9. Обрабатываем события в React

Содержание урока

- Обзор;
- Обработка событий;
- Синтетическое событие;
- Подведём итоги.

Обзор

Привет! 👋🏼 В этой части конспекта мы разузнаем о том, как обрабатывать события в React, а также о продвинутом концепте React, лежащем в основе обработки событий — «синтетическом событии». 🐯

К делу! 👊🏼

Обработка событий

Механизм обработки событий в React очень похож на механизм обработки событий в DOM, с учётом некоторых синтаксических поправок:

- Имена слушателей событий написаны, следуя практике `camelCase`, а не `lowerCase`, как в DOM;
- Обработчики событий — функции, а не строки, как в DOM.

Например, вот так слушатель и обработчик выглядит в HTML:

```
<button onclick="hadnlecheckout()">
    Checkout
</button>
```

А так — в React:

```
<button onClick = { this.handleCheckout }>
    Checkout
</button>
```

Ещё одна существенная разница — предотвращение дефолтного поведения элементов. В React нельзя возвращать `false` из обработчика для предотвращения дефолтного поведения. Вместо этого необходимо вызвать метод `preventDefault` из объекта события в явном виде.

Например, предотвратить дефолтное поведение ссылки в HTML можно так:

```
<a href="#" onclick="console.log('The link was clicked.'); return false">
  Click me
</a>
```

В React следует описать желаемое поведение по-другому:

```
const ActionLink = () => {
    const handleClick = (event) => {
        event.preventDefault();
        console.log('The link was clicked.');
    }

    return (
        <a href = '#' onClick = { handleClick }>
            Click me
        </a>
    );
}
```

Идентификатор `event`, доступный в области видимости функции `handleClick` — это объект `синтетического события`. 🔮

✅ Хорошая практика:

> В React не стоит регистрировать слушателей событий посредством вызова `addEventListener` на DOM-элементах напрямую. Вместо этого лучше объявлять слушателей в JSX, в методе `render`.

Синтетическое событие

`Синтетическое событие` — особая концепция, следуя которой React обрабатывает ошибки. В основе данной концепции лежит кроссбраузерная обёртка вокруг обычной модели событий DOM, спрототипированная согласно спецификации W3C. Объект `синтетического события` React обладает таким же интерфейсом, что и объект `нативного события` DOM, включая методы `stopPropagation` и `preventDefault`, с поправкой на полную кроссбраузерность. 👯🎉

При срабатывании обработчик события React получает доступ к экземпляру `синтетического события`.

Совсем недавно тёмные силы космических просторов пытались проникнуть в этот мир и разрушить его. 🔮 Но у Оскара есть план: создать генератор магических палочек и воспользоваться им, чтобы прогнать злые силы. 👿 Давай поможем Оскару в этом. 🎩

🖥️ Пример кода 9.1:

```
export default class MagicWandGenerator extends Component {
    state = {
        wandsQuantity: 0
    }

    _generateWand = () => {
        this.setState(({ wandsQuantity }) => ({
            wandsQuantity: wandsQuantity + 1
        }));
    }

    render () {
        const { wandsQuantity } = this.state;

        return (
            <>
                <span>Magic wands generated: { wandsQuantity }.</span>
                <button onClick = { this._generateWand }>
                    Generate new magic wand!
                </button>
            </>
        );
    }
}
```

В `примере кода 9.1` на `строке кода 18` создан элемент `<button>` с регистрацией слушателя события `onClick`. Это событие вызовет привязанный к нему обработчик `this._generateWand` при клике на кнопку. При вызове на `строке кода 6` обработчик выполнит свое тело и обновит стейт компонента, увеличив количество магических палочек на `1`. 🎋

Это самая простая модель обработки события в React.

Давай рассмотрим более продвинутый вариант с обработкой событий клавиатуры. 🎹 Хотя эта, пожалуй, лучше впишется в контекст. → ⌨️

Генератор магических палочек из примера кода `9.1` сработал хорошо, однако тёмные силы космических просторов позвали на помощь, открыв портал в ад, и теперь нам предстоит сражаться с демоническими силами. 😈 Эй, Оскар, нам срочно нужно улучшить эффективность генератора магических палочек!

🖥️ Пример кода 9.2:

```
export default class MagicWandGenerator extends Component {
    state = {
        wandsQuantity: 0
    }

    _generateWand = (event) => {
        let { wandsQuantity } = this.state;

        event.shiftKey
            ? wandsQuantity += 10
            : wandsQuantity += 1;

        this.setState({
            wandsQuantity
        });
    }

    render () {
        const { wandsQuantity } = this.state;

        return (
            <section>
                <span>Magic wands generated: { wandsQuantity }.</span>
                <button onClick = { this._generateWand }>
                    Generate new magic wand!
                </button>
            </section>
        );
    }
}
```

В `примере кода 9.2` на `строке кода 6` в обработчике задействован объект `синтетического события`. На `сроке кода 9` описано обращение к объекту `синтетического события` с целью определить, была ли нажата клавиша клавиатуры `shift` в момент срабатывания события. ⌨️ Свойство `shiftKey` содержит булевое значение.

Таким образом, кликнув по кнопке `Generate new magic wand!` с зажатой клавишой `shift`, генератор магических палочек создаст их 10 штук за один раз вместо одной, что, в свою очередь, поможет справиться со злыми силами. 🤼

В `7`-й части этого конспекта мы познакомились с приёмом использования контролируемых компонентов React, но на тот момент у нас еще не было понимания концепции `синтетических событий`. Теперь можно оценить картину более полным взглядом.

🖥️ Пример кода 9.3:

```
1  export default class MagicWandName extends Component {
2      state = {
3          magicWandName: ''
4      }
5
6      _updateName = (event) => {
7          this.setState({
8              magicWandName: event.target.value
9          });
10     }
11
12     render () {
13         const { magicWandName } = this.state;
14
15         return (
16             <>
17                 <div>A new magic wand name is: { magicWandName }.</div>
18                 <input
19                     type = 'text'
20                     value = { magicWandName }
21                     onChange = { this.updateName }
22                 />
23             </>
24         );
25     }
26 }
```

В `примере кода 9.3` на `строке кода 6` в обработчике события `onChange` задействован объект `синтетического события`. В случае обработки события `onChange` в объекте `синтетического события` становится доступным свойство `target`, являющееся ссылкой на DOM-элемент. В данном случае это элемент `<input>` на `строке кода 18`. Таким образом, посредством обращения к свойству `event.target.value` можно получить актуальное значение элемента `<input>` и вследствие обновить стейт компонента, завершив логическу цепочку контролируемого компонента.

Кроме рассмотренных выше, существуют следующие свойства объекта `синтетического события`:

Свойство	Значение
bubbles	boolean
cancelable	boolean
currentTarget	DOMEventTarget
defaultPrevented	boolean
eventPhase	number
isTrusted	boolean
nativeEvent	DOMEvent
preventDefault()	void
isDefaultPrevented()	boolean
stopPropagation()	void
isPropagationStopped()	boolean
target	DOMEventTarget
timeStamp	number
type	string

Однако, в случае обработки события `мыши` (`MouseEvent`), свойства будут иметь другой вид:

Свойство	Значение
altKey	boolean
button	number
buttons	number
clientX	number
clientY	number
ctrlKey	boolean
getModifierState(key)	boolean
metaKey	boolean
pageX	number
pageY	number
relatedTarget	DOMEventTarget
screenX	number
screenY	number
shiftKey	boolean

А вот так может выглядеть объект `синтетического события` при обработке события `клавиатуры` (`KeyboardEvent`):

Свойство	Значение
altKey	boolean
charCode	number
ctrlKey	boolean
getModifierState(key)	boolean
key	string
keyCode	number
locale	string
location	number
metaKey	boolean
repeat	boolean
shiftKey	boolean
which	number

Подведём итоги

React реализует кастомную модель обработки событий в виде объекта `синтетического события`, представляющего собой универсальную кроссбраузерную обёртку, следующую спецификации W3C.

Спасибо, что остаёшься с нами! 👋🏽 В следующей части конспекта мы разберём подходы объявления `пользовательских методов` в компонентах React. До встречи! 🐯

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