Интенсив-курс по React JS

www.andersenlab.com

Telegram channel

Программа курса:

1. Основы React
2. Продвинутый React
3. State-менеджеры
4. Типизация
5. Тестирование
6. Code style
7. Процесс и методологии разработки
8. Подведение итогов. Подготовка к срезу знаний

УРОК 1

Основы REACT

www.andersenlab.com

Telegram channel

Темы урока:

1. React. Введение
2. Настройка окружения
3. JSX
4. Типы компонентов
5. State, Props, Ref
6. Обработка событий
7. Virtual DOM
8. Жизненный цикл компонентов

React. Введение

React - это декларативная JavaScript библиотека для создания пользовательских интерфейсов.

Основные концепции:

- *компонентный подход;*
- *однонаправленный поток данных;*
- *“виртуальный” DOM;*

Преимущества:

- *переиспользование компонентов;*
- *возможность хранения данных в стейте компонента;*
- *быстрота (благодаря Virtual DOM);*
- *простота (из-за JSX);*

```
import React from "react";
import ReactDOM from "react-dom";

class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState( state: state => ({
      seconds: state.seconds + 1
    }));
  }

  componentDidMount() {
    this.interval = setInterval( handler: () => this.tick(),  timeout: 1000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    return (
      <div>
        Секунды: {this.state.seconds}
      </div>
    );
  }
}

ReactDOM.render(
  <Timer />,
  document.getElementById( elementId: 'timer-example')
);

```

Настройка окружения

Если вы хотите быстро создать новое React приложение, используйте Create React App.

Для этого вам необходимо сначала установить Node.js. Рекомендуется устанавливать последнюю LTS версию.

(1)

Данный инструмент настраивает среду для использования новейших возможностей JavaScript, оптимизирует приложение для продакшена и обеспечивает комфорт во время разработки.

```
MINGW64:/c/Users/borov/OneDrive/Desktop/FrontEnd/Work/Andersen_Courses

borov@DESKTOP-U3H224E MINGW64 ~/OneDrive/Desktop/FrontEnd/Work/Andersen_Courses
$ npx create-react-app <your-app-name>
```

(2)

```
MINGW64:/c/Users/borov/OneDrive/Desktop/FrontEnd/Work/Andersen_Courses

Created git commit.

Success! Created your-app-name at C:\Users\borov\OneDrive\Desktop\FrontEnd\Work\
Andersen_Courses\your-app-name
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd your-app-name
  npm start

Happy hacking!

borov@DESKTOP-U3H224E MINGW64 ~/OneDrive/Desktop/FrontEnd/Work/Andersen_Courses
$
```

(3)

JSX

JSX — это расширение языка JavaScript (синтаксический сахар). Который напоминает язык шаблонов, наделённый силой JavaScript. С помощью JSX описывать UI компонента стало куда проще.

JSX компилируется с помощью Bable в React.createElement().

Компонент с JSX:

```
import React from "react";
import ReactDOM from "react-dom";

class Hello extends React.Component {
  render() {
    return <div>Hello {this.props.toWhat}</div>;
  }
}

ReactDOM.render(
  <Hello toWhat="World" />,
  document.getElementById( elementId: 'root')
);

```

Компонент без JSX:

```
import React from "react";
import ReactDOM from "react-dom";

class Hello extends React.Component {
  render() {
    return React.createElement(
      type: 'div',
      props: null,
      children: `Hello ${this.props.toWhat}`
    );
  }
}

ReactDOM.render(
  React.createElement(Hello, props: {toWhat: 'World'}, children: null),
  document.getElementById( elementId: 'root')
);

```

Типы компонентов

Компоненты в React по спобобу создания разделяют на *классовые* – объявленны как классы. И *функциональные* – как функции.

Так же компоненты разделяют на *stateless* – компоненты которые не несут в себе какой-либо логики, кроме отрисовки. И *statefull* – компоненты которые помимо отрисовки несут в себе какую-то логику (например обработку событий или получение данных с сервера) имеют состояние.

По типу управления компоненты бывают *управляемыми* и *неуправляемыми*. В управляемых компонентах данные форм (form, input) обрабатываются самим компонентом (с помощью обработчиков) и они хранятся в state компонента. В неуправляемых компонентах данных формы храняться в DOM и вместо того чтобы писать обработчик события для каждого поля input мы можем считывать эти данные прямо из DOM с помощью Ref (React.createRef() или useRef()).

Классовые компоненты

- Используют методы жизненных циклов.
- Для управления состоянием используют *this.setState(state, [callback])*.
- Более сложные для понимания новичкам.
- Считаются устаревшими. Все еще встречаются в проектах. Комманда React разработчиков рекомендует писать новый код на функциональных компонентах.

```jsx
import React from "react";
import ReactDOM from "react-dom";

class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  componentDidMount() {
    this.timerID = setInterval(
      handler: () => this.tick(),
      timeout: 1000
    );
  }

  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  tick() {
    this.setState( state: {
      date: new Date()
    });
  }

  render() {
    return (
      <div>
        <h1>Привет, мир!</h1>
        <h2>Сейчас {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}

```

Функциональные компоненты

- Используют Hooks (Хуки).
- Более простые для понимания.
- Конечный размер бандла приложения меньше чем на компонентах построенных с помощью классов.

```jsx
import React, {useEffect, useState} from "react";
import ReactDOM from "react-dom";

function Clock() {
  const [date, setDate] = useState( initialState: { date: new Date() });

  useEffect( effect: () => {
    const timerId = setInterval(
      handler: () => tick(),
      timeout: 1000
    );

    return () => {
      clearInterval(timerId);
    }
  }, deps: []);

  const tick = () => {
    setDate( value: {
      date: new Date()
    });
  }

  return (
    <div>
      <h1>Привет, мир!</h1>
      <h2>Сейчас {date.toLocaleTimeString()}.</h2>
    </div>
  );
}
```

State

Компоненты в React могут иметь свой state для хранения в нем состояния.

Состоянием компонента можно управлять с помощью функции *this.setState(state, [callback])*.

Данная функция добавляет в очередь изменения состояния компонента. Так же она указывает React, что компонент и его дочерние элементы должны быть повторно отрисованы с уже обновленным состоянием.

Функция *this.setState(state, [callback])* является ассинхронной.

```
import React from "react";
import ReactDOM from "react-dom";

class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  componentDidMount() {
    this.timerID = setInterval(
      handler: () => this.tick(),
      timeout: 1000
    );
  }

  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  tick() {
    this.setState( state: {
      date: new Date()
    });
  }

  render() {
    return (
      <div>
        <h1>Привет, мир!</h1>
        <h2>Сейчас {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}

```

Особенности использования setState(state, [callback])

1. Первый аргумент setState() может быть как объектом нового состояния так и функцией. Если ваше новое состояние основывается на значении предыдущего состояния, то во избежание багов всегда передавайте в качестве первого аргумента функцию.
2. Функция, как первый аргумент this.setState(), в качестве аргументов получает предыдущее состояние state и обновленные props компонента. Для того чтобы произошла перерисовка компонента данная функция должна вернуть объект с новым состоянием.
3. Если нам необходимо по тем или иным причинам избежать перерисовки, то данная функция должна вернуть *null*.
4. Вторым аргументом this.setState() принимает callback функцию которая ввыполниться сразу после завершения обновления состояния.
5. Никогда не вызывайте this.setState() в конструкторе – иначе вы получите ошибку «Can only update a mounted or mounting component». Поэтому для инициализации переменных внутри конструктора следует использовать только this.state.

Props

Компоненты в React могут иметь свойства (props). Пропсы это данные которые родительский компонент может передать дочернему.

Пропсы нельзя изменять и мутировать. В React есть одно обязательное правило - *компоненты обязаны вести себя как чистые функции по отношению к своим пропсам*.

```jsx
import ReactDOM from "react-dom";

function Avatar(props) {
  return (
    <img
      className="Avatar"
      src={props.user.avatarUrl}
      alt={props.user.name}
    />
  );
}

function UserInfo(props) {
  return (
    <div className="UserInfo">
      <Avatar user={props.user} />
      <div className="UserInfo-name">{props.user.name}</div>
    </div>
  );
}

function Comment(props) {
  return (
    <div className="Comment">
      <UserInfo user={props.author} />
      <div className="Comment-text">{props.text}</div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}

const comment = {
  date: new Date(),
  text: 'I hope you enjoy learning React!',
  author: {
    name: 'Hello Kitty',
    avatarUrl: 'https://placekitten.com/g/64/64',
  },
};
ReactDOM.render(
  <Comment
    date={comment.date}
    text={comment.text}
    author={comment.author}
  />,
  document.getElementById( elementId: 'root')
);
```

Ref (reference, ссылки)

С помощью Ref'ов мы можем получить прямой доступ к DOM объекту, либо к компоненту.

Ref'ы часто использую для того чтобы имплеме-нтировать такой функционал как:

- Фокус на указанный DOM объект, выделение текста или воспроизведение медиа.
- Скрол (например scrollToTop)
- Чтобы получить текущие размеры DOM объекта с помощью свойст offsetWidth и offsetHeight

```
import React from "react";

class CustomTextInput extends React.Component {
  constructor(props) {
    super(props);
    this.textInput = React.createRef();
    this.focusTextInput = this.focusTextInput.bind(this);
  }

  focusTextInput() {
    this.textInput.current.focus();
  }

  render() {
    return (
      <div>
        <input
          type="text"
          ref={this.textInput} />
        <input
          type="button"
          value="Фокус на текстовом поле"
          onClick={this.focusTextInput}
        />
      </div>
    );
  }
}
```

Ref (reference, ссылки)

При передаче ссылки ref классовому компоненту, мы получаем доступ к его инстансу. И таким образом, например, можем вызывать его методы из-под родительского компонента.

```jsx
import React from "react";

export class Parent extends React.Component {
  constructor(props) {
    super(props);
    this.child = React.createRef();
  }

  onClick = () => {
    this.child.current.getAlert();
  };

  render() {
    return (
      <div>
        <Child ref={this.child} />
        <button onClick={this.onClick}>Click</button>
      </div>
    );
  }
}

class Child extends React.Component {
  getAlert() {
    alert('getAlert from Child');
  }

  render() {
    return <h1>Hello</h1>;
  }
}
```

Особенности использования ссылок (Ref)

- Для создания ссылки в классовых компонентах мы используем React.createRef(). В функциональных – хук useRef();
- При передачи ссылки в свойство ref наш узел DOM или инстанс компонента будет находиться в поле .current;
- Для перенаправления ссылки дочернему компоненту используем forwardRef((props, ref) => ComponentLogic);
- У ссылок есть одна малоизвестная особенность. В них можно хранить любые актуальные значения к которым нам необходим достп внутри замыкания. Как известно, при создании замыкания, при обращении внутри него к обычной переменной, мы получим состояние переменной на момент создания замыкания. Что в некоторых случях неудобно. Ссылки ломают эту логику и с помощью них мы можем получить доступ к актуальному значению (например свойства или состояния) в любое время;

Ref (reference, ссылки)

Пример неуправляемого компонента. Состояние данного input'а храниться в самом DOM объекте. Обработка события изменения value в данном случае ложется на сам браузер.

Мы только считываем значение с помощью ссылки (ref).

```jsx
import React from "react";

class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.handleSubmit = this.handleSubmit.bind(this);
    this.input = React.createRef();
  }

  handleSubmit(event) {
    alert('Отправленное имя: ' + this.input.current.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Имя:
          <input type="text" ref={this.input} />
        </label>
        <input type="submit" value="Отправить" />
      </form>
    );
  }
}
```

Обработка событий

Обработка событий в React-элементах очень похожа на обработку событий в DOM-элементах.

Но есть несколько синтаксических отличий:

- события в React именуются в стиле camelCase вместо нижнего регистра.
- с JSX вы передаёте функцию как обработчик события вместо строки.

Вместо того чтобы привязывать контекст явно с помощью .bind(), можно воспользоваться особенностями стрелочных функций и объявлять методы компонента используя их.

```
import React from "react";

class MyInput extends React.Component {
  constructor(props) {
    super(props);
    this.handleChange = this.handleChange.bind(this);
    this.state = {
      input: ''
    }
  }

  handleChange(event) {
    this.setState( state: () => ({
      input: event.target.value
    }));
  }

  componentDidUpdate(prevProps, prevState, snapshot) {
    console.log(this.state.input);
  }

  render() {
    return (
      <label>
        Имя:
        <input
          type="text"
          onChange={this.handleChange}
          value={this.state.input}
        />
      </label>
    );
  }
}
```

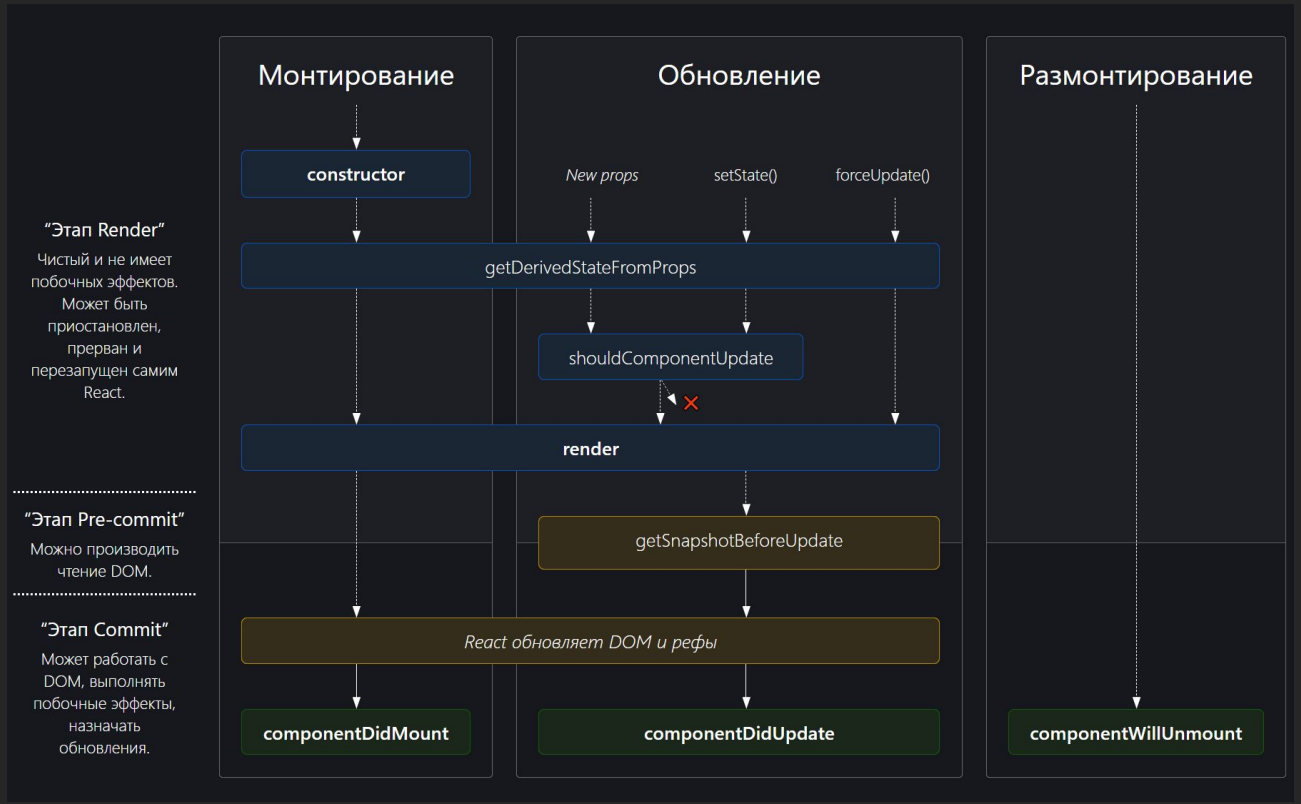
Virtual DOM

Виртуальный DOM (VDOM) — это концепция программирования, в которой идеальное или «виртуальное» представление пользовательского интерфейса хранится в памяти и синхронизируется с «настоящим» DOM при помощи библиотеки, такой как ReactDOM. Это процесс называется согласованием.

Когда мы меняем какой-то компонент и он должен перерисоваться, то вначале изменения вносят в VDOM, после чего происходит сравнение(согласование) с реальным DOM и перерендаривается лишь изменившаяся часть.

Самый последний механизм согласования был введен в React 16 версии, и несет имя Fiber.

Стадии жизненного цикла компонента



Методы жизненного цикла компонента

1) *getDerivedStateFromProps* - запускается перед вызовом метода render() и при каждом повторном рендеринге. Он используется в редких случаях, когда нам требуется производное состояние. Для получения более подробной информации смотрите если вам требуется производное состояние.
2) *componentDidMount* - выполняется после первого рендеринга, здесь выполняются AJAX-запросы, обновляется DOM или состояние компонента, регистрируются обработчики событий.
3) *shouldComponentUpdate* - определяет, должен ли компонент обновляться. Значением по умолчанию является true. Если вы уверены в том, что компонент не нуждается в повторном рендеринге при изменении состояния или пропов, тогда можете вернуть ложное значение. Это подходящее место для улучшения производительности, позволяющее предотвратить ненужные рендеринги при получении компонентом новых пропов.
4) *getSnapshotBeforeUpdate* - выполняется перед применением результатов рендеринга к DOM. Любое значение, возвращенное этим методом, передается в componentDidUpdate(). Это может быть полезным для получения информации из DOM, например, позиции курсора или величины прокрутки.
5) *componentDidUpdate* - в основном, используется для обновления DOM в соответствии с изменением состояния или пропов. Не выполняется, если shouldComponentUpdate() возвращает false.
6) *componentWillUnmount* - используется для отмены сетевых запросов или удаления обработчиков событий, связанных с компонентом.

Дополнительные материалы для самостоятельного изучения

1. Понимание жизненного цикла React-компонента
2. Подробный обзор React Fiber
3. Медленнее, плавнее: разбираемся с React Fiber
4. Как и почему React использует связные списки в архитектуре Fiber

Спасибо за ваше внимание!

www.andersenlab.com

Telegram channel