Лекция 11 - Компонентный подход в React.js

Тлеубаева А.О., магистр технических наук, ст.преподаватель

Введение

React.js - это одна из самых популярных JavaScript библиотек для создания пользовательских интерфейсов. Основываясь на компонентном подходе, React позволяет строить сложные интерфейсы из небольших, независимых и повторно используемых компонентов.

React, полное название - React.js, является одной из самых популярных и широко используемых библиотек JavaScript для разработки пользовательских интерфейсов, особенно для веб-приложений. Она была создана инженерами Facebook и впервые представлена в 2013 году. С тех пор React стал фундаментальным инструментом в современной веб-разработке.

Ключевые особенности React:

Компонентный подход: React строится на концепции компонентов. Компоненты - это переиспользуемые и независимые блоки кода, которые могут включать в себя как логику, так и разметку. Они помогают создавать сложные интерфейсы, делая код более управляемым и модульным.

Декларативное программирование: React позволяет описывать, как компоненты интерфейса должны выглядеть и вести себя, а затем сам заботится об их отрисовке и обновлении. Это делает код более понятным и упрощает процесс разработки.

Virtual DOM (Виртуальный DOM): React использует концепцию Виртуального DOM, что позволяет оптимизировать обновление интерфейса. Вместо того чтобы обновлять весь DOM при каждом изменении, React сначала применяет изменения к Виртуальному DOM, а затем эффективно обновляет только те части реального DOM, которые изменились.

JSX (JavaScript XML): В React используется JSX, расширение синтаксиса для JavaScript, которое позволяет писать компоненты интерфейса в стиле, близком к HTML, непосредственно в JavaScript-коде. Это упрощает создание шаблонов и повышает читаемость кода.

Однонаправленный поток данных: В React данных следуют от родительских компонентов к дочерним. Это делает поток данных более предсказуемым и упрощает отладку приложений.

Хуки: С введением хуков в React 16.8, функциональные компоненты получили возможность использовать состояние и другие возможности React, ранее доступные только в классовых компонентах.

Преимущества использования React:

Быстродействие: Благодаря оптимизированному обновлению DOM, приложения на React работают быстро и плавно.

Масштабируемость: Компонентная архитектура делает React идеальным для создания больших и сложных приложений.

Большое сообщество и экосистема: React имеет огромное сообщество разработчиков и богатую экосистему инструментов, библиотек и расширений.

Гибкость: React может использоваться в сочетании с другими библиотеками и фреймворками.

Использование в реальных проектах:

React нашел широкое применение в разработке веб-приложений, от небольших личных проектов до крупномасштабных корпоративных систем. Крупные компании, такие как Facebook, Instagram, Netflix и Airbnb, используют React для создания своих продуктов.

Основы React.js

Что такое React?

React - это открытая JavaScript библиотека, созданная Facebook для разработки интерактивных пользовательских интерфейсов.

Основные Принципы React

Декларативность: React позволяет описывать, как компоненты интерфейса должны выглядеть и вести себя, а библиотека заботится об эффективном обновлении и рендеринге компонентов.

Компонентный подход: Интерфейс разбивается на мелкие, независимые части - компоненты, которые можно легко переиспользовать.

Компонентный Подход

Что такое Компоненты?

Компоненты в React - это независимые и повторно используемые части вашего интерфейса, написанные как функции или классы JavaScript.

Классовые и Функциональные Компоненты

Классовые компоненты: Более старая форма компонентов, поддерживающая состояние и жизненный цикл.

Функциональные компоненты: Более современный и лаконичный способ создания компонентов, используется совместно с хуками для управления состоянием и эффектами.

Примеры Компонентов

Создание Простого Компонента

Пример функционального компонента "Приветствие":

```
function Greeting({ name }) {
  return <h1>Привет, {name}!</h1>;
}
```

Композиция Компонентов

Компоненты могут включать другие компоненты в своём выводе, что позволяет создавать сложные интерфейсы.

Состояние и Жизненный Цикл

Управление Состоянием

Состояние компонента управляется с помощью хука useState в функциональных компонентах.

Жизненный Цикл Компонента

Классовые компоненты предоставляют методы жизненного цикла (например, componentDidMount), позволяющие выполнять действия в определённые моменты.

Управление Состоянием в React

Состояние в React - это объект, который содержит данные, влияющие на отрисовку компонента. Изменения состояния приводят к повторной отрисовке компонента.

useState в Функциональных Компонентах

useState - это хук, введенный в React 16.8, который позволяет функциональным компонентам использовать внутреннее состояние.

```
import React, { useState } from 'react';

function Example() {
  // Объявляем новую переменную состояния "count"
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>Вы нажали {count} раз</p>
      <button onClick={() => setCount(count + 1)}>
        Нажми на меня
      </button>
    </div>
  );
}
```

В этом примере useState инициализируется значением 0 (начальное состояние). count используется для отображения состояния, а setCount для его обновления.

Классовые Компоненты

В классовых компонентах состояние управляется через this.state и обновляется с помощью this.setState.

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  render() {
    return (
      <div>
        <p>Вы нажали {this.state.count} раз</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Нажми на меня
        </button>
      </div>
    );
  }
}
```

Жизненный Цикл Компонента

Жизненный цикл компонента - это серия методов, которые автоматически вызываются в процессе создания, обновления и уничтожения компонента.

Основные Методы Жизненного Цикла

Mounting (Монтирование):

- constructor(): Инициализация состояния и привязка методов.
- componentDidMount(): Вызывается сразу после монтирования компонента. Идеальное место для запросов к API, подписок и т.д.

Updating (Обновление):

- shouldComponentUpdate(): Определяет, должен ли компонент обновляться.
- componentDidUpdate(): Вызывается сразу после обновления компонента. Не вызывается после первого рендера.

Unmounting (Размонтирование):

- componentWillUnmount(): Вызывается перед удалением компонента из DOM. Используется для очистки подписок, таймеров и т.д.

Управление Событиями

Управление событиями в React - это важная часть создания интерактивных веб-приложений. React обрабатывает события похожим на JavaScript способом, но с некоторыми отличиями и особенностями, характерными для его экосистемы.

Как Работает Управление Событиями в React

1. **Обработчики Событий**: В React события обрабатываются с помощью обработчиков событий, которые являются функциями JavaScript. Эти функции прикрепляются к элементам JSX с использованием атрибутов, подобных onClick, onChange, onSubmit и так далее.
2. **Синтетические События**: React оборачивает события браузера в собственные "синтетические события" (Synthetic Events). Это обеспечивает кросс-браузерную совместимость событий, то есть события будут работать одинаково во всех браузерах.
3. **Передача Данных**: Часто в обработчик события необходимо передать дополнительные данные. В React это можно сделать, используя стрелочные функции или метод bind.

- **Примеры Кода**

1.Обработка Клика:

```
function MyComponent() {
  function handleClick() {
    console.log('Кнопка нажата');
  }

  return <button onClick={handleClick}>Нажми на меня</button>;
}
```

- **Обработка Изменения в Инпуте**:

```
function MyForm() {
  function handleChange(event) {
    console.log('Введенное значение: ', event.target.value);
  }

  return <input type="text" onChange={handleChange} />;
}
```

- **Передача Параметров в Обработчик Событий**:

```
function MyComponent() {
  function handleClick(id) {
    console.log('Выбранный ID: ', id);
  }

  return <button onClick={() => handleClick(1)}>Нажми на меня</button>;
}
```

Особенности

Использование camelCase: В React имена событий пишутся в стиле camelCase, а не в нижнем регистре, как в обычном HTML (например, onClick вместо onclick).

e.preventDefault(): В React, чтобы предотвратить поведение по умолчанию, используется e.preventDefault(), как и в обычном JavaScript.

Пул событий: React использует пул событий для повышения производительности. Это значит, что объект события, передаваемый в ваш обработчик, будет переиспользован. Если вам нужно сохранить событие для асинхронного доступа, вызовите event.persist().

Управление событиями в React - это мощный инструмент для создания интерактивных интерфейсов. Он обеспечивает удобный способ обработки пользовательских взаимодействий, сохраняя при этом консистентность и кросс-браузерную совместимость.

Хуки в Функциональных Компонентах

Хуки, такие как useEffect, предоставляют функциональность жизненного цикла в функциональных компонентах.

```
import React, { useState, useEffect } from 'react';

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

  // Аналогично componentDidMount и componentDidUpdate:
  useEffect(() => {
    // Обновляем заголовок документа с помощью API браузера
    document.title = `Вы нажали ${count} раз`;
  });

  return (
    <div>
      <p>Вы нажали {count} раз</p>
      <button onClick={() => setCount(count + 1)}>
        Нажми на меня
      </button>
    </div>
  );
}
```

```
import React, { useState, useEffect } from 'react';

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

  // Аналогично componentDidMount и componentDidUpdate:
  useEffect(() => {
    // Обновляем заголовок документа с помощью API браузера
    document.title = `Вы нажали ${count} раз`;
  });

  return (
    <div>
      <p>Вы нажали {count} раз</p>
      <button onClick={() => setCount(count + 1)}>
        Нажми на меня
      </button>
    </div>
  );
}
```

В этом примере useEffect выполняется после каждого рендеринга, обеспечивая функциональность, аналогичную componentDidMount и componentDidUpdate.

Итого

- Управление состоянием и понимание жизненного цикла компонентов в React - ключевые аспекты для создания динамичных и отзывчивых веб-приложений. Благодаря хукам, функциональные компоненты теперь могут использовать такие же возможности, как и классовые, делая код более компактным и читаемым.

Практические Примеры в React

Работа с Формами

В React формы обычно управляются через состояние компонента. Для каждого поля формы обычно создается свое состояние. Это позволяет легко отслеживать изменения и управлять отправкой формы.

Пример: Управляемый Компонент

```
import React, { useState } from 'react';

function MyForm() {
  const [name, setName] = useState('');

  const handleSubmit = (event) => {
    event.preventDefault();
    alert('Отправленное имя: ' + name);
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Имя:
        <input type="text" value={name} onChange={(e) => setName(e.target.value)} />
      </label>
      <button type="submit">Отправить</button>
    </form>
  );
}
```

В этом примере состояние name используется для хранения значения инпута. Каждый раз, когда пользователь вводит символ, состояние обновляется через setName.

Использование API

Для взаимодействия с внешними API, React предоставляет хук useEffect, который позволяет выполнять побочные эффекты в функциональных компонентах, включая загрузку данных.

Пример: Загрузка Данных из API

```
import React, { useState, useEffect } from 'react';

function MyComponent() {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch('https://api.example.com/data')
      .then(response => response.json())
      .then(data => setData(data))
      .catch(error => console.error('Ошибка при загрузке данных:', error));
  }, []); // Пустой массив зависимостей означает, что эффект запустится только
при монтировании компонента

  return (
    <div>
      {data ? <div>{data.someField}</div> : <div>Загрузка...</div>}
    </div>
  );
}
```

В этом примере используется useEffect для выполнения запроса к API при первом рендере компонента. Данные сохраняются в состоянии data после успешного получения ответа.

Заключение

- React.js, безусловно, оказал значительное влияние на мир веб-разработки. Его компонентный подход революционизировал способ создания пользовательских интерфейсов. Этот подход позволяет разработчикам разбивать сложные интерфейсы на меньшие, независимые и повторно используемые компоненты, что облегчает разработку, тестирование и поддержку приложений.
- Управление состоянием и жизненным циклом компонентов в React позволяет создавать динамичные и интерактивные пользовательские интерфейсы. С помощью хуков в функциональных компонентах и классовых методов жизненного цикла, разработчики могут точно контролировать рендеринг и поведение компонентов, реагируя на изменения данных и контекста приложения.
- Кроме того, React способствует написанию декларативного кода, что делает приложения более предсказуемыми и упрощает отладку. Его совместимость с различными библиотеками и инструментами, а также мощная экосистема делают React одним из лучших выборов для разработки современных веб-приложений.
- В заключение, React.js - это мощный и гибкий инструмент для разработки веб-приложений, который постоянно развивается и адаптируется к новым трендам и потребностям индустрии, поддерживая при этом огромное сообщество разработчиков и множество успешных проектов по всему миру.