

6. Знакомство с «children» в React

Содержание урока

- Обзор;
- Что такое «children» ?;
- «Children» в виде строк;
- «Children» в виде JavaScript-выражений;
- «Children» в виде функций;
- «Children» в виде Boolean, null или undefined;
- Итераторы «children» ;
- Уникальность и счёт «children» ;
- Подведём итоги.

Обзор

Привет! 🙌 В этом уроке мы рассмотрим необычную концепцию управления контентом, называемую «children». «Children» тесно переплетена с компонентной природой React, поэтому понимание «children» существенно повысит понимание происходящего в приложении в целом. 🐰

Что такое «children» ?

Всякий раз, когда рендерящийся компонент включает закрывающий и открывающий тэг, контент между этими тегами передаётся в специальный пропс с именем `children` (доступен по адресу `this.props.children`).

! Важно:

«Children» — это **непрозрачная структура данных** (англ. "opaque data structure"), что означает то, что `props.children` может быть любого типа: массивом, функцией, строкой, объектом и другими типами данных. Данную нестабильность стоит учитывать при работе с «children». 🙌

«Children» в виде строк

Можно передать компоненту строку между открывающим и закрывающим тэгами. Тогда эта строка будет доступна посредством обращения к `this.props.children` изнутри компонента.

```
1 <BlackCat>Meow.</BlackCat>
```

Пример выше — валидный JSX. А обращение к `this.props.children` в компоненте `BlackCat` вернет строку 'Meow.'.

При работе с «children» в виде строк React будет автоматически вырезать пробелы в начале и в конце линий кода, а также будет удалять пустые строки кода. React также будет сжимать большие пробелы внутри строки до размера одного символа пробела.

 Пример кода 6.1:

```
1 export default class FortuneTeller extends Component {
2   render () {
3     return (
4       <section>
5         <p>          Hard work pays off.</p>
6         <p>
7           Stay          focused.
8         </p>
9         <p>
10          Live
11          your
12          life.
13        </p>
14        <p>
15
16          Don't forget to tell the truth.
17        </p>
18      </section>
19    );
20  }
21 }
```

В примере кода 6.1 компонент-гадалка отрендерит свои предсказания в отформатированном виде:

- На строке кода 5 : длинный пробел после открывающего тэга будет вырезан;
- На строках кода 6–8 : длинный пробел между словами и переносы строк будут вырезаны;
- На строках кода 9–13 : переносы строк будут вырезаны;
- На строках кода 14–17 : переносы строк и пустая строка будут вырезаны.

В результате каждый элемент `<p>` из примера выше отрендерится в одну строку.


«Children» в виде JavaScript-выражений

«Children» может принимать любое JavaScript-выражение, если поместить его в фигурные скобки `{}`.

 Пример кода 6.2:

```
1 <Fate>Your life will be happy and peaceful.</Fate>
2
3 <Fate>12</Fate>
4
5 <Fate>{ 'Your life will be happy and peaceful.' }</Fate>
```

В примере кода 6.2 во всех случаях «children» для компонента Fate — JavaScript-выражения.

 Хозяйке на заметку:

Данная особенность приходится кстати, когда нужно отрендерить произвольное количество элементов. Например, рендерить список элементов произвольной длины, итерируя по массиву данных методом `map` и возвращая JSX-элемент на каждую итерацию, поместив вышеописанную инструкцию в виде выражения в фигурных скобках.

«Children» в виде функций

Как мы уже знаем, обычно «children» для компонента вычисляется как строка, React-элемент или список строк. Однако компоненту также можно передать функцию в виде «children».

 Пример кода 6.3:

```

1  const Order = ({ children }) => {
2
3      return children();
4  };
5
6  const OrderMagicDust = () => {
7
8      return (
9          <Order>
10             { () => <span>Bring me some magic dust with delivery please!
11             </span> }
12             </Order>
13         );
14     };

```

В примере кода 6.3 Оскар создал компоненты для заказа магической пыли в интернет-магазине. 🧙

В результате рендера компонента `OrderMagicDust` на UI отобразится элемент `` с его контентом. Так происходит потому, что, передав функцию компоненту `Order` в виде «children» на строке кода 10, мы её деструктурируем (строка кода 1) и вызываем (строка кода 3) внутри тела компонента `Order`.

🔍 Хозяйке на заметку:

К слову, мы уже рассматривали подход передачи «children» в виде функции в уроке 5 (пример кода 5.1, файл `Book.js`, строка кода 9).

«Children» в виде Boolean, null или undefined

Такие JavaScript-значения, как `true`, `false`, `null` и `undefined` — валидные «children», и все они означают отсутствие значения для рендера. 🧙

💻 Пример кода 6.4:

```

1  export default class IgnoredValues extends Component {
2      render () {
3
4          return (
5              <>
6                  <div />
7
8                  <div></div>
9
10                 <div>{ false }</div>
11
12                 <div>{ null }</div>

```

```

13
14         <div>{ undefined }</div>
15
16         <div>{ true }</div>
17
18         <div>{ ' ' }</div>
19
20         <div>But me will be rendered! Because I'm a string!
</div>
21     </>
22     );
23 }
24 }

```

Данный подход полезен в применении приёма `условного рендеринга`. 🧑

🖥️ Пример кода 6.5:

```

1  // Fate.js
2
3  import React from 'react';
4  import Fortune from './Fortune';
5
6  const isUnfold = true;
7
8  const Fate = () => {
9
10     return (
11         <>
12             <h1>Your fate is { isUnfold ? 'unfold' : 'hidden' }.</h1>
13             { isUnfold && <Fortune /> }
14         </>
15     );
16 };
17
18 export default Fate;

```

```

1  // Fortune.js
2
3  import React from 'react';
4
5  const Fortune = () => {
6
7     return <p>The one you love is closer than you think.</p>;
8 };
9
10 export default Fortune;

```

В результате выполнения примера кода 6.5 на UI отобразится строка `Your fate is hidden`, если идентификатор `isUnfold` содержит значение `false`, или строка `Your fate is unfold` и компонент `Fortue`, если идентификатор `isUnfold` содержит значение `true`. 🙋

В случае, когда ты хочешь отобразить `false`, `true`, `null` или `undefined` на экране явно, это значение нужно привести к строке.

🖥️ Пример кода 6.6:

```
1  const NullToRender = () => {
2
3    return <span>{ String(null) }</span>;
4  };
```

Учитывая динамическую природу JavaScript, мы можем совершать различные гибкие операции с «children». Можно передавать им специальные свойства или решать, отрендерить или нет.

В следующей части этого урока мы рассмотрим несколько специализированных методов «children», которые помогут нам в этом.

Итераторы «children»

React предоставляет набор методов для удобного управления непрозрачной структурой данных «children». Один из самых полезных — `React.Children.map`, который работает точно так же, как соответствующий метод массива, однако он универсален, учитывая непрозрачность «children», то есть умеет перебирать «children» любого типа, будь-то строка, функция, объект или массив. 🧑

🖥️ Пример кода 6.7:

```
1  // Fortunes.js
2  import React, { Component } from 'react';
3  import FilteredFortunes from './FilteredFortunes';
4
5  export default class Fortunes extends Component {
6    render () {
7
8      return (
9        <FilteredFortunes>
10         <span>It's time to make new friends.</span>
11         <span>Good times are coming your way.</span>
12         <span>Keep Calm and Stay Cool.</span>
13       </FilteredFortunes>
14     );
15  }
```

```
16 }
```

```
1 // FilteredFortunes.js
2 import React, { Component } from 'react';
3
4 export default class FilteredFortunes extends Component {
5   render () {
6     const { children } = this.props;
7     const filteredChildren = React.Children.map(children, (child,
8 index) => {
9
10         return index % 2 === 0 ? child : null;
11     });
12
13     return <section>{ filteredChildren }</section>;
14 }
15 }
```

В примере кода 6.7 компонент `<FilteredFortunes />` итерирует по «children», отфильтровывая каждый второй «children» на строке кода 9.

! Важно:

Для успешного выполнения операции из примера кода выше необходимо использовать именно специализированный метод `React.Children.map`. Попытка использования обычного метода `map` массива (`this.props.children.map(...)`) может привести к ошибке, в случае если передать в «children» функцию.

Существует также похожий метод `React.Children.forEach`, действующий точно так же, как `React.Children.map`, только не возвращающий никакого значения.

Уникальность и счёт «children»

`React.Children.count` возвращает количество сущностей, содержащихся в «children».

🖥️ Пример кода 6.8:

```
1 // Fortunes.js
2 import React, { Component } from 'react';
3 import FilteredFortunes from './FilteredFortunes';
4
5 export default class Fortunes extends Component {
6   render () {
7
8     return (
```

```

9         <FilteredFortunes>
10             <span>It's time to make new friends.</span>
11             <span>Good times are coming your way.</span>
12             <span>Keep Calm and Stay Cool.</span>
13         </FilteredFortunes>
14     );
15 }
16 }

```

```

1 // FilteredFortunes.js
2 import React, { Component, Children } from 'react';
3
4 export default class FilteredFortunes extends Component {
5     render () {
6         const { children } = this.props;
7         const filteredChildren = Children.map(children, (child, index)
=> {
8
9             return index % 2 === 0 ? child : null;
10         });
11         const totalChildrenQuantity = Children.count(children);
12
13         return (
14             <section>
15                 <p>Total children: { totalChildrenQuantity }</p>
16                 <p>Only first children: { filteredChildren }</p>
17             </section>
18         );
19     }
20 }

```

В примере кода 6.8, в файле `FilteredFortunes.js`, перед отфильтрованными «children», на строке кода 15 отрендерится общее количество «children», которым располагает компонент. 100

Иногда также необходимо отрендерить только первый и единственный «children» из имеющихся. Для этого существует специализированный метод `React.Children.only`. Этот метод также возбудит ошибку при попытке отрендерить несколько «children» вместо одного.

 Пример кода 6.9:

```

1 // Fortunes.js
2 import React, { Component } from 'react';
3 import OnlyFirstFortune from './OnlyFirstFortune';
4
5 export default class Fortunes extends Component {
6     render () {

```



```

7
8     return (
9         <OnlyFirstFortune>
10            <span>It's time to make new friends.</span>
11            <span>Good times are coming your way.</span>
12            <span>Keep Calm and Stay Cool.</span>
13        </OnlyFirstFortune>
14    );
15 }
16 }

```

```

1  // OnlyFirstFortune.js
2  import React, { Component, Children } from 'react';
3
4  export default class OnlyFirstFortune extends Component {
5      render () {
6          const { children } = this.props;
7          const theOnlyChildren = Children.only(children);
8
9          return <section>{ theOnlyChildren }</section>;
10     }
11 }

```

В примере кода 6.9 React возбудит ошибку вследствие попытки передачи нескольких «children» в связке с инструкцией `React.Children.only`. 🦊

Подведём итоги

«Children» помогает компонентам React более гибко управлять разметкой.

«Children» могут быть следующих типов данных:

- Строка;
- Компонент;
- Выражение;
- Любое falsy значение;
- Функция;
- Объект;
- Массив.

React также предоставляет набор утилит для работы с непрозрачной структурой данных «children».

Спасибо, что остаёшься с нами! 🦊 В следующем уроке мы разузнаем, как делать компонент ещё более полноценным и самостоятельным, используя механизм инкапсулированного состояния React. До встречи! 🙌

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