BÁO CÁO TỔNG QUAN VỀ REACTJS

MỤC LỤC

1. Giới thiệu về ReactJS
2. Virtual DOM
3. JSX
4. Components
5. Props và State
6. React Lifecycle
7. Kết luận

## 1. GIỚI THIỆU VỀ REACTJS

### **1.1. Lịch sử và xuất xứ**

ReactJS là một thư viện JavaScript mã nguồn mở được phát triển bởi Facebook (nay là Meta) vào năm 2011. Ban đầu, React được tạo ra để giải quyết các vấn đề về hiệu suất và bảo trì trong các ứng dụng web phức tạp của Facebook. Nó chính thức được công bố mã nguồn mở tại hội nghị JSConf US vào tháng 5/2013.

### **1.2. Định nghĩa**

ReactJS là một thư viện JavaScript dùng để xây dựng giao diện người dùng (UI), đặc biệt tập trung vào việc xây dựng các ứng dụng single-page (SPA). React sử dụng component-based architecture, cho phép phát triển các thành phần giao diện độc lập, tái sử dụng và dễ bảo trì.

### **1.3. Ưu điểm**

* **Hiệu suất cao**: Nhờ cơ chế Virtual DOM
* **Tái sử dụng component**: Giảm thiểu code trùng lặp
* **Dễ học và sử dụng**: Cú pháp đơn giản, tài liệu phong phú
* **Cộng đồng lớn**: Nhiều thư viện và công cụ hỗ trợ
* **Hỗ trợ mobile**: React Native cho phát triển ứng dụng di động

### **1.4. Kiến trúc tổng quan**

React hoạt động dựa trên mô hình declarative programming, nơi developer mô tả UI nên trông như thế nào ở các trạng thái khác nhau, và React sẽ tự động cập nhật DOM khi dữ liệu thay đổi.

## 2. VIRTUAL DOM

### **2.1. Khái niệm**

Virtual DOM (DOM ảo) là một khái niệm then chốt trong React. Đây là một bản sao JavaScript của DOM thật, được lưu trữ trong bộ nhớ. Khi state của ứng dụng thay đổi, React tạo một Virtual DOM mới và so sánh với phiên bản trước đó để xác định những thay đổi cần thiết.

### **2.2. Cơ chế hoạt động**

Quy trình làm việc của Virtual DOM:

1. **Khi state thay đổi**: React tạo một Virtual DOM mới
2. **So sánh (Diffing)**: So sánh Virtual DOM mới với phiên bản cũ
3. **Xác định thay đổi**: Tìm ra chính xác những phần tử cần cập nhật
4. **Cập nhật DOM thật**: Chỉ cập nhật những phần thực sự thay đổi

### **2.3. Ưu điểm**

* **Hiệu suất**: Giảm thiểu thao tác trực tiếp với DOM thật
* **Tối ưu hóa**: Tự động nhóm các thay đổi để cập nhật một lần
* **Trừu tượng hóa**: Developer không cần quan tâm đến chi tiết cập nhật DOM

### **2.4. Ví dụ minh họa**

// Trước khi thay đổi

const oldVirtualDOM = {

type: 'div',

props: { className: 'container', children: 'Hello World' }

};

// Sau khi thay đổi

const newVirtualDOM = {

type: 'div',

props: { className: 'container', children: 'Hello React' }

};

// React sẽ chỉ cập nhật text content từ 'Hello World' thành 'Hello React'

## 3. JSX (JAVASCRIPT XML)

### **3.1. Định nghĩa**

JSX là một cú pháp mở rộng của JavaScript, cho phép viết mã HTML-like trong JavaScript. Nó không bắt buộc trong React nhưng được khuyến khích sử dụng vì tính rõ ràng và dễ hiểu.

### **3.2. Đặc điểm**

* **Không phải chuỗi**: JSX biểu diễn các React element
* **Biên dịch thành JavaScript**: Babel chuyển đổi JSX thành các lệnh React.createElement()
* **Biểu thức JavaScript**: Có thể nhúng biểu thức JS trong JSX bằng {}

### **3.3. Cú pháp cơ bản**

const name = 'John Doe';

const element = <h1>Hello, {name}!</h1>;

// JSX với thuộc tính

const element = <img src={user.avatarUrl} alt={user.name} />;

// JSX với nhiều phần tử

const element = (

<div>

<h1>Hello!</h1>

<p>Welcome to React</p>

</div>

);

### **3.4. Quy tắc quan trọng**

* **Một phần tử gốc**: JSX phải có một phần tử cha duy nhất
* **Đóng thẻ**: Tất cả thẻ phải được đóng đúng cách
* **className thay cho class**: Vì class là từ khóa trong JavaScript
* **CamelCase cho thuộc tính**: onClick thay vì onclick

### **3.5. Biên dịch JSX**

JSX được biên dịch thành:

// JSX

const element = <h1 className="greeting">Hello, world!</h1>;

// Sau khi biên dịch

const element = React.createElement(

'h1',

{className: 'greeting'},

'Hello, world!'

);

## 4. COMPONENTS

### **4.1. Khái niệm**

Components là các khối xây dựng cơ bản của ứng dụng React. Mỗi component là một phần độc lập, có thể tái sử dụng, đóng gói giao diện và logic của nó.

### **4.2. Phân loại component**

#### **4.2.1. Function Components (Component hàm)**

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

// Hoặc với Arrow Function

const Welcome = (props) => {

return <h1>Hello, {props.name}</h1>;

};

#### **4.2.2. Class Components (Component lớp)**

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

### **4.3. Component composition**

Các component có thể lồng vào nhau:

function App() {

return (

<div>

<Welcome name="Alice" />

<Welcome name="Bob" />

<Welcome name="Charlie" />

</div>

);

}

### **4.4. Ưu điểm của component-based architecture**

* **Tái sử dụng**: Một component có thể dùng nhiều nơi
* **Bảo trì**: Dễ dàng sửa đổi và kiểm thử từng phần
* **Phân chia công việc**: Nhiều developer có thể làm việc song song
* **Tổ chức code**: Code được cấu trúc rõ ràng, dễ hiểu

## 5. PROPS VÀ STATE

### **5.1. Props (Properties)**

#### **5.1.1. Định nghĩa**

Props là dữ liệu được truyền từ component cha xuống component con. Props là read-only và không thể thay đổi bởi component nhận chúng.

#### **5.1.2. Truyền và nhận props**

// Truyền props

<UserProfile name="John" age={25} isActive={true} />

// Nhận props trong function component

function UserProfile(props) {

return (

<div>

<p>Name: {props.name}</p>

<p>Age: {props.age}</p>

<p>Active: {props.isActive ? 'Yes' : 'No'}</p>

</div>

);

}

// Nhận props trong class component

class UserProfile extends React.Component {

render() {

return (

<div>

<p>Name: {this.props.name}</p>

<p>Age: {this.props.age}</p>

</div>

);

}

}

#### **5.1.3. Default Props**

function Greeting(props) {

return <h1>Hello, {props.name}</h1>;

}

Greeting.defaultProps = {

name: 'Guest'

};

#### **5.1.4. PropTypes (kiểm tra kiểu dữ liệu)**

import PropTypes from 'prop-types';

function UserProfile(props) {

// ...

}

UserProfile.propTypes = {

name: PropTypes.string.isRequired,

age: PropTypes.number,

isActive: PropTypes.bool

};

### **5.2. State**

#### **5.2.1. Định nghĩa**

State là dữ liệu nội bộ của component, có thể thay đổi theo thời gian. Khi state thay đổi, component sẽ re-render.

#### **5.2.2. State trong function component (với Hook)**

import { useState } from 'react';

function Counter() {

const [count, setCount] = useState(0);

return (

<div>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>

Click me

</button>

</div>

);

}

#### **5.2.3. State trong class component**

class Counter extends React.Component {

constructor(props) {

super(props);

this.state = {

count: 0

};

}

render() {

return (

<div>

<p>You clicked {this.state.count} times</p>

<button onClick={() => this.setState({ count: this.state.count + 1 })}>

Click me

</button>

</div>

);

}

}

#### **5.2.4. So sánh Props và State**

| **Tiêu chí** | **Props** | **State** |
| --- | --- | --- |
| **Truyền dữ liệu** | Từ cha → con | Nội bộ component |
| **Thay đổi** | Read-only | Có thể thay đổi |
| **Ảnh hưởng** | Component con re-render | Component hiện tại re-render |
| **Khởi tạo** | Từ component cha | Trong component |

### **5.3. Quy tắc khi làm việc với State**

* **Không thay đổi state trực tiếp**: Luôn dùng setState() hoặc hàm setter từ useState()
* **State updates có thể bất đồng bộ**: React có thể gộp nhiều setState() lại
* **State updates được merge**: Trong class component, setState() merge object

## 6. REACT LIFECYCLE

### **6.1. Khái niệm**

Vòng đời component là các giai đoạn mà một component trải qua từ khi được tạo, cập nhật đến khi bị hủy. Hiểu vòng đời giúp kiểm soát hành vi component ở các thời điểm khác nhau.

### **6.2. Vòng đời trong class component**

#### **6.2.1. Mounting (Gắn vào DOM)**

1. **constructor()**: Khởi tạo state và bind methods
2. **static getDerivedStateFromProps()**: Cập nhật state dựa trên props
3. **render()**: Trả về JSX để render
4. **componentDidMount()**: Được gọi sau khi component được render lần đầu

#### **6.2.2. Updating (Cập nhật)**

1. **static getDerivedStateFromProps()**: Cập nhật state khi props thay đổi
2. **shouldComponentUpdate()**: Quyết định component có nên re-render không
3. **render()**: Render lại component
4. **getSnapshotBeforeUpdate()**: Lấy thông tin từ DOM trước khi cập nhật
5. **componentDidUpdate()**: Được gọi sau khi update hoàn tất

#### **6.2.3. Unmounting (Gỡ khỏi DOM)**

1. **componentWillUnmount()**: Dọn dẹp trước khi component bị hủy

#### **6.2.4. Error Handling (Xử lý lỗi)**

1. **static getDerivedStateFromError()**: Cập nhật state khi có lỗi
2. **componentDidCatch()**: Xử lý lỗi

### **6.3. Vòng đời trong function component (với Hooks)**

#### **6.3.1. useEffect Hook**

import { useEffect, useState } from 'react';

function Example() {

const [count, setCount] = useState(0);

// Tương đương componentDidMount và componentDidUpdate

useEffect(() => {

document.title = `You clicked ${count} times`;

});

// Tương đương componentDidMount

useEffect(() => {

console.log('Component mounted');

}, []);

// Tương đương componentDidUpdate với điều kiện

useEffect(() => {

console.log('Count changed:', count);

}, [count]); // Chỉ chạy khi count thay đổi

// Tương đương componentWillUnmount

useEffect(() => {

return () => {

console.log('Component will unmount');

};

}, []);

return (

<div>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>

Click me

</button>

</div>

);

}

### **6.4. So sánh lifecycle methods và hooks**

| **Class Component** | **Function Component với Hooks** |
| --- | --- |
| componentDidMount | useEffect với dependency array rỗng |
| componentDidUpdate | useEffect với dependencies cụ thể |
| componentWillUnmount | Return cleanup function từ useEffect |
| shouldComponentUpdate | React.memo hoặc useMemo |

## 7. KẾT LUẬN

ReactJS đã cách mạng hóa cách phát triển ứng dụng web với kiến trúc component-based và cơ chế Virtual DOM. Các khái niệm cơ bản như JSX, Components, Props, State và Lifecycle tạo nền tảng vững chắc cho việc xây dựng ứng dụng React hiệu quả.

### **7.1. Tóm tắt điểm mạnh**

* **Hiệu suất**: Virtual DOM tối ưu hóa render
* **Tái sử dụng**: Component architecture cho phép tái sử dụng code
* **Dễ bảo trì**: Code được tổ chức rõ ràng, dễ hiểu
* **Cộng đồng**: Ecosystem phong phú với nhiều thư viện hỗ trợ

### **7.2. Xu hướng phát triển**

* **Function components với Hooks**: Xu hướng chính thay thế class components
* **Server Components**: React Server Components cho render phía server
* **Concurrent Features**: Cải thiện khả năng responsive của ứng dụng

### **7.3. Khuyến nghị**

Đối với người mới học React, nên:

1. Bắt đầu với function components và hooks
2. Hiểu rõ sự khác biệt giữa props và state
3. Thực hành với các dự án nhỏ trước
4. Tận dụng React Dev Tools để debug

ReactJS tiếp tục là một trong những thư viện frontend phổ biến nhất và là kỹ năng quan trọng cho các developer web hiện đại.

**Tài liệu tham khảo:**

* React Official Documentation: <https://reactjs.org/>
* React Beta Documentation: <https://beta.reactjs.org/>
* "Learning React" by Alex Banks and Eve Porcello
* "React Up and Running" by Stoyan Stefanov