React / Angular Meeting

# Attendees

Christopher Chedeau (react)

Igor Minar (angular)

Lee Byron (react)

Miško Hevery (angular)

Sebastian Markbage (react)

Yegor Jbanov (angular)

Victor Savkin (angular)

Matias Niemela (angular)

John Lindquist (egghead.io)

# Collaboration points

* Best practices for NPM as package manager
  + AI: Sebastian to send model for ensuring smooth upgrades
* Performance measurement tooling and techniques
  + AI [done]: Brad to share Benchpress techniques/arcana (see our [Benchpress docs on Github](https://github.com/angular/angular/blob/master/modules/benchpress/docs/index.md))
* Using “other renderers” like iOS and Web Worke r support to prove model for updates to Web platform
  + AI: Brad to nominate Angular team members to collaborate with Christopher
* Immutable data structures: identify nice patterns in Angular apps as support for inclusion as part of TC39 spec
  + AI: Victor to add specific support for [immutable-js](https://github.com/facebook/immutable-js) to Angular 2 change detection
* Types in standards
  + AI: Angular team explores if we can better support developers using [Flow](http://flowtype.org/) types and toolchain (though maybe perfectly usable as-is)

# Discussion Notes

* CLI
  + Igor: There are a lot of JS tools, but none of them do what we want. We’re trying to coordinate them. We want to provide a good default experience out of the box, so we’re building a CLI to:
    - scaffold
    - skeleton files
    - set up build
    - set up testing environment
    - possibly even deployment
  + Also, enable editor vendors to build on top of it to understand layout and choices in an Angular project.  
      
    We’re working with the Ember CLI team who are extracting reusable bits. Working with Joe from broccoli and reusing those bits. Current changing the Angular build from gulp to broccoli. Working with the NPM team on package management and resolution. The package managers that exist today aren’t good, but NPM is the closest of all of them.
  + Lee: We have two things. One is the jsappserver built with react native in mind. It’s a runtime similar to webpack that tries to join together linting, type checking, transpilation, bunding. Provides a static resource server. Great for ReactNative as you can just reload from the server to see your new code running. It’s prescriptive about every tool for how the integrate. We’re anti-any one tool. We don’t require Flow, linter, etc. We’ve pretty much landed on Babel for transpilation. Currently linters don’t understand Flow...the mix of tools is hard to integrate.  
      
    We’re interested in doing more of giving folks a recommended suite together.
  + Sebastian: We might unify on a toolchain, or we can go down an NPM route so folks can declare dependencies like for JSX transpilation. It should be as easy as ‘npm install’ and everything works.  
      
    We’re working on a best practice that you guys might be interested in. We’ll use semver and slowly add deprecation warnings and eventually tell you that you’re about to fall off the cliff. Writing this down is my next step.
* Web Components
  + Igor: What’s your take on Web Components? Some folks seem interested in wrapping Web Components.
  + Sebastian: We’re not going to use it at all at Facebook. We’re not going to build React on it because there’s a strong model difference -- imperative in Web Components to declarative in React. Web Components doesn’t have an idiomatic way to define things like where events go. How do you pass data when everything is a string? We see it more as an interop layer that lets various frameworks talk to each other.  
      
    In talking to the Atom team, this doesn’t solve different framework idioms as it doesn’t have an opinion on how they relate.
  + Lee: React needs to know enough about web components -- given a leaf node, how does React deal with it. We wrap stateful DOM elements to create stateless DOM elements. Web Components would be the same so we can make them appear stateless -- move the ownership of the state into our environment. If DOM owns the state and can change it, it’ll get out of sync with your model.
  + Sebastian: Whatever we express downwards, you should respect it.
  + Yegor: How do you deal with focus declaratively?
  + Lee: The hard thing about this is that focus is global. We wrap a text box to be stateless to capture the events, the developer then decides what the state should be. Focus is more difficult because of when something becomes focused, something else becomes unfocused. We want to think about things as sub-trees.
  + Yegor: Unless you map everything that can have focus to your model, your model isn’t representative. We’re thinking of representing this as an ‘action’ that isn’t part of your model. You send an event and forget about its existence.
  + Sebastian: That’s sort of what we do. We do have another idea. Focus and browser selection aren’t global -- they’re sub-tree specific. In an iframe you can have selection around and within. We might want to introduce as context for a sub-tree. Normally, that would be just the current page, but they could be nested. One thing you don’t want to do is to have the currently selected element state to flow it through every element. You want to say for this component and its sub-tree, what is the currently selected element. Then you want to be able to request focus that can be bubbled up to the next focusable root. If anything in a sub-tree is focused, it means that everything above it is focused too. If the top node says ‘you are focused’, the next thing can say ‘i know i’m focused, but the next thing down may be more specific to me so I bubble it down’.
* Benchpress
  + Igor: For a long time, we tried to measure performance in a predictable way. We knew that microbenchmarks weren’t very useful for framework code. We tried many things to move to macro-benchmarks.   
      
    We got close on a first version of Benchpress that was a runner that executed instructions in your app and measured the timing. But we had variable results. With Chrome canary, we could run GC and approximate how much garbage was created/collected on each pass. This got us excited. Wasn’t perfect, was Chrome-only, etc. Also, having it running in the page made things difficult.  
      
    We now use Webdriver to control the browser. Write an app to represent the use case. We then use chrome debugging timeline to annotate the code through the debugging protocol. There are similar protocols for Safari, Firefox, etc. so it’s not chrome only. We now have a test suite for Angular to verify different scenarios.  
      
    The other part of the puzzle were figuring out which test cases to use to know how Angular.  
      
    It’s not Angular specific. We run various version of Angular, Polymer, etc. in it.
  + Lee: One thing we’re interested in is from a developer perspective that they can try ideas and know if it improves or degrades performance.
  + Yegor: Yes, we use our build system to run comparing with previous builds.
  + Lee: We care less about the browser frontend but want to test the React internals and how that performs. We want to know the different performance of various JS VMs. Turn off JIT. Etc.
  + Brad: What are you doing about locking down the kernal, turning off extensions, etc.?
  + Lee: Nothing yet, but we should. We’d ideally like this to be runnable on Travis CI.
  + Brad: Sounds like good area for collaboration. We should also share our doc on ways we control variance.
  + Sebastian: I want a way to test un-primed actions like initial rendering or primed actions within an app.
* Animations
  + Matias: We’ve recently moved to a declarative animations format. I’m curious where you guys are on this.
  + Lee: There’s the easy case where you can just rely on CSS animations. Then there’s descriptive fire and forget animations for view to view transitions, modals, etc. Anything where you’re transitioning between states. Then there’s interruptible animations which makes state transitions stop. You might snap back to the original. You need a way to describe these interruptions and how they map to the gesture model.
  + Sebastian: The problem we have with our current API is that we don’t have a way to abstract things multiple levels deep - a component deep in the tree may want to animate when a component high up is deleted. The thing below may do an animation and tell the upper levels when they can go away.
  + <matias demonstrated new declarative timeline-based format>
* Other renderers: canvas, native, etc.
  + Sebastian: The first thing I built was a canvas renderer for React. It was an SVG subset. This powers all our charts at FB. That was the first thing that created a second rendering environment and led to the rendering on iOS. There’s lots of benefits when you don’t target another imperative API -- you have much more granular control over state/memory usage. On iOS, it’s the same model as Web/DOM where you have parents nesting children. On iOS there’s CA, View Hierarchy, and View Controllers, but we just target the view controllers. There’s a port that targets threejs.
  + Lee: We do retained mode except for Canvas where we do direct/half-and-half mode. We’re looking towards more GL surfaces for rendering.
  + Christopher: One big issue we deal with is layout. Most folks do they layout manually -- compute top left, width, height and go from there. I had to reimplement the layout with a subset of CSS using the Flexbox spec. I wish I could have reused Chrome or some similar layout engine. Turned out too hard to extract this.
  + Sebastian: Folks have used this layout with charts, Flipboards ReactCanvas, etc. Problem is with text measurements in the browser. If you render the same text in canvas it will look different than other text on the page.
  + Christopher: The layout takes a tree of components and computes top-left, height, width for all elements, independent of other bits.
  + Lee: We can do layout in a single pass and move that process to a separate thread. The concern of reflow that it’s expensive, but when you’re async a 5ms delay is imperceptible.
  + Sebastian: One problem we still deal with is that we can’t do text measurements in a web worker.
  + Christopher: One idea on ReactNative is to let you write native code and so long as the interface is async, you can do it. Image decoding on another thread? No problem.
  + Lee: This is good for helping us understand what we want out of the web better. We can try out the model on high performance system. We can implement it on ReactNative, then go to the browser vendors with a proposal. If Angular came to the same conclusion, we could go in together.
  + Christopher: The end game isn’t ReactNative. We want the web to win. Would be great for Angular to try to implement on top of our same primitives to see if we could share the work.
* Immutable JavaScript data structures
  + Victor: For us, extensible records are very interesting. As a developer, I would rather us that rather than using maps.
  + Sebastian: One proposal is syntactic sugar for extending an object that spreads values from original one, but you’re missing immutability. Might be interesting to extend this to annotate that object is frozen. One thing we’re talking about with React is around optimization around knowing what is static. I’d like to be able to reuse objects and know they haven’t changed.
  + Victor: Would be useful for us for change detection in skipping sub-trees.
  + Sebastian: Another thing we’d like is to know deep immutability.
  + Lee: Would good to get developers using Angular 2 to chime in with good patterns and what doesn’t work. Help us figure out patterns and incremental steps.
  + Brad: We’ve built A2 so we can support mixed immutable and observables/streams. Is that a thing for React?
  + Sebastian: It makes sense -- big DAGs have cycles and can be expensive to update. Mixing immutable with observables let you control this better.
* i18n
  + Igor: We’ve been trying different things with i18n. We knew that translation messages was something we wanted to get to, but was hard. We just did formatting of dates, numbers, etc. Internally, folks needed translation messages and built hacky solutions. Extracted messages from Angular templates that then built templates per locale. Was fine internally, but didn’t work externally.  
      
    Building a new system that works both at Google and for others. Will be for Angular 1 and Angular 2. Takes HTML templates with markers.
  + Lee: This works pretty well in React today as it’s really good at composing JS functions into output.
  + Sebastian: Yahoo has a fairly advanced localization tool. We have a requirement to annotate files or tag translations belong to a particular project so i18n teams can split up the work.
  + <igor demonstrated localization format>
  + <christopher demonstrated fbt tag usage -- Facebook-specific>
* Standards: types, annotations
  + Lee: Most excited about Flow as it divorces transpilation from semantics. Lets us incrementally compile from only dependencies that flow down from that. We have an IDE called Neucluide that plugs into this model.
  + Igor: How is that different from TypeScript language services?
  + Sebastian: I don’t think it’s as fast. I’m not familiar with it as much. Flow has goal from Hack to make it possible to parallelize work. Provide type annotations only within boundaries of a file so that you can do inference in parallel between files so you don’t have to annotate everything. I’m excited about it because it embraces existing JS coding style rather than putting new idioms on top of it. From a fundamental level, TypeScript seems inspired by C# vs Flow being inspired by ML or F#.
  + Lee: Annotations/decorators isn’t a clear thing we need in React. Maybe in class property initializers.
  + Sebastian: This is the killer feature we need. It’s like the TypeScript static x = value. Can also be instance level.
  + Igor: We need that to type properties.
  + Lee: You can then have default values for them or call a function like UID generator to populate. We also have this thing called autobinding that we can use it. Property initializers give us a better way to do this by passing a lambda to initializer.
  + Misko: What about need to describe properties of components to the framework?
  + Sebastian: Mix-ins work fine for this.
  + Victor: Since we don’t have inheritance, we don’t do it this way.
  + Sebastian: In the new version, we don’t extend.
* Web Workers
  + Yegor (Angular)
    - Goal is to build a non-blocking UI thread
    - Angular did a first prototype with a React-like virtual DOM
    - Had good performance results
    - Now doing production version that handles asynchrony of events
    - Building a way to express platform-specific features in a unified way
  + Christopher: You want animation to happen on the UI thread?
  + Yegor: Yes, to avoid complexity of synchronizing.
  + Christopher: We went that way, but complex animations need to know about the rendering pipeline.
  + Sebastian: We originally ran React all in Web Worker. In ReactNative we had to provide a different API. We do an async wrapper around UIKit APIs. We’re thinking about doing this for the Web as well. The interesting part is where you draw the line. What do you give access to the developer?
  + Yegor: For us, the thing that sits in the middle between UI and developer is like a state machine. At specific moments, with Zones, we decide when to flush. The idea is that Animations should be able to hook into the list of instructions that are going to the UI.
  + Christopher: For a scrolling header example, the easiest way is to do it in the Web Worker where you have all the context.
  + Misko: In Angular 2, we don’t have a virtual dom. Instead, we have a View Tree that deals with UI chunks that are opaque to Angular. We marshal on the level of Views and not DOM elements. We do the expansion to DOM nodes in the main thread. On the Web Worker, we have no notion of HTML. We can create views. We can set properties. Developers can write arbitrary JS for custom renderers.
  + Christian: What’s the level of the view?
  + Misko: Its the smallest indivisible part. Think of it as a document fragment. Individual rows in a repeater would be views.
  + Christian: Current solution is that everything on the main thread is ObjC or Java.
  + Sebastian: Currently, takes advantage of parallelizsm to get performance. Sometimes, web workers are in control of every tick. We think we can build a lot of the management into the framework to time slice automatically. We may also build boundaries where everything in a category must be fast (UI). The other world would not be as performance sensitive.
  + Lee: We’re currently unsatisfied with performance of web workers given the overhead. Immutable data structures in JS would make this basically free. There are many performance hacks we can do when this is true, like moving work between threads. Every single rendering can happen on an arbitrary thread. We’re trying to use this as a model to pull it back into JavaScript. Given architecture we’re doing on iOS to prove out arch, want to have it in JS. We’re looking to produce the same mental model across any and all platforms/environments. Folks don’t have to worry about which thread they’re on, they just write code. This is our long term vision. You just write code/components and don’t have to decide.
  + Yegor: What serialization strategies have you tried?
  + Sebastian: I’m experimenting with what is most efficient. It ends up varying across environments. The reason the DOM apis are slow is because of overhead between environments. We serialize using built-in JSON mechanisms because of the overhead of going between environments -- the built-in mechanism bypasses those boundaries.
  + Yegor: I’m currently struggling with wanting to use more complex structures than JSON. You then have to build objects again. It seems that if it’s always a tree, you might serialize into array and never turn back into objects. Prevents creating objects or garbage, but gives you a terrible API because you don’t have classes. Have to implement a visitor pattern and deal with tokens.
  + Lee: Shared memory is great, but doesn’t work across client/server. I’ve come to the same conclusions. Pay the cost to turn it back to original data structure if you want.
  + Brad: How could we share work on this?
  + Sebastian: Start with describing our API surfaces.
  + Lee: We have concrete use cases from iOS apps. Sebastian is working on polyfills that may be slow but would have the right API. We can then reframe this as an optimization with standards committees. Potentially build experiment in V8 that enable this kind of thing.
  + Yegor: I like the idea of coming up with what the protocol should look like for app developers.
  + Lee: The conversation I want to have with TC39 is what we want to come up with to get to the protocols we want.