Netcracker

Front-end development course

JavaScript

Lecture #9

Components lifecycle

In applications with many components, it's very important to free up resources taken by the components when they are destroyed, and for that case ***lifecycle hooks*** are implemented for React components.

Components have several groups of lifecycle methods, such as: mounting, updating,

Common lifecycle hooks:

- componentDidMount()
- componentWillUnmount()
- shouldComponentUpdate()
- componentDidCatch()

Full list of lifecycle methods and good article.

Data flow

React enforce usage of what is commonly called is “top-down” or “unidirectional” data flow. Any state is always owned by some specific component, and any data or UI derived from that state can only affect components “below” them in the tree.

The major benefit of this approach is that data flows throughout your app in a single direction and you have better control over it. Also the application state is contained in specific stores and as a result different components of your app remain loosely coupled.

Event handling

Handling events with React elements is very similar to handling events on DOM elements. There are some syntactic differences:

```
function handleClick(e) {
    e.preventDefault();
  }

<a href="#" onClick={handleClick}>Click me</a>;
```

When using Class components it is common in React to define events handlers using experimental ES “Class field”syntax.

Guide to handling events here.

Arrays of element and components

You can easily render list of React elements of components just by using JavaScript expressions and array extra methods, like so:

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li>{number}</li>
);
ReactDOM.render(<ul>{listItems}</ul>, document.getElementById('root'));
```

Virtual DOM

All nowadays frameworks are trying to implement declarative approach - it means that we describe how UI should look depending on some circumstances.

React doesn't translate immediately everything that your component method **render** returns, instead it builds ReactElement (Virtual DOM node) from that result (Virtual DOM Node is just an JS object).

Virtual DOM - is just a special technique, when underlying framework or library builds real DOM from plain JS object with maximum efficiency. Good article on that topic.

Homework

- You should create a repository for you final assignment (or use an existing one).
- You should configure your development environment with parcel and babel according to guide.
- You should think of you final project theme. (some ideas can be found here https://github.com/toddmotto/public-apis)

Q&A

Thank You