TECHNOLOGY

JavaScript's History and How it Led To ReactJS

25 Jul 2014 9:02am, by Chris Dawson

Facebook today offered a fantastic presentation on ReactJS, "A JavaScript library for building user interfaces." The presenter, Christopher Chedeau from Facebook, spoke briefly about the history of JavaScript and why Facebook wrote ReactJS. If you've played with ReactJS and understand at least the basics of "how" to write ReactJS, this talk will explain the "why."

ReactJS started as a JavaScript port of XHP, a version of PHP which Facebook released four years ago. XHP was principally concerned with minimizing Cross Site Scripting (XSS) attacks. XSS attacks are facilitated when a malicious user enters content that is intended to inflict harm on the viewer of that content. Typical attack vectors are to enter content with embedded and hidden JavaScript (the language which runs inside every web browser), and then use that embedded JavaScript to steal information or otherwise compromise the user who views the content. XHP removes the burden of scrubbing user submitted information.

But, there was a distinct problem with XHP: dynamic web applications require many roundtrips to the server, and XHP did not solve this problem. So, a Facebook engineer negotiated with his manager to take XHP into the browser using JavaScript and was granted six months to try it. The result is ReactJS.

If you've been paying attention to JavaScript development for years, you'll know that "DOM manipulation is expensive" has become a mantra. In other words, taking your JavaScript application data and "rendering" it into a browser is a costly operation. It is surprising, therefore, that ReactJS is the first framework to take this to an obvious conclusion: optimizing for DOM manipulation leads to a fast library, and in this case, a library which enables web applications that require very little code. ReactJS works by storing the state of your application internally, and only re-rendering your content into the browser (the DOM manipulation) when the state changes. You interact with ReactJS by telling it when the state has changed and ReactJS handles all the

visual changes to your application for you. This abstraction is brilliant. The implementation is similar to AngularJS which handles DOM manipulation for you through two-way data binding, but ReactJS takes things a step further because it knows when things have changed and when they have not, which makes a difference in large applications. AngularJS relies on dirty checking and a digest loop, and my own experience is that the AngularJS interface does not have the ability to be as intelligent in determining when a change to the DOM is unnecessary and skip the DOM manipulation. For large applications, I believe this will make a difference.

If you are a computer scientist this talk will be fascinating as it documents the steps to build an effective "diff-ing" algorithm that understands when the state tree has changed. Christopher showed how initial approaches had an O(n^3) algorithmic complexity. Anyone that has ever interviewed for a computer engineering position knows that this is a high cost algorithm to be avoided with anything other than tiny data sets. The presenter noted that with a large web application with 10,000 DOM nodes, this would mean diff'ing the tree would require 17 minutes on a 1 GHz CPU. As Facebook engineers worked further on ReactJS they were able to optimize the algorithm to first achieve O(n^2) complexity, a huge improvement, but then were able to achieve O(n) complexity by using a hash map of DOM elements containing unique keys. This part of the talk was a fascinating entry point into the real CS fundamentals underlying this lovely library.

Knowing the steps used to achieve this algorithmic complexity helps explain why the library has the interface it does. ReactJS relies on a "batch" step and "pruning" step when determining what to render, steps to which your application can offer hints, and understanding the algorithm behind the scenes made lightbulbs go off in my head as I finally got why it worked the way it does. There is a learning curve with ReactJS in that you typically need to marshal data back and forth between views and models which Facebook has built to handle for you (AngularJS revolutionized this step). The API to do this in ReactJS makes more sense when you understand why it asks you to provide hints at the times it does and this talk clarified a lot for me.

In essence, the ReactJS interface is different than other js libraries which are more imperative, meaning you tell them to change the DOM directly. Jquery is like this. ReactJS is more like: here is my state, and here is how you should interpret my state and how it will change. Now, I can sit back and let ReactJS handle the expensive and complicated task of making this visible to the user in the browser.

Christopher also mentioned a very exciting framework called "Om" from David Nolan written in ClojureScript that wraps the ReactJS library. Because Om uses immutable data structures (a feature of Clojure and derivatives like ClojureScript) Om can optimize the "shouldComponentUpdate" method inside of ReactJS and provide the proper hints automatically to applications so that the ReactJS algorithm most efficiently re-renders the DOM. Implementing undo in an application like Om is trivial, as David notes, and asks that we

consider how we could do this in other JS libraries. ReactJS is a solid foundation to write interesting libraries upon.

Lastly, Christopher spoke about a performance optimization technique that comes free with ReactJS. If your application uses ReactJS, you can use the integrated performance monitoring tools inside ReactJS to analyze which parts of the application are incorrectly subjecting the application to re-rendering. Simply call "perf.printWasted()" inside your application and you'll see an analysis of places where your application wasted render cycles because ReactJS can tell if a render was called but no state changes were made. You can then hint to your application that it should not do this, improving responsiveness. I'm not aware of other frameworks that are smart enough to tell you when the code is incorrectly written and what to do about it.

ReactJS is a fantastic approach to building web applications in JavaScript. This talk teaches you the "why" which helps a lot after you have learned the "how." It is well worth watching when the OSCON videos are posted.

Feature image via Flickr Creative Commons

THE NEW STACK UPDATE

A newsletter digest of the week's most important stories & analyses.

Email Address

Subscribe

We don't sell or share your email. By continuing, you agree to our Terms of Use and Privacy Policy.

SPONSORED FEED

logdna

July Newsletter – Series C, New CEO, and Web Server Template
JULY 30, 2020

THE LINUX FOUNDATION

Fledge, an LF Edge Project, Enters Growth Stage as Release 1.8 Enables Open Industrial Edge Software with AI/ML, and Public Cloud Integration
JULY 30, 2020

Announcing the New AWS Community Builders Program!
JULY 30, 2020

Spread the Love: Appreciating Our Pollinators Community
JULY 30, 2020

Infrastructure as code, part 3: automate Kubernetes deployments with continuous integration and deployment
JULY 30, 2020

MongoDB 4.4: User-Driven Engineering. Ready for You
JULY 30, 2020

Jaeger Project Journey Report: A 917% increase in companies contributing code
JULY 30, 2020

Dynatrace Managed release notes version 1.198
JULY 30, 2020

Kubera: MayaData's unique cloud native approach to back-ups
JULY 30, 2020

Service Mesh Fundamentals Training Course Now Available
JULY 30, 2020

Community Highlight: How InfluxDB Enables IoT Monitoring of Gas Station Tanks
JULY 30, 2020

pagerduty

What's New: Updates to Visibility Console, Event Intelligence, Analytics, and More! by Vera Chan
JULY 30, 2020

HashiCorp

Making a Mesh in Multi-Cloud with Consul
JULY 30, 2020

New Relic.

Everything You Need to Know About Your New Observability Platform
JULY 30, 2020

SENTRY

Automate Release Management with the Sentry Release GitHub Action
JULY 29, 2020

cloudbees.

The Software Agents - Episode 3: Automated Wealth Management Advice
JULY 29, 2020

GitLab

Understand Kubernetes terminology from namespaces to pods
JULY 29, 2020

DIAMANTI

Reduce Time-to-Insight Running Splunk on Diamanti
JULY 29, 2020

SALTSTACK.

Creating an Open Source Business Model
JULY 29, 2020

redislabs
HOME OF REDIS

RediSearch 2.0 Hits Its First Milestone
JULY 29, 2020

sonatype	Nexus Intelligence Insights:CVE-2020-13935 - Apache Tomcat Websocket - Denial of Service (DoS) JULY 29, 2020
RED HAT OPENSHIFT	Red Hat OpenShift certified HPE CSI Operator for Kubernetes available now! JULY 29, 2020
APPDYNAMICS	Protect Your Bottom Line and Change the Game on SAP Monitoring with APM JULY 28, 2020
GRAVITATIONAL	Ensuring Safer Connections with SOC2 Certification JULY 28, 2020
PRISMA BY PALO ALTO NETWORKS	CN-Series Firewalls: Comprehensive Network Security for Kubernetes JULY 28, 2020
ASPEN MESH	What's a Service Mesh? JULY 28, 2020
Cockroach Labs	Build a Python App with SQLAlchemy + CockroachDB JULY 28, 2020
CLOUD FOUNDRY	Bi-weekly Round-Up: Technical + Ecosystem Updates from Cloud Foundry | 7.28.20 JULY 28, 2020
DATASTAX	Apache Cassandra Benchmarking: 4.0 Brings the Heat with New Garbage Collectors ZGC and Shenandoah JULY 28, 2020

What Every Technical Leader Should Know About Streaming Data Pipelines
JULY 26, 2020

Announcing HAProxy Data Plane API 2.1
JULY 24, 2020

Fundamental Library Styles version 0.11.0 has been released
JULY 23, 2020

Our Application Powered World Requires Stronger Infrastructure
JULY 22, 2020

How to backup and restore MySQL on Kubernetes
JULY 21, 2020

Citrix ADC CPX and Ingress Controller on the Azure Marketplace
JULY 15, 2020

Kubernetes our Own Way
JULY 08, 2020

Join us at our new blog home
JULY 02, 2020

RedMonk – The coming SMOKEstack: rethinking and retooling “multi-cloud”
JUNE 30, 2020

Securing Infrastructure as Code Using Terrascan
JUNE 24, 2020

Discover package vulnerabilities with the Snyk integration for JSDelivr
JUNE 08, 2020

Getting storage engines ready for fast storage devices
MARCH 16, 2020