React:

Session 1

May 15th 2021

Introduction to Client Side Rendering

JavaScript

- First release: December 4, 1995; 25 years ago
- Brendan Eich of Netscape initially

- **ECMAScript** is a general-purpose programming language, standardised by Ecma International according to the document ECMA-262. It is a JavaScript standard meant to ensure the interoperability of web pages across different web browsers

- ES6 is also called ECMAScript 2015

STOP DOING JAVASCRIPT

- DOCUMENTS WERE NOT SUPPOSED TO BE TURING-COMPLETE
- YEARS OF WEB TECHNOLOGY yet NO REAL-WORLD USE FOUND for doing more than HTML FORMS
- Wanted to do more anyway for a laugh? We had a tool for that: It was called “ADOBE FLASH”
- “Yes please connect the Redux thunk to a Suspense. Please instantiate WebAssembly streaming.” - Statements dreamed up by the utterly Deranged

LOOK at what Web Developers have been demanding your Respect for all this time with all the computers & compilers we built for them

(This is REAL Javascript, done by REAL Web Developers)

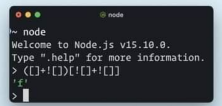
slack

?????

node_modules

807 items

???????



??????????????????

“Hello I would like **undefined** apples please”

They have played us for absolute fools

```
> typeof NaN
< "number"
> 9999999999999999
< 10000000000000000
> 0.5+0.1==0.6
< true
> 0.1+0.2==0.3
< false
> Math.max()
< -Infinity
> Math.min()
< Infinity
> []+[]
< ""
> []+{}
< "[object Object]"
> {}+[]
< 0
> true+true+true===3
< true
> true-true
< 0
> true==1
< true
> true===1
< false
> (!+[]+[]+![]).length
< 9
> 9+"1"
< "91"
> 91-"1"
< 90
> []==0
< true
```

Client Side JS Frameworks

- Angular
- AngularJS
- VueJS
- EmberJS

- ReactJS library

React

Initial release: May 29, 2013; 7 years ago
Stable release: March 22, 2021; 54 days ago

- React.js is
  a. declarative
  b. simple
  c. component based
  d. extensive
  e. fast
  f. easy to learn

- React.js supports server side rendering

Virtual DOM

When you render a JSX element, every single virtual DOM object gets updated

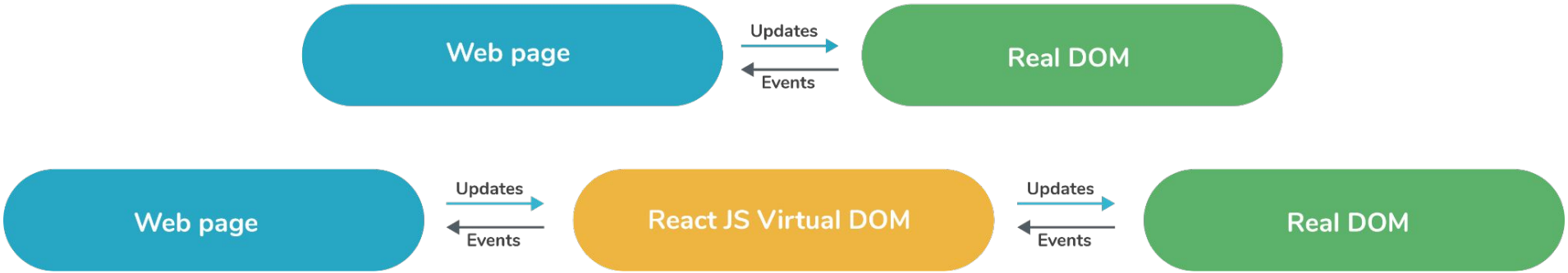
This sounds incredibly inefficient, but the cost is insignificant because the virtual DOM can update so quickly

Once the virtual DOM has updated, then React compares the virtual DOM with a virtual DOM *snapshot* that was taken right before the update

By comparing the new virtual DOM with a pre-update version, React figures out *exactly which virtual DOM objects have changed.* This process is called "**diffing**."

Once React knows which virtual DOM objects have changed, then React updates those objects, *and only those objects,* on the real DOM

Real & Virtual DOMs



- JSX, or JavaScript XML, is an extension to the JavaScript language syntax
- Similar in appearance to HTML, JSX provides a way to structure component rendering using syntax familiar to many developers
- React components are typically written using JSX, although they do not have to be (components may also be written in pure JavaScript)

JSX example

HTML-like means its outward appearance is similar to HTML markup. Here's a line of JSX:

```
<div id="foo">Hello world</div>
```

There is no visual difference between that and a similar HTML code, but this is still JavaScript. It translates to:

```
React.createElement("div", {id: "foo"}, "Hello world");
```

JSX/JS: javascript -> vdom

Template: string -> javascript -> vdom

Adding custom styling to an element uses different syntax

JSX uses JS objects to style the element and the CSS property containing dashes are converted into camel Case when styling

HTML → <input style='border: 1px solid black; border-radius: 4px'/>
JSX → <input style={{border: '1px solid black', borderRadius: '4px'}}/>

Attribute names are different in JSX

HTML uses **class** and **for** attributes whereas JSX uses **className** and **htmlFor** attributes to not conflict with existing JS keywords

HTML → `<button class="foo" for="bar">`

JSX → `<button className="foo" htmlFor="bar"/>`

JS variables can be directly used in JSX

JSX → {myVariable}

For HTML, this vanilla javascript should be run within the <script> of the HTML:

document.querySelector('#root').textContent = myVariable;

Terminology

<></>

shorthand for React.Fragment

Toolchain

Webpack (gathers code into a single bundle and listens for file changes to reload this bundle to show the updated code)

Babel (converts ES6 and JSX to plain JavaScript)

npx itself is a Node tool which allows you to run a package with Create React App, which allows you to easily start a new React project

The server that you see is simply to allow for the reloading of the app in response to file changes in real time

The server is only for use in development

node vs npm vs npx vs nvm

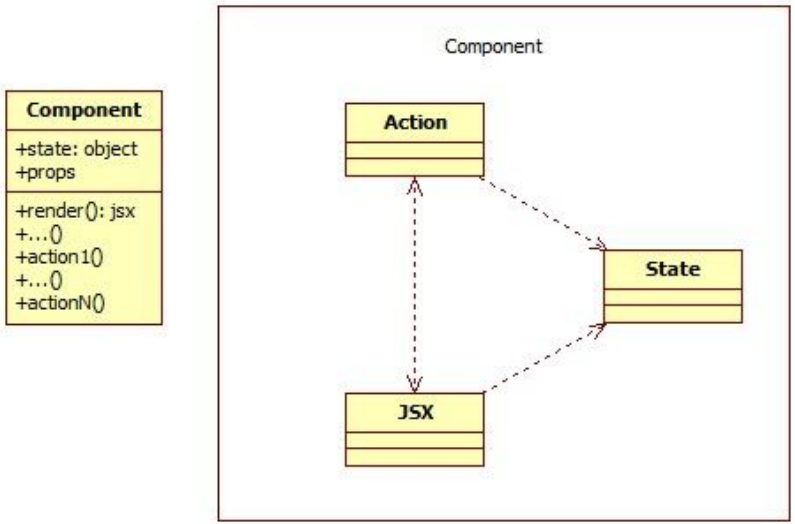
Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine.

npm is a package manager for the JavaScript programming language

The **npx** stands for Node Package Execute and it comes with the npm, when you installed npm above 5.2.0 version then automatically npx will installed

nvm is a version manager for node.js, designed to be installed per-user, and invoked per-shell

React Architecture



React Architecture

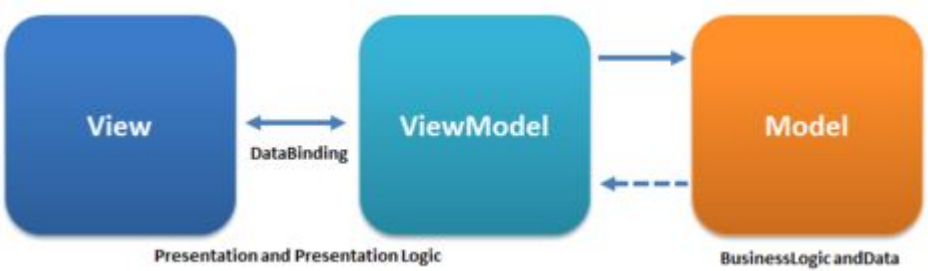
- Actions (serve a role of Controllers in MVC). These are javascript functions that react to the interactions originating mainly from the JSX (view). At the end of its execution, the action typically dispatches (by calling setState()) a state (model) to which the JSX can bind later in the rendering lifecycle. Everything matches with the responsibilities of a Controller from MVC pattern except the fact that sometimes you know too much about the view. e.g., to process the entire form data in action (such as after submitting a form), you will need to get hold of the HTML form element and retrieve values from the fields in it. In an ideal MVC implementation, you should be receiving a plain JSON object containing values from all form fields. Another sad example is having to receive an event as an action argument when you need to do custom stuff such as prevent the default behavior of propagation. Such little things break the MVC pattern, and therefore the React becomes just an attempt for it

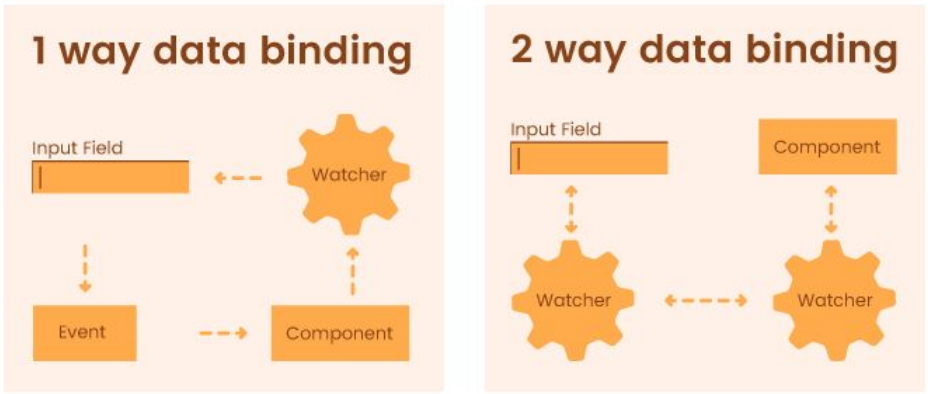
- State (serves a role of Model in MVC) gets dispatched by the action (Controller), and JSX (View) binds to it. This part fits well within the MVC pattern

- JSX (serves a role of View in MVC) binds to the state (Model) received from an action (Controller). Furthermore, it can invoke a controller per user interaction by explicitly calling it. Therefore, in React, View is aware of the Controller, which again follows the MVC pattern. If only the controller could know less about the view - that would avoid the circular dependencies in the architecture, but probably in another version or another life

- Component, while it helps to modularize the overall React applications, also comes with an architectural smell: It forces you to put actions, state, and JSX (all three elements of the MVC pattern) together. Furthermore, these three elements always become tightly coupled with each other, making it impossible to develop or unit-test them separately (e.g., with traditional MVC, we would want to unit test Controllers separately). By the way, speaking of unit tests, the official React website does not even explain how to unit test the component - maybe that is because it's not so easy to achieve? Either way, the component comes with the described architectural anti-pattern, and unfortunately, there is no easy way out of it.

Model-View-ViewModel pattern





Flux pattern is sometimes expressed as "properties flow down, actions flow up"