hillel IT school

Lesson 34

План заняття

- Context;
- useContext;
- Reducer;
- useReducer;

Rander props

Рендер-проп - це проп-функція, яка повертає дерево React-елементів. По факту - це проп, який уміє рендерити що-то.

Компонент, який приймає такий проп, викликає функцію і рендерить те, що вона вернет. Він може передавати в рендер-проп які-то параметри, вираховані дані, що зручно. Сам рендер-проп може використовувати ці параметри при рендері.

Рендер-пропи дозволені:

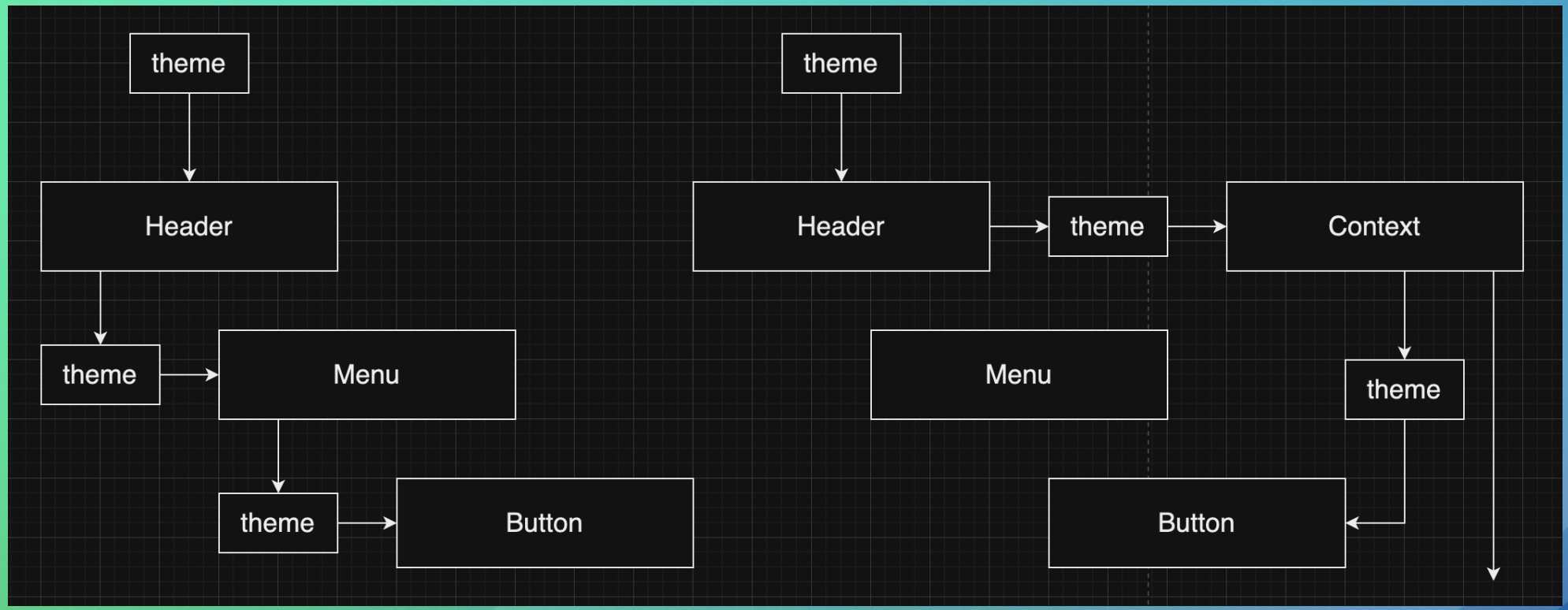
- переіспользовать какую-то логіку в різних компонентах
- розділяти різні блоки рендера і залишати їх у різних «слотах».

Context

Контекст потрібен щоб передавати дані на будь-який рівень вкладеності, без проміжної передачі пропсів.

Детальніше тут https://react.dev/learn/passing-data-deeply-with-context

І тут https://react.dev/reference/react/createContext



Context

Викличте createContext поза будь-якими компонентами, щоб створити контекст.

Синтаксис:

createContext(defaultValue);

defaultValue: значення, яке потрібно мати контексту, якщо в дереві над компонентом, який читає контекст, немає відповідного постачальника контексту. Якщо у вас немає значущого значення за замовчуванням, вкажіть null. Значення за замовчуванням означає «останній засіб». Він статичний і ніколи не змінюється з часом.

createContext повертає контекстний об’єкт.

Сам об’єкт контексту не містить жодної інформації. Він представляє, який контекст читають або надають інші компоненти. Як правило, ви використовуєте SomeContext.Provider у компонентах вище, щоб указати значення контексту, і викликаєте useContext(SomeContext) у компонентах нижче, щоб прочитати його. Об'єкт контексту має кілька властивостей:

- SomeContext.Provider дозволяє надавати значення контексту компонентам.
- SomeContext.Consumer — альтернативний і рідко використовуваний спосіб читання значення контексту.

Provider

Оберніть свої компоненти в постачальник контексту, щоб указати значення цього контексту для всіх компонентів усередині.

```
<SomeContext.Provider  value={someValue}>
    <Page />
</SomeContext.Provider>
```

value: значення, яке ви хочете передати всім компонентам, які читають цей контекст у цьому провайдері, незалежно від того, наскільки глибоко. Значення контексту може бути будь-якого типу. Компонент, який викликає useContext(SomeContext) усередині провайдера, отримує значення самого внутрішнього відповідного провайдера контексту над ним.

Consumer

До появи useContext існував старіший спосіб читання контексту.

```
<ThemeContext.Consumer>
    {theme => (
      <button className={theme} />
    )}
</ThemeContext.Consumer>
```

children: Функція. React викличе передану вами функцію з поточним значенням контексту, визначеним за тим же алгоритмом, що й useContext(), і відобразить результат, який ви повертаєте з цієї функції. React також повторно запускатиме цю функцію та оновлюватиме інтерфейс кожного разу, коли контекст батьківських компонентів змінюватиметься.

useContext

useContext — це хук React, який дозволяє читати контекст із вашого компонента та підписуватися на нього.

Синтаксис:

const value = useContext(SomeContext);

SomeContext: контекст, який ви раніше створили за допомогою createContext. Контекст сам по собі не містить інформації, він лише представляє тип інформації, яку ви можете надати або зчитати з компонентів.

useContext повертає значення контексту для викликаючого компонента. Воно визначається як значення, передане найближчому SomeContext.Provider над викликаючим компонентом у дереві. Якщо такого постачальника немає, тоді повернуте значення буде defaultValue, яке ви передали createContext для цього контексту. Повернене значення завжди актуальне. **React автоматично повторно рендерить компоненти, які читають певний контекст, якщо він змінюється.**

useContext

Важливо!

- На виклик useContext() у компоненті не впливають постачальники, які повертаються з того самого компонента. Відповідний <Context.Provider> має бути над компонентом, який виконує виклик useContext().
- React автоматично повторно рендерить усіх дочірніх елементів, які використовують певний контекст, починаючи з постачальника, який отримує інше значення. Попереднє та наступне значення порівнюються за допомогою порівняння Object.is. Пропуск повторних візуалізацій за допомогою memo не перешкоджає дочірнім елементам отримувати нові контекстні значення.
- Якщо ваша система збирання створює дублікати модулів у виводі (що може статися з символічними посиланнями), це може порушити контекст. Передача чогось через контекст працює, лише якщо SomeContext, який ви використовуєте для надання контексту, і SomeContext, який ви використовуєте для його читання, є абсолютно однаковими об’єктами, як визначено порівнянням ===.

useReducer

useReducer — це хук React, який дозволяє додавати reducer до вашого компонента.

Синтаксис:

const [state, dispatch] = useReducer(reducer, initialArg, init?)

- reducer: функція редуктора, яка визначає, як оновлюється стан. Він має бути чистим, має сприймати стан і дію як аргументи та повертати наступний стан. Стан і дія можуть бути будь-якими.
- initialArg: значення, з якого обчислюється початковий стан. Це може бути значення будь-якого типу. Те, як початковий стан обчислюється з нього, залежить від наступного аргументу init.
- необов'язковий init: функція ініціалізатора, яка має повертати початковий стан. Якщо його не вказано, початковий стан встановлено як initialArg. В іншому випадку початковий стан встановлюється як результат виклику init(initialArg).

Докладніше про useReducer ви можете почитати тут https://react.dev/reference/react/useReducer

Докладніше про reducer ви можете почитати тут https://react.dev/learn/extracting-state-logic-into-a-reducer

useReducer

useReducer повертає масив рівно з двома значеннями:

- state. Під час першого рендерингу встановлюється значення init(initialArg) або initialArg (якщо немає init).
- Функція диспетчеризації, яка дозволяє оновлювати стан до іншого значення та запускати повторне відтворення.

Важливо!

- useReducer — 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 ваш редуктор та ініціалізатор, щоб допомогти вам знайти випадкові домішки. Це поведінка лише для розробки і не впливає на виробництво. Якщо ваш редуктор та ініціалізатор є чистими (як вони повинні бути), це не повинно впливати на вашу логіку. Результат одного з викликів ігнорується.

dispatch function

Функція диспетчеризації, яку повертає useReducer, дозволяє оновити стан до іншого значення та запустити повторне відтворення. Вам потрібно передати дію як єдиний аргумент функції dispatch.

Синтаксис:

```
dispatch({ type: 'incremented' });
```

React встановить наступний стан на результат виклику функції редюсера, яку ви надали разом із поточним станом, і дію, яку ви передали dispatch.

Параметри:

action: дія, яку виконує користувач. Це може бути значення будь-якого типу. За домовленістю дія зазвичай є об’єктом із властивістю типу, що його ідентифікує, і, необов’язково, іншими властивостями з додатковою інформацією.

dispatch function

Важливо!

- Функція диспетчеризації лише оновлює змінну стану для наступного відтворення. Якщо ви прочитаєте змінну стану після виклику функції диспетчеризації, ви все одно отримаєте старе значення, яке було на екрані до вашого виклику.
- Якщо нове значення, яке ви надаєте, ідентичне поточному стану, як визначено порівнянням Object.is, React пропустить повторне відтворення компонента та його дочірніх компонентів. Це оптимізація. React може все одно викликати ваш компонент, перш ніж ігнорувати результат, але це не повинно вплинути на ваш код.
- React пакетно оновлює стан. Він оновлює екран після того, як усі обробники подій запущені та викликають свої встановлені функції. Це запобігає багаторазовому повторному рендерингу під час однієї події. У рідкісних випадках, коли вам потрібно змусити React оновити екран раніше, наприклад, щоб отримати доступ до DOM, ви можете використовувати flushSync.

Reducer function

Функція редуктора оголошується так:

```
function reducer(state, action) {
  // ...
}
```

Потім потрібно заповнити код, який обчислить і поверне наступний стан. За домовленістю прийнято писати як оператор switch. Для кожного випадку в перемикачі обчислити та повернути наступний стан.

```
function reducer(state, action) {
  switch (action.type) {
    case 'incremented_age': {
      return {
        name: state.name,
        age: state.age + 1
      };
    }
}
```

Reducer function

Action можуть мати будь-яку форму. За домовленістю прийнято передавати об’єкти з властивістю type, що ідентифікує дію. Він повинен містити мінімально необхідну інформацію, яка необхідна редуктору для обчислення наступного стану. За звичай ця інформація зберігається у payload.

Назви type у action є локальними для вашого компонента. Кожна дія описує одну взаємодію, навіть якщо це призводить до кількох змін у даних. Форма state є довільною, але зазвичай це буде об’єкт або масив.

Важливо!

Стан доступний лише для читання. Не змінюйте жодних об’єктів або масивів у стані.

Дякую за увагу