Node Program

Lesson 1: React Basics

React.js version: 15
Last updated: Nov 2016

© Node Program, 2016

Quick Intro

© Node Program, 2016

The Definition

What is React.js?

It's not about templates, or data binding, or DOM manipulation. It's about using functional programming with a virtual DOM representation to build ambitious, high-performance apps with JavaScript.

Pete Hunt http://bit.ly/1U53Rb2

© Node Program, 2016

Key Differences

- Virtual DOM
- Declarative (not imperative)
- Functional
- No DOM manipulation
- No templates
- No event listeners or handlers

© Node Program, 2016

MVC

React.js is not a model-view-controller (MVC) framework/ library.

© Node Program, 2016

You need to bring your own models and routers.

React.js is only VIEW.

© Node Program, 2016

React.js can work with other MVC-like framework such as Backbone.js and Angular.js.

© Node Program, 2016

React.js is often used with Flux and Meteor.

© Node Program, 2016

Hello World

React.js CDN

Or hotlink to Facebook CDN:

```
<script src="https://unpkg.com/react@15/dist/react.js"></script>
<script src="https://unpkg.com/react-dom@15/dist/react-dom.js"></script>
```

© Node Program, 2016

HTML Structure

The `div` in the body of `hello-world.html`:

```
  <body>
    <div id="example"></div>
    <script type="text/javascript">
    ... // React.js code
    </script>
  </body>
</html>
```

© Node Program, 2016

H1 Element

The following snippet creates the h1 React.js object with content 'Hello world!':

```
React.createElement('h1', null, 'Hello world!')
```

© Node Program, 2016

H1 Element

The following snippet creates the h1 React.js object with content 'Hello world!':

```
React.DOM.h1(null, 'Hello world!')
```

© Node Program, 2016

Rendering

Once the element is created we render it to the DOM element with ID example and renders it:

```
ReactDOM.render(
  React.createElement('h1', null, 'Hello world!'),
  document.getElementById('example')
)
```

© Node Program, 2016

Running The Page

Open `hello-world.html` and check if you see Hello world!

© Node Program, 2016

Inspecting HTML

If we inspect the HTML generated by React.js, it will have this attribute:

```
<h1>Hello world!</h1>
```

HTML Arguments

Virtually all HTML arguments are supported so you can pass them like this:

```
React.createElement('div', {style: {color: red}}, 'Hello ', 'world!')
```

Note: You can add many parameters at the end to combine them.

© Node Program, 2016

for and class Attributes

If you need to use `for` or `class` attributes, their names are `forHtml` and `className`. For example,

```
React.createElement('div', {className: 'hide'}, 'Hello world!')
```

© Node Program, 2016

Demo

Hello World without JSX

© Node Program, 2016

Meet JSX

What is JSX

JSX is a combination of JavaScript and XML. HTML is a form of XML:

```
ReactDOM.render(
  <h1>Hello world!</h1>,
  document.getElementById('example')
)
```

© Node Program, 2016

What is JSX (Cont.)

JSX is compiled into native/regular JavaScript. JSX allows for easier and faster writing HTML views and elements along with JavaScript

https://jsx.github.io/

© Node Program, 2016

Why Use JSX

The mix of XML and JS looks weird, but its the recommended way of writing React.js apps because it provides syntax for components, layouts and hierarchy.

Note: As you've seen from the previous Hello World! example, JSX is optional.

© Node Program, 2016

Ways to use JSX

1. Pre-process with `babel-cli`: production recommended
2. Build with Gulp, Grunt, Webpack and Babel: production recommended
3. Run-time via `babel-core`: development only

© Node Program, 2016

Let's use JSX, Webpack, Babel and npm to make it close to real web development flow

© Node Program, 2016

Real Dev Setup

- Babel
- Babel React presets
- Webpack
- npm scripts

© Node Program, 2016

Why Use Webpack?

© Node Program, 2016

What Webpack Will do for You

1. Bundle JS, CSS, etc.
2. Minification
3. Dependency management
4. React/JSX transpilation

© Node Program, 2016

Babel

The JavaScript compiler.

http://babeljs.io/

Compiling JSX with Babel

For more realistic and production-like example, we'll use Babel. This will allow us to compile JSX into native JS and run only native JS in the browser. This will increase performance in production React.js apps.

© Node Program, 2016

What is Babel

Babel allows you to use ECMAScript 6 now by compiling ES6 code into ES5-friendly code to support browsers that don't have ES6 yet.

https://babeljs.io/

New Project with npm

```
$ mkdir react-project
$ cd react-project
$ npm init
```

© Node Program, 2016

Install the React Deps!

```
$ npm i react@15 react-dom@15 -SE
```

© Node Program, 2016

Install the Babel Deps!

```
$ npm i babel-core@6 babel-loader@6 babel-preset-react@6 -SE
```

© Node Program, 2016

Dependencies in `package.json`

```
"devDependencies": {
  "babel-core": "6.13.2",
  "babel-loader": "6.2.4",
  "babel-preset-react": "6.5.0",
  "react": "15.2.1",
  "react-dom": "15.2.1",
  "webpack": "1.13.3"
}
```

© Node Program, 2016

`package.json` for Babel

```
"babel": {
  "presets": ["react"]
},
```

© Node Program, 2016

Scripts

```
"scripts": {
  "build": "./node_modules/.bin/webpack",
  "build-watch": "./node_modules/.bin/webpack -w"
},
```

© Node Program, 2016

Webpack Config

`webpack.config.js`

webpack.config.js:

```
module.exports = {
  entry: './jsx/app.jsx',
  output: {
    path: __dirname + '/js/',
    filename: 'bundle.js'
  },
```

© Node Program, 2016

webpack.config.js:

```
  module: {
    loaders: [
      {
        test: /\.jsx?$/,
        exclude: /(node_modules)/,
        loaders: ['babel']
      }
    ]
  }
}
```

© Node Program, 2016

Test Setup

1. `index.html` with `<script src="js/bundle.js"></script>`
2. script with `console.log('start')` in `jsx/app.jsx`
3. Run build with `npm run build` (not `npm build`)

`code/react/react-project`

© Node Program, 2016

JSX Code

Change `React.createElement` to `<h1>...</h1>`:

```
ReactDOM.render(
  <h1>Hello world!</h1>,
  document.getElementById('example')
)
```

© Node Program, 2016

Running the Code

Open `hello-world-jsx.html` and check if you see Hello world!

© Node Program, 2016

Composable Components

The concept of components is the foundation of React.js philosophy. They allow you to reuse code and logic. They are like templates only better.

© Node Program, 2016

Types of React.js Components

React.js component types:

» Regular HTML elements such as h1, p, div, etc.

» Custom or composable components

© Node Program, 2016

Difference Between Regular and Custom Components

If it's a regular HTML tag name, then React.js will create such element. Otherwise, it will look for the custom component definition.

Note: React.js uses lower-case vs. upper case to distinguish between HTML tags and components.

© Node Program, 2016

Defining a Component

Composable components are created with `React.createClass` and must have `render` method that returns regular component (div, h1, etc.):

```
const React = require('react')

class HelloWorld extends React.Component {
  render() {
    return <h1>Hello world!</h1>
  }
}
```

© Node Program, 2016

Sidenote `createClass()`

old style:

```
var React = require('react')

var HelloWorld = React.createClass({
  render: function() {
    return (
      <h1>Hello world!!!</h1>
    )
  }
})
```

© Node Program, 2016

Refactoring with a HelloWorld Component

The `hello-world-component.jsx` file has a custom component

```
const React = require('react')
const ReactDOM = require('react-dom')

class HelloWorld extends React.Component {
  render() {
    return (
      <h1>Hello world!!!</h1>
    )
  }
}

ReactDOM.render(
  <HelloWorld/>,
  document.getElementById('example')
)
```

© Node Program, 2016

HTML Skeleton

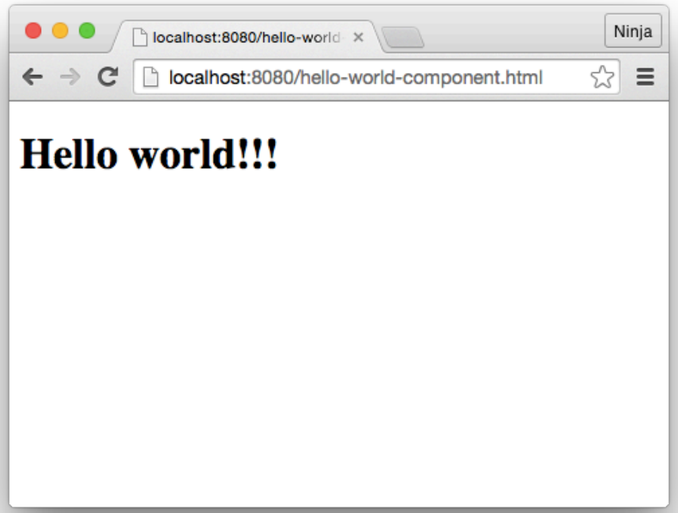
Point your `hello-world-component.html` to use `hello-world-component.js`, not `hello-world-component.jsx`:

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <div id="example"></div>
    <script src="hello-world-component.js"></script>
  </body>
</html>
```

© Node Program, 2016

Running the Code

Open `hello-world-component.html` and check if you see Hello world!



© Node Program, 2016

Auto Update

© Node Program, 2016

Nested Elements

Nesting React.js components is easy.

© Node Program, 2016

Rendering Title and Text

This is how we can nest `<h1>` and `<p>` inside of `<div>`:

```
ReactDOM.render(
  <div>
    <h1>
      Core React.js
    </h1>
    <p>This text is very useful for learning React.js.</p>
  </div>,
  document.getElementById('example')
)
```

© Node Program, 2016

Single Top-Level Tag

Remember to always have only one element as the top level tag! For example, this is a **no go**:

```
ReactDOM.render(
  <h1>
    Core React.js
  </h1>
  <p>This text is very useful for learning React.js.</p>
  ,
  document.getElementById('content')
)
```

© Node Program, 2016

Obviously, we can create nested structures in custom components:

```
const Content extends React.Component {
  render() {
    return (
      <div>
        <h1>
          Core React.js
        </h1>
        <p>This text is very useful for learning React.js.</p>
      </div>
    )
  }
}
```

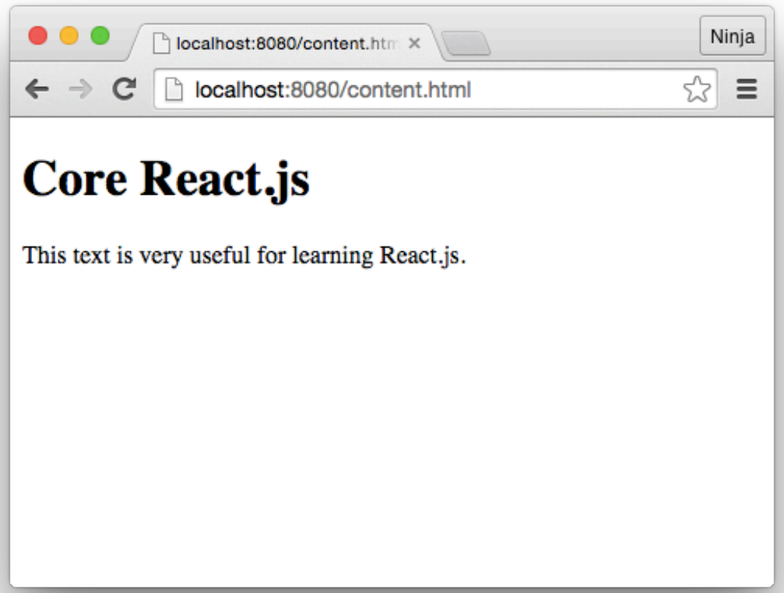
© Node Program, 2016

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

© Node Program, 2016

Running the Code

Open `content.html` and check if you see title and text!



© Node Program, 2016

Order of the Code

Remember that the content element (`<div id="content"></div>`) must precede the React.js code (`<script ...`), for the `getElementById` method to locate the proper DOM element:

```
...
    <div id="content"></div>
    <script type="text/jsx">
      var Content = React.createClass({
      ...
      })
      ReactDOM.render(
        <Content />,
        document.getElementById('content')
      )
    </script>
...
```

© Node Program, 2016

Variables

Use {} to render variable inside of JSX:

```
{a}
{' '}
{b}
```

Variable Example

In the `variable/script.jsx` file, we output the value of `a`:

```
class Content extends React.Component {
  render() {
    let a = 1
    return (
      <div>
        <h1>
          {a}. Core React.js
        </h1>
        <p>This text is very useful for learning React.js.</p>
      </div>
    )
  }
}
```

© Node Program, 2016

States

States are mutable properties of components meaning they can change. When state changes the corresponding view changes, but everything else in DOM remains intact.

Initial State

```
class Content extends React.Component {
  constructor(props){
    super(props)
    this.state = {a: 0}
  },
  render() {
    // ...
  }
}
```

© Node Program, 2016

Sidenote: ES5

The initial state is set by the `getInitialState` method which is called once when the element is created.

Let's use this method to return `a`:

```
var Content = React.createClass({
  getInitialState: function(){
    return {a: 0}
  },
  render: function() {
    // ...
  }
})
```

© Node Program, 2016

Updating State

State is updated with `this.setState()`, so this code will update the value with a random number every 300 milliseconds:

```
class Content extends React.Component {
  constructor(props){
    super(props)
    this.state = {a: 0}
    setInterval(()=>{
      this.setState({a: Math.random()})
    }, 300)
  }
  render() {
    // ...
  }
}
```

© Node Program, 2016

Outputting The State

To output the state property `a`, we use `{this.state.a}`:

```
  render() {
    return (
      <div>
        <h1>Changing the State</h1>
        <p>This value is random: {this.state.a}</p>
      </div>
    )
  }
}
```

© Node Program, 2016

Rendering

The rendering didn't change:

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

http://plnkr.co/edit/S2gjlc?p=preview

© Node Program, 2016

Component Methods

© Node Program, 2016

Calling Methods

It's possible to invoke components methods from the {} interpolation:

```
class Content extends React.Component {
  getA() {
    return 10
  }
  render() {
    return (
      <div>
        <p>This value is return by the method: {this.getA()} </p>
      </div>
    )
  }
}
```

© Node Program, 2016

Component Events

© Node Program, 2016

Events

Components have normalized (cross-browser) events such as

`onClick onContextMenu onDoubleClick onDrag onDragEnd onDragEnter onDragExit onDragLeave onDragOver onDragStart onDrop onMouseDown onMouseEnter onMouseLeave onMouseMove onMouseOut onMouseOver onMouseUp`

© Node Program, 2016

Declaring Events

React.js is declarative, not imperative. So we won't attach event like we would do with jQuery, instead we declare them in the JSX and classes:

```
class Content extends React.Component {
  constructor(props) {
    super(props)
    this.state = {counter: 0}
    this.click = this.click.bind(this)
  }
  click(event) {
    this.setState({counter: ++this.state.counter})
  }
  render() {
    // ...
  }
}
```

© Node Program, 2016

Button onClick Event

The button has the `onClick={this.click}`.

© Node Program, 2016

The name must match the method of the `Content` component class:

```
...
  render() {
    return (
      <div>
        <button onClick={this.click}>Don't click me {this.state.counter} times!</button>
      </div>
    )
  }
})
```

http://plnkr.co/edit/owbmK9?p=preview

© Node Program, 2016

Props

Props or properties are immutable meaning they don't change. They are passed by parent components to their children.

© Node Program, 2016

Using Props

```
class ClickCounterButton extends React.Component {
  render() {
    return <button onClick={this.props.handler}>Don't click me {this.props.counter} times! </button>
  }
}
```

© Node Program, 2016

Supplying Props

Provide props to the ClickCounterButton component:

```
class Content extends React.Component {
  constructor(props) {
    super(props)
    this.state = {counter: 0}
    this.click = this.click.bind(this)
  }
  click(event) {
    this.setState({counter: ++this.state.counter})
  }
  render() {
    return (
      <div>
        <ClickCounterButton counter={this.state.counter} handler={this.click}/>
      </div>
    )
  }
}
```

http://plnkr.co/edit/3HqvdG?p=preview

© Node Program, 2016

Where to Put logic

In this example, click event handler was in the parent element. You can put the event handler on the child itself, but using parent allows you to exchange info between children components.

Let's have a button:

```
class ClickCounterButton extends React.Component {
  render() {
    return <button onClick={this.props.handler}>Don't click me! </button>
  }
}
```

© Node Program, 2016

Exchanging Props Between Children

This is a new component which displays value prop:

```
class Counter extends React.Component {
  render() {
    return <span>Clicked {this.props.value} times.</span>
  }
}
```

© Node Program, 2016

Parent Component

The parent component provides props one of which is a handler:

```
class Content extends React.Component {
  constructor(props) {
    super(props)
    this.state = {counter: 0}
    this.click = this.click.bind(this)
  }
  click(event) {
    this.setState({counter: ++this.state.counter})
  }
  render() {
    return (
      <div>
        <ClickCounterButton handler={this.click}/>
        <br/>
        <Counter value={this.state.counter}/>
      </div>
    )
  }
}
```

© Node Program, 2016

Summary

Summary

- You don't need JSX to work with React.js, but its the recommended syntax for React.js components. JSXTransformer for run-time JSX (development only).
- React.js can be installed via multiple sources: npm, website, and CDN

© Node Program, 2016

Summary (Cont.)

- You create React.js elements with `<...>` or `React.createElement()` and render them with `ReactDOM.render()`
- States are mutable, and props are immutable
- Using Babel can watch for file changes with `-w` flag
- Regular vs. custom components: lower-case first letter

© Node Program, 2016

Summary (Cont.)

» `class NAME extends React.Component` allows to create custom components

» `class NAME extends React.Component` needs `render` method that return other React.js component (always one).

© Node Program, 2016

Summary (Cont.)

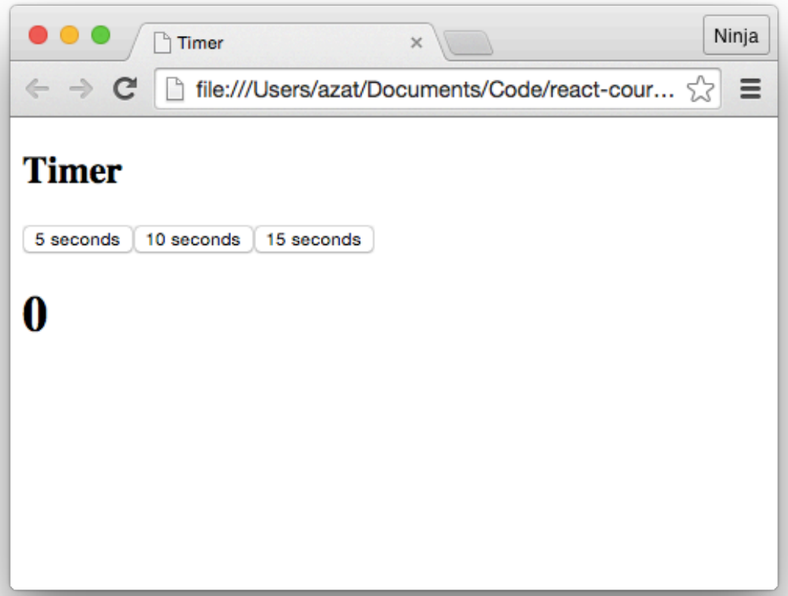
- for and class are forHtml and className attributes in React.js components
- `{}` is a way to render variables and JS in the JSX code
- `this.state.NAME` and `this.props.NAME` are ways to access state and props variables respectively

© Node Program, 2016

Questions and Exercises

😕❓➡️😸

Demo: Timer



`code/react/timer`

© Node Program, 2016

Workshop: Timer

» 3+ buttons

» 3+ components: 2 presentational and one smart

» Webpack, JSX, Babel, modules, npm

» Static (`node-static`) is not included

`code/react/timer`

© Node Program, 2016

Workshop: Timer

1. Move `Timer` and `Button` to separate files
2. Create a pause and resume buttons (could be one button as a toggle)
3. Create a reset button (separate component)
4. Create a custom timer with an input field (separate component)
5. Change to minutes, not seconds

© Node Program, 2016