Java Script

NEXT

План

Основи популярної бібліотеки

Створення зборки

використання React на прикладах

В даному розділі ми оглянемо документацію React та пов’язані з ним ресурси.

React — це JavaScript-бібліотека для створення інтерфейсів користувача.
Відвідайте нашу головну сторінку або вступ, аби скласти перше враження про React. - https://uk.reactjs.org/docs/getting-started.html

Для навчання та маленьких/середніх проектів рекомендується використовувати утиліту від авторів React.
Абстрагує всю конфігурацію, дозволяючи зосередитись на написанні коду
Включає необхідні інструменти: Webpack, Babel, ESLint тощо.
Розширюється додатковими пакетами із екосистеми React
Має функцію вилучення, яка видаляє абстракцію та відкриває конфігурацію

для створення збірки:

- npx create-react-app *ім*я_папки_проекту

React — це декларативна, ефективна і гнучка JavaScript-бібліотека, призначена для створення інтерфейсів користувача. Вона дозволяє компонувати складні інтерфейси з невеликих окремих частин коду — “компонентів”.

Найменший приклад React виглядає наступним чином:

```
ReactDOM.render(<h1>Hello, world!</h1>,
  document.getElementById('root'));
```

На сторінці з’явиться заголовок “Hello, world!”

React-елементи – це найменші будівельні блоки React, елементи Virtual DOM. Елементи – це звичайні JS-об'єкти, тому створювати їх дуже швидко.

Функція React.createElement() це найголовніший метод, що надається React API. Подібно до document.createElement() для DOM, React.createElement() це функція для створення React-елементів. Повертає об'єкт елемент Virtual DOM.

```
React.createElement(type, [props], [...children])
```

type - ім'я вбудованого React-елемента який у Virtual DOM відповідає майбутньому HTML-тегу.

props - об'єкт містить HTML-атрибути та кастомні властивості. Можливо, null або порожній об'єкт, якщо передавати нічого не потрібно.

children - довільна кількість аргументів після другого - це діти створюваного елемента. Так створюється дерево елементів

Компоненти - основні будівельні блоки React-програм, за допомогою яких інтерфейс ділиться розділити на незалежні частини.

Розробник створює невеликі компоненти, які можна поєднувати, щоб сформувати більші або використовувати їх як самостійні елементи інтерфейсу. Найголовніше в цій концепції те, що і великі, і маленькі компоненти можна використати повторно і в поточному, і в новому проекті.

Елемент описує те, що ви хочете бачити на екрані:

const element = <h1>Привіт, світе</h1>;
На відміну від DOM-елементів, елементи React — звичайні об’єкти, легкі для створення. React DOM бере на себе оновлення DOM для його відповідності React-елементам.

Оновлення відображеного елемента

React-елементи є незмінними. Як тільки елемент створений, ви не можете змінювати його дочірні елементи чи атрибути. Елемент схожий на кадр із фільму: він відображає інтерфейс користувача в певний момент часу.

З нашими поточними знаннями, єдиний спосіб оновити інтерфейс користувача — створити новий елемент і передати його в ReactDOM.render().
Розглянемо наступний приклад годинника:

```
function tick() {const element = (<div>
<h1>Hello, world!</h1>
<h2>Зараз {new Date().toLocaleTimeString()}.</h2>
</div>);
ReactDOM.render(element, document.getElementById('root'));}setInterval(tick, 1000);
```

Він щосекунди викликає ReactDOM.render() у функції зворотнього виклику setInterval().

Примітка:
На практиці, більшість React-додатків викликає ReactDOM.render() лише раз. У наступних розділах ми дізнаємось, як такий код інкапсулюється в компоненти зі станом.
Ми рекомендуємо вам не пропускати ці теми, тому що вони залежать одна від одної.

Компоненти-функції

У найпростішій формі компонент це JavaScript-функція з дуже простим контрактом: функція отримує об'єкт властивостей, який називається props і повертає дерево React-елементів.

Ім'я компонента обов'язково має починатися з великої літери. Назви компонентів із маленької літери зарезервовані для HTML-елементів. Якщо ви спробуєте назвати компонент card, а не Card, при рендері, React проігнорує його та відрендерує тег <card></card>.

Властивості компонента (props)

Властивості (пропси) є однією з основних концепцій React. Компоненти приймають довільні властивості і повертають React-елементи, що описують, що має відрендеритися в DOM.

Пропси використовуються передачі даних від батька до дитини.
Пропси передаються лише по дереву від батьківського компонента.
При зміні пропсів React ререндерить компонент і, можливо, оновлює DOM.
Пропси доступні лише для читання, змінити їх у дитині не можна.
Пропс може бути текст кнопки, картинка, url, будь-які дані для компонента.
Пропси можуть бути рядками або результатом JS-виразу. Якщо передано лише ім'я пропсу - це буль, за замовчанням true.

Властивість props.children

Концепція дочірніх елементів дозволяє просто робити композицію компонентів.
У вигляді дітей можна передавати компоненти як вбудовані так і кастомні. Це дуже зручно під час роботи зі складними складовими компонентами.
Властивість children автоматично доступна в кожному компоненті, його вмістом є те, що стоїть між відкриваючим та закриваючим JSX-тегом.
У функціональних компонентах звертаємось як props.children.
Значенням props.children може бути практично будь-що.

Властивість defaultProps

Що якщо компонент чекає на якесь значення, а його не передали? - при зверненні до якості об'єкта props, отримаємо undefined.
Для того щоб вказати значення властивостей за замовчуванням, компоненти мають статичну властивість defaultProps, в якій можна вказати об'єкт з дефолтними значеннями пропів (не обов'язково всіх). Цей об'єкт буде злитий з об'єктом props, що прийшов.

```
const Product = ({ imgUrl, name, price }) => (
  <div>
    <img src={imgUrl} alt={name} width="640" />
    <h2>{name}</h2>
    <p>Price: {price}$</p>
    <button type="button">Add to cart</button>
  </div>
);
Product.defaultProps = {
  imgUrl:
    'https://тут_посилання_на_фото',
};
```

Рендер за умовою

Для рендеру розмітки за умовою використовуються оператори розгалужень та умов. Умови можна перевіряти перед поверненням розмітки або прямо в JSX. Якщо за умовою нічого не повинно бути відрендеровано, можна повернути null, undefined чи false, вони не рендеряться.

if за допомогою логічного оператора &&

```
const Mailbox = ({ unreadMessages }) => (
  <div>
    <h1>Hello!</h1>
    {unreadMessages.length > 0 && (
      <p>You have {unreadMessages.length} unread messages.</p>
    )}
  </div>
);
```

if...else за допомогою тернарного оператора

```
const Mailbox = ({ name, unreadMessages }) => (
  <div>
    <h1>Hello {name}.</h1>
    {unreadMessages.length > 0 ? (
      <p>You have {unreadMessages.length} unread messages.</p>
    ) : (
      <p>No unread messages.</p>
    )}
  </div>
);
```

або

```
const Mailbox = ({ name, unreadMessages }) => (
  <div>
    <h1>Hello {name}.</h1>
    <p>
      {unreadMessages.length > 0
        ? `You have ${unreadMessages.length} unread messages.`
        : 'No unread messages.'}
    </p>
  </div>
);
```

GENIUS space

Ознайомлення з Хуками

Хуки — це новинка в React 16.8. Вони дозволяють вам використовувати стан та інші можливості React без написання класу.

import React, { useState } from 'react';function Example()

{ // Створюємо нову змінну стану, яку назвемо "count"
const [count, setCount] = useState(0);return (
Ви натиснули {count} разів
<button onClick={() => setCount(count + 1)}> Натисни мене </button>
 );}

Нова функція useState є першим “хуком”, про який ми дізнаємося більше. Цей приклад є лише тизером. Не хвилюйтеся, якщо вам поки що нічого не зрозуміло! Почати вивчення хуків можна на наступній сторінці. На цій ми пояснимо, чому додаємо хуки до React та як вони можуть допомогти у написанні чудових застосунків.

Примітка

React 16.8.0 — це перший реліз, який підтримує хуки. При оновленні не забудьте оновити всі бібліотеки, включаючи React DOM. React Native підтримує хуки з релізу версії 0.59.

На React Conf 2018 Софі Алперт (Sophie Alpert) та Ден Абрамов (Dan Abramov) презентували хуки, а потім Райан Флоренс (Ryan Florence) показав як переписати застосунок, використовуючи хуки. Дивіться відео тут: https://www.youtube.com/watch?v=dpw9EHDh2bM

Перед тим, як ми продовжимо, зверніть увагу що хуки:

- Необов'язкові. Можна спробувати хуки в декількох компонентах без переписування будь-якого існуючого коду. Немає нагальної потреби у вивченні чи використанні хуків, якщо вам цього не хочеться.
- 100% зворотна сумісність. хуки нічого не ламають.
- Доступні вже зараз. хуки доступні у релізі 16.8.0.

Мотивація

Хуки вирішують великий діапазон скоріш за все непоєднаних проблем у React, на які ми натрапили при написанні та підтриманні десятків тисяч компонентів протягом п'яти років. Неважливо, ви тільки вивчаєте React, чи використовуєте його щодня, чи навіть віддаєте перевагу іншій бібліотеці зі схожою системою компонентів, можливо, ви впізнаєте деякі з цих проблем.

Важко перевикористати логіку станів між компонентами

React не має можливості “прикріпити” до компонента поведінку, яку можна перевикористати (наприклад підключення до сховища). Якщо ви вже доволі давно працюєте із React, то скоріше за все ви знайомі із такими петернами як рендер-пропи та компоненти вищого порядку, що мають на меті вирішити цю проблему. Але при використанні цих шаблонів проектування, розробник вимушений реструктурувати компоненти, через що код стає заплутаним та важко зрозумілим.

Якщо поглянути у React Devtools на типовий застосунок написаний на React, з великою імовірністю ви побачите таку ситуацію, коли компоненти обгортаються у шари провайдерів, споживачів, компонентів вищого порядку, рендер-пропів та інших абстракцій, які ми лагідно назвемо “пекло з обгорток”. Незважаючи на те, що їх можливо відфільтрувати у Devtools, стає зрозуміло, що необхідно створити кращий спосіб повторно використовувати логіку навколо стану.

За допомогою хуків ви можете виокремити логіку стану з компонента, щоб протестувати її або повторно використати. З хуками з'являється можливість перевикористання логіки з відслідкуванням стану без зміни ієрархії ваших компонентів. Через це стає легко ділитися хуками поміж іншими компонентами чи зі спільнотою.

Хук стану

Розглянемо приклад, в якому рендериться лічильник. Коли ви натискаєте кнопку, значення лічильника збільшується:

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

function Example() {
  // Оголошуємо нову змінну стану "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Ви натиснули {count} разів</p>
      <button onClick={() => setCount(count + 1)}>
        Натисни мене
      </button>
    </div>
  );
```

У цьому прикладі, useState — це хук (визначення хуку наведенно нижче). Ми викликаємо його для того, щоб надати внутрішній стан нашому компоненту. React буде зберігати цей стан між повторними рендерами. Виклик useState повертає дві речі: поточне значення стану та функцію, яка дозволяє оновлювати цей стан. Ви можете викликати цю функцію де завгодно, наприклад, в обробнику події. Вона подібна до this.setState у класах, за винятком того, що не об’єднує новий та старий стан. Порівняння хука useState та this.state приведено на сторінці Використання хука стану.

Єдиним аргументом для useState є початкове значення стану. У наведеному вище прикладі — це 0, тому що наш лічильник починається з нуля. Зауважте, що на відміну від this.state, у нашому випадку стан може, але не зобов’язаний, бути об’єктом. Початкове значення аргументу використовується тільки під час першого рендера.

Оголошення декількох змінних стану

Ви можете використовувати хук стану більше одного разу в одному компоненті:

```
function ExampleWithManyStates() {
  // Оголошуємо декілька змінних стану!
  const [age, setAge] = useState(42);
  const [fruit, setFruit] = useState('банан');
  const [todos, setTodos] = useState([{ text: 'Вивчити хуки' }]);
  // ...
}
```

Синтаксис деструктуризації масивів дозволяє нам по різному називати змінні стану, які ми оголошуємо при виклику useState. Ці імена не є частиною API useState. Натомість, React припускає, що якщо ви викликаєте useState багато разів, то ви робите це в тому ж порядку під час кожного рендеру. Ми пояснимо, чому це працює та коли це стане в нагоді, трохи пізніше.

Що ж таке хук?
Хуки — це функції, за допомогою яких ви можете “зачепитися” за стан та методи життєвого циклу React з функційних компонентів. Хуки не працюють всередині класів — вони дають вам можливість використовувати React без класів. (Ми не рекомендуємо відразу ж переписувати наявні компоненти, але за бажанням, ви можете почати використовувати хуки у своїх нових компонентах.)

React містить кілька вбудованих хуків, таких як useState. Ви також можете створювати власні хуки, щоб повторно використовувати їх в інших своїх компонентах. Для початку, розглянемо вбудовані хуки.

Хук ефекту

Вам, напевно, доводилося створювати запити даних, робити підписки або вручну змінювати DOM з React-компонента. Ми називаємо ці операції “побічними ефектами” (або скорочено “ефекти”), так як вони можуть впливати на роботу інших компонентів і не можуть бути виконані під час рендеринга.

За допомогою хука ефекту useEffect ви можете виконувати побічні ефекти із функційного компонента. Він виконує таку ж саму роль, що і componentDidMount, componentDidUpdate та componentWillUnmount у React-класах, об’єднавши їх в єдиний API. (Ми порівняємо useEffect з іншими методами на сторінці Використання хука ефекту.)

Наприклад, цей компонент встановлює заголовок документа після того, як React оновлює DOM:

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

function Example() {
  const [count, setCount] = useState(0);
  // Подібно до componentDidMount та componentDidUpdate:
  useEffect(() => {
    // Оновлюємо заголовок документа, використовуючи API браузера
    document.title = `Ви натиснули ${count} разів`;
  });
  return (
    <div>
      <p>Ви натиснули {count} разів</p>
      <button onClick={() => setCount(count + 1)}>
        Натисни мене
      </button>
    </div>
  );
}
```

Коли ви викликаєте useEffect, React отримує вказівку запустити вашу функцію з “ефектом” після того, як він відправив зміни у DOM. Оскільки ефекти оголошуються всередині компонентів, то у них є доступ до пропсів та стану. За замовчуванням, React запускає ефекти після кожного рендеру, включаючи перший рендер. (Ми розглянемо більш докладно, як це відрізняється від класових методів життєвого циклу на сторінці Використання ефекту хука.)

Ефект також може повертати функцію, яка вказуватиме, як за ним слід здіснити “прибрання”. Наприклад, цей компонент використовує ефект, щоб підписатися на статус друга в мережі, і виконує прибирання, відписуючись від нього:

```
function FriendStatus(props) {
  const [isOnline, setIsOnline] = useState(null);
  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }
  useEffect(() => {
    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
    };
  });

  if (isOnline === null) {
    return 'Завантаження...';
  }
  return isOnline ? 'В мережі' : 'Не в мережі';
}
```

Додаткові матеріали

- https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=en
- https://uk.reactjs.org/docs/getting-started.html
- https://legacy.reactjs.org/blog/2019/08/15/new-react-devtools.html
-

Домашнє завдання

https://uk.reactjs.org/tutorial/tutorial.html

створити гру в хрестики-нолики по посібнику

як ускладнення - фінальний код переписати на хуки