REACTJS 01

FDP 5.0

Nội dung

- Lifecycle
- Event

LIFECYCLE

- Tìm hiểu vòng đời của một Component React (React Componenent), khi nào và sử dụng ra sao ?
- Mỗi thành phần trong React có một vòng đời mà bạn có thể theo dõi và thao tác trong ba giai đoạn chính của nó.
- Ba giai đoạn là:
  - Mounting
  - Updating
  - Unmounting

LIFECYCLE

“Render Phase”

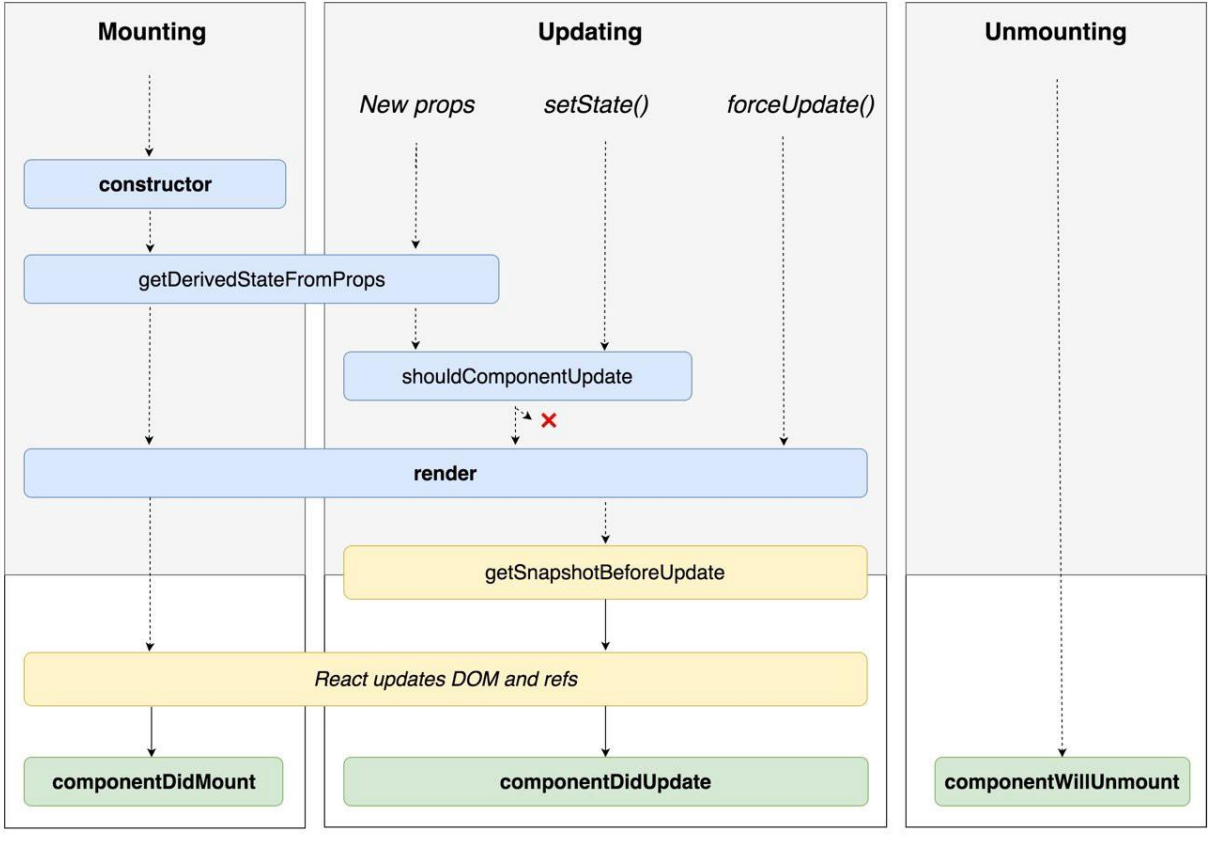
Pure and has no side effects. May be paused, aborted or restarted by React.

“Pre-Commit Phase”

Can read the DOM.

“Commit Phase”

Can work with DOM, run side effects, schedule updates.



LIFECYCLE

- Mounting có nghĩa là thêm phần tử đó vào trong DOM
- Gồm các phương thức có thứ tự như sau:
  - constructor()
  - static getDerivedStateFromProps()
  - render()
  - componentDidMount()
- Trong đó phương thức render() là bắc buộc, các phương thức còn lại sẽ sử dụng khi nào có nhu cầu

LIFECYCLE

- Constructor() sẽ được gọi đầu tiên trước bất kỳ phương thức nào khác. Hay còn gọi là phương thức khởi tạo.
- Nếu không khởi tạo state cho component thì không cần gọi phương thức này

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    );
  }
}

ReactDOM.render(<Header />, document.getElementById('root'));
```

LIFECYCLE

- static getDerivedStateFromProps() được gọi ngay trước khi gọi render()
- Rất hiếm khi sử dụng, chỉ khi giá trị của **state** phụ thuộc vào **props**
- Nó nhận vào **state** như tham số đầu vào, và return 1 object **state** đã được thay đổi.
- Nên hạn chế sử dụng hàm này, vì làm logic hiển thị của component rất khó hiểu, hãy nghĩ đến những cách implement đơn giản hơn bằng những lifecycle khác.

LIFECYCLE

Mounting - static getDerivedStateFromProps

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  static getDerivedStateFromProps(props, state) {
    return {favoritecolor: props.favcol };
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    );
  }
}

ReactDOM.render(<Header favcol="yellow"/>, document.getElementById('root'));
```

LIFECYCLE

- Render() đây là phương thức bắt buộc duy nhất khi tạo ra một component, bắt buộc trả về một trong những giá trị.
  - React element
  - Arrays and fragments
  - Portals
  - String and numbers
  - Booleans or null
- Hàm này sẽ không được gọi nếu shouldComponentUpdate() **return false**
- Khi state thay đổi hàm này sẽ được gọi lại

LIFECYCLE

Mounting - render

```
class Header extends React.Component {
  render() {
    return (
      <h1>This is the content of the Header component</h1>
    );
  }
}

ReactDOM.render(<Header />, document.getElementById('root'));
```

LIFECYCLE

Mounting - componentDidMount

- componentDidMount() được gọi sau khi 1 component đã được render ra.
- Tại component này chúng ta có thể gọi AJAX (API) ở đây.
- Các thao tác với DOM cũng được phép sử dụng tại đây.
- Có thể setState tại đây

LIFECYCLE

Mounting - componentDidMount

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({favoritecolor: "yellow"})
    }, 1000)
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    );
  }
}

ReactDOM.render(<Header />, document.getElementById('root'));
```

LIFECYCLE

- Các phương thức này sẽ được gọi khi có sự thay đổi của state hoặc props
  - static getDerivedStateFromProps()
  - shouldComponentUpdate()
  - render()
  - getSnapshotBeforeUpdate()
  - componentDidUpdate()
- Trong đó phương thức render() là bắc buộc, các phương thức còn lại sẽ sử dụng khi nào có nhu cầu

LIFECYCLE

Updating - getDerivedStateFromProps

- getDerivedStateFromProps() được gọi sau khi 1 state thay đổi.
- Được gọi trước hàm render()

LIFECYCLE

Mounting - getDerivedStateFromProps

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  static getDerivedStateFromProps(props, state) {
    return {favoritecolor: props.favcol };
  }
  changeColor = () => {
    this.setState({favoritecolor: "blue"});
  }
  render() {
    return (
      <div>
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
      <button type="button" onClick={this.changeColor}>Change color</button>
      </div>
    );
  }
}

ReactDOM.render(<Header favcol="yellow"/>, document.getElementById('root'));
```

LIFECYCLE

Updating - shouldComponentUpdate

- shouldComponentUpdate() được sinh ra nhằm cải thiện hiệu suất của React.
- Mặc định phương thức này return **true**
- Mỗi khi thay đổi state nhưng không muốn render lại thì chỉ cần return **false**

LIFECYCLE

Updating - shouldComponentUpdate

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  shouldComponentUpdate() {
    return false;
  }
  changeColor = () => {
    this.setState({favoritecolor: "blue"});
  }
  render() {
    return (
      <div>
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
      <button type="button" onClick={this.changeColor}>Change color</button>
      </div>
    );
  }
}

ReactDOM.render(<Header />, document.getElementById('root'));
```

LIFECYCLE

- render() sẽ được gọi mỗi khi component có sự thay đổi
- Nó sẽ re-render HTML ở khu vực mà thành phần HTML được tác động thay đổi dữ liệu

LIFECYCLE

Updating - render

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  changeColor = () => {
    this.setState({favoritecolor: "blue"});
  }
  render() {
    return (
      <div>
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
      <button type="button" onClick={this.changeColor}>Change color</button>
      </div>
    );
  }
}

ReactDOM.render(<Header />, document.getElementById('root'));
```

LIFECYCLE

Updating - getSnapshotBeforeUpdate

- getSnapshotBeforeUpdate() gọi ngay trước khi render xuống DOM, cho phép lấy một số thông tin của DOM (ví dụ vị trí thanh scroll), các giá trị return từ hàm này sẽ đưa cho componentDidUpdate()
- Ở phương thức này bạn có quyền truy cập vào các **props** và **state** trước khi cập nhật, có nghĩa là ngay cả sau khi cập nhật, bạn vẫn có thể kiểm tra các giá trị trước khi cập nhật.
- Nếu sử dụng phương thức này thì cần phải sử dụng thêm phương thức componentDidUpdate() nếu không sẽ phát sinh lỗi

LIFECYCLE

Updating - getSnapshotBeforeUpdate

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({favoritecolor: "yellow"})
    }, 1000)
  }
  getSnapshotBeforeUpdate(prevProps, prevState) {
    document.getElementById("div1").innerHTML =
    "Before the update, the favorite was " + prevState.favoritecolor;
  }
  componentDidUpdate() {
    document.getElementById("div2").innerHTML =
    "The updated favorite is " + this.state.favoritecolor;
  }
  render() {
    return (
      <div>
        <h1>My Favorite Color is {this.state.favoritecolor}</h1>
        <div id="div1"></div>
        <div id="div2"></div>
      </div>
    );
  }
}

ReactDOM.render(<Header />, document.getElementById('root'));
```

LIFECYCLE

Updating - componentDidUpdate

- componentDidUpdate() ngay sau khi component được cập nhập, phương thức này sẽ được gọi.
- Không gọi trong lần render đầu.
- Đây cũng có thể là nơi để tạo một network request khi chúng ta so sánh prop hiện tại với prop ở thời điểm trước đó
- Nếu muốn gọi setState ở đây, phải đưa nó trong câu điều kiện, nếu không sẽ bị lặp vô tận
- Nếu có implement phương thức getSnapshotBeforeUpdate(), giá trị return của getSnapshotBeforeUpdate() sẽ được đưa vào snapshot, nếu không thì là undefined
- Phương thức này sẽ không được gọi nếu giá trị của shouldComponentUpdate() là **false**

LIFECYCLE

Updating - componentDidUpdate

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({favoritecolor: "yellow"})
    }, 1000)
  }
  componentDidUpdate() {
    document.getElementById("mydiv").innerHTML =
    "The updated favorite is " + this.state.favoritecolor;
  }
  render() {
    return (
      <div>
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
      <div id="mydiv"></div>
      </div>
    );
  }
}

ReactDOM.render(<Header />, document.getElementById('root'));
```

LIFECYCLE

- Đây là giai đoạn cuối của 1 component khi được remove khỏi DOM.
- Chỉ có một phương thức được cấp phát trong giai đoạn này
  - componentWillUnmount()

LIFECYCLE

Unmounting - componentWillUnmount-

- Phương thức được gọi trước khi remove component khỏi DOM

LIFECYCLE

Unmounting - componentWillUnmount

```
class Container extends React.Component {
  constructor(props) {
    super(props);
    this.state = {show: true};
  }
  delHeader = () => {
    this.setState({show: false});
  }
  render() {
    let myheader;
    if (this.state.show) {
      myheader = <Child />;
    };
    return (
      <div>
      {myheader}
      <button type="button" onClick={this.delHeader}>Delete Header</button>
      </div>
    );
  }
}

class Child extends React.Component {
  componentWillUnmount() {
    alert("The component named Header is about to be unmounted.");
  }
  render() {
    return (
      <h1>Hello World!</h1>
    );
  }
}

ReactDOM.render(<Container />, document.getElementById('root'));
```

LIFECYCLE

- Ngoài ra còn 1 phương thức quan trọng là componentDidCatch().
- Thông thường khi xảy ra lỗi ở khu vực nào đó trong component nó sẽ gây ra hiện tượng chết ứng dụng
- Khi gọi phương thức này nó sẽ quăng lỗi ở đây. Từ đó có thể sử dụng để hiển thị lỗi UI nhằm che đậy lỗi bên dưới.

LIFECYCLE

componentDidCatch

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  componentDidCatch(error, info) {
    // Display fallback UI
    this.setState({ hasError: true });
    // You can also log the error to an error reporting service
    logErrorToMyService(error, info);
  }

  render() {
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return <h1>Something went wrong.</h1>;
    }
    return this.props.children;
  }
}

<ErrorBoundary>
  <MyWidget />
</ErrorBoundary>
```

EVENT

- Cũng giống như HTML, React có thể thực hiện các hành động dựa trên các sự kiện của người dùng.
- React có các sự kiện giống như HTML: nhấp chuột, thay đổi, di chuột qua, v.v.

React:

```
<button onClick={shoot}>Take the Shot!</button>
```

Run Example »

HTML:

```
<button onclick="shoot()">Take the Shot!</button>
```

EVENT

- Đối với các phương thức trong React, từ khóa this phải đại diện cho thành phần sở hữu phương thức
- Đó là lý do tại sao bạn nên sử dụng các hàm mũi tên(arrow function). Với các hàm mũi tên, điều này sẽ luôn đại diện cho đối tượng đã xác định hàm mũi tên.

EVENT

- Trong các class component, từ khóa this không được định nghĩa theo mặc định, vì vậy với các hàm thông thường, từ khóa this đại diện cho đối tượng được gọi là phương thức, có thể là đối tượng cửa sổ chung, nút HTML hoặc bất cứ thứ gì.
- Nếu bạn phải sử dụng các hàm thông thường thay vì các hàm mũi tên, bạn phải liên kết nó với cá thể thành phần bằng phương thức bind ()
- Nếu không có ràng buộc, từ khóa này sẽ trả về không xác định.

EVENT

Tại sao là arrow function

```
class Football extends React.Component {
  constructor(props) {
    super(props)
    this.shoot = this.shoot.bind(this)
  }
  shoot() {
    alert(this);
    /*
    Thanks to the binding in the constructor function,
    the 'this' keyword now refers to the component object
    */
  }
  render() {
    return (
      <button onClick={this.shoot}>Take the shot!</button>
    );
  }
}

ReactDOM.render(<Football />, document.getElementById('root'));
```

EVENT

Tham số đầu vào cho function của event

- Nếu muốn sử dụng đối số đầu vào của phương thức được gọi bằng event thì phải sử dụng arrow fucntion để thực hiện
- Hoặc nếu sử dụng 1 function bình thường thì phải **bind** từ khóa **this**. Từ khóa this sẽ là tham số đầu tiên của function

```
class Football extends React.Component {
  shoot = (a) => {
    alert(a);
  }
  render() {
    return (
      <button onClick={() => this.shoot("Goal")}>Take the shot!</button>
    );
  }
}

ReactDOM.render(<Football />, document.getElementById('root'));
```

```
class Football extends React.Component {
  shoot(a) {
    alert(a);
  }
  render() {
    return (
      <button onClick={this.shoot.bind(this, "Goal")}>Take the shot!</button>
    );
  }
}

ReactDOM.render(<Football />, document.getElementById('root'));
```

EVENT

- Khi sử lý 1 sự kiện nào đó trong React, thì cũng sẽ nhận được các thông số của sự kiện đó.
- Để lấy event của 1 sự kiện nào đó, cần truyền vào tham số đầu vào của arrow function mà gọi tới hàm được xử lý bằng sự kiện đó.
- Nếu không sử dụng arrow function thì event sẽ là đối số cuối của function được gọi.

EVENT

Event object

```
class Football extends React.Component {
  shoot = (a, b) => {
    alert(b.type);
    /*
    'b' represents the React event that triggered the function,
    in this case the 'click' event
    */
  }
  render() {
    return (
      <button onClick={(ev) => this.shoot("Goal", ev)}>Take the shot!</button>
    );
  }
}

ReactDOM.render(<Football />, document.getElementById('root'));
```

EVENT

Event object – Sử dụng function bình thường

```
class Football extends React.Component {
  shoot = (a, b) => {
    alert(b.type);
    /*
    'b' represents the React event that triggered the function,
    in this case the 'click' event
    */
  }
  render() {
    return (
      <button onClick={this.shoot.bind(this, "Goal")}>Take the shot!</button>
    );
  }
}

ReactDOM.render(<Football />, document.getElementById('root'));
```

Thank you and
happy learning !!!