Báo Cáo: Các Khái Niệm Cơ Bản Của ReactJS

Mục Lục:

[1. Giới Thiệu Về ReactJS 2](#_Toc207788760)

[Lịch Sử Phát Triển 2](#_Toc207788761)

[Ưu Điểm Của React 2](#_Toc207788762)

[Nhược Điểm 3](#_Toc207788763)

[Cài Đặt React 3](#_Toc207788764)

[2. Virtual DOM 4](#_Toc207788765)

[Khái Niệm Cơ Bản 4](#_Toc207788766)

[Cách Hoạt Động 4](#_Toc207788767)

[Ưu Điểm 5](#_Toc207788768)

[Nhược Điểm 5](#_Toc207788769)

[3. JSX 6](#_Toc207788770)

[Khái Niệm Cơ Bản 6](#_Toc207788771)

[Quy Tắc JSX 6](#_Toc207788772)

[Ưu Điểm 7](#_Toc207788773)

[Nhược Điểm 7](#_Toc207788774)

[4. Components 7](#_Toc207788775)

[Loại Components 7](#_Toc207788776)

[Composition 8](#_Toc207788777)

[Best Practices 9](#_Toc207788778)

[5. Props Và State 9](#_Toc207788779)

[Props (Properties) 9](#_Toc207788780)

[State 9](#_Toc207788781)

[So Sánh Props Và State 11](#_Toc207788782)

[6. React Lifecycle 11](#_Toc207788783)

[Giai Đoạn Lifecycle 11](#_Toc207788784)

[Với Hooks 12](#_Toc207788785)

[7. Kết Luận 13](#_Toc207788786)

[8. Tài Liệu Tham Khảo 13](#_Toc207788787)

## 1. Giới Thiệu Về ReactJS

ReactJS, thường được gọi tắt là React, là một thư viện JavaScript mã nguồn mở được phát triển bởi Facebook (nay là Meta) vào năm 2013. React được thiết kế để xây dựng giao diện người dùng (UI) cho các ứng dụng web đơn trang (Single Page Applications - SPA) một cách hiệu quả và dễ dàng mở rộng. Thay vì là một framework đầy đủ như Angular hay Vue.js, React tập trung vào việc xử lý phần view layer, cho phép các lập trình viên xây dựng các thành phần UI có thể tái sử dụng.

### Lịch Sử Phát Triển

React được giới thiệu lần đầu tiên tại hội nghị JSConf US năm 2013 bởi Jordan Walke, một kỹ sư phần mềm tại Facebook. Ban đầu, nó được sử dụng nội bộ để giải quyết vấn đề hiệu suất trong việc cập nhật giao diện trên Facebook và Instagram. Đến năm 2015, React Native được ra mắt, mở rộng React sang phát triển ứng dụng di động. Hiện nay, React là một trong những thư viện phổ biến nhất, với hơn 10 triệu lượt tải hàng tuần trên npm (Node Package Manager).

### Ưu Điểm Của React

* **Hiệu Suất Cao**: Nhờ cơ chế Virtual DOM, React giảm thiểu việc thao tác trực tiếp với DOM thực, giúp ứng dụng nhanh hơn.
* **Tái Sử Dụng Code**: Các component cho phép chia nhỏ UI thành các phần độc lập, dễ quản lý và tái sử dụng.
* **Cộng Đồng Lớn**: Hỗ trợ từ Meta, cùng với hàng ngàn thư viện bên thứ ba như Redux, React Router.
* **Dễ Học**: Nếu bạn đã biết JavaScript, việc học React khá đơn giản, tập trung vào các khái niệm cốt lõi.
* **Tích Hợp Linh Hoạt**: Có thể kết hợp với các công nghệ khác như Node.js, Express cho backend.

### Nhược Điểm

* **Không Phải Framework Đầy Đủ**: Cần kết hợp với các thư viện khác để xử lý state management, routing, v.v.
* **Học Đường Cong Ban Đầu**: Các khái niệm như hooks (từ React 16.8) có thể phức tạp với người mới.
* **Cập Nhật Thường Xuyên**: Phiên bản mới có thể thay đổi cách làm việc, đòi hỏi cập nhật kiến thức.

React sử dụng mô hình declarative programming, nghĩa là bạn mô tả UI nên trông như thế nào dựa trên state, và React sẽ tự động cập nhật khi state thay đổi. Điều này khác biệt với imperative programming trong vanilla JavaScript, nơi bạn phải chỉ rõ từng bước thao tác DOM.

### Cài Đặt React

Để bắt đầu, bạn có thể sử dụng Create React App (CRA):

bash

npx create-react-app my-app

cd my-app

npm start

Điều này sẽ tạo một dự án cơ bản với Babel (chuyển đổi JSX) và Webpack (bundle code).

React đã phát triển qua nhiều phiên bản, với React 18 (2022) giới thiệu concurrent rendering để cải thiện hiệu suất.

## 2. Virtual DOM

Virtual DOM (Document Object Model ảo) là một trong những tính năng cốt lõi làm nên sức mạnh của React. Nó là một bản sao nhẹ của DOM thực tế, được lưu trữ trong bộ nhớ, giúp React tối ưu hóa việc cập nhật giao diện mà không cần thao tác trực tiếp với DOM browser – một quá trình thường chậm và tốn kém.

### Khái Niệm Cơ Bản

DOM thực là một cấu trúc cây đại diện cho HTML của trang web. Mỗi thay đổi (như thêm/xóa element) yêu cầu browser reflow và repaint, dẫn đến hiệu suất kém nếu cập nhật thường xuyên. Virtual DOM giải quyết bằng cách:

* Tạo một bản sao ảo của DOM.
* Khi state thay đổi, React tạo một Virtual DOM mới.
* So sánh (diffing) Virtual DOM mới với cũ để tìm sự khác biệt (reconciliation).
* Chỉ cập nhật những phần thay đổi lên DOM thực (batching updates).

Quá trình này được gọi là "Reconciliation" và được thực hiện bởi thuật toán React Fiber (từ React 16).

### Cách Hoạt Động

1. **Render Ban Đầu**: React xây dựng Virtual DOM từ JSX và render thành DOM thực.
2. **Cập Nhật**: Khi props/state thay đổi, phương thức render() được gọi lại, tạo Virtual DOM mới.
3. **Diffing**: React sử dụng heuristic algorithm để so sánh:
   * So sánh type của element (nếu khác, thay thế toàn bộ subtree).
   * So sánh attributes (props).
   * Duyệt đệ quy các child elements.
   * Sử dụng key prop để tối ưu hóa list rendering.
4. **Commit**: Áp dụng thay đổi tối thiểu lên DOM thực.

Ví dụ mã:

jsx

import React, { 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>

);

}

Khi click, state count thay đổi, Virtual DOM cập nhật chỉ phần text {count}, không render lại toàn bộ component.

### Ưu Điểm

* **Hiệu Suất**: Giảm thiểu DOM manipulations, đặc biệt hữu ích cho ứng dụng lớn.
* **Cross-Platform**: Áp dụng cho React Native (Virtual DOM cho native views).
* **Debug Dễ Dàng**: Công cụ như React DevTools hiển thị Virtual DOM tree.

### Nhược Điểm

* **Bộ Nhớ**: Virtual DOM chiếm bộ nhớ, nhưng thường không đáng kể.
* **Overhead**: Diffing có thể chậm nếu tree lớn mà không tối ưu (sử dụng shouldComponentUpdate hoặc React.memo).

So sánh với DOM thực: Trong vanilla JS, bạn dùng document.getElementById('id').innerHTML = newValue;, nhưng React làm điều này hiệu quả hơn.

## 3. JSX

JSX (JavaScript XML) là một cú pháp mở rộng của JavaScript, cho phép viết code giống HTML ngay trong JS. Nó không phải là HTML mà là syntactic sugar, được Babel chuyển đổi thành các lời gọi React.createElement().

### Khái Niệm Cơ Bản

JSX giúp viết UI declarative, kết hợp logic JS với markup. Ví dụ:

jsx

const element = <h1>Hello, world!</h1>*;*

Babel chuyển thành:

js

const element = React.createElement('h1', null, 'Hello, world!');

### Quy Tắc JSX

* **Tag Phải Đóng**: <img /> hoặc <div></div>.
* **Attribute**: Sử dụng camelCase (e.g., className thay vì class, onClick thay vì onclick).
* **JS Expression**: Sử dụng {} để nhúng JS: <p>{2 + 2}</p>.
* **Style**: Là object: style={{ color: 'red' }}.
* **Children**: Có thể là text, element, hoặc array.
* **Fragment**: Sử dụng <></> hoặc <React.Fragment> để nhóm elements mà không thêm node.

Ví dụ phức tạp:

jsx

function Greeting({ name }) {

return (

<div className="greeting">

<h1>Hello, {name}!</h1>

{name === 'Admin' ? <p>Admin panel</p> : <p>User dashboard</p>}

</div>

);

}

### Ưu Điểm

* **Đọc Dễ**: Code giống HTML, dễ hình dung UI.
* **Tích Hợp JS**: Dễ dàng sử dụng variables, functions, loops (sử dụng map() cho lists).
* **An Toàn**: JSX tự động escape để tránh XSS attacks.

### Nhược Điểm

* **Cần Transpiler**: Phải dùng Babel.
* **Học Ban Đầu**: Trông lạ nếu quen vanilla JS.

JSX là nền tảng cho việc viết components, làm React trở nên trực quan.

## 4. Components

Components là các khối xây dựng cơ bản của React, đại diện cho một phần UI độc lập và tái sử dụng. Chúng có thể là function hoặc class, nhận input (props) và trả về JSX.

### Loại Components

1. **Function Components**: Đơn giản, stateless (trước hooks). Từ React 16.8, có thể dùng hooks cho state.

jsx

function Welcome(props) {

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

}

1. **Class Components**: Sử dụng ES6 class, hỗ trợ state và lifecycle methods.

jsx

class Welcome extends React.Component {

render() {

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

}

}

1. **Higher-Order Components (HOC)**: Function nhận component và trả về component mới, dùng cho code reuse (e.g., withAuth).
2. **Render Props**: Pattern chia sẻ code qua prop là function render.

### Composition

Components có thể nest:

jsx

function App() {

return (

<div>

<Header />

<Content />

<Footer />

</div>

);

}

### Best Practices

* **Single Responsibility**: Mỗi component chỉ làm một việc.
* **Pure Functions**: Tránh side effects trong render.
* **Key Prop**: Cho lists để tối ưu diffing: <li key={id}>...</li>.
* **Controlled vs Uncontrolled**: Controlled components quản lý state qua React.

Components làm React modular, dễ test (sử dụng Jest/Enzyme).

## 5. Props Và State

Props và State là hai khái niệm quản lý dữ liệu trong React.

### Props (Properties)

* Input cho component, immutable (không thay đổi bởi component nhận).
* Truyền từ parent sang child: <Child name="John" />.
* Default props: Child.defaultProps = { name: 'Guest' };.
* PropTypes: Để validate (dùng PropTypes library hoặc TypeScript).

Ví dụ:

jsx

function Child(props) {

return <p>{props.name}</p>;

}

### State

* Dữ liệu nội bộ của component, mutable (có thể thay đổi).
* Khởi tạo trong constructor (class) hoặc useState (function).
* Cập nhật bằng setState() (class) hoặc setter từ hook.

Ví dụ class:

jsx

class Counter extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

inc*rement = () => {*

this.setState({ count: this.state.count + 1 });

};

render() {

return <button onClick={this.inc*rement}>{this.state.count}</button>;*

}

}

Ví dụ hook:

jsx

import { useState } from 'react'*;*

function Counter() {

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

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

}

### So Sánh Props Và State

| **Đặc Điểm** | **Props** | **State** |
| --- | --- | --- |
| Mutable? | Không | Có |
| Truyền Từ | Parent | Nội bộ component |
| Sử Dụng | Dữ liệu tĩnh, config | Dữ liệu động, user input |
| Cập Nhật | Parent re-render | setState/useState |

* **Lifting State Up**: Di chuyển state lên parent để chia sẻ giữa children.
* **useReducer**: Cho state phức tạp, tương tự Redux.

Quản lý props/state tốt giúp tránh bugs như stale state.

## 6. React Lifecycle

Lifecycle methods là các hàm đặc biệt trong class components được gọi tại các giai đoạn khác nhau của component's life. Với function components, sử dụng useEffect hook để simulate.

### Giai Đoạn Lifecycle

1. **Mounting** (Tạo):
   * constructor(): Khởi tạo state/props.
   * getDerivedStateFromProps(): Cập nhật state từ props (rare).
   * render(): Trả về JSX.
   * componentDidMount(): Sau khi render, dùng cho API calls, event listeners.
2. **Updating** (Cập Nhật):
   * getDerivedStateFromProps().
   * shouldComponentUpdate(): Quyết định có re-render không (tối ưu).
   * render().
   * getSnapshotBeforeUpdate(): Lấy snapshot trước update (rare).
   * componentDidUpdate(): Sau update, xử lý side effects.
3. **Unmounting** (Hủy):
   * componentWillUnmount(): Cleanup (remove listeners, timers).

### Với Hooks

useEffect thay thế nhiều methods:

jsx

useEffect(() => {

*// componentDidMount + componentDidUpdate*

return () => {

*// componentWillUnmount*

};

}, [dependencies]);

Ví dụ full lifecycle class:

jsx

class Timer extends React.Component {

constructor(props) {

super(props);

this.state = { seconds: 0 };

}

componentDidMount() {

this.interval = setInterval(() => this.setState({ seconds: this.state.seconds + 1 }), 1000);

}

componentWillUnmount() {

clearInterval(this.interval);

}

render() {

return <div>Seconds: {this.state.seconds}</div>;

}

}

Lifecycle giúp quản lý side effects như fetch data, animations.

## 7. Kết Luận

ReactJS với các khái niệm cốt lõi như Virtual DOM, JSX, Components, Props/State, và Lifecycle cung cấp nền tảng vững chắc để xây dựng ứng dụng web hiện đại. Bằng cách nắm vững chúng, bạn có thể tạo UI responsive, hiệu suất cao. Để đi sâu hơn, học hooks, context, và Redux. React tiếp tục phát triển, với focus trên server-side rendering (Next.js) và concurrent mode.

## 8. Tài Liệu Tham Khảo

* Official React Docs: reactjs.org
* MDN Web Docs for DOM
* Books: "Learning React" by Alex Banks & Eve Porcello
* Online: freeCodeCamp React Tutorial, Codecademy