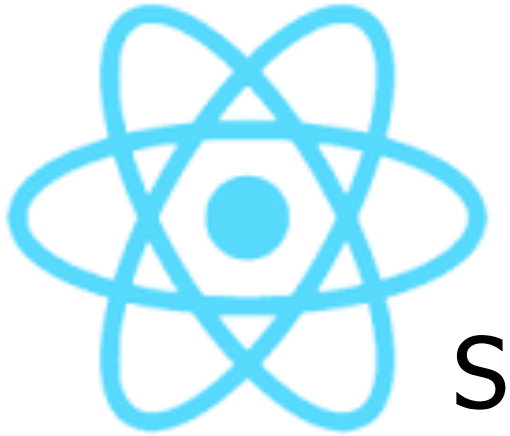


React

Say "No" to Complexity!

Mark Volkmann, Object Computing, Inc.
Email: mark@ociweb.com
Twitter: @mark_volkmann
GitHub: mvolkmann
Website: http://ociweb.com/mark

http://ociweb.com/mark/MidwestJS/react.pdf

https://github.com/mvolkmann/react-examples

OCI | WE ARE SOFTWARE ENGINEERS.

Intro.

- Meaning behind the talk title - 2 kinds of complexity
  - other frameworks
  - state management approaches: thunks, sagas, epics, effects, GraphQL, Relay, Falcor, ...
- Why are the slides so dense?

What is OCI?

- **Software development** (on-site and off-site), **consulting**, and **training**
- Home of **Grails**,
  "An Open Source high-productivity framework
  for building fast and scalable web applications"
- Open Source Transformation Services
  - helping clients move from commercial to open source software
- Industrial Internet of Things (IIoT)
- DevOps

Overview ...

- Web app library from Facebook
  - http://facebook.github.io/react/
- Focuses on view portion
  - not full stack like other frameworks such as AngularJS and EmberJS
  - use other libraries for non-view functionality
    - some are listed later
- "One-way reactive data flow"
  - UI reacts to "state" changes
  - not two-way data binding like in AngularJS 1
    - what triggered a digest cycle?
    - should I manually trigger it?
  - easier to follow flow of data
    - events -> state changes -> component rendering

As of 8/6/16, **React** was reportedly **used by** Airbnb, Angie's List, Atlasssian, BBC, Capitol One, Clash of Clans, Codecademy, Coursera, Docker, Dropbox, Expedia, **Facebook**, Feedly, Flipboard, HipChat, IMDb, **Instagram**, Intuit, Khan Academy, Lyft, New York Times, NFL, NHL, **Netflix**, **Paypal**, Periscope, Reddit, Salesforce, Squarespace, Tesla Motors, **Twitter**, Uber, Visa, WhatsApp, Wired, Wolfrum Alpha, Wordpress, Yahoo, Zendesk, and many more.
Source: https://github.com/facebook/react/wiki/Sites-Using-React

… Overview

- Defines components that are composable
  - whole app can be one component that is built on others
- Components get data to render from "state" and/or "props"
- Can render in browser, on server, or both
  - ex. could only render on server for first page
    and all pages if user has disabled JavaScript in their browser
  - great article on this at https://24ways.org/2015/universal-react/
- Can render output other than DOM
  - ex. HTML5 Canvas, SVG, Android, iOS, …

use "React Native"
for Android and iOS

- Can use in existing web apps that use other frameworks
  - start at leaf nodes of UI and gradually work up,
    replacing existing UI with React components
- Supports IE9, Chrome, Firefox, Safari

ThoughtWorks Tech Radar 4/16

ADOPT

82. ES6
83. React.js
84. Spring Boot
85. Swift

TRIAL

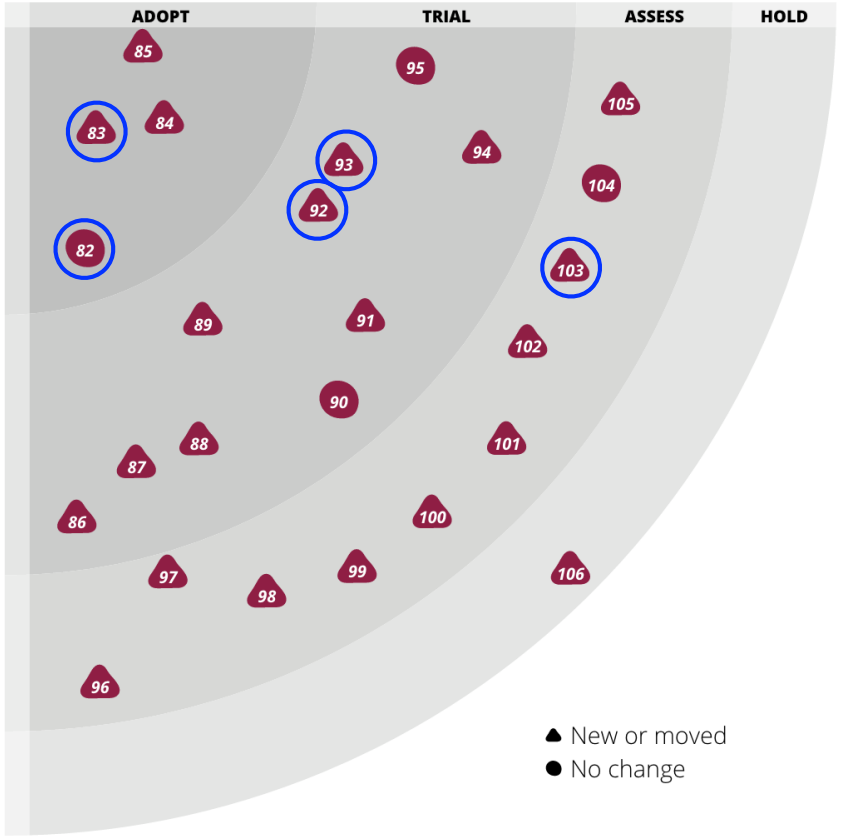
86. Butterknife new
87. Dagger new
88. Dapper new
89. Ember.js
90. Enlive
91. Fetch new
92. React Native
93. Redux new
94. Robolectric new
95. SignalR

ASSESS

96. Alamofire new
97. AngularJS
98. Aurelia new
99. Cylon.js new
100. Elixir
101. Elm
102. GraphQL new
103. Immutable.js new
104. OkHttp
105. Recharts new

demoted from TRIAL

We "have certainly seen codebases become overly complex from a combination of two-way binding and inconsistent state-management patterns."



ThoughtWorks Quotes

- In the avalanche of front-end JavaScript frameworks, **React.js stands out** ...
  - due to its design around a reactive data flow. Allowing only **one-way data binding greatly simplifies the rendering logic** and avoids many of the issues that commonly plague applications written with other frameworks. We're seeing the benefits of React.js on a growing number of projects, large and small, while at the same time we continue to be concerned about the state and the future of other popular frameworks like AngularJS. This has led to React.js becoming **our default choice for JavaScript frameworks**.
- **Redux is a great, mature tool** that has helped many of our teams
  - reframe how they think about **managing state in client-side apps**. Using a Flux-style approach, it enables a loosely coupled state-machine architecture that's **easy to reason about**. We've found it a good companion to some of our favored JavaScript frameworks, such as Ember and React.
- **Immutability** is often emphasized in the functional programming paradigm,
  - and most languages have the ability to create immutable objects, which cannot be changed once created. **Immutable.js** is a library for JavaScript that **provides many persistent immutable data structures, which are highly efficient** on modern JavaScript virtual machines. ... Our teams have had value using this library for tracking mutation and maintaining state, and it is a library we encourage developers to investigate, especially when it's combined with the rest of the Facebook stack.

Virtual DOM

- Secret sauce that makes React fast
- An in-memory representation of DOM
- Rendering steps
  1) create new version of virtual DOM (fast)
  2) diff that against previous virtual DOM (very fast)
  3) make minimum updates to actual DOM, only what changed
     (only slow if many changes are required)

from Pete Hunt, formerly on Instagram and Facebook React teams ...
"**Throwing out your whole UI and re-rendering it every time the data changes is normally prohibitively expensive, but with our fake DOM it's actually quite cheap.**
We can quickly diff the current state of the UI with the desired state and compute the minimal set of DOM mutations (which are quite expensive) to achieve it.
We can also **batch** together these mutations such that the UI is updated all at once in a single animation frame."

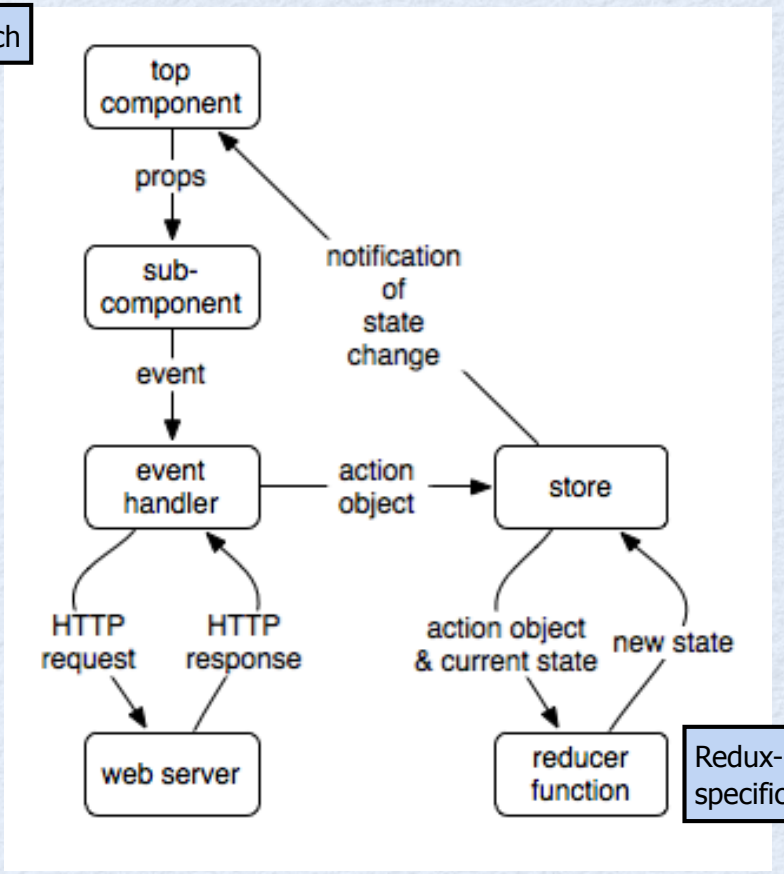
Client-side Model

- Three options for holding client-side data ("state") used by components

- 1) Every component holds its own state
  - not recommended; harder to manage
- 2) Only a few top-level components hold state
  - these pass data to sub-components via props
- 3) "Stores" hold state
  - with **Flux** architecture there can be multiple "stores"
  - with **Redux** there is one store

Simplified Thought Process

- What DOM should each component produce with given state and props?
  - use JSX to produce DOM
- When events occur in this DOM, what should happen?
  - dispatch an action or make an Ajax call? assuming Flux approach
- Ajax calls
  - what HTTP method and URL?
  - what data to pass? pass in query string or request body?
  - update a persistent store?
  - what data will be returned in response body?
  - dispatch an action, perhaps including data from Ajax call?
- Action processing
  - how should state be updated?
- Notification of state change
  - which components need to be re-rendered?
  - just an optimization; can re-render all from top



Related Libraries

- Use other libraries for non-view functionality
- **react-bootstrap** for styling and basic widgets
  - such as modal dialogs
  - also consider Material Design and Foundation
- **Fetch** or **axios** for Ajax
- **react-router** for routing
  - maps URLs to components that should be rendered
  - supports nested views
- **Immutable** for persistent data structures with structural sharing
  - important for holding app state
  - also from Facebook - https://facebook.github.io/immutable-js/
- **Redux** for data management
  - variation on **Flux** architecture
  - uses a single store to hold all state for app
  - uses **reducer functions** that take an action and the current state, and return the new state

version of Todo app using **Redux** and **Immutable** is at https://github.com/mvolkmann/react-examples/blob/master/todo-redux-rest

Flux architecture
component -> event -> action -> dispatcher -> stores -> components

Recommended Learning Order

- From Pete Hunt
  - "You don't need to learn all of these to be productive with React."
  - "Only move to the next step if you have a problem that needs to be solved."

1. **React** itself
2. **npm** - for installing JavaScript packages
3. JavaScript bundlers - like **webpack** supports use of ES6 modules
4. **ES6** (ES 2015)

5. routing - **react-router**
6. state management with Flux - **Redux** is preferred
7. immutable state - **Immutable** library is preferred
8. Ajax alternatives - Relay (uses GraphQL), Falcor, ... Currently I would skip this.

npm

- Node Package Manager
  - even though they say it isn't an acronym
- Each project/library is described by a **`package.json`** file
  - lists all dependencies (development and runtime)
  - can define scripts to be run using the "npm run" command
- To generate **`package.json`**
  - **`npm init`**
  - answer questions
- To install a package globally
  - **`npm install -g name`**
- To install a package locally and add dependency to **`package.json`**
  - for development dependencies, **`npm install --save-dev`** ***`name`*** or **`npm i -D`** ***`name`***
  - for runtime dependencies, **`npm install --save`** ***`name`*** or **`npm i -S`** ***`name`***

To find outdated dependencies, **`npm outdated`**

`package.json` Scripts

- Defined by **`scripts`** property object value
  - keys are script names
  - values are strings of shell commands to run

with some care, it's possible to write scripts that are compatible with both *nix and Windows

- Manually add script tasks
  - to do things like
    start a server,
    run a linter,
    run tests, or
    delete generated files
- To run a script, **`npm run name`**
  - can omit **`run`** keyword for special script names
- See example ahead

Special Script Names

- **`prepublish`**, **`publish`**, **`postpublish`**
- **`preinstall`**, **`install`**, **`postinstall`**
- **`preuninstall`**, **`uninstall`**, **`postuninstall`**
- **`preversion`**, **`version`**, **`postversion`**
- **`pretest`**, **`test`**, **`posttest`**
- **`prestart`**, **`start`**, **`poststart`**
- **`prestop`**, **`stop`**, **`poststop`**
- **`prerestart`**, **`restart`**, **`postrestart`**

React Setup

- Install React with **`npm install --save react react-dom`**
  - **`react-dom`** is used when render target is web browsers
- Can use **`browser.js`** to compile React code in browser at runtime, but not intended for production use
- Let's start serious and use **webpack**
  - details on next slide

webpack

- https://webpack.github.io
- Module bundler
  - combines all JavaScript files starting from "entry" by following imports
  - can also bundle CSS files references through imports
- Tool automation
  - through loaders
  - ex. ESLint, Babel, Sass, ...
- Install by running **`npm install --save-dev`** on each of these:
  - babel-core, babel-loader
  - eslint, eslint-loader, eslint-plugin-react
  - webpack, webpack-dev-server

requires configuration via **`.eslintrc`**

webpack-dev-server

- HTTP server for development environment
- Provides watch and hot reloading
- Bundles are generated in memory
  and served from memory for performance
- If another server must be used
  - for example, when REST services are implemented in Java and served from Tomcat
  - use **`webpack --watch`** and webpack-livereload-plugin
  - start from an npm script with
    `"start": "webpack --watch"`
  - see https://github.com/statianzo/webpack-livereload-plugin

webpack.config.js

- Create **`webpack.config.js`**
  - **`entry`** is main JavaScript file that imports others
  - use babel-loader to transpile ES6 code to ES5
  - use eslint-loader to check for issues in JavaScript files
  - use css-loader to resolve URL references in CSS files
  - use style-loader to provide hot reloading of CSS
- To generate **`bundle.js`** file
  - run **`webpack`** for non-minimized
  - run **`webpack -p`** for minimized (production)

```
module.exports = {
  entry: './src/demo.js',
  output: {
    path: __dirname,
    filename: 'build/bundle.js'
  },
  module: {
    loaders: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        loader: 'babel!eslint'
      },
      {
        test: /\.css$/,
        exclude: /node_modules/,
        loader: 'style!css'
      },
    ]
  }
};
```

webpack.config.js

"Loading CSS requires the css-loader and the style-loader. They have two different jobs. The css-loader will go through the CSS file and find url() expressions and resolve them. The style-loader will insert the raw css into a style tag on your page."

package.json

```
{
  "name": "my-project-name",
  "version": "1.0.0",
  "description": "my project description",
  "scripts": {
    "start": "webpack-dev-server --content-base . --inline"
  },
  "author": "my name",
  "license": "my license",
  "devDependencies": {
    "babel-core": "^6",
    "babel-eslint": "^5",
    "babel-loader": "^6",
    "babel-preset-es2015": "^6",
    "babel-preset-react": "^6",
    "css-loader": "^0",
    "eslint": "^2",
    "eslint-loader": "^1",
    "eslint-plugin-react": "^4",
    "style-loader": "^0",
    "webpack": "^1",
    "webpack-dev-server": "^1"
  },
  "dependencies": {
    "react": "^0",
    "react-dom": "^0"
  }
}
```

to start server and watch process, enter "**npm start**"

Simplest Possible Demo

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>React Simplest Demo</title>
  </head>
  <body>
    <div id="content"></div>
    <script src="build/bundle.js"></script>
  </body>
</html>
```

`build/bundle.js` is generated from `src/demo.js` by webpack

`build/bundle.js` isn't actually generated when using webpack-dev-server, it's all done in memory

src/demo.js

```
import React from 'react';
import ReactDOM from 'react-dom';

ReactDOM.render(
  <h1>Hello, World!</h1>,
  document.getElementById('content'));
```

JSX

can render into any element, and can render into more than one element

Do not render directly to document.body!
Browser plugins and other JS libraries sometimes add elements to body which can confuse React.

- Steps to run
  - `npm start`
    - assumes `package.json` configures this to start webpack-dev-server
  - browse `localhost:8080`

HTML in My JS?

- Yes, but ...
- Typically React apps have three primary kinds of JS files
  - component definition
  - event handling
  - state management (ex. Redux reducer functions)
- HTML only appears in JS files that define components
- Every line in those files is focused on deciding what to render
- So HTML is not out of place there ... same concern

JSX ...

- JavaScript XML
- Inserted directly into JavaScript code
  - can also use in TypeScript
- Very similar to HTML
- Babel finds this and converts it to calls to JavaScript functions that build DOM
- Many JavaScript editors and tools support JSX
  - **editors**: Atom, Brackets, emacs, Sublime, Vim, WebStorm, ...
  - **tools**: Babel, ESLint, JSHint, Gradle, Grunt, gulp, ...

from Pete Hunt ...
"We think that **template languages are underpowered** and are bad at creating complex UIs.
Furthermore, we feel that they are **not a meaningful implementation of separation of concerns** — markup and display logic both share the same concern, so why do we introduce artificial barriers between them?"

Great article on JSX from Corey House at http://bit.ly/2001RRy

... JSX ...

- Looks like HTML, but it isn't!
  - all tags must be terminated, following XML rules
  - insert JavaScript expressions by enclosing in braces - `{ js-expression }`

    not statements!
    ex. ternary instead of `if`
  - switch back to JSX mode with a tag
  - `class` attribute -> `className`
  - `label for` attribute -> `htmlFor`

    supposedly because `class` and `for` are reserved keywords in JavaScript

    Why?
  - camel-case all attributes: ex. `autofocus` -> `autoFocus` and `onclick` -> `onClick`
  - value of event handling attributes must be a function, not a call to a function
  - `style` attribute value must be a JavaScript object, not a CSS string
  - camel-case all CSS property names: ex. `font-size` -> `fontSize`
  - `<textarea>value</textarea>` -> `<textarea value="value"/>`
  - cannot use HTML/XML comments

    can use `{/* comment */}`

    Why?
  - HTML tags start lowercase; custom tags start uppercase

... JSX

- Repeated elements (ex. `li` and `tr`) require a `key` attribute
  - often an `Array` of elements to render is created using `map` and `filter` methods
  - `key` value must be unique within parent component
  - used in "reconciliation" process to determine whether a component needs to be re-rendered or can be discarded
  - will get warning in browser console if omitted
- Comparison to Angular
  - Angular provides **custom syntax** (provided directives and filters/pipes) used in **HTML**
  - React provides **JSX** used in **JavaScript**, a much more powerful language

Lifecycle Methods

Order of Invocation

- Mount (initial render)
  - `getDefaultProps`
  - `getInitialState`
  - `componentWillMount`
  - **`render`**
  - **`componentDidMount`**
- Prop Change
  - `componentWillReceiveProps`
  - **`shouldComponentUpdate`**
  - `componentWillUpdate`
  - **`render`**
  - **`componentDidUpdate`**
- Unmount
  - `componentWillUnmount`
- State Change
  - **`shouldComponentUpdate`**
  - `componentWillUpdate`
  - **`render`**
  - **`componentDidUpdate`**

Props

- JSX attributes create "props"
  - see "`name`" in next example

both standard HTML attributes and custom attributes

Reserved prop names

`dangerouslySetInnerHTML`, `children`, `key`, and `ref`

- Props specified on a JSX component can be accessed
  - **inside component methods** with `this.props` whose value is an object holding name/value pairs
  - **inside "functional components"** via props object argument to the function
  - often ES6 destructuring is used to extract specific properties from props object

see examples of these two forms of defining components ahead

- Used to pass read-only data and functions (ex. event handling callbacks) into a component
- To pass value of a variable or JavaScript expression, enclose in braces instead of quotes
  - will see in Todo example

Components

- Custom components can be referenced in JSX
  - names must start uppercase to distinguish from HTML elements
- Two kinds, smart and dumb
  - **smart components** have state and/or define lifecycle methods
  - **dumb components** get all their data from props
    and can be defined in a more concise way
    ("stateless functional component" form)
    - essentially only equivalent of `render` method; no "lifecycle methods"
- Want a minimal number of smart components at top of hierarchy
- Want most components to be dumb
- Defining each component in a separate `.js` file
  allows them to be imported where needed

Component Example

react-examples/component

src/greeting.js

```
import React from 'react';

class Greeting extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }
}

export default Greeting;
```

demonstrates using ES6 `class` syntax;
can also define by calling `React.createClass`

```
import React from 'react';

export default ({name}) =>
  <h1>Hello, {name}!</h1>;
```

stateless functional component form
"like a React class with only a `render` method"

props is passed and destructured

src/demo.js

```
import Greeting from './greeting';
import React from 'react';
import ReactDOM from 'react-dom';

ReactDOM.render(
  <Greeting name="Mark"/>,
  document.getElementById('content'));
```

must have this even though it is not directly referenced

Hello, Mark!

State

- Holds data for a component that may change over lifetime of component instance, unlike props which do not change for that component instance
  - the component may be re-rendered with different prop values
- To add/modify state properties, pass an object describing new state to `this.setState`
  - replaces values of specified properties and keeps others
  - performs a shallow merge
  - triggers DOM modifications
    - unless modified state properties aren't used by the component
- To access state data, use `this.state.name`
  - example: `const foo = this.state.foo;`
  - alternative using destructuring: `const {foo} = this.state;`
- Never directly modify `this.state`
  - can cause subtle bugs

two kinds of data, app data and UI data (ex. selected sort order and filtering applied)

Events

- HTML event handling attributes (like **`onclick`**) must be camel-cased in JSX (**`onClick`**)
- Set to a function reference, not a call to a function
  - three ways to use a component method
    1. arrow function; ex. `onClick={e => this.handleClick(e)}`
    2. function **`bind`**; ex. `onClick={this.handleClick.bind(this)}`
    3. pre-bind in constructor ← best option; with other options a different value is passed as the prop value in each render which makes rendering optimization more difficult
  - see **`onChange`** in example ahead

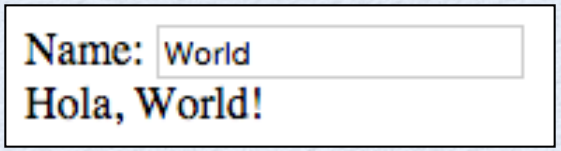
With **Redux** there is no need to use **`this`** in event handling methods, so **`bind`** isn't needed!

- Registers React-specific event handling on a DOM node
- The function is passed a React-specific event object where **`target`** property refers to React component where event occurred

State/Event Example ...

react-examples/events

- This example demonstrates an alternative to two-way data binding that is often shown in example AngularJS code




```
import Greeting from './greeting';
import React from 'react';
import ReactDOM from 'react-dom';

ReactDOM.render(
  <Greeting greet="Hola"/>,
  document.getElementById('content'));
```

src/demo.js

defined on next slide

assumes same HTML as on slide 20

… State/Event Example

src/greeting.js

```
import React from 'react';

class Greeting extends React.Component {
  constructor() {
    super(); // must call before accessing "this"
    this.state = {name: 'World'}; // initial state
    this.setName = this.setName.bind(this); // pre-bind
  }

  setName(event) {
    this.setState({name: event.target.value});
  }

  render() {
    return (
      <form>
        <div>
          <label>Name: </label>
          <input type="text" value={this.state.name}
            onChange={this.setName}/>
        </div>
        <div>
          {this.props.greet}, {this.state.name}!
        </div>
      </form>
    );
  }
}
```

optional **prop validation** that identifies JSX errors

```
const {string} = React.PropTypes;
Greeting.propTypes = {
  greet: string
};

Greeting.defaultProps = {
  greet: 'Hello'
};

export default Greeting;
```

Prop Validation ...

- Optional, but highly recommended to find JSX errors faster
- Not performed in production builds
- Specified via component **`propTypes`**
  - an object where keys are property names and values are validation specifications
  - defined by properties on **`React.PropTypes`**

```
MyComponent.propTypes = { ... };
```

- Example

```
const {func, object} = React.PropTypes;
Todo.propTypes = {
  todo: object.isRequired,
  onToggleDone: func.isRequired,
  onDeleteTodo: func.isRequired
};
```

... Prop Validation

- Validation options
  - primitive types: **`bool`**, **`number`**, **`string`**
  - function: **`func`**
  - DOM types: **`element`**, **`node`**
  - enums: **`oneOf`**, **`oneOfType`**
  - arrays: **`array`**, **`arrayOf`**
  - objects: **`object`**, **`objectOf`**, **`instanceOf`**, **`shape`**
  - custom: a function that takes **`props`**, **`propName`**, and **`componentName`**
    - useful for complex validation such as evaluating values of other properties
    - access value to be validated with **`props[propName]`**
    - return an **`Error`** object if validation fails; nothing otherwise
  - any type: **`any`**

`oneOf` specifies an array of allowed literal values

`oneOfType` specifies an array of validation options

`shape` specifies properties that must be present in an object, and their types (see example later)

only useful when type doesn't matter, but prop must be present

- Props are optional by default
  - add **`.isRequired`** at end of validation option to make required

Todo List App ...

react-examples/todo

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>React Todo App</title>
  </head>
  <body>
    <div id="content"></div>
    <script src="build/bundle.js"></script>
  </body>
</html>
```

todo.css

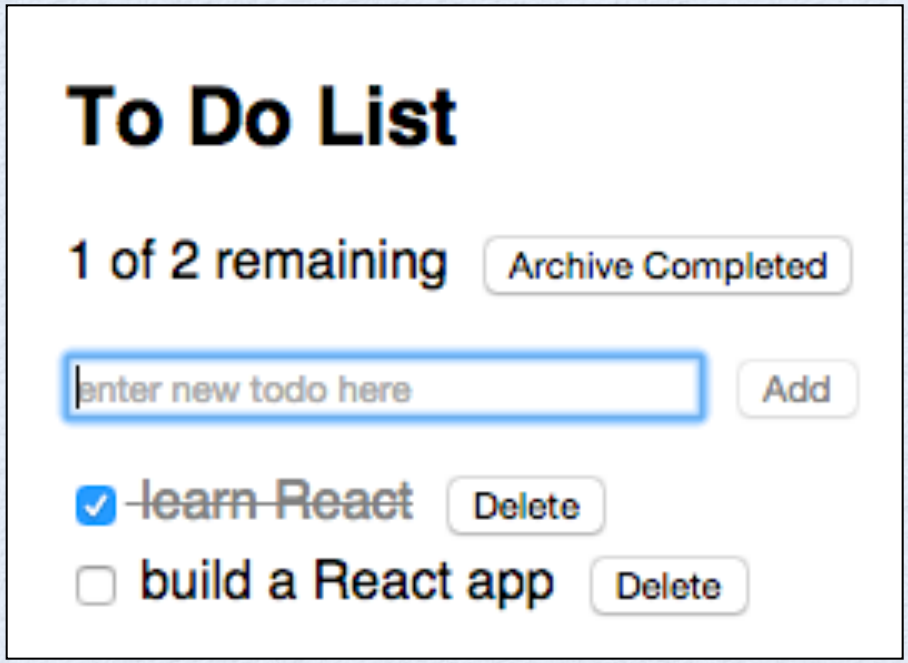
```
body {
  font-family: sans-serif;
  padding-left: 10px;
}

button {
  margin-left: 10px;
}

li {
  margin-top: 5px;
}

ul.unstyled {
  list-style: none;
  margin-left: 0;
  padding-left: 0;
}

.done-true {
  color: gray;
  text-decoration: line-through;
}
```



To run:
`npm start`
browse `localhost:8080`

... Todo List App ...

todo.js

```
import React from 'react';

// props is passed to this function and destructured.
const Todo = ({todo, onToggleDone, onDeleteTodo}) =>
  <li>
    <input type="checkbox"
      checked={todo.done}
      onChange={onToggleDone}/>
    <span className={'done-' + todo.done}>{todo.text}</span>
    <button onClick={onDeleteTodo}>Delete</button>
  </li>;

const {func, object} = React.PropTypes;
Todo.propTypes = {
  todo: object.isRequired,
  onToggleDone: func.isRequired,
  onDeleteTodo: func.isRequired
};

export default Todo;
```

a stateless functional component

event props specify a function reference, not a call to a function

Validating todo prop using shape

```
const {bool, func, shape, string} = React.PropTypes;
Todo.propTypes = {
  todo: shape({
    done: bool.isRequired,
    text: string.isRequired
  }).isRequired,
  onDeleteTodo: func.isRequired,
  onToggleDone: func.isRequired
};
```

... Todo List App ...

todo-list.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import Todo from './todo';
import './todo.css';

let lastId = 0;

class TodoList extends React.Component {
  constructor() {
    super(); // must call before accessing "this"

    this.state = {
      todoText: '', // must initialize
      todos: [
        TodoList.createTodo('learn React', true),
        TodoList.createTodo('build a React app')
      ]
    };

    // Pre-bind event handling methods.
    this.onAddTodo = this.onAddTodo.bind(this);
    this.onArchiveCompleted = this.onArchiveCompleted.bind(this);
    this.onTextChange = this.onTextChange.bind(this);
  }

  static createTodo(text, done = false) {
    return {id: ++lastId, text, done};
  }
```

... Todo List App ...

```
get uncompletedCount() {
  return this.state.todos.filter(t => !t.done).length;
}
```

todo-list.js

```
onAddTodo() {
  const newTodo = TodoList.createTodo(this.state.todoText);
  this.setState({
    todoText: '',
    todos: this.state.todos.concat(newTodo)
  });
}

onArchiveCompleted() {
  this.setState({
    todos: this.state.todos.filter(t => !t.done)
  });
}
```

just deleting in this simple version

... Todo List App ...

todo-list.js

```
onDeleteTodo(todoId) {
  this.setState({
    todos: this.state.todos.filter(t => t.id !== todoId)
  });
}

onTextChange(event) {
  this.setState({todoText: event.target.value});
}

onToggleDone(todo) {
  const id = todo.id;
  const todos = this.state.todos.map(t =>
    t.id === id ?
      {id, text: todo.text, done: !todo.done} :
      t);
  this.setState({todos});
}
```

Using **Immutable** would be good here because it can efficiently produce a new version of a `List` where an object at a given "key path" is updated.

... Todo List App

todo-list.js

```
  render() {
    const todos = this.state.todos.map(todo =>
      <Todo key={todo.id} todo={todo}
        onDeleteTodo={this.onDeleteTodo.bind(this, todo.id)}
        onToggleDone={this.onToggleDone.bind(this, todo)}/>);

    return (
      <div>
        <h2>To Do List</h2>
        <div>
          {this.uncompletedCount} of {this.state.todos.length} remaining
          <button onClick={this.onArchiveCompleted}>Archive Completed</button>
        </div>
        <br/>
        <form>
          <input type="text" size="30" autoFocus
            placeholder="enter new todo here"
            value={this.state.todoText}
            onChange={this.onTextChange}/>
          <button disabled={!this.state.todoText}
            onClick={this.onAddTodo}>Add</button>
        </form>
        <ul className="unstyled">{todos}</ul>
      </div>
    );
  }
}

ReactDOM.render(<TodoList/>, document.getElementById('content'));
```

`Array` `map` method is often used to create a collection of DOM elements from an array

can use any JavaScript to create DOM, not just a custom syntax like in templating languages or Angular

not 2-way binding

Wrapping this in a `form` causes the button to be activated when input has focus and return key is pressed.

create-react-app

- Tool that creates a great starting point for new React apps
- Installs and configures many tools and libraries
  - Babel, ESLint, Immutable.js, lodash, React, react-dom,
    webpack (including webpack-dev-server, html-webpack-plugin, css-loader, and style-loader),
    whatwg-fetch, and more
- Provides watch and live reload
- Steps to use
  - **`npm install -g create-react-app`**
  - **`create-react-app`** ***`my-app-name`*** creates and populates directory; installs all dependencies
  - **`cd`** ***`my-app-name`***
  - **`npm start`** starts local server and loads app in default browser
- Configuration is in node_modules/react-scripts
  - see "scripts" property near bottom of **`package.json`**
- For more information, see https://github.com/facebookincubator/create-react-app

Biggest Issues

- For large apps, need to choose a way to efficiently modify state
  - **Immutable** library from Facebook is a good choice
- Often need to use `Function bind` for event handlers
  - not needed when a Flux library is used
- Cannot use external HTML files
  - must specify DOM in JavaScript, typically using JSX
- JSX is like HTML, but it's not
  - it seems there could be fewer differences

Biggest Benefits

- Emphasizes using JavaScript rather than custom template syntax to build views
  - ex. JavaScript `if` and ternary operator versus `ng-if`, `ng-show`, and `ng-hide`
  - ex. JavaScript `Array` `map` method versus `ng-repeat`
- Easier to create custom React components than to create Angular directives
  - just need a `render` method or stateless functional component
- Fast due to use of virtual DOM and DOM diffing
- One way data flow makes it easier to understand and test components
  - most components only use data that is directly passed to them via props
- Very easy to write component tests
- Can use same approach for rendering to DOM, Canvas, SVG, Android, iOS, ...

comparing to Angular 1

Big Questions

- Is it easier to learn and use React
  than other options?
- Should my team use React
  in a new, small project
  to determine if it is a good fit for us?

The End

- Thanks so much for attending my talk!
- Feel free to find me later and ask questions about React or anything in the JavaScript world
- **Contact me**

Mark Volkmann, Object Computing, Inc.

Email: mark@ociweb.com

Twitter: @mark_volkmann

GitHub: mvolkmann

Website: http://ociweb.com/mark