5. Передача данных по контексту

Содержание урока

- Обзор;
- Контекст;
- Подведём итоги.

Обзор

Привет! 👋🏼 В этом уроке мы рассмотрим продвинутый подход передачи данных в React, называемый `«контекст»` (или `«context»`).

Чем хорош контекст? В чем его отличия от пропсов? И какие ограничения?

Это нам и предстоит узнать. 👊🏼

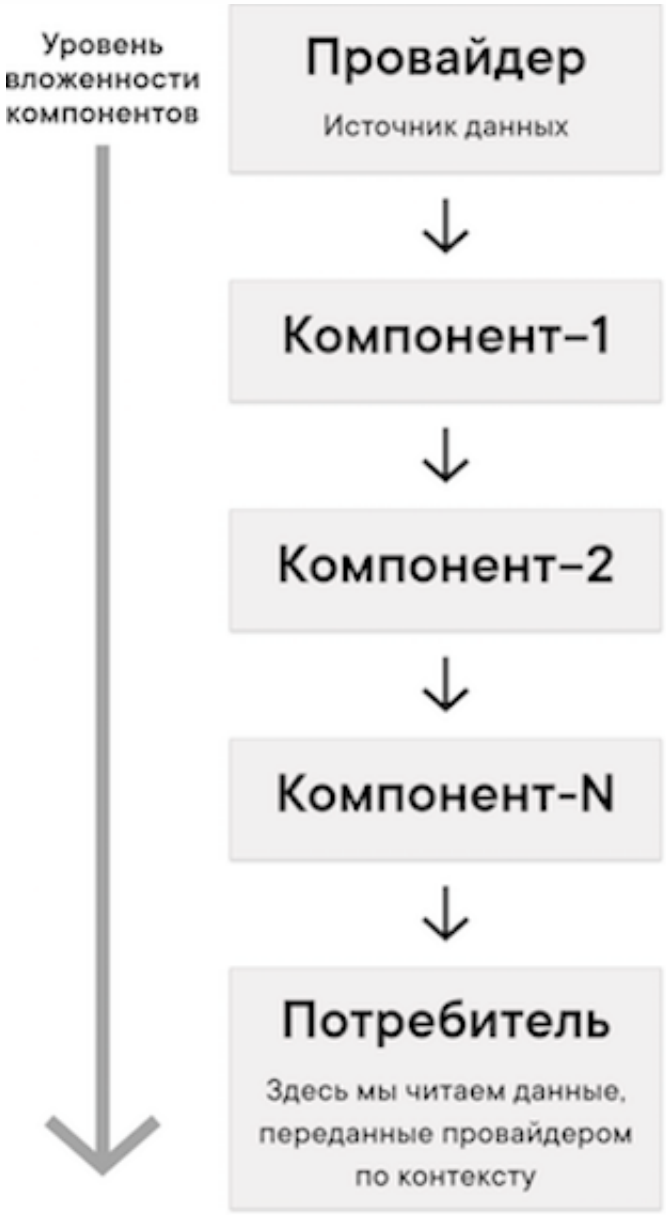
Контекст

Учитывая подход `«однонаправленного потока данных»`, передача информации осуществляется по пропсам. Этот подход работает хорошо до тех пор, пока не приходится иметь дело с данными, которые необходимы компонентам в приложении во многих местах и на глубоком уровне вложенности. 🦞

Зато с этой задачей хорошо справляется `«контекст»`.

`«Контекст»` представляет собой способ передачи данных вниз по иерархии дерева компонентов без необходимости пробрасывать пропсы на каждом уровне. 🦅

Механика использования `«контекста»` предусматривает наличие `«провайдера»` и `«потребителя»`, компонентов-ключевых точек в отношении обмена данных.



В промежутке между «провайдером» и «потребителем» может быть вложенность компонентов любой глубины. Также из контекста, предоставленного «провайдером», может читать любое количество «потребителей». 👯

Для создания «контекста» существует API со следующей подписью:

```
const { Provider, Consumer } = React.createContext(defaultValue);
```

Где:

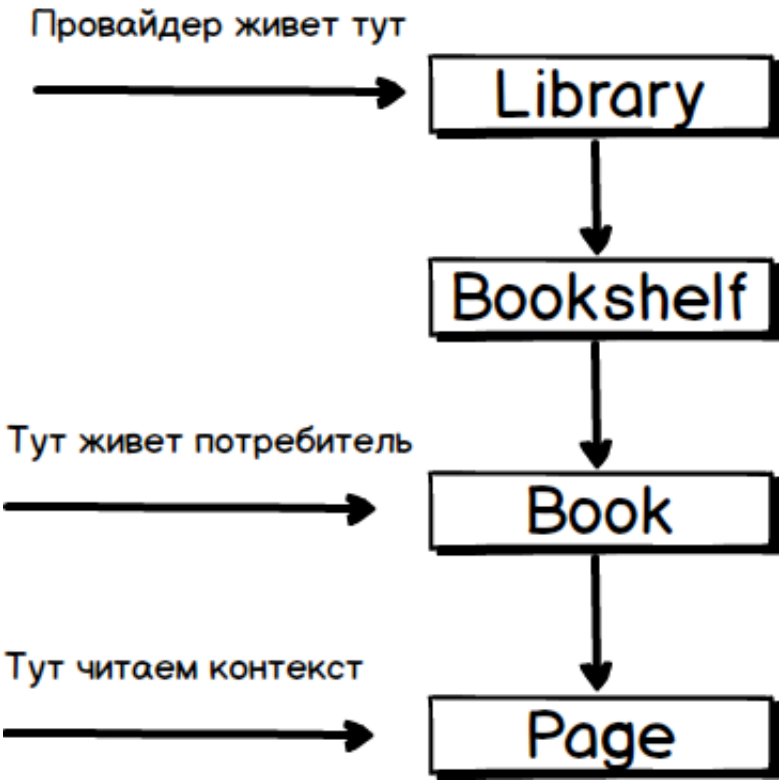
- `Provider` — это компонент-«провайдер» контекста;
- `Consumer` — это компонент-«потребитель» контекста;
- `defaultValue` — фолбэк (на случай, если «потребитель» захочет использовать контекст без соответствующего «провайдера»).

Оскар любит практиковать магию. Для этого ему нужна библиотека с книгами заклинаний. Давай поможем Оскару построить библиотеку, используя `«контекст»`, чтобы проложить эффективный канал данных. Учитывая отборный эстетический вкус Оскара, мы создадим «темизированную» библиотеку, с возможностью переключать темы UI между светлой и темной, а информацию о том, какая тема приложения выбрана, будем хранить в контексте. 👨‍🎨

Сперва давай разобьём библиотеку на компоненты. У нас будет:

- Родительский компонент `Library` — «управляющий» компонент, содержащий `«провайдер»` контекста;
- Промежуточный компонент `Bookshelf` — ребёнок по отношению к компоненту `Library` и родитель для компонента `Book`;
- Компонент `Book`, содержащий `«потребителя»` контекста;
- Компонент `Page`, из которого мы будем читать контекст.

Схематически структуру компонентов нашего мини-приложения `«библиотека»` можно выразить так:



Учитывая наличие нескольких компонентов, каждый файл в следующем примере будет описан непосредственно после примера. 👩‍🎨 Мы также коснёмся нескольких новых концепций React: `«состояния»`, `«методов класса»` и `«children»`. Пускай их присутствие тебя не пугает — эти темы мы разберем в следующих уроках.

🖥️ Пример кода 5.1:

```
// context.js
import React from 'react';

```

```
4  export const themes = {
5      dark: {
6          foreground: '#FFF',
7          background: '#222',
8      },
9      light: {
10         foreground: '#000',
11         background: '#EEE',
12     },
13 };
14
15 export const ThemeContext = React.createContext(themes.dark);
```

Файл `context.js`:

- `На строке кода 4`: мы инициализируем идентификатор `themes` — эта структура данных будет хранить информацию о доступных темах;
- `На строке кода 15`: мы создаём модуль `ThemesContext`, содержащий два компонента:
  - `Provider`: его мы используем в компоненте `Library`, где инициализируем контекст с данными о теме;
  - `Consumer`: его мы используем в компоненте `Book`, чтобы прочитать из контекста данные о теме.
- `На строке кода 15`: мы также задаём дефолтное значение для контекста в виде `темной` темы.

```
1  // Library.js
2  import React, { Component } from 'react';
3  import { themes, ThemeContext } from './context';
4  import { Bookshelf } from './Bookshelf';
5
6  export default class Library extends Component {
7      constructor() {
8          super();
9
10         this._toggleTheme = () => {
11             this.setState(state => ({
12                 theme: state.theme === themes.dark ? themes.light :
   themes.dark,
13             }));
14         };
15
16         this.state = {
17             theme:       themes.light,
18             _toggleTheme: this._toggleTheme,
19         };
20     }
```

```
21
22      render() {
23          return (
24              <ThemeContext.Provider value = { this.state }>
25                  <Bookshelf />
26              </ThemeContext.Provider>
27          );
28      }
29  }
```

Файл `Library.js`:

- `На строке кода 7`: происходит нечто новое для нас — мы описываем `«контструктор класса»`. В нем мы объявляем `«состояние»` компонента и `«метод класса»`:
  - `На строке кода 10`: создаём метод класса `_toggleTheme`: он будет переключать темы со светлой на темную и обратно;
  - `На строке кода 16`: инициализируем `состояние` компонента, объявив свойство `theme` — описание текущей выбранной темы, и `_toggleTheme` — ссылки на метод, которая будет переключать темы. Оба эти свойства будут доступны в контексте `«потребителям»` после передачи в `«провайдер»`.

```
1  // Bookshelf.js
2  import React from 'react';
3  import { Book } from './Book';
4
5  export const Bookshelf = () => {
6      return (
7          <>
8              <Book />
9          </>
10     );
11 };
```

Файл `Bookshelf.js` описывает промежуточный компонент, который просто рендерит ребёнка.

```
1  // Book.js
2  import React from 'react';
3  import { ThemeContext } from './context';
4  import { Page } from './Page';
5
6  export const Book = () => {
7      return (
8          <ThemeContext.Consumer>
9              { context => (
10                 <Page
```

```
                    theme = { context.theme }
                    _toggleTheme = { context._toggleTheme}
                />
            )}
        </ThemeContext.Consumer>
    );
};
```

Файл `Book.js`:

- `На строке кода 8`: рендерим `«потребителя»` контекста;
- `На строке кода 9`: рендерим `«children»` компонента-`«потребителя»` — функцию, параметром которой станут данные, переданные `«провайдеру»` в пропс `value` в файле `Library.js`, а затем эти данные передаём компоненту `Page` по пропсам.

```
// Page.js
import React from 'react';

export const Page = props => {
    const style = {
        backgroundColor: props.theme.background,
        color:           props.theme.foreground,
    };

    return (
        <button onClick = { props._toggleTheme } style = { style }>
            Click!
        </button>
    );
};
```

Файл `Page.js`:

- `На строке кода 5`: объявляем идентификатор `styles`, содержащий описание стилей, исходя из темы, выбранной в контексте (читаем ее из пропсов за счёт того, что передали эти данные уровнем выше);
- `На строке кода 11`: привязываем метод-обработчик `_toggleTheme` к слушателю `onClick`. В результате данной привязки при каждом клике на кнопку метод будет менять выбранную тему в компоненте `Library`.

Таким образом, мы передали информацию о теме UI по контексту и её прочитали.

🖐🏽 Совет бывалых:

> Не стоит использовать контекст лишь для того, чтобы не передавать данные по пропсам. Используй контекст лишь в тех сценариях, в которых одни и те же данные необходимы многим компонентам на разных уровнях вложенности.

Подведём итоги

`«Контекст»` предоставляет возможность передавать и читать данные из одной точки приложения в любую другую ниже по иерархии компонентов, без необходимости пробрасывать данные по пропсам на каждом уровне компонентов. 🤷‍♀️

Ключевые концепции `«контекста»`:

- Метод для инициализации `«контекста»` доступен в именной области библиотеки React, принимает дефолтное значение для контекста в аргумент и возвращает `«провайдер»` контекста и его `«потребителя»`;
- `«Провайдер»` — это компонент, служащий `родителем` в отношении потока данных;
- `«Потребитель»` — это компонент, служащий `ребёнком` в отношении потока данных;
- `«Контекст»` может быть динамическим, если связать логику его изменения с методом класса.

Спасибо, что остаёшься с нами! 🙌🏻 В следующем уроке мы рассмотрим новую для нас концепцию `«children»` в React. До встречи! 👋🏻

> Мы будем очень признательны, если ты оставишь свой фидбек в отношении этой части конспекта на нашу электропочту `hello@lectrum.io`.