**ĐỀ CƯƠNG BÁO CÁO**

**EVENT FORM TRONG REACTJS**

**Chương 1. Tổng quan về Event trong ReactJS**

Trong React, "event" (sự kiện) là các hành động mà người dùng thực hiện trên giao diện người dùng (UI), như click chuột, nhập liệu vào form, hoặc gửi form. React xử lý các sự kiện này thông qua hệ thống SyntheticEvent, một lớp "vỏ bọc" đa trình duyệt để đảm bảo hoạt động nhất quán trên các trình duyệt khác nhau. Khi một sự kiện xảy ra, một hàm xử lý sự kiện được gọi để phản hồi lại hành động đó, cho phép bạn xây dựng các ứng dụng web tương tác.

1. Hệ thống SyntheticEvent:

React sử dụng SyntheticEvent, một trình bao bọc xung quanh sự kiện gốc của trình duyệt, cung cấp một API thống nhất cho các sự kiện này bất kể trình duyệt nào.

2. Xử lý sự kiện:

Bạn định nghĩa các hàm xử lý (event handlers) để chạy khi một sự kiện xảy ra.

Đặt tên quy ước: Các thuộc tính sự kiện dùng tên theo kiểu camelCase (ví dụ: onClick, onChange).

Truyền hàm xử lý: Hàm xử lý được truyền dưới dạng một prop (thuộc tính) cho component.

Ví dụ về các sự kiện phổ biến

onClick: Kích hoạt khi một phần tử được nhấp.

onChange: Kích hoạt khi giá trị của một trường nhập thay đổi.

onSubmit: Kích hoạt khi một form được gửi đi.

onKeyDown: Kích hoạt khi một phím được nhấn xuống.

Xử lý hành động mặc định

Trong nhiều trường hợp, các thành phần có hành vi mặc định đi kèm với một sự kiện (ví dụ: gửi form sẽ làm tải lại trang).

Bạn có thể sử dụng event.preventDefault() bên trong hàm xử lý để ngăn chặn hành vi mặc định này và thay thế bằng logic tùy chỉnh của riêng bạn.

3.Cú pháp sự kiện trong React

Trong React, cú pháp xử lý sự kiện sử dụng tên sự kiện viết hoa dạng camelCase (ví dụ: onClick, onChange) và truyền một hàm callback vào prop của sự kiện thay vì một chuỗi.

Ví dụ:

Mã

<button onClick={handleClick}>

Nhấn vào đây

</button>

Trong ví dụ trên:

onClick là tên sự kiện, được viết theo dạng camelCase.

handleClick là một hàm callback được truyền vào prop onClick để thực thi khi sự kiện xảy ra.

1. So sánh với HTML truyền thống

### ****Cú pháp****

* **HTML truyền thống**:

html

<button onclick="handleClick()">Click</button>

* + Sử dụng thuộc tính onclick (viết thường) để gắn một hàm JavaScript (handleClick()) trực tiếp vào phần tử HTML.
  + Tên hàm được gọi dưới dạng chuỗi, và thường được định nghĩa trong một tệp JavaScript riêng hoặc trong thẻ <script>.
* **React**:

jsx

<button onClick={handleClick}>Click</button>

* + Sử dụng cú pháp JSX, một phần mở rộng của JavaScript, trông giống HTML nhưng được biên dịch thành JavaScript.
  + Thuộc tính sự kiện được viết theo kiểu **camelCase** (onClick thay vì onclick).
  + Giá trị của onClick là một **tham chiếu đến hàm** (handleClick) chứ không phải chuỗi hay gọi hàm trực tiếp (handleClick()).

### 2. ****Cách xử lý sự kiện****

* **HTML truyền thống**:
  + Sự kiện được xử lý trực tiếp trong DOM, thông qua thuộc tính onclick được gắn vào phần tử.
  + Hàm handleClick phải được định nghĩa trong phạm vi toàn cục (global scope) hoặc được truy cập qua chuỗi mã JavaScript.
  + Dễ dẫn đến vấn đề về phạm vi (scope) hoặc xung đột tên hàm nếu không quản lý tốt.
  + Hiệu suất có thể kém hơn trong các ứng dụng lớn do sự kiện được gắn trực tiếp vào DOM.
* **React**:
  + React sử dụng **Synthetic Events** (sự kiện tổng hợp), một lớp bao bọc quanh các sự kiện DOM gốc, giúp chuẩn hóa hành vi sự kiện giữa các trình duyệt.
  + Hàm handleClick được truyền dưới dạng tham chiếu, cho phép React quản lý sự kiện hiệu quả hơn thông qua **Virtual DOM**.
  + React kiểm soát việc gắn và gỡ sự kiện (event delegation), giảm số lượng trình xử lý sự kiện trực tiếp trên DOM, cải thiện hiệu suất.

### 3. ****Quản lý trạng thái và khả năng tái sử dụng****

* **HTML truyền thống**:
  + Không có cơ chế quản lý trạng thái tích hợp. Nếu muốn cập nhật giao diện sau khi click, bạn cần thao tác DOM thủ công (ví dụ: document.getElementById).
  + Khó tái sử dụng logic hoặc giao diện vì HTML tĩnh không hỗ trợ các thành phần (components).
* **React**:
  + React khuyến khích sử dụng **components** và quản lý trạng thái thông qua useState hoặc các cơ chế khác.
  + Hàm handleClick thường được định nghĩa trong phạm vi của component, dễ dàng truy cập trạng thái (state) hoặc props, giúp logic xử lý sự kiện linh hoạt và tái sử dụng.
  + Khi trạng thái thay đổi, React tự động cập nhật giao diện thông qua cơ chế **re-rendering**.

### 4. ****Khả năng bảo trì và mở rộng****

* **HTML truyền thống**:
  + Phù hợp cho các trang web tĩnh hoặc ứng dụng nhỏ, nơi logic đơn giản.
  + Khi ứng dụng phức tạp, việc duy trì mã JavaScript và HTML riêng lẻ trở nên khó khăn, dễ dẫn đến mã "spaghetti".
* **React**:
  + Phù hợp cho các ứng dụng giao diện người dùng phức tạp, nơi cần quản lý trạng thái và tái sử dụng component.
  + JSX cho phép kết hợp giao diện và logic trong cùng một tệp, giúp mã dễ đọc và bảo trì hơn.
  + Hỗ trợ các công cụ như hooks (useState, useEffect) để xử lý logic phức tạp.

### 5. ****Hiệu suất****

* **HTML truyền thống**:
  + Mỗi phần tử có thể có một trình xử lý sự kiện riêng, dẫn đến tiêu tốn tài nguyên khi có nhiều phần tử.
  + Thao tác DOM trực tiếp (như element.innerHTML) thường chậm trong ứng dụng lớn.
* **React**:
  + Sử dụng Virtual DOM để giảm thiểu thao tác trực tiếp trên DOM thật, chỉ cập nhật những phần thay đổi.
  + Event delegation giúp giảm số lượng trình xử lý sự kiện, cải thiện hiệu suất trong các ứng dụng lớn.

### 6. ****Ví dụ minh họa****

* **HTML truyền thống**:

html

<button onclick="handleClick()">Click</button><script> function handleClick() { alert("Button clicked!"); }</script>

* **React**:

jsx

function Button() { const handleClick = () => { alert("Button clicked!"); }; return <button onClick={handleClick}>Click</button>;}

### Kết luận

* **HTML truyền thống** phù hợp cho các trang web đơn giản, tĩnh, hoặc khi không cần quản lý trạng thái phức tạp. Tuy nhiên, nó thiếu linh hoạt và khó bảo trì trong các ứng dụng lớn.
* **React** cung cấp cách tiếp cận hiện đại, mạnh mẽ hơn, với JSX, Virtual DOM, và cơ chế quản lý trạng thái, phù hợp cho các ứng dụng giao diện người dùng động và phức tạp.

Nếu bạn cần giải thích chi tiết hơn về bất kỳ phần nào, hãy cho tôi biết!

**Chương 2. Các sự kiện phổ biến**

### 1. ****Sự kiện chuột****

* **onClick**:
  + **Mô tả**: Kích hoạt khi người dùng nhấp chuột (thường là chuột trái) vào một phần tử.
  + **HTML truyền thống**:

html

<button onclick="handleClick()">Click</button><script> function handleClick() { console.log("Clicked!"); }</script>

* + **React**:

jsx

<button onClick={() => console.log("Clicked!")}>Click</button>

* + **Khác biệt**: HTML dùng chuỗi gọi hàm, React dùng tham chiếu hàm và camelCase (onClick).
* **onDoubleClick**:
  + **Mô tả**: Kích hoạt khi người dùng nhấp đúp chuột vào phần tử.
  + **HTML**:

html

<div ondblclick="handleDblClick()">Double Click</div>

* + **React**:

jsx

<div onDoubleClick={() => console.log("Double Clicked!")}>Double Click</div>

* + **Ghi chú**: Trong HTML, thuộc tính là ondblclick (viết thường), còn React dùng onDoubleClick.
* **onMouseEnter**:
  + **Mô tả**: Kích hoạt khi con trỏ chuột di chuyển vào phần tử.
  + **HTML**:

html

<div onmouseenter="handleEnter()">Hover me</div>

* + **React**:

jsx

<div onMouseEnter={() => console.log("Mouse entered!")}>Hover me</div>

* + **Khác biệt**: HTML dùng onmouseenter, React dùng onMouseEnter.
* **onMouseLeave**:
  + **Mô tả**: Kích hoạt khi con trỏ chuột rời khỏi phần tử.
  + **HTML**:

html

<div onmouseleave="handleLeave()">Leave me</div>

* + **React**:

jsx

<div onMouseLeave={() => console.log("Mouse left!")}>Leave me</div>

### 2. ****Sự kiện bàn phím****

* **onKeyDown**:
  + **Mô tả**: Kích hoạt khi một phím được nhấn xuống.
  + **HTML**:

html

<input onkeydown="handleKeyDown()" />

* + **React**:

jsx

<input onKeyDown={(e) => console.log(e.key)} />

* + **Ghi chú**: React cung cấp đối tượng sự kiện (e) để truy cập thông tin phím (như e.key).
* **onKeyPress**:
  + **Mô tả**: Kích hoạt khi một phím được nhấn và tạo ra một ký tự (lưu ý: đã bị deprecated trong nhiều trình duyệt).
  + **HTML**:

html

<input onkeypress="handleKeyPress()" />

* + **React**:

jsx

<input onKeyPress={(e) => console.log("Key pressed!")} />

* + **Khuyến nghị**: Trong React, nên dùng onKeyDown thay vì onKeyPress vì onKeyPress không còn được hỗ trợ tốt.
* **onKeyUp**:
  + **Mô tả**: Kích hoạt khi một phím được thả ra.
  + **HTML**:

html

<input onkeyup="handleKeyUp()" />

* + **React**:

jsx

<input onKeyUp={(e) => console.log("Key released!")} />

### 3. ****Sự kiện form****

* **onChange**:
  + **Mô tả**: Kích hoạt khi giá trị của phần tử form (như input, select) thay đổi.
  + **HTML**:

html

<input onchange="handleChange()" />

* + **React**:

jsx

<input onChange={(e) => console.log(e.target.value)} />

* + **Khác biệt**: Trong React, onChange kích hoạt ngay khi giá trị thay đổi, không cần đợi mất focus như HTML ở một số trường hợp.
* **onSubmit**:
  + **Mô tả**: Kích hoạt khi form được gửi (submit).
  + **HTML**:

html

<form onsubmit="handleSubmit()"> <button type="submit">Submit</button></form>

* + **React**:

jsx

<form onSubmit={(e) => { e.preventDefault(); console.log("Form submitted!"); }}> <button type="submit">Submit</button></form>

* + **Ghi chú**: Trong React, thường cần gọi e.preventDefault() để ngăn hành vi mặc định của form (như làm mới trang).
* **onFocus**:
  + **Mô tả**: Kích hoạt khi phần tử nhận được focus.
  + **HTML**:

html

<input onfocus="handleFocus()" />

* + **React**:

jsx

<input onFocus={() => console.log("Focused!")} />

* **onBlur**:
  + **Mô tả**: Kích hoạt khi phần tử mất focus.
  + **HTML**:

html

<input onblur="handleBlur()" />

* + **React**:

jsx

<input onBlur={() => console.log("Blurred!")} />

### 4. ****Sự kiện khác****

* **onLoad**:
  + **Mô tả**: Kích hoạt khi một phần tử (như hình ảnh hoặc trang) được tải xong.
  + **HTML**:

html

<img src="image.jpg" onload="handleLoad()" />

* + **React**:

jsx

<img src="image.jpg" onLoad={() => console.log("Image loaded!")} />

* **onError**:
  + **Mô tả**: Kích hoạt khi có lỗi xảy ra (ví dụ: hình ảnh không tải được).
  + **HTML**:

html

<img src="invalid.jpg" onerror="handleError()" />

* + **React**:

jsx

<img src="invalid.jpg" onError={() => console.log("Image load failed!")} />

* **onScroll**:
  + **Mô tả**: Kích hoạt khi người dùng cuộn nội dung trong phần tử có thanh cuộn.
  + **HTML**:

html

<div style="height: 100px; overflow: auto;" onscroll="handleScroll()">...</div>

* + **React**:

jsx

<div style={{ height: '100px', overflow: 'auto' }} onScroll={() => console.log("Scrolled!")}>...</div>

### So sánh tổng quan

| **Tiêu chí** | **HTML truyền thống** | **React** |
| --- | --- | --- |
| **Cú pháp** | Thuộc tính viết thường (onclick, onchange) | CamelCase (onClick, onChange) |
| **Giá trị sự kiện** | Chuỗi gọi hàm ("handleClick()") | Tham chiếu hàm (handleClick) |
| **Quản lý sự kiện** | Gắn trực tiếp vào DOM, dễ gây tốn tài nguyên | Synthetic Events, tối ưu qua Virtual DOM |
| **Hiệu suất** | Kém hơn trong ứng dụng lớn | Tốt hơn nhờ event delegation và Virtual DOM |
| **Khả năng tái sử dụng** | Khó tái sử dụng, logic tách biệt | Dễ dàng với components và hooks |
| **Đối tượng sự kiện** | Truy cập qua event trong hàm | Truy cập qua tham số (e) trong hàm |

### Lưu ý quan trọng

* **HTML truyền thống**: Phù hợp cho các trang web tĩnh hoặc ứng dụng nhỏ, nhưng khó mở rộng và bảo trì khi logic phức tạp.
* **React**: Sử dụng **Synthetic Events**, cung cấp tính nhất quán giữa các trình duyệt và hiệu suất tốt hơn. Tuy nhiên, cần học JSX và cách quản lý trạng thái.
* Trong React, luôn sử dụng tham chiếu hàm (onClick={handleClick}) thay vì gọi hàm trực tiếp (onClick={handleClick()}) để tránh gọi hàm ngay khi component render.
* Một số sự kiện như onKeyPress đã bị deprecated, nên ưu tiên onKeyDown trong các ứng dụng hiện đại.

**Chương 3. Form trong React**

### 1. ****Khái niệm về Form****

* **Định nghĩa**: Trong cả HTML và React, **form** là một phần tử giao diện dùng để thu thập dữ liệu từ người dùng. Các thành phần phổ biến của form bao gồm:
  + <input>: Nhập văn bản, số, email, mật khẩu, v.v.
  + <checkbox>: Hộp kiểm cho lựa chọn bật/tắt.
  + <select>: Danh sách thả xuống để chọn một hoặc nhiều giá trị.
  + <textarea>: Vùng nhập văn bản nhiều dòng.
  + Các thành phần khác như <radio>, <file>, v.v.
* **Mục đích**: Thu thập dữ liệu người dùng (ví dụ: thông tin đăng nhập, biểu mẫu đăng ký) và gửi dữ liệu đó (thường qua onSubmit) để xử lý (gửi lên server, lưu trữ, v.v.).
* **HTML truyền thống**:
  + Form sử dụng thuộc tính action để chỉ định URL nhận dữ liệu và method (thường là GET hoặc POST).

Ví dụ:

html

<form action="/submit" method="POST"> <input type="text" name="username" /> <button type="submit">Submit</button></form>

* + Dữ liệu được gửi trực tiếp qua HTTP request khi form được submit, không có trạng thái (state) quản lý giá trị đầu vào.
* **React**:
  + Form thường được quản lý thông qua **state** (trạng thái) trong React, sử dụng **Controlled Components** hoặc **Uncontrolled Components**.
  + React không sử dụng action hay method trực tiếp mà xử lý logic trong hàm JavaScript (thường qua onSubmit).

### 2. ****Controlled Component trong React****

* **Định nghĩa**:
  + Một **Controlled Component** là phần tử form (như <input>, <select>, <textarea>) có giá trị được điều khiển bởi **state** của React.
  + Giá trị của phần tử được liên kết với một biến state, và mọi thay đổi (do người dùng nhập) được xử lý thông qua sự kiện onChange để cập nhật state.
  + Điều này đảm bảo UI luôn đồng bộ với state, theo triết lý "single source of truth" (nguồn dữ liệu duy nhất) của React.
* **Cách hoạt động**:
  + **State** lưu trữ giá trị của phần tử form.
  + **onChange** được gắn vào phần tử để cập nhật state mỗi khi người dùng tương tác (nhập, chọn, v.v.).
  + Khi state thay đổi, React tự động re-render để hiển thị giá trị mới trên UI.
* **Ví dụ Controlled Component**:

jsx

import { useState } from 'react';function MyForm() { const [inputValue, setInputValue] = useState(''); const handleChange = (event) => { setInputValue(event.target.value); }; const handleSubmit = (event) => { event.preventDefault(); console.log('Submitted:', inputValue); }; return ( <form onSubmit={handleSubmit}> <input type="text" value={inputValue} onChange={handleChange} /> <button type="submit">Submit</button> </form> );}

* + **Giải thích**:
    - inputValue là state lưu giá trị của <input>.
    - Thuộc tính value={inputValue} liên kết giá trị của input với state.
    - onChange={handleChange} cập nhật inputValue mỗi khi người dùng nhập.
    - onSubmit={handleSubmit} xử lý khi form được gửi, với event.preventDefault() để ngăn làm mới trang.

### So sánh Form trong HTML truyền thống và Controlled Component trong React

| **Tiêu chí** | **HTML truyền thống** | **Controlled Component trong React** |
| --- | --- | --- |
| **Quản lý giá trị** | Giá trị nằm trong DOM, truy cập qua name hoặc id. | Giá trị được lưu trong state của React. |
| **Cập nhật UI** | Cập nhật thủ công qua DOM (element.value). | Tự động re-render khi state thay đổi. |
| **Sự kiện onChange** | Chỉ kích hoạt khi mất focus (ở một số trường hợp). | Kích hoạt ngay khi giá trị thay đổi. |
| **Sự kiện onSubmit** | Gửi dữ liệu qua action và method. | Xử lý logic trong hàm JavaScript, cần e.preventDefault(). |
| **Tính đồng bộ** | Không có cơ chế đồng bộ giá trị với logic. | Giá trị luôn đồng bộ với state. |
| **Tính linh hoạt** | Phù hợp cho form đơn giản, tĩnh. | Linh hoạt, dễ xử lý logic phức tạp (validation, điều kiện). |
| **Hiệu suất** | Nhanh cho ứng dụng nhỏ, nhưng khó quản lý khi phức tạp. | Tốt hơn cho ứng dụng lớn nhờ Virtual DOM. |

### Liên kết với các sự kiện Form

Các sự kiện form (onChange, onSubmit, onFocus, onBlur) đóng vai trò quan trọng trong **Controlled Component**:

* **onChange**:
  + Bắt buộc trong Controlled Component để cập nhật state mỗi khi người dùng thay đổi giá trị.
  + Ví dụ: <input onChange={(e) => setInputValue(e.target.value)} />.
* **onSubmit**:
  + Xử lý logic khi form được gửi, thường đi kèm e.preventDefault() để ngăn hành vi mặc định (làm mới trang).
  + Ví dụ: Gửi dữ liệu đến server hoặc hiển thị thông báo.
* **onFocus**:
  + Dùng để xử lý khi người dùng focus vào input, ví dụ: hiển thị gợi ý hoặc thay đổi giao diện.
  + Ví dụ: <input onFocus={() => console.log('Input focused!')} />.
* **onBlur**:
  + Dùng để xử lý khi input mất focus, thường để kiểm tra dữ liệu (validation).
  + Ví dụ: <input onBlur={() => console.log('Input blurred!')} />.

### Ưu điểm và nhược điểm của Controlled Component

* **Ưu điểm**:
  + Dễ dàng kiểm soát và xác thực dữ liệu (validation) trong state.
  + Dễ tích hợp với logic phức tạp (ví dụ: cập nhật giao diện theo giá trị nhập).
  + Đảm bảo UI và dữ liệu luôn đồng bộ.
* **Nhược điểm**:
  + Cần viết thêm code để quản lý state và sự kiện onChange, đặc biệt với form lớn có nhiều trường.
  + Có thể gây giảm hiệu suất trong form rất lớn nếu không tối ưu.

### Uncontrolled Component (so sánh ngắn gọn)

* Trong React, ngoài **Controlled Component**, còn có **Uncontrolled Component**, nơi giá trị của phần tử form không được quản lý bởi state mà được truy xuất qua **ref**.
* Ví dụ Uncontrolled Component:

jsx

import { useRef } from 'react';function UncontrolledForm() { const inputRef = useRef(); const handleSubmit = (event) => { event.preventDefault(); console.log('Submitted:', inputRef.current.value); }; return ( <form onSubmit={handleSubmit}> <input type="text" ref={inputRef} /> <button type="submit">Submit</button> </form> );}

* **Khác biệt chính**:
  + Controlled: Giá trị được lưu trong state, đồng bộ với UI.
  + Uncontrolled: Giá trị nằm trong DOM, truy xuất qua ref, ít code hơn nhưng khó kiểm soát logic phức tạp.

### Kết luận

* **Form** trong HTML truyền thống và React đều dùng để thu thập dữ liệu, nhưng React cung cấp cách tiếp cận hiện đại hơn với **Controlled Component**, nơi giá trị được quản lý bởi state, đảm bảo đồng bộ và dễ xử lý logic phức tạp.
* **Controlled Component** sử dụng onChange để cập nhật state, kết hợp với các sự kiện như onSubmit, onFocus, onBlur để tạo trải nghiệm người dùng linh hoạt.
* So với HTML truyền thống, Controlled Component trong React mạnh mẽ hơn, nhưng đòi hỏi thêm code để quản lý state. Trong trường hợp form đơn giản, **Uncontrolled Component** có thể là lựa chọn gọn nhẹ hơn.