REACT.JS

The DOM as a Persistent Data Structure

@zerokarmaleft

(

ABOUT ME

- Senior Software Engineer
  at Laureate Institute for Brain Research
- Build stuff for the web
- Build stuff to process data

WHAT IS REACT.JS?

- “a JavaScript library for creating user interfaces”
- “the V in **MVC**”
- simple (i.e. not tied to mutable state)
- declarative
- controversial (maybe)

SERVER-SIDE **MVC**

On every request:

- render the current application state

CLIENT-SIDE **MVC**

React to browser events:

1. update model client-side
2. update model server-side
3. update DOM

CLIENT-SIDE **MVC**

- consistent world view
- efficiency
- modularity

TEMPLATES

- weak abstractions
- not very composable
- weak expressive power
- not very coherent

“However **isolated scope** creates a new problem: if a **transcluded** DOM is a child of the widget isolated scope then it will not be able to **bind** to anything. For this reason the transcluded scope is a **child** of the **original scope**, before the widget created an isolated scope for its local variables. This makes the transcluded and widget isolated scope **siblings**.”

– Angular.js **directives** documentation

http://docs.angularjs.org/guide/directive

COMPONENTS

- composable
- cohesive
- loosely-coupled
- expressive
- testable

```
var TodoList = React.createClass({
  render: function() {
    var createItem = function(itemText) {
      return React.DOM.li(null, itemText);
    };
    return React.DOM.ul(null, this.props.items.map(createItem));
  }
});

var TodoApp = React.createClass({
  getInitialState: function() {
    return { items: [], text: '' };
  },
  onChange: function(e) {
    this.setState({ text: e.targetValue });
  },
  handleSubmit: function(e) {
    e.preventDefault();
    var nextItems = this.state.items.concat([this.state.text]);
    var nextText = '';
    this.setState({ items: nextItems, text: nextText });
  },
  render: function() {
    return (
      React.DOM.div(null,
        React.DOM.h3(null, "TODO"),
        TodoList({ items: this.state.items }),
        React.DOM.form({ onSubmit: this.handleSubmit },
          React.DOM.input({ onChange: this.onChange,
                            value:    this.state.text }),
          React.DOM.button(null, 'Add #' + (this.state.items.length + 1))
        )
      )
    );
  }
});
```

```
var TodoApp = React.createClass({
  getInitialState: function() {
    return { items: [], text: '' };
  },
  onChange: function(e) {
    this.setState({ text: e.targetValue });
  },
  handleSubmit: function(e) {  ← abstraction
    e.preventDefault();
    var nextItems = this.state.items.concat([this.state.text]);
    var nextText = '';
    this.setState({ items: nextItems, text: nextText });
  },
  render: function() {
    return (
      React.DOM.div(null,
        React.DOM.h3(null, "TODO"),
        TodoList({ items: this.state.items }), ← composition
        React.DOM.form({ onSubmit: this.handleSubmit },
          React.DOM.input({ onChange: this.onChange,
                            value:    this.state.text }),
          React.DOM.button(null, 'Add #' + (this.state.items.length + 1))
        )
      )
    );
  }
});
```

JSX

- convenience for working with designers
- entirely optional
- transforms HTML-like syntax to lower-level compositional functional calls

```
render: function() {
  return (
    <div>
      <h3>TODO</h3>
      <TodoList items={this.state.items} />
      <form onSubmit={this.handleSubmit}>
        <input onChange={this.onChange} value={this.state.text} />
        <button>{'Add #' + (this.state.items.length + 1)}</button>
      </form>
    </div>
  );
}
```

```javascript
render: function() {
  return (
    React.DOM.div(null,
      React.DOM.h3(null, "TODO"),
      TodoList({ items: this.state.items }),
      React.DOM.form({ onSubmit: this.handleSubmit },
        React.DOM.input({ onChange: this.onChange, value: this.state.text }),
        React.DOM.button(null, 'Add #' + (this.state.items.length + 1))
      )
    )
  );
}
```

"Co-located view logic and HTML? Blasphemy!!!"

– first reaction by anyone used to MVC

VIRTUAL DOM

- DOM operations are slow
- Unbatched DOM operations are slow
- DOM reflows are slow
- Too many event handlers are slow

VIRTUAL DOM

- declarative abstraction over explicitly optimizing DOM operations and event-delegation
- no manual data synchronization
- no magic data-binding
- no complex model dirty-checking

ARCHITECTURE

1. Game state
2. Game logic loop
3. Scene intermediate representation
4. Optimized OpenGL operations
5. GPU

ARCHITECTURE

1. Application state
2. Application logic loop
3. Virtual DOM
4. Optimized DOM operations
5. Browser

OM

- an opinionated ClojureScript interface to React.js
- DSL for defining component DOM
- builds on top of core principles from React.js with CLJS's:
  - immutable data structures
  - powerful concurrency primitives

OM

- React.js diffing relies on `shouldComponentUpdate`
- Om uses immutable data structures
- Implementing `shouldComponentUpdate` amounts to a simple reference equality check
- UI state is always serializable, always snapshottable

```
user=> (def x (atom 0))
#’user/x
user=> x
#<Atom@4c640782: 0>
user=> (deref x)
0
user=> @x
0
user=> (swap! x inc)
1
user=> (reset! x 100)
100
user=> (swap! x (fn [n] (* n n)))
```

```clojure
(def app-state   (atom {:showing :all, :todos []}))
(def app-history (atom [@app-state]))

(add-watch app-state :history
  (fn [_ _ _ new-state]
    (when-not (= (last @app-history) new-state)
      (swap! app-history conj new-state))
    (set! (.-innerHTML (.getElementById js/document “message”))
      (let [c (count @app-history)]
        (str c “ Saved “ (pluralize c “State”))))))

(aset js/window “undo”
  (fn [e]
    (when (> (count @app-history) 1)
      (swap! app-history pop)
      (reset! app-state (last @app-history)))))
```

```
(def app-state   (atom {:showing :all, :todos []}))
(def app-history (atom [@app-state]))

(add-watch app-state :history
  (fn [_ _ _ new-state]
    (when-not (= (last @app-history) new-state)
      (swap! app-history conj new-state))
    (set! (.-innerHTML (.getElementById js/document “message”))
      (let [c (count @app-history)]
        (str c “ Saved “ (pluralize c “State”))))))

(aset js/window “undo”
  (fn [e]
    (when (> (count @app-history) 1)
      (swap! app-history pop)
      (reset! app-state (last @app-history)))))
```

```clojure
(def app-state   (atom {:showing :all, :todos []}))
(def app-history (atom [@app-state]))

(add-watch app-state :history
  (fn [_ _ _ new-state]
    (when-not (= (last @app-history) new-state)
      (swap! app-history conj new-state))
    (set! (.-innerHTML (.getElementById js/document “message”))
      (let [c (count @app-history)]
        (str c “ Saved “ (pluralize c “State”))))))

(aset js/window “undo”
  (fn [e]
    (when (> (count @app-history) 1)
      (swap! app-history pop)
      (reset! app-state (last @app-history)))))
```

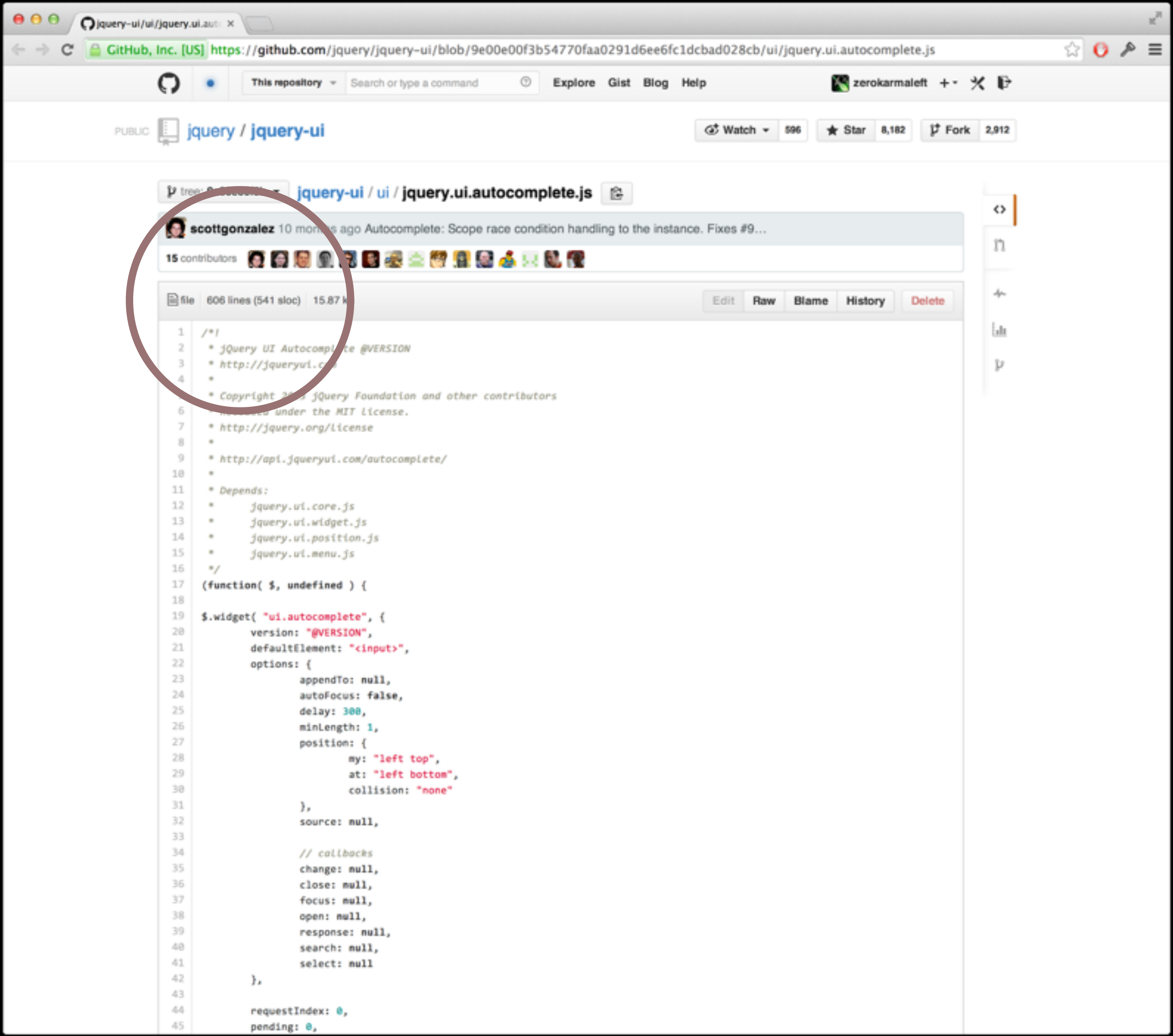
```clojure
(def app-state   (atom {:showing :all, :todos []}))
(def app-history (atom [@app-state]))

(add-watch app-state :history
  (fn [_ _ _ new-state]
    (when-not (= (last @app-history) new-state)
      (swap! app-history conj new-state))
    (set! (.-innerHTML (.getElementById js/document “message”))
      (let [c (count @app-history)]
        (str c “ Saved “ (pluralize c “State”))))))

(aset js/window “undo”
  (fn [e]
    (when (> (count @app-history) 1)
      (swap! app-history pop)
      (reset! app-state (last @app-history)))))
```

OM

- Communication between components can be streamlined with core.async
- Manage complicated state models for logical concurrent UI processes
- Goodbye to **CALLBACK HELL**



jquery-ui/ui/jquery.ui.aut

GitHub, Inc. [US] https://github.com/jquery/jquery-ui/blob/9e00e00f3b54770faa0291d6ee6fc1dcbad028cb/ui/jquery.ui.autocomplete.js

This repository · Search or type a command · Explore · Gist · Blog · Help · zerokarmaleft

PUBLIC jquery / jquery-ui

Watch 596 · Star 8,182 · Fork 2,912

tree: [illegible] jquery-ui / ui / jquery.ui.autocomplete.js

scottgonzalez 10 months ago Autocomplete: Scope race condition handling to the instance. Fixes #9…

15 contributors

file · 606 lines (541 sloc) · 15.87 kb

Edit · Raw · Blame · History · Delete

```
/*!
 * jQuery UI Autocomplete @VERSION
 * http://jqueryui.com
 *
 * Copyright [illegible] jQuery Foundation and other contributors
 * [illegible] under the MIT license.
 * http://jquery.org/license
 *
 * http://api.jqueryui.com/autocomplete/
 *
 * Depends:
 *	jquery.ui.core.js
 *	jquery.ui.widget.js
 *	jquery.ui.position.js
 *	jquery.ui.menu.js
 */
(function( $, undefined ) {

$.widget( "ui.autocomplete", {
	version: "@VERSION",
	defaultElement: "<input>",
	options: {
		appendTo: null,
		autoFocus: false,
		delay: 300,
		minLength: 1,
		position: {
			my: "left top",
			at: "left bottom",
			collision: "none"
		},
		source: null,

		// callbacks
		change: null,
		close: null,
		focus: null,
		open: null,
		response: null,
		search: null,
		select: null
	},

	requestIndex: 0,
	pending: 0,
```

```clojure
(defn autocompleter* [{:keys [focus query select cancel menu] :as opts}]
  (let [out         (chan)
        [query raw] (r/split r/throttle-msg? query)]
    (go (loop [items nil focused false]
          (let [[v sc] (alts! [raw cancel focus query select])]
            (cond
              (= sc focus)
              (recur items true)

              (= sc cancel)
              (do (-hide! menu)
                (>! (:query-ctrl opts) (h/now))
                (recur items (not= v :blur)))

              (and focused (= sc query))
              (let [[v c] (alts! [cancel ((:completions opts) (second v))])]
                (if (or (= c cancel) (zero? (count v)))
                  (do (-hide! menu)
                    (recur nil (not= v :blur)))
                  (do
                    (-show! menu)
                    (-set-items! menu v)
                    (recur v focused))))

              (and items (= sc select))
              (let [_      (reset! (:selection-state opts) true)
                    _      (>! (:query-ctrl opts) (h/now))
                    choice (<! ((:menu-proc opts) (r/concat [v] select)
                                 (r/fan-in [raw cancel]) menu items))]
                (reset! (:selection-state opts) false)
                (-hide! menu)
                (if (= choice ::cancel)
                  (recur nil (not= v :blur))
                  (do (-set-text! (:input opts) choice)
                    (>! out choice)
                    (recur nil focused))))

              :else
              (recur items focused)))))
    out))
```

DEMOS

- TodoMVC
- life

OM BENCHMARKS

- add a bunch of TodoItems
- add a bunch of TodoItems, repeatedly toggle the completion status of all TodoItems, delete all of the TodoItems

REFERENCES

- React.js
  http://facebook.github.io/react/index.html
- Pete Hunt, "Rethinking Best Practices". JSConf 2013.
  http://www.slideshare.net/floydophone/react-preso-v2
- Speed up your JavaScript, Part 4
  http://www.nczonline.net/blog/2009/02/03/speed-up-your-javascript-part-4/
- Event delegation in JavaScript
  http://www.nczonline.net/blog/2009/06/30/event-delegation-in-javascript/
- David Nolen, "The Future of JavaScript MVC Frameworks"
  http://swannodette.github.io/2013/12/17/the-future-of-javascript-mvcs/
- David Nolen, "Time Travel".
  http://swannodette.github.io/2013/12/31/time-travel/
- David Nolen, "Comparative Literate Programming".
  http://swannodette.github.io/2013/08/17/comparative/

)