existing elements.

Keith Cirkel: That the customized elements share a surface with autonomous custom elements it is hard to discern when reading a class, if it is running into those issues. Personally I think there are limited use cases that justify the need specifically for customized built ins vs some other solution like attributes.

Justin: It sounds to me like the overall theme is that since they dont behave like regular custom elements it shouldn't have the custom api shape

Keith C: They should have many of those apis because it's too many caveats to internalize, whereas if we had a new, quite separate and limited version - it is much easier to teach, read, retain and conceptualize. As a thought experiment - if you use an autonomous CE you can create & customize the shadow dom, if you customise an <input> you cannot. Who can tell me off-hand which built-ins can have a shadowroot?

Rniwa: For customized built-ins you don't have access to elementinternals either.

Keith: I haven't internalized a list of which all of those apply to.

Justin: I would like to clarify the constraints. I can't recall if we have some canonical list of use cases? I think progressive enhancement is listed in a few use cases

Rniwa: Commonly cited

- Enhancing existing form controls
- Inheriting a11y from existing builtins
- Progressive enhancement from the built in element
- Supporting special HTML parsing rules where you cannot put a custom elements
- polyfills

Brian: I'll add that when Google tried to do Toast, part of that was intended to launch a discussion about how we answer polyfilling legitimate elements - can we improve the story. While not the primary purpose we should see if there's alignment here.

Rniwa: I think the first one form associated custom elements provides a good solution already. There might be certain things that can improved (making it submittable, or just a button not just a submit button). I think those are just improvements we can make to elementinternals

Annevk: Those would be fairly straightforward.

Justin: I've heard from clients that they don't really want to recreate and specify all of that.

Keith: What are the use cases for subclassing a button?

Justin: Clients telling me it's styling, adding new capabilities such as icon buttons or leading/trailing visuals, dropdown visuals. Being able to take an adopted stylesheet and applying it to buttons would be useful.

Rniwa: That wouldn't work with buttons, right? Adopted stylesheets wouldn't work

Justin: It doesn't work, but the use case is what I am talking about

Annevk: If they wanted to use adopted stylesheets, currently they would have to use JS already, they could use element internals

Annevk: If you claim to be a button, I feel like the aria stuff should mostly be there.

Justin: Our clients are dealing with a lot of complexity and reimplementing things where they want it to be additive.

Annevk: It reminds me of declarative custom elements.

Rniwa: I don't really understand the use case. You want it to be a button, what else?

Justin: It maybe adds some affordance for an icon, brings it's own styles, whatever from their design system

Annevk: It sounds like it is not a good case for customized builtins either since it can't do those things either, right?

Justin: Our customers think it solves

Annevk: How?

(some confusion in the room about customized builtins feedback vs related feedback)

Rniwa: This is like the second case I was talking about - inheriting from existing elements. Let's talk about solutions

1. Element Behaviors: instead of is="" you can do has="" and we do customElementBehaviors.define(...) or something and you get a lifecycle and callbacks similar to custom elements - but it doesn't pretend to subclass the element. Another benefit is that you can have multiple “has” values. You can have the markup or different elements. That's one advantage of this approach. You can easily create a global that can add to any element.

   Brian: I'm glad we are finally having the 'mixins' conversation

2. Custom Enhancements - merging a custom property with a custom attribute..
3. Custom Attributes - like the other two but instead of having a single attribute, you can specify something as a custom attribute like "bg-color" so that it can have N attributes.

Rniwa: All of these are quite similar in that they create a separation between the element they are enhancing and the element itself, and you can have N of them.

Justin: The one thing I don't see in these 3 is they talk about adding .. With the mixin idea you could take the JavaScript mixin pattern and apply it to the prototype of the instance and swizzle the prototype. Another way to do it would be to take a bag of properties and patch it onto the instance. There are important things to clarify - how collisions are handled. In all of these when they talk about adding - they have to think about the drawbacks

Rniwa: One drawback is that if you are really trying to add something for one _specific_ element, then this is kind of the wrong answer. The use cases that come up seem to be a better fit generally with these types of ideas though.

Bruce Anderson: ???

Lea: Why not element types?

Bruce Anderson: What do you mean?

Lea: I am talking about class names not tag names, tag names can be ambiguous

Bruce Anderson: I guess in a scoped registry…

Lea: Yes, scoped registries, also namespaces, you don't want to deal with that…

Keith: Class names are also ambiguous, HTMLElement encompasses many tags. What about autonomous elements that have not yet been registered too? HTMLUnknownElement. There's timing issues there.

Anne: I think we are having a lot of topics at the same time

(some discussion about ?scoped registries?)

Anne: My assumption was that those would be global

Ollie:

Bruce: I agree you should be able to target a specific element.

Justin: What do we mean by class

Anne: Class would be HTMLInputElement, name is input, namespace is HTML namespace. For anything in this space you want to do targeting by name and namespace – headings for example are not specific enough… If you have a name and a namespace you can get the class

Lea: That is a good point but are there use cases where you want to support them in both ways

Annevk: Or targeting multiple elements

Lea: it is not uncommon today for people to not do the define in their libraries and consumers define.

Keith: In a connectedCallback for a customElement you could add a whenDefined callback. Would you ever want to have a custom attribute for an HTMLUnknownElement. Any of these needs to have some kind of feature that scopes when the definition is applied based on the tag

Annevk: the has attribute is very clear - you don't need this

Keith: if you take a form and I want to add ajax, I only want my lifecycle callbacks to only run on forms – I can look and see if it is a form and not do it. You just don't attach listeners

Annevk: so the form has a has atr with has="ajax"

Keith: Lets say I also had the same ajax attribute for buttons and they do something different. It's not great to name them the same thing, but…

Lea: they can come from entirely different libraries and just have a common name

Keith: more than any of the reasons of that - if the API doesn't allow you to subset based on the tag, then you need to do that check in every lifecycle callback or something

Justin: You probably do anyway - if this goes down the road of scoping, they are in charge of registration - they probably have to add that logic anyway to be defensive. It seems like annevk was thinking that each element would leave it to the person using it?

Lea: At least for custom attributes there are two designs: it could be defined either on the class or on the registration point, as an option in CustomAttributeRegistry#define(). It sounds like the latter might solve those use cases better?

Keith: Like - there's a way to declare in a custom element definition - a static getter that declares which options can be used to define…prevent shadow roots.

Rniwa: There are multiple things being discussed here – the attribute name, and other is some kind of filter based on the element type… we probably need to support both.

Annevk: there was also name conflicts

Rniwa: It is just like custom elements. In that case the solution is scoping. It's not a problem this needs to address

Keith: It's more around the sorts of conditions you'd want and that you'd want them

Lea: It's probably not great if it is an arbitrary function because then UAs won't be able to optimize, it needs to be a static list

Keith: I think it should be on the class rather than the define call. Static

Rniwa: I think we want to do just like custom elements, not dynamically update

Lea: What if it is both? You can set it and the define can override it

Keith: The author of the customization should win, not the person applying it in their markup. Or the latter should only be able to scope *further*.

Lea: I agree

Justin: There are multiple questions here, I agree. the initial question, how do we associate a behavior with an element? how do we filter what behaviors go on what elements? how do we patch those elements? In addition to all of the things we said, are their timing questions? What happens when you define a behavior that matches existing DOM, what happens if you upgrade a custom element that has a behavior attached?… creating a list of those for each would help us decide.

Rniwa: We are about out of time, but I agree that coming up with a list of issues we want to ask is good.

Olli: So we probably do need scoped versions of this like custom elements

Rniwa: It's an interesting question about whether this is a new registry or is it something we can build into the existing registry. If you are scoping one, you seem to need to scope the other. I am initially inclined to think that the custom element registry might be the place to do this.

Keith: The timing issues can be handwaved away for now. The sticking point is how they are instantiated- the author

Justin: I think how javascript is applied is the bigger question, are you patching the prototype like how does it 'mix in'?

Keith: I would imagine custom attributes (which I prefer) are basically custom elements but for attribute nodes - so you would actually have custom attribute nodes.

Annevk: I'm not sure that works, if you add them on the prototype we'd have to add them on each instance. I feel like these proposals are like "that's a js problem and you can solve it in user land" and what justin is saying is that that's not really adequate. I'm not sure where decorators are

Justin: I know where they are but I don't think they apply - they are executed during definition and cannot be applied post-hoc.

Bruce: In our proposals, the elements can be attached directly without having to pass through attributes.

Lea: It sounds like there are 2 orthogonal issues here - one is if it is one new attribute (e.g. has="") with multiple values vs arbitrary hyphenated names. The other is whether custom attributes are sufficient or whether we also need the ability to add methods, decorators etc. What if we did *both*? A custom attribute registry, AND enhancements that can *include* custom attributes, as well as other things. If your enhancement *only* involves custom attributes, then you just register custom attributes, if it's something more complex, you define an enhancement plus custom attributes.

… Personally I prefer custom attributes (and proposed them in 2017) than a single attribute with identifiers. WC are all about making elements that feel first-class, and custom attributes are more on par with that. Also, custom attributes are more on par with paving the cowpaths, since developers have already been defining hyphenated attributes with a common prefix (e.g. VueJS, older Angular, AlpineJS, htmx etc).

… Also, custom attributes have so much potential beyond augmenting built-ins: e.g. if they automate attribute-property reflection (which is currently really hard), they could even be useful in defining a WCs *own* attributes.