Signals community call
May 28, 2024

Attendees:

- Daniel Ehrenberg (littledan)
- Alistair Lynn (prophile)
- Chengzhong Wu
- Chris de Almeida (ctcpip)
- Matthieu Riegler

Agenda:

- Intros
- Topics:
    - Alistair's list of grumbling
        - Call went through a number of open PRs and issues and closed them
    - Alternative watcher proposals: making an "is dirty" flag available vs the current getPending API. Call didn't have strong opinions on the subject.
    - Ways to help:
        - Adding more tests - tests now live in the polyfill repository, but are quite incomplete. Every existing framework with signals has a signals test suite, so it would be great to work on enhancing our current tests.
        - Unwatch is still very slow for reasons unknown, but should be relatively straightforward to fix in the polyfill.
        - Matthieu was interested in working on PRs for the polyfill – Daniel pointed them towards the preact signals test suite as a good source of test coverage to port, as well as the Vue signals test suite. Vue integrates the reactive objects into their main signals API, where we have that as a separate package (`signal-utils`). A third is Solid's signals, which is mixed with a concept of ownership. With these parts factored out we could port their test suites over.

Signals community call
May 14, 2024

Attendees:

- Shay Lewis (shaylew)
- Alistair Lynn (prophile)
- Preston (NullVoxPopuli)
- Devin Weaver (@sukima / suki@tritarget.org)
- Meghan Denny (@nektro)

- David Neil
- Dolan Miu
- Jonathan Kuperman (jkup)
- Mathieu Hofman
- Steve Orvell

Agenda:

- Intros
- Topics:
    - Alistair: generalizing notify phase
    - Preston: watcher performance
    - Steve: effects (contentious!), polyfill status
    - Shay: alternative watcher API discussion (topic missed due to no time)
- notify phase:
    - Alistair: doing certain things from callbacks can break internal invariants of the system (both the polyfill and the specified algorithm). Should the notify flag – or other "being inside a callback" situations – freeze the graph to preserve invariants?
    - Steve: what actually *can* you do in notify callbacks? why call notify in this tricky internal state of the graph where dirtyness status isn't completely propagated? would be great if there was a better justification for the model here, of the "asynchronous only" nature of effects.
    - Shay: sort of two things here: proposal/polyfill state (could be made more robust by using two passes) and library internal state stored in signals, which might have invariants they want to maintain
    - Alistair: as to the watched/unwatched callbacks – getting the semantics right when doing this recursively (e.g. watch from a `watched` callback) is hard to get right, [general agreement] about restricting this with the equivalent of `notifying = false` until we know that modifying the graph from a watched callback is something we would need
- watcher performance:
    - Preston: https://github.com/tc39/proposal-signals/issues/215 – clear that there's some sort of problem with watcher performance, was wondering if anyone had ideas about this situation
    - David: do we expect the polyfill to be performant right now? it seems to have a lot of wrappers
        - Shay: right now? not exactly, but eventually it'd be nice to have it more state of the art, so that we can catch any pieces of the spec that turn out to not be able to make fast
    - Preston: it'd be great if we had some in-polyfill benchmark testing so we could track change over time / benchmark PRs
    - David: got permission to publish Lucid's internal benchmarking graph https://github.com/DavidANeil/signal-benchmarks , which could be adapted to benchmark the spec

    - Steve: (sort of a hot take) more general concern with the model of signals: “how big is my component tree” is easier (or at least more familiar) to think about than “how big is my signal graph”. how to make it easy for users to reason about and understand performance?
    - Alistair: Excel has good developer tooling for this (for people who might be using Excel for things they shouldn’t) – could be a source of inspiration
- effects
    - Steve: README should spell out exactly why “no synchronous effects” is an intentional part of the design. existing systems (even some that wish they weren’t) are synchronous or have synchronous escape hatches. so prohibiting it isn’t necessarily wrong but should have a really clear, solid justification/explanation.
    - Steve: other big design decision about effects is that it’s not built in. the system as it stands is really beginner friendly, aside from not having effects and watchers being scary. there’s a lot of potential here for this to be like promises – simple, understandable, really core – but it’s hard to get there without effects
    - David: starting to think it might be a mistake to not have synchronous reactions. `eagerlyRecalculateToPreventDirtyPropagation` etc need it for expressiveness/correctness
    - Steve: the “set trap” escape hatch to allow synchronous reactions makes things not interoperable – this really defeats the purpose of the spec if people end up having to resort to that
- the polyfill
    - Steve: is there a release cadence for the polyfill? people are trying it out, there’s been some PRs, what’s the plan here?
    - Preston: release automation is set up, waiting on Rob for access to get the ball rolling on releases
        - Update: Rob has granted NullVoxPopuli NPM access and automation is currently being set up. Should be ready soon.

Signals community call
April 30, 2024

Attendees:

- Shay Lewis (shaylew)
- Daniel Ehrenberg (littledan)
- Dominic Gannaway (trueadm)
- Alistair Lynn (prophile)
- Chris de Almeida (ctcpip)

- Justin Fagnani
- Erik Yeomans
- Ryan Carniato
- Kristen
- David Neil
- Steve Orvell

Agenda:
- Intros
- What do each of the newer participants think of the proposal?
    - Alastair: Seems good! I'm happy to see this as a primitive coming in. I've been working on this in Python, and it's tricky to nail down semantics. Good to see something fast, etc.
    - Erik: Seems great, took me a bit to understand nuances/algorithms. It'd be great to pull this core stuff in as a standard. Then we can pull this into our internal framework, so it's locked in, good to go, and we can build on top of it.
    - Chris: Meta-thought: it'd be great if we could parallelize this effort with the observables effort. We've had some issues getting feedback incorporated into that effort.
        - Dan: The relationship is misleading; the mechanics are opposite so the APIs shouldn't be the same.
        - Ryan: They can work together, but they have different purposes. In observables, there was a lot of conversation about composability of operators. Signals are the opposite–you end up wrapping/hiding the pieces, and they are the magic behind the scene. With signals, you don't need operators, because autotracking reverses the scope.
        - Dan: And we have the interop story: use watched/unwatched to subscribe/unsubscribe from signals so you can convert an observable to a signal
        - Chris: How does that work, and how do you create a signal from an observable like in Solid?
        - Ryan: It's normal to want this, when going from events to signals, but things become tricky when going the other way, in terms of preserving timing, since you expect synchronous updates, but the proposed "cold" signals don't update at the right time reliably.
        - Shay: If you think of observables as streams, you can consume that. But with signals being lazy, the main thing you respect is the dependency order, so scheduling relies on no "secret" dependency edges, where something leaves the graph, turns into an observable, and then you get pushed back into the graph on the other side–this defeats topological sorting, so you may have to run things more than once. So if you do try to create a stream of changes as the output from signals, there's no canonical time of when to pull them, and you need to decide which things come out in which order.

- Alastair: In FRP, the literature separates “behaviors” or “reactives” – a signal, a function of time – vs an “event stream”. Observables are very close to the latter and signals are close to the former. Probably there’s good stuff in the literature. They’re dual to each other. If you want to make signals respond to observables independent of things arriving at the same time, you can make an API to get a window to see a set of events received over a period of time. There are ways of doing it, but this may or may not be useful.
- Justin: I’ve seen applications where people build graphs of observables, approaching signals. These graphs often have path length dependencies, which signals don’t have. When observables fork and join, you can get timing differences based on just how many links there are. This is a misuse of observables which signals avoid. You need a window to absorb the difference between the shortest and longest path. You could track dependencies but this doesn’t tend to be done.
- Shay: The most fully baked solution in that direction is Jane Street’s incremental, which calculates the height. This isn’t compatible with autotracking because height needs to be discoverable before running code. You may be able to build this out of just observables, if you want to fix the arrival order problems.
- Dan: So, back to the original question, I don’t see much interop stuff that we should actually *do*.
- Chris: I’m worried that, when people try to interop between signal and observable [...]
- Alastair: If both of these systems make it into the web platform, people will build interoperability between them. Given that it’s subtle, many people will do it badly if it’s not spelled out. If interoperability is declared out of scope, that almost guarantees it’s done badly. So I’d want to include one or a few operators to interop well.
- Dan: I can imagine one operator (observable => signal) but are there others?
- Alastair: In my Python system, there are three operators. I’ll file an issue to explain more.
- Shay: This is about watchers–how can I observe the signal graph? People seem not totally satisfied with watchers, partly because people wish things are simpler than they are, and also because the API isn’t so clear. There’s some sort of relationship between the stream timing and the watcher notification timing, people may want a “faster” way to react. You can poll to get an observable out of a signal. A solution to how you observe the signals as a discrete stream of events comes down to timing, since we already know how to describe values.
- Justin: I wonder if we’re more likely to see integration attempts starting with APIs coming as observables
- Shay: Luckily that side is much easier

    - Justin: Tricky cases when handling multiple observables?
    - Alastair: We have a signal.computed which emits an observable event, then we have an observer which calls .set. (In my Python system, this isn't possible by construction, so no analogy there; that system is more about data engineering.)
    - Dominic: One of the things difficult about mixing both systems together is the lossiness of signals. You only see things at the times you pull them. In event streams, that is important, since the coherence is important to the UI. It can be glitchy if an event is dropped! So you have to be careful to not mix/match them, and instead treat them as entirely different. You can read observables into the signal graph, but not bring signals to observables coherently.
    - Kristen: Agreed, we shouldn't encourage this. There may be backwards compatibility issues, trying to transition systems, existing data layers written with observables. Users will just need to craft their APIs well to make sure that they're not creating spaghetti, and I'm not sure if we can do anything. This is like any async action, e.g., sending messages to the worker, and getting information back (except this might be sync).
    - Alastair: I agree that Signals > Observables is dangerous, but I'm also sure that people will build it whether or not we think they should. From a harm reduction point of view it might be better to figure out a "least bad" option here than leave it to whatever people end up doing IMO
    - Shay: About synchronous reactions: if you do something within a tracking context, in a computed body, and that causes other things to react synchronously, you'd better do that within an untrack, or it will make you rerun later! A gotcha for any integration. Also a thing for sync effects.
- David Neil: I'm from Lucid Software, and we have signals. My main concern is the observability of the performance difference between live/unlive signals. In our system, we don't use watchers. We do explicitly dispose of computeds, but we never need the notification part of the watcher. We're experimenting with WeakRefs which might be the best of both worlds.
    - Dan: It'd be great if you could share your benchmarks on this. We've been concerned about this performance.
    - Kristen: The Ember team previously made a very highly optimized version of this. (namely the @track annotation and the way that relates to the rendering layer, which avoids two-way communication–signals do not keep track of their consumers). Could you use watchers though?
    - David: I'd prefer to not have to dispose of nodes, it's a huge developer painpoint. I'd be concerned about a performance cliff where we forget to watch some subgraph.
    - Kristen: Yes, both paths have to be reasonably performant.
    - Shay: You probably don't ever get to a world where you don't have to dispose of *behaviors*, but you should ideally not have to deal with disposing *data*. So, watchers let you avoid needing to dispose

intermediate computeds. This is what unwatch gives you over computed ownership. Also I’m wondering if we should lay out performance guarantees more explicitly–if you try to write this down, you get things like “you traverse each edge at most twice/per write-read pair”. People don’t know how bad/not bad each of these is. I was more spooked by the thing that Ember uses as. I think the polyfill has the right guarantees but might be needlessly slow implementation-wise.

- Kristen: I’ll take a look at the polyfill and see if we can make it more performant.
- Ember’s signal impl == `@tracked`