hillel IT school

Lesson 33

План заняття

- useEffect;
- useLayoutEffect;
- useRef;
- useMemo;
- useCallback;

useEffect

useEffect — це хук React, який дозволяє синхронізувати компонент із зовнішньою системою. Позволяє працювати з сайд-ефектами при:

1. монтування (аналог компонентаDidMount)
2. оновлення стану або пропсов (аналог компонентаDidUpdate)
3. розмонтування (аналог компонентаWillUnmount)

Спрацьовує гарантовано після оновлення DOM для поточного компонента.

Снтаксис:

useEffect(setup, dependencies?)

setup: функція з логікою вашого ефекту. Ваша функція налаштування також може повертати функцію очищення. Коли ваш компонент додано до DOM, React запустить вашу функцію налаштування. Після кожного повторного рендерингу зі зміненими залежностями React спочатку запустить функцію очищення (якщо ви її надали) зі старими значеннями, а потім запустить вашу функцію налаштування з новими значеннями. Після видалення компонента з DOM React запустить вашу функцію очищення.

dependencies: список усіх реактивних значень, на які посилається код налаштування. Реактивні значення включають властивості, стан і всі змінні та функції, оголошені безпосередньо в тілі компонента. Список залежностей повинен мати постійну кількість елементів і бути записаним у рядку, як [dep1, dep2, dep3]. React порівнює кожну залежність з її попереднім значенням за допомогою порівняння Object.is. Якщо ви пропустите цей аргумент, ваш ефект буде запускатися повторно після кожного повторного відтворення компонента.

useEffect

Важливо!

- useEffect — це хук, тому ви можете викликати його лише на верхньому рівні вашого компонента або ваших власних хуків. Ви не можете викликати внутрішні цикли або умови. Якщо вам це потрібно, витягніть новий компонент і перемістіть у нього стан.
- Якщо ви не намагаєтеся синхронізуватися з якоюсь зовнішньою системою, можливо, вам не потрібен ефект.
- Коли строгий режим увімкнено, React запустить один додатковий цикл налаштування+очищення лише для розробки перед першим справжнім налаштуванням. Це стрес-тест, який гарантує, що ваша логіка очищення "віддзеркалює" вашу логіку налаштування та зупиняє або скасовує будь-які дії налаштування. Якщо це викликає проблему, застосуйте функцію очищення.

useEffect

Важливо!

- Якщо деякі з ваших залежностей є об’єктами або функціями, визначеними всередині компонента, існує ризик, що вони спричинять повторний запуск Ефекту частіше, ніж потрібно. Щоб виправити це, видаліть непотрібні залежності об’єктів і функцій. Ви також можете отримати оновлення стану та нереактивну логіку за межами вашого ефекту.
- Якщо ваш ефект не був спричинений взаємодією (наприклад, клацанням), React, як правило, дозволить браузеру спочатку намалювати оновлений екран перед запуском вашого ефекту. Якщо ваш ефект виконує щось візуальне (наприклад, позиціонує спливаючу підказку), і затримка помітна (наприклад, він мерехтить), замініть useEffect на useLayoutEffect.
- Навіть якщо ваш ефект був спричинений взаємодією (наприклад, клацанням), браузер може перефарбувати екран перед обробкою оновлень стану всередині вашого ефекту. Зазвичай це те, що ви хочете. Однак, якщо вам потрібно заблокувати браузеру перемальовування екрана, вам потрібно замінити useEffect на useLayoutEffect.
- Ефекти запускаються лише на клієнті. Вони не запускаються під час рендерингу сервера.

useEffect

Ефект дозволяє синхронізувати ваш компонент із зовнішньою системою. Тут зовнішня система означає будь-який фрагмент коду, який не контролюється React, наприклад:

- Таймер, керований за допомогою setInterval() і clearInterval().
- Підписка на подію за допомогою window.addEventListener() і window.removeEventListener().
- Бібліотека анімації третьої сторони з такими API, як animation.start() і animation.reset().

Якщо ви не підключаєтесь до жодної зовнішньої системи, вам, ймовірно, не потрібен ефект.

useLayoutEffect

useLayoutEffect — це версія useEffect, яка запускається до того, як браузер перефарбує екран.

Синтаксис:

useLayoutEffect(setup, dependencies?)

Більшості компонентів не потрібно знати їх положення та розмір на екрані, щоб вирішити, що відображати. Вони повертають лише деякі JSX. Потім браузер розраховує їх розташування (положення та розмір) і перефарбовує екран. Але іноді нам потрібно знати розміри елементів перш ніж відобразити щось (наприклад підказку) на екрані.

Для цього потрібно виконати рендеринг у три проходи:

- Візуалізуйте спливаючу підказку будь-де (навіть у неправильному місці).
- Виміряйте його висоту та вирішіть, де розмістити підказку.
- Знову відобразіть підказку в потрібному місці.

useLayoutEffect

Важливо!

- useLayoutEffect — це хук, тому ви можете викликати його лише на верхньому рівні вашого компонента або ваших власних хуків. Ви не можете викликати внутрішні цикли або умови. Якщо вам це потрібно, витягніть компонент і перемістіть туди ефект.
- Коли строгий режим увімкнено, React запустить один додатковий цикл налаштування+очищення лише для розробки перед першим справжнім налаштуванням. Це стрес-тест, який гарантує, що ваша логіка очищення «віддзеркалює» вашу логіку налаштування та зупиняє або скасовує будь-які дії налаштування. Якщо це викликає проблему, застосуйте функцію очищення.
- Якщо деякі з ваших залежностей є об’єктами або функціями, визначеними всередині компонента, існує ризик, що вони спричинять повторний запуск Ефекту частіше, ніж потрібно. Щоб виправити це, видаліть непотрібні залежності об’єктів і функцій. Ви також можете отримати оновлення стану та нереактивну логіку за межами вашого ефекту.
- Ефекти запускаються лише на клієнті. Вони не запускаються під час рендерингу сервера.
- Код усередині useLayoutEffect і всі заплановані з нього оновлення стану блокують браузер від перефарбування екрана. При надмірному використанні це робить ваш додаток повільним. Якщо можливо, віддайте перевагу useEffect.

useRef

useRef — це хук React, який дозволяє посилатися на значення, непотрібне для рендерингу.

Синтаксис:

const ref = useRef(initialValue);

initialValue: початкове значення поточної властивості об’єкта ref. Це може бути значення будь-якого типу. Цей аргумент ігнорується після початкового відтворення.

useRef повертає об’єкт з єдиною властивістю current: спочатку встановлено початкове значення, яке ви передали. Пізніше ви можете встановити для нього щось інше. Якщо ви передасте об’єкт ref до React як атрибут ref до вузла JSX, React встановить його поточну властивість.

Під час наступних візуалізацій useRef поверне той самий об’єкт.

Детальніше тут https://react.dev/reference/react/useRef

useRef

Важливо!

- Ви можете змінити властивість ref.current. На відміну від стану, він змінний. Однак, якщо він містить об’єкт, який використовується для візуалізації (наприклад, частину вашого стану), тоді вам не слід змінювати цей об’єкт.
- Коли ви змінюєте властивість ref.current, React не рендерить ваш компонент повторно. React не усвідомлює, коли ви його змінюєте, оскільки посилання є звичайним об’єктом JavaScript.
- Не записуйте та не читайте ref.current під час візуалізації, за винятком ініціалізації. Це робить поведінку вашого компонента непередбачуваною.
- У строгому режимі React двічі викличе вашу функцію компонента, щоб допомогти вам знайти випадкові домішки. Це поведінка лише для розробки і не впливає на виробництво. Кожен об’єкт посилання буде створено двічі, але одна з версій буде відкинута. Якщо ваша функція компонента чиста (як і має бути), це не повинно впливати на поведінку.

useMemo

useMemo — це хук React, який дозволяє кешувати результат обчислень між повторними візуалізаціями.

Синтаксис:

const memoParam = useMemo(callback, dep)

calculateValue: функція, що обчислює значення, яке ви хочете кешувати. Він має бути чистим, не приймати аргументів і повертати значення будь-якого типу. React викличе вашу функцію під час початкового рендерингу. Під час наступних рендерів React знову повертатиме те саме значення, якщо залежності не змінилися з часу останнього рендеру. В іншому випадку він викличе CalculateValue, поверне результат і збереже його, щоб потім можна було повторно використати.

dep: список усіх реактивних значень, на які посилаються всередині коду calculateValue. Реактивні значення включають властивості, стан і всі змінні та функції, оголошені безпосередньо в тілі компонента. Список залежностей повинен мати постійну кількість елементів і бути записаним у рядку, як [dep1, dep2, dep3]. React порівнює кожну залежність з її попереднім значенням за допомогою порівняння Object.is.

useMemo

Важливо!

- useMemo — це хук, тому ви можете викликати його лише на верхньому рівні вашого компонента або ваших власних хуків. Ви не можете викликати внутрішні цикли або умови. Якщо вам це потрібно, витягніть новий компонент і перемістіть у нього стан.
- У строгому режимі React двічі викличе вашу функцію обчислення, щоб допомогти вам знайти випадкові домішки. Це поведінка лише для розробки і не впливає на виробництво. Якщо ваша функція обчислення чиста (як і має бути), це не повинно впливати на вашу логіку. Результат одного з викликів буде проігноровано.
- React не буде викидати кешоване значення, якщо для цього немає конкретної причини. Наприклад, під час розробки React видаляє кеш, коли ви редагуєте файл свого компонента. Як під час dev, так і під час prod, React викидає кеш, якщо ваш компонент призупиняється під час початкового монтування. У майбутньому React може додати більше функцій, які використовують переваги викидання кешу — наприклад, якщо React додасть вбудовану підтримку віртуалізованих списків у майбутньому, було б доцільно викинути кеш для елементів, які прокручуються з вікно перегляду віртуальної таблиці. Це добре, якщо ви покладаєтеся на useMemo виключно як на оптимізацію продуктивності. В іншому випадку змінна стану або посилання можуть бути більш доречними.

useCallback

useCallback — це хук React, який дозволяє кешувати визначення функції між повторними візуалізаціями.

Синттаксис:

const cachedFn = useCallback(fn, dependencies)

fn: значення функції, яке потрібно кешувати. Він може приймати будь-які аргументи та повертати будь-які значення. React поверне (не викличе!) вашу функцію під час початкового рендерингу. Під час наступних рендерів React знову надасть вам ту саму функцію, якщо залежності не змінилися з часу останнього рендеру. В іншому випадку він надасть вам функцію, яку ви передали під час поточного візуалізації, і збереже її на випадок, якщо її можна буде повторно використати пізніше. React не буде викликати вашу функцію. Функція повертається до вас, щоб ви могли вирішити, коли та чи викликати її.

dependencies: список усіх реактивних значень, на які посилається код fn. Реактивні значення включають властивості, стан і всі змінні та функції, оголошені безпосередньо в тілі компонента. Якщо ваш лінтер налаштований для React, він перевірить, чи кожне реактивне значення правильно вказано як залежність. Список залежностей повинен мати постійну кількість елементів і бути записаним у рядку, як [dep1, dep2, dep3]. React порівнює кожну залежність з її попереднім значенням за допомогою алгоритму порівняння Object.is.

useCallback

useCallback — це хук React, який дозволяє кешувати визначення функції між повторними візуалізаціями.

Синттаксис:

const cachedFn = useCallback(fn, dependencies)

fn: значення функції, яке потрібно кешувати. Він може приймати будь-які аргументи та повертати будь-які значення. React поверне (не викличе!) вашу функцію під час початкового рендерингу. Під час наступних рендерів React знову надасть вам ту саму функцію, якщо залежності не змінилися з часу останнього рендеру. В іншому випадку він надасть вам функцію, яку ви передали під час поточного візуалізації, і збереже її на випадок, якщо її можна буде повторно використати пізніше. React не буде викликати вашу функцію. Функція повертається до вас, щоб ви могли вирішити, коли та чи викликати її.

dependencies: список усіх реактивних значень, на які посилається код fn. Реактивні значення включають властивості, стан і всі змінні та функції, оголошені безпосередньо в тілі компонента. Якщо ваш лінтер налаштований для React, він перевірить, чи кожне реактивне значення правильно вказано як залежність. Список залежностей повинен мати постійну кількість елементів і бути записаним у рядку, як [dep1, dep2, dep3]. React порівнює кожну залежність з її попереднім значенням за допомогою алгоритму порівняння Object.is.

useCallback

Важливо!

- useCallback — це хук, тому ви можете викликати його лише на верхньому рівні вашого компонента або ваших власних хуків. Ви не можете викликати внутрішні цикли або умови. Якщо вам це потрібно, витягніть новий компонент і перемістіть у нього стан.
- React не викине кешовану функцію, якщо для цього немає конкретної причини. Наприклад, під час dev React видаляє кеш, коли ви редагуєте файл свого компонента. Як під час dev, так і під час prod, React викидає кеш, якщо ваш компонент призупиняється під час початкового монтування. У майбутньому React може додати більше функцій, які використовують переваги викидання кешу — наприклад, якщо React додасть вбудовану підтримку віртуалізованих списків у майбутньому, було б доцільно викинути кеш для елементів, які прокручуються з вікно перегляду віртуальної таблиці. Це має відповідати вашим очікуванням, якщо ви покладаєтеся на useCallback як оптимізацію продуктивності. В іншому випадку змінна стану або посилання можуть бути більш доречними.

Дякую за увагу