Related Posts

Getting Started with Auth0 Lock Series: Implementing Lock
Kassandra Perch

More Benchmarks: Virtual DOM vs Angular 1 & 2 vs Others
Sebastián Peyrott

Incremental DOM 101: What is it and why I should care?
Pablo Terradillos

BLOG

More Benchmarks: Virtual DOM vs Angular 1 & 2 vs Others

You asked? We listened! Benchmark series is back with even more frameworks and metrics!

Sebastián Peyrott

January 07, 2016

191 152 132

We have extended our previous benchmarks to include other popular DOM manipulation libraries: Angular 1 and 2, Mithril.js, cito.js and the standalone independent implementation of React's Virtual DOM algorithm. We have also added more metrics (including memory use). Read on! **Update:** we have released an updated version of this article including the use of track-by and array key ids, check it out.

nmarks article, here is what we did: we took a sample implementation of DBMonster along ools from this post, reimplemented it for use with different DOM libraries, ran the scripts and collected the results. For this post we have also extended browser-perf to generate memory usage statistics.

> We will submit our additions to browser-perf for review, which is why we haven't linked to the original browser-perf repo above.

What follows is an overview of the contenders. Or you can jump straight to the results.

Angular.js 1.x

The venerable Angular 1.x library is part of an opinionated but powerful framework that saw its initial release in the year 2009. Since then it has gathered a tremendous following and inspired many other libraries. Often criticized for being hard to integrate into existing projects, and questioned for its performance in certain use cases, it is still used in many high-ranking websites. Angular extends standard digest cycle is the method Angular uses to detect changes in the model that trigger changes in the view. This method has particular performance characteristics as can be seen in the results below.

```
<td ng-repeat="query in database.topFiveQueries"
    ng-class="getClassName(query)">
    <div class="popover left">
        <div class="popover-content">

        </div>
        <div class="arrow"></div>
    </div>
>
```

And part of the associated JavaScript code, including dependency injection semantics:

```
dbmonControllers.controller('MainController', ['$scope', '$timeout',
    function($scope, $timeout) {
        $scope.loadCount = 0;

                                                n(query) {
                                                short";
                                                ) {
                                                warn_long";
            } else if (query.elapsed >= 1.0) {
                className = "elapsed warn";
            }
            return "Query " + className;
        };
        //(...)
    }
);
```

Angular.js 2

The still-in-flux Angular 2 library has recently been declared in *beta* state. Still under heavy scrutiny by many developers that do not consider a breaking change to be the right way to fix all of Angular 1.x problems, Angular 2 carries on with its opinionated approach.

model has changed (in contrast to the need to explicitly tell Angular 1 so in certain cases).

An Angular 2 view template:

```
<td *ngFor="#query of database.topFiveQueries"
    [ngClass]="getClassName(query)">
    <div class="popover left">
        <div class="popover-content">
        </div>
        <div class="arrow"></div>
    </div>
</td>
```

And its associated JavaScript code:

```
AppComponent = ng.core
    .Component({
        selector: 'my-app',
        templateUrl: './app/component.html'
                                            {
                                          short";
                                          ) {
                className = "elapsed warn_long";
            } else if (query.elapsed >= 1.0) {
                className = "elapsed warn";
            }
            return "Query " + className;
        }
    ]);
```

> Our Angular 2 DBMonster code was developed using JavaScript rather than the recommended TypeScript to reuse as much code as possible from other versions of the benchmark. Angular 2 code is more idiomatic using TypeScript and was developed with its features in mind.

Virtual DOM is an independent implementation of React's tree-diffing algorithm. It provides an API that allows users to describe a DOM tree directly in JavaScript. JSX is not used.

```
function renderQuery(query) {
    var className = "elapsed short";
    if (query.elapsed >= 10.0) {
        className = "elapsed warn_long";
    } else if (query.elapsed >= 1.0) {
        className = "elapsed warn";
    }

    return h('td', { className: 'Query ' + className }, [
        query.elapsed ? formatElapsed(query.elapsed) : '',
        h('div', { className: 'popover left' }, [
            h('div', { className: 'popover-content' }, query.query),
            h('div', { className: 'arrow' })
        ])
    ]);
}
```

Mithril.js

A React-like library aimed at being simple, small and fast. It provides a convenient API to describe DOM-trees in JavaScript, and a nto simple JSON objects for additional speed. The API for DOM-trees is similar to that of view objects are simple enough to be integrated easily into existing bodies of code. Much like tree made outside library boundaries to be notified for redrawing. Data-binding is performed

```
function renderQuery(query) {
    var className = "elapsed short";
    if (query.elapsed >= 10.0) {
        className = "elapsed warn_long";
    } else if (query.elapsed >= 1.0) {
        className = "elapsed warn";
    }

    return m('td', { className: 'Query ' + className }, [
        query.elapsed ? formatElapsed(query.elapsed) : '',
        m('div', { className: 'popover left' }, [
            m('div', { className: 'popover-content' }, query.query),
            m('div', { className: 'arrow' })
        ])
```

A minimalist Virtual DOM-like library. A virtual DOM is constructed from plain JSON-like objects. A simple call tells the library to compare the existing tree to a new one and perform the necessary updates.

Related Posts

Getting Started with Auth0 Lock Series: Implementing Lock

Kassandra Perch

More Benchmarks: Virtual DOM vs Angular 1 & 2 vs Others

Sebastián Peyrott

Incremental DOM 101: What is it and why I should care?

Pablo Terradillos

```
 tion renderQuery(query) {
  var className = "elapsed short";
  if (query.elapsed >= 10.0) {
    className = "elapsed warn_long";
  } else if (query.elapsed >= 1.0) {
    className = "elapsed warn";
  }

  return {
    tag: 'td',
    attrs: { 'class': 'Query ' + className },
    children: [
      query.elapsed ? formatElapsed(query.elapsed) : '',
      {
        tag: 'div',
        attrs: { 'class': 'popover left' },
        children: [

                 ss': 'popover-content' },
                 ry.query.toString()


                 ss': 'arrow' }
          }
        ]
      }
    ]
  };
}
```

Ember 1

Another powerful and opinionated framework. Ember makes use of their own template language which allows for easy differentiation between static and dynamic parts of the DOM. Their diffing algorithm is optimized to take this into account. Data binding is handled through a properties system.

```
    <td class="Query ">
      <div class="popover left">
        <div class="popover-content"></div>
        <div class="arrow"></div>
      </div>
    </td>
```

And its associated JavaScript code:

```
ort default Ember.Component.extend({
 agName: 'tr',

  queries: function() {
    var samples = this.get('attrs.db.value.samples');
    return samples[samples.length - 1].queries;
  }.property('attrs.db'),

  topFiveQueries: function() {
    var queries = this.get('queries');
    var topFiveQueries = queries.slice(0, 5);
                   {
                   });
                  n(query, index) {
      return {
        key: index+'',
        query: query.query,
        elapsed: query.elapsed ? formatElapsed(query.elapsed) : '',
        className: elapsedClass(query.elapsed)
      };
    });
  }.property('queries'),

  //(...)
});
```

Ember 2

The Ember 2 code in this article is 100% compatible with the Ember 1.x version, so no example is provided here.

React.js

Facebook's popular library is gaining ground day-by-day. Its simple integration model, flexibility and speed make it a no-brainer for many projects. The crux of React is its smart diffing algorithm: a virtual tree is constructed by making JavaScript calls. When a new DOM tree is constructed, React can find the optimal number of operations to transform the old tree into the new one. React is usually paired with the JSX preprocessor, which allows an extended form of HTML to be embedded in JavaScript to describe components in a convenient way.

```
  Query = React.createClass({
  render: function() {
    var className = "elapsed short";
    if (this.props.elapsed >= 10.0) {
      className = "elapsed warn_long";
    }
    else if (this.props.elapsed >= 1.0) {
      className = "elapsed warn";
    }

    return (
                   Name}>
                   lapsed(this.props.elapsed) : ''}
                   >
                   tent">{this.props.query}</div>

      </td>
    );
  }
})
```

Incremental DOM

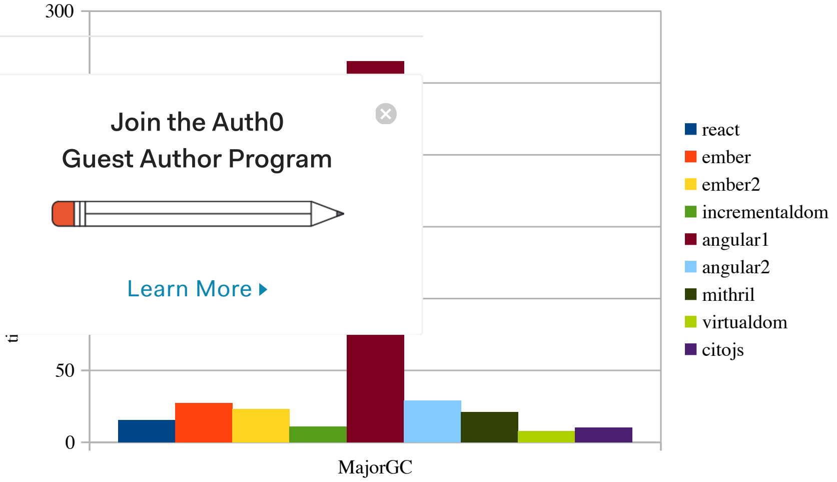
A Google project, Incremental DOM aims to develop a memory-efficient library that can perform in-place updates of the DOM tree. It is intended to be used as a compilation target for different template languages. In practice, its API is similar to that of Virtual DOM, Mithril or cito.js. The main benefit of a memory efficient approach is reduced waits during garbage collection cycles. This can result in improved performance and fewer dropped frames during rendering.

```
      if (query.elapsed >= 10.0) {
        className = "elapsed warn_long";
      } else if (query.elapsed >= 1.0) {
                                       wa

      elementOpen('td', null, null, 'class', "Query " + className);
        text(query.elapsed ? formatElapsed(query.elapsed) : '');
        elementOpen('div', null, ['class', 'popover left']);
          elementOpen('div', null, ['class', 'popover-content']);
            text(query.query);
          elementClose('div');
          elementVoid('div', null, ['class', 'arrow']);
        elementClose('div');
      elementClose('td');
    }
```

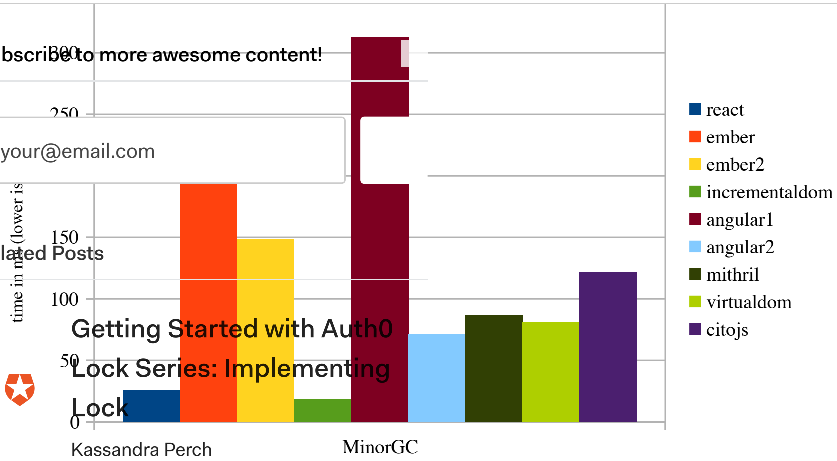
The Results

Here are the summarized results of our tests:

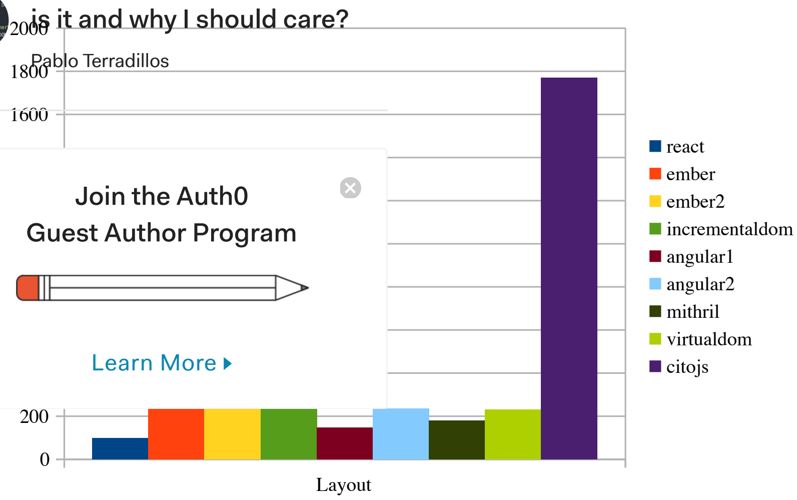


Angular 1.x results make this graph hard to read.

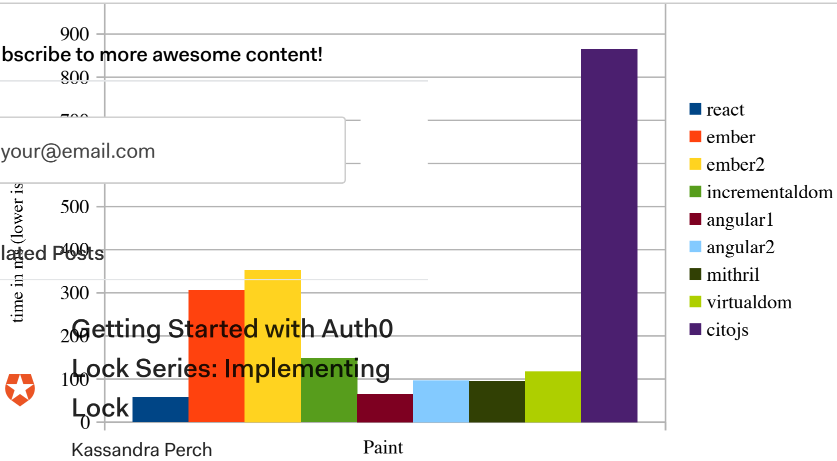
Angular 1.x performance problems are known and it shows in this graph. Major GC collections take several times longer compared to the alternatives. As expected, lightweight libraries such as mithril and cito.js shine here, but the absolute winner is Virtual DOM. Incremental DOM is also among the winners. The difference between Angular 1 and 2 is outstanding. It appears a big break was necessary to fix many of the shortcomings Angular 1 had.



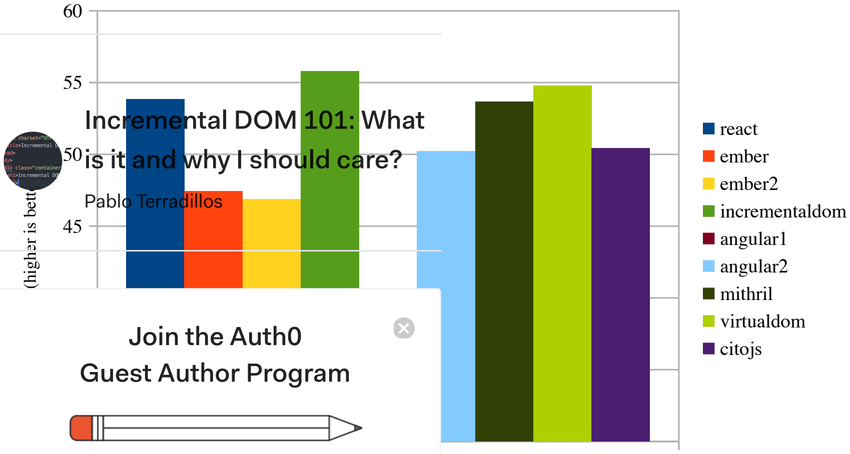
Incremental DOM shines here. Being developed for lower memory use pays off. React is a surprise as well, sporting excellent results in spite of its two-copies virtual DOM algorithm. The one that draws the attention is Virtual DOM, with its independent React-like virtual dom implementation: it shows longer pauses that React. This is probably the result of the optimizations Facebook developers have done over time to React. Angular 1.x is still the slowest and the difference with Angular 2 is still surprising. Ember 1 and 2 are the next in the pack in front of Angular 1. The difference between Ember 1 and 2 is significant. It appears many optimizations have been carried out under the hood.



This graph shows the time spent by the browser doing re-layout operations (i.e. creating an internal representation of the DOM tree after changes). I find it surprising that cito.js is the slowest in this case. Subjective performance would tell otherwise, as both cito.js and Incremental DOM feel quite snappy when interacting directly with the browser. Ember 1 and 2 are the next in the pack, and in this case the subjective feeling matches with the result. React is black-magic in this case, being the fastest. Incremental DOM sits somewhere in the middle along with the rest.



This graph is almost a copy of the previous one. It shows the amount of time spent rasterizing the elements in the internal DOM tree of the browser. There is not much to add here other than restate that the subjective feeling for Incremental DOM and cito.js does not match what is shown in the graph (really, try it!).



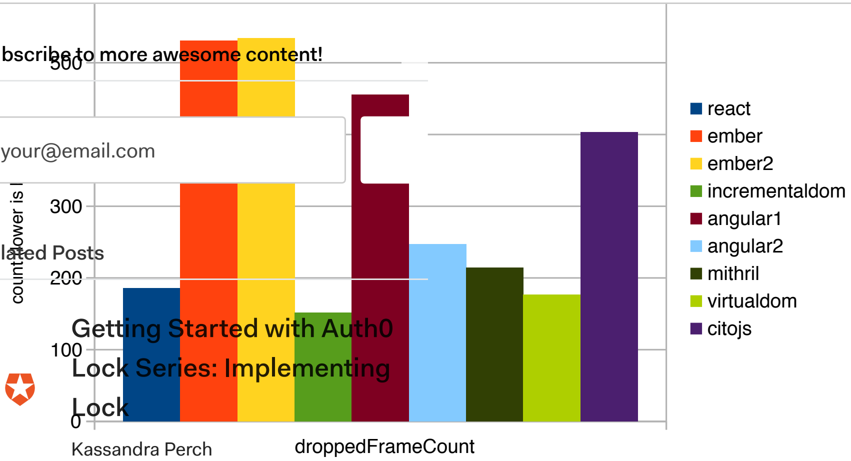
Deviation among most libraries is small in this graph, so one would expect hands-on usage to be pretty much the same for all of them except one (Angular 1). This is not the case. Even though Angular 1.x does feel sluggish compared to the pack, the difference between cito.js and Incremental DOM compared to the others is much bigger than it would appear. There must be something else going on behind the scenes (or this metric is either not measured correctly or useless).

Related Posts

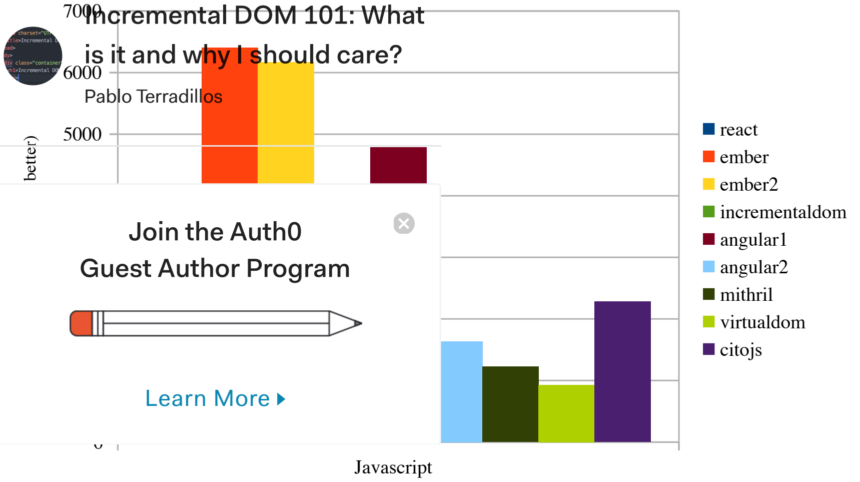
Getting Started with Auth0 Lock Series: Implementing Lock
Kassandra Perch

More Benchmarks: Virtual DOM vs Angular 1 & 2 vs Others
Sebastián Peyrott

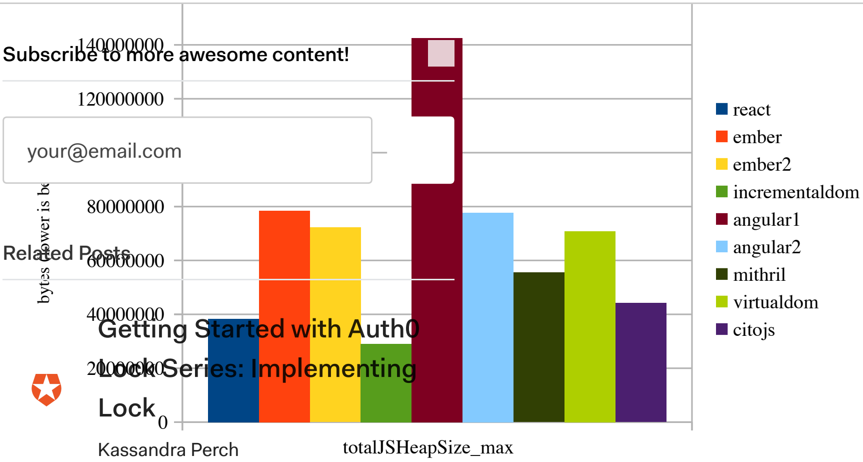
Incremental DOM 101: What is it and why I should care?
Pablo Terradillos

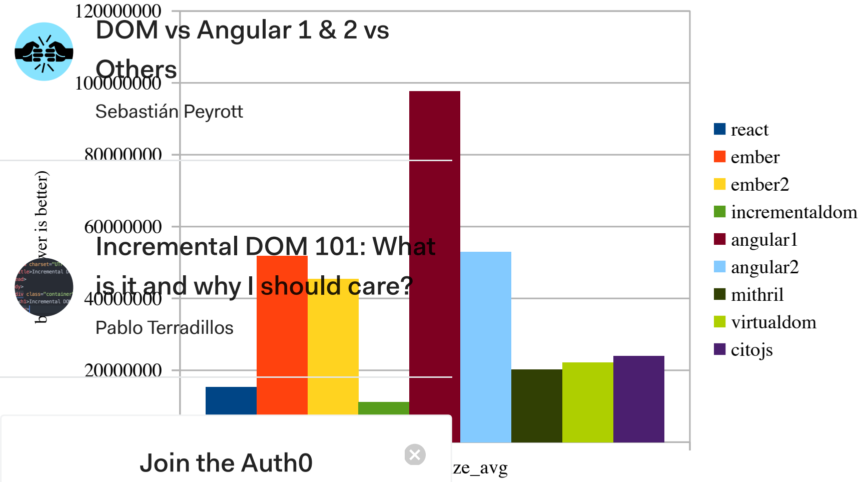


This chart shows the number of frames that take longer than 16.6ms to be drawn. They account for stuttering and other non-smooth behavior so it is an important metric for the subjective feeling of an application. Ember 1 and 2 are outshined by almost all of the other libraries. Incremental DOM, on the other hand shines again, followed closely by Mithril and React. Again the results for cito.js are odd. We will review our testing environment to make sure background processes are not affecting these results.



This graph shows the total amount of time spent by the browser executing JavaScript code. Ember 1 and 2 are the slowest here, followed closely by Angular 1.x. Again the difference between Angular 1 and 2 is abysmal. In my limited experience with Angular 2, I would say it is a step in the right direction. Incremental DOM and React shine here, followed closely by Mithril and Virtual DOM. If it serves as any indication, when it comes to CPU metrics, React appears to be the most balanced, scoring wins and near wins in several benchmarks.





These two graphs are deeply related so there is only

the amount of memory used by the browser when running the benchmark. The first one ding unused free space; while the second shows an average of the amount of memory used only by active objects. They look mostly the same. In general, Incremental DOM used the least amount of memory, followed closely by React (a real surprise!). React optimizations have paid off and it still looks as all-around balanced library. Mithril, cito.js and Virtual DOM all do very well in these tests. Again, of note are the improvements when going from Angular 1 to Angular 2. Angular 2 has improved every single metric when compared to Angular 1, and in a significant way. Angular 1 almost doubles the memory use of its follower.

Check out the full code for all tests.

Aside: React at Auth0

At Auth0 we care about maintainability and performance. We routinely look at competing solutions to find out the right tool for the job. For our Passwordless Lock library, we found that React has great performance characteristics, allows for an easy integration path, and

Conclusion

e of the Incremental DOM library. Not only in numbers, but also in the way it *feels* faster than the alternatives. Google has done a tremendous job optimizing its internals to produce a faster and leaner library. The only other library that has achieved this subjective feeling of speed is cito.js. In the case of cito.js we are concerned by its apparent lack of development activity at GitHub. When it comes to bigger frameworks, we cannot recommend React enough. It sports the right balance of pe mance, memory use, support and community mindshare. Mithril.js surprised us with its ease of use and great performance. We consider it to be a rightful contender to React, with an all-around great balance between speed, memory usage and ease of use. Between the bigger frameworks, both Angular and Ember are good choices. We consider them great for many use cases: in particular, for developers that adhere to their specific ideologies (if you like TypeScript, you will love Angular 2 as a library that is developed for it from the ground up). Angular's performance issues are by now known by all developers. If you want to use it, take this into co eration. In particular, consider that Angular 2 is significantly better than Angular 1 in every metric (to be fair, Angular 1 and 2 are different enough to be considered totally different libraries, only inspired by similar concepts).

> "Mithril.js surprised us with its ease of use and great performance."
>
> TWEET THIS

Some oddities in the results (for instance cito.js appearing slower in dropped frame counts and FPS counts when subjective tests tell otherwise) make us think something else is going on behind the scenes, so take these results as what they are: a small and probably biased

s probably need an improvement.

nk! Cheers!

13 Comments Auth0 Blog 1 Login

♡ Recommend 5 Share Sort by Best

Join the discussion…

Donny Rolanda • 8 months ago

Антон Кудрис • 7 months ago

em... am I blind or you aren't using "track by" in "ng-repeat" in Angular 1 code in templates? I mean what's the point of making benchmarks if you have a poor code in test examples from the very beginning?

Sebastian Peyrott Mod ➦ Антон Кудрис • 7 months ago

We have a note at the top about this. We did run these tests using track-by for Angular in an updated version of the benchmarks: https://auth0.com/blog/upda...

2 ^ | ⌄ • Reply • Share ›

Ajinkya ➦ Антон Кудрис • 7 months ago

angular 1 doesnt do much even if you track by if the profiles to be rendered in dom are more than 1000

^ | ⌄ • Reply • Share ›

Tim Zadorozhniy ➦ Антон Кудрис • 7 months ago

The problem is that Angular 1 doesn't provide clear API by default, to achieve good performance you need to know all these hacks like "track by", not triggering scope digest and counting.

^ | ⌄ • Reply • Share ›

Антон Кудрис ➦ Tim Zadorozhniy • 7 months ago

Angular is far from perfection but it's a job of programmer to learn how to use tool effectively. And if you fail to learn and understand how your tool works, you'll create poor code with any framework

"track by" isn't a hack and clearly described in the docs https://docs.angularjs.org/...

and the debug mode is ON by default in Angular. You can turn it off and gain some performance boost. It's also in the docs.

3 ^ | ⌄ • Reply • Share ›

Tim Zadorozhniy • 7 months ago

Despite all the tests, the main value of the framework is a way to mange complexity in the growing application. So far I had good experience with React and component based approach + redux for state management.

ago

not component based or that it can not use redux or better yet fluxible? How come?

ıs ago

/ should show the graphs, and delete everything else they wrote. As they got everything about react wrong, and it's the best framework out of the lot, easily. IMO

^ | ⌄ • Reply • Share ›

Sebastian Peyrott Mod ➦ dsont • 6 months ago

The article remarks that picking a library is a much more complex matter than just looking at performance in a single very specific case:

> "When it comes to bigger frameworks, we cannot recommend React enough. It sports the right balance of performance, memory use, support and community mindshare."

Of course, application complexity and maintainability are key aspects as well.

^ | ⌄ • Reply • Share ›

dsont ➦ dsont • 6 months ago

I as well use redux + react. And the crux (the decisive or most important point at issue) of react is not dom diffing, that's an implementation detail.

^ | ⌄ • Reply • Share ›

Marcelo Cantos • 9 months ago

Log graphs would be nice. They are make ratio comparisons easier when there's a wide range of values.

Subscribe to more awesome content!

your@email.com

Related Posts

Getting Started with Auth0 Lock Series: Implementing Lock

Kassandra Perch

More Benchmarks: Virtual DOM vs Angular 1 & 2 vs Others

Sebastián Peyrott

Incremental DOM 101: What is it and why I should care?

Pablo Terradillos

Join the Auth0 Guest Author Program

Learn More ▸

Hi Sebastian,

Very nice article, can you help us how did you calculated all metrics, meaning which tool you have used to collect them and how did you measured. can you explain us step by step video or something.

ov

similar type of code with the results in message box and graphs in excel sheet with original values.

Thanks in Advance

^ | ˅ • Reply • Share ›

✉ Subscribe Add Disqus to your siteAdd DisqusAdd 🔒 Privacy

Subscribe to more awesome content!

your@email.com

Related Posts

Getting Started with Auth0 Lock Series: Implementing Lock

Kassandra Perch

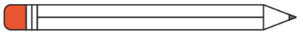
More Benchmarks: Virtual DOM vs Angular 1 & 2 vs Others

Sebastián Peyrott

Incremental DOM 101: What is it and why I should care?

Pablo Terradillos

Join the Auth0 Guest Author Program



Learn More ▸

USE AUTH0 FOR FREE

Subscribe to more awesome content!

your@email.com

Related Posts

Getting Started with Auth0 Lock Series: Implementing Lock

Kassandra Perch

More Benchmarks: Virtual DOM vs Angular 1 & 2 vs Others

Sebastián Peyrott

Incremental DOM 101: What is it and why I should care?

Pablo Terradillos

PRODUCT

Pricing

Why Auth0

How It Works

COMPANY

About Us

Blog

Jobs

Press

LEARN

Help & Support

Documentation

Open Source

EXTEND

Lock

WordPress

API Explorer

10900 NE 8th St.
Suite 700
Bellevue, WA 98004

+1 (888) 235-2699

+1 (425) 312-6521

Support Center

Subscribe to more awesome content!

your@email.com

Like 13K

Related Posts

Getting Started with Auth0 Lock Series: Implementing Lock

Kassandra Perch

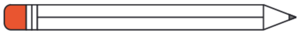
More Benchmarks: Virtual DOM vs Angular 1 & 2 vs Others

Sebastián Peyrott

Incremental DOM 101: What is it and why I should care?

Pablo Terradillos

Join the Auth0
Guest Author Program



Learn More ▸

Privacy Policy Terms of Service