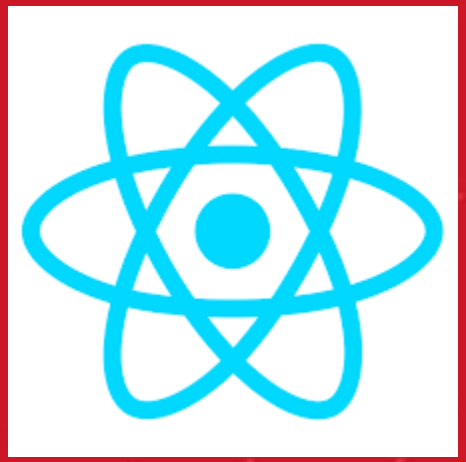
ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

ReactJs



Content

Overview of ReactJS
Virtual DOM
JSX
Components
React Component Lifecycle

Overview of ReactJS

Overview of ReactJS

- React(also known as React.js or ReactJS) is an open-source, front end, JavaScript library for building user interfaces or UI components
  - It maintained by Facebook and a community of individual developers and companies

- React was created by Jordan Walke, a software engineer at Facebook, who released an early prototype of React called "FaxJS“. It was first deployed on Facebook's News Feed in 2011 and later on Instagram in 2012.
  - On September 26, 2017, React 16.0 was released to the public.
  - On February 16, 2019, React 16.8 was released to the public.
  - On August 10, 2020, the React team announced the first release candidate for React v17.0, notable as the first major release without major changes to the React developer-facing API

Virtual DOM

Virtual DOM

What is DOM?

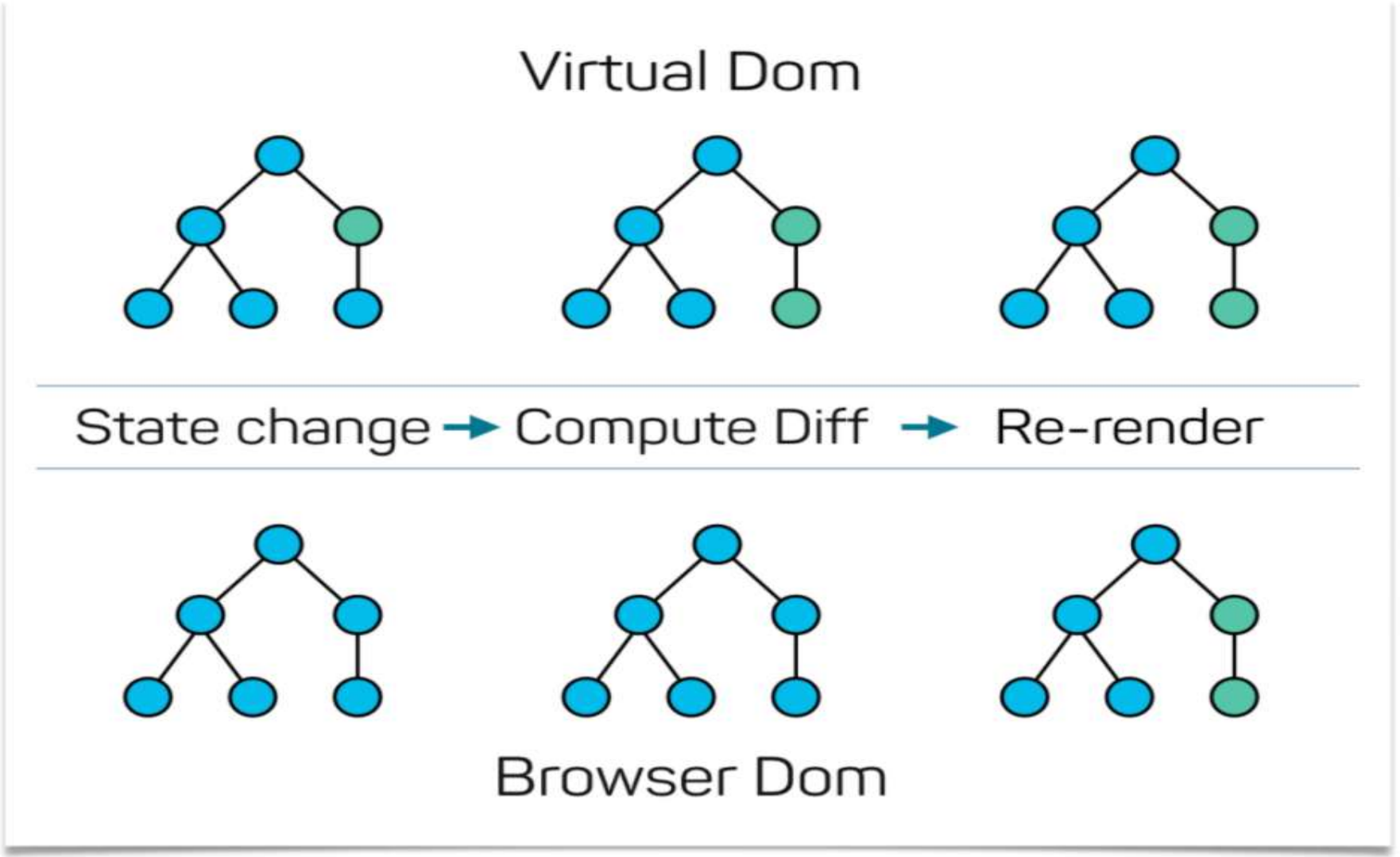
- The Document Object Model (DOM) is a cross-platform and language-independent interface that treats an XML or HTML document as a tree structure wherein each node is an object representing a part of the document.
  - The DOM represents a document with a logical tree.
  - Each branch of the tree ends in a node, and each node contains objects. DOM methods allow programmatic access to the tree; with them one can change the structure, style or content of a document.

$\Rightarrow$In React, Virtual DOM exists which is like a lightweight copy of the actual DOM

- every object that exists in the original DOM there is an object for that in React Virtual DOM
- Manipulating DOM is slow, but manipulating Virtual DOM is fast as nothing gets drawn on the screen.

How Virtual DOM helps React?

- Each time we change something in our JSX file, all the objects that are there in the virtual DOM gets updated.
- React maintains two Virtual DOM at each time, one contains the updated Virtual DOM and one which is just the pre-update version of this updated Virtual DOM.
- Now it compares the pre-update version with the updated Virtual DOM and figures out what exactly has changed in the DOM. This process is known as ‘diffing’
- Once React finds out what exactly has changed then it updated those objects only, on real DOM. This significantly improves the performance and is the main reason why Virtual DOM is much loved by developers all around.


Virtual Dom
State change → Compute Diff → Re-render
Browser Dom

JSX

JSX

- JSX = Javascript + XML
- 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 uses JSX for templating instead of regular JavaScript => but, It is not necessary to use it
- however, following are some pros that come with it.
  - It is faster because it performs optimization while compiling code to JavaScript.
  - It is also type-safe and most of the errors can be caught during compilation.
  - It makes it easier and faster to write templates, if you are familiar with HTML.

JSX

```
import React from 'react';

class App extends React.Component {
   render() {
      return (
         <div>
            Hello World!!!
         </div>
      );
   }
}
export default App;
```

JSX looks like a regular HTML in most cases.

JSX - Nested Elements

```
import React from 'react';

class App extends React.Component {
   render() {
      return (
         <div>
            <h1>Header</h1>
            <h2>Content</h2>
            <p>This is the content!!!</p>
         </div>
      );
   }
}
export default App;
```

If we want to return more elements, we need to wrap it with one container element.

JSX - Attributes

```
import React from 'react';

class App extends React.Component {
   render() {
      return (
         <div>
            <h1>Header</h1>
            <h2>Content</h2>
            <p data-myattribute = "somevalue">This is the content!!!</p>
         </div>
      );
   }
}
export default App;
```

We can use our own custom attributes in addition to regular HTML properties and attributes

JSX - JavaScript Expressions

```
import React from 'react';

class App extends React.Component {
   render() {
      return (
         <div>
            <h1>{1+1}</h1>
         </div>
      );
   }
}
export default App;
```

JavaScript expressions can be used inside of JSX. We just need to wrap it with curly brackets **{}**

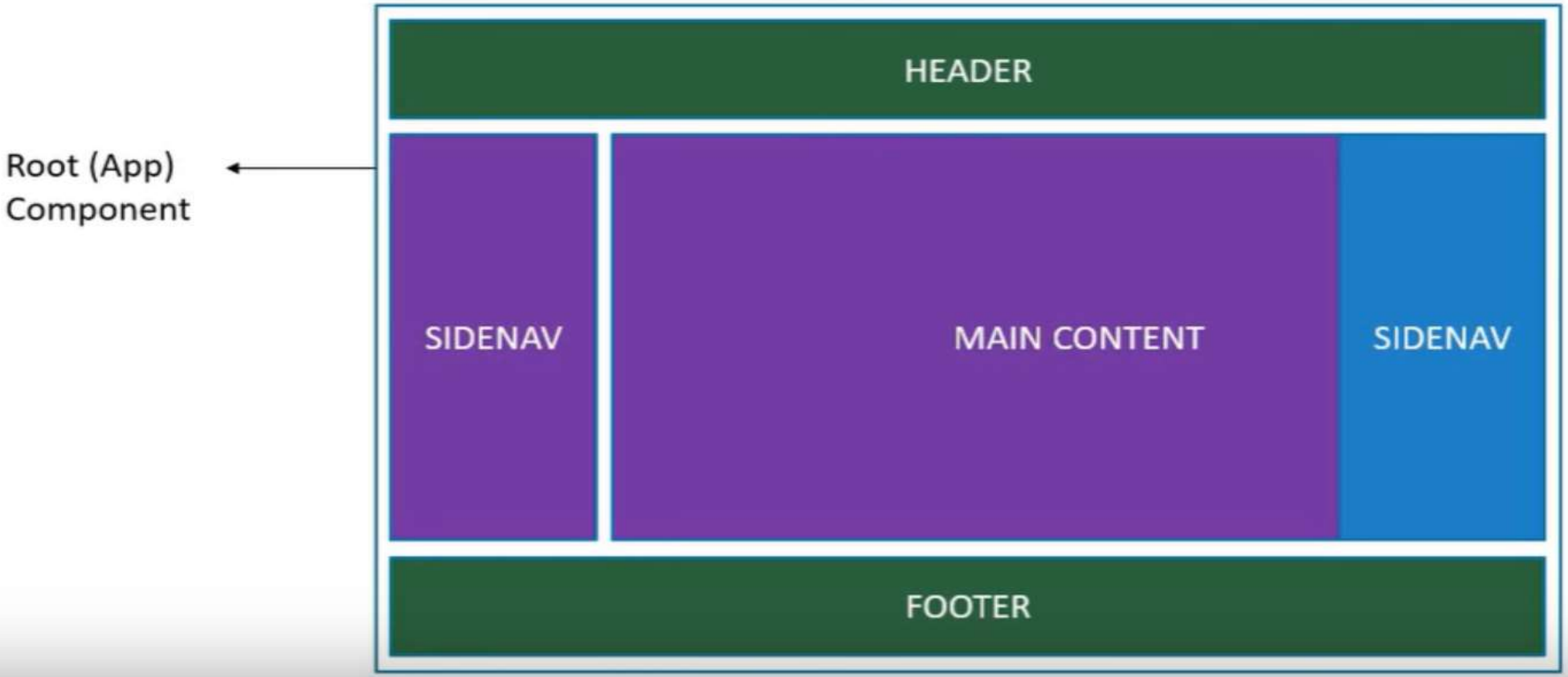
JSX - Styling

```
import React from 'react';
class App extends React.Component {
   render() {
      var myStyle = {
         fontSize: 100,
         color: '#FF0000'
      }
      return (
         <div>
            <h1 style = {myStyle}>Header</h1>
         </div>
      );
   }
}
export default App;
```

React recommends using inline styles.

Components

Components


Root (App)
Component
HEADER
SIDENAV
MAIN CONTENT
SIDENAV
FOOTER

Components

- Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML via a render() function.

- Components come in two types:
  - Class component
  - Function component

Function Component

- Functional components are declared with a function that then returns some JSX.

```
function Car() {
  return <h2>Hi, I am also a Car!</h2>;
}
```

Class Component

- Class-based components are declared using ES6 classes.

```
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}
```

Differences between Functional Components and Class Components

Functional Components	Class Components
A functional component is just a plain JavaScript function that accepts props as an argument and returns a React element.	A class component requires you to extend from React. Component and create a render function which returns a React element.
There is no render method used in functional components.	It must have the render() method returning HTML
Also known as Stateless components as they simply accept data and display them in some form , that is they are mainly responsible for rendering UI.	Also known as Stateful components because they implement logic and state.
React lifecycle methods (for example, componentDidMount) cannot be used in functional components.	React lifecycle methods can be used inside class components (for example, componentDidMount).

Component - props

- Props is what you pass into the Component via attributes
- Props is only way to input data(Or you can use **Redux**)
- Props are immutable
- Container component will define data that can be changed
- Child Component will received data from parent component via props

Component - props

```
import React from 'react';
class App extends React.Component{
   render(){
       return(
          <div>
             <h1>{this.props.headerProp}</h1>
          </div>
   );
  }
}

export default App;
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(
    <App headerProp="Header from props" />
    document.getElementById('app')
);

export default App;
```

Component - State

- Private data of component
- When state change => re-render component
- Can’t read from outside Component

Component - State

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {brand: "Ford"};
  }
  render() {
    return (
      <div>
        <h1>My Car</h1>
      </div>
    );
  }
}
```

React Component Lifecycle

React Component Lifecycle

- React enables to create components by invoking the React.createClass() method which expects a render method and triggers a lifecycle that can be hook into via a number of so called lifecycle methods
- This short article should shed light into all the applicable functions
- Understanding the component lifecycle will enable you to perform certain actions when a component is created or destroyed. Further more it gives you the opportunity to decide if a component should updated in the first place and to react props or state changes accordingly

React Component Lifecycle - Mounting

- Mounting means putting elements into the DOM.
- React has four built-in methods that gets called, in this order, when mounting a component:
  - constructor()
  - getDerivedStateFromProps()
  - render()
  - componentDidMount()
- The render() method is required and will always be called, the others are optional and will be called if you define them.

React Component Lifecycle - Updating

- This lifecycle will be call when a component is updated.

- A component is updated whenever there is a change in the component's state or props.
- React has five built-in methods that gets called, in this order, when a component is updated:
  - getDerivedStateFromProps()
  - shouldComponentUpdate()
  - render()
  - getSnapshotBeforeUpdate()
  - componentDidUpdate()
- The render() method is required and will always be called, the others are optional and will be called if you define them.

React Component Lifecycle - Unmounting

- The next phase in the lifecycle is when a component is removed from the DOM or unmounting as React likes to call it.
- React has only one built-in method that gets called when a component is unmounted:
  - componentWillUnmount()

Out next step

- Flux/ redux
- React Native

Reference

- https://reactjs.org/
- https://www.w3schools.com/react/
- https://www.tutorialspoint.com/reactjs/index.htm
- https://github.com/facebook/react/