

10. Объявляем пользовательские методы в компонентах

Содержание урока

- Обзор;
- Объявление пользовательских методов в виде инициализаторов свойств класса;
- Объявление пользовательских методов в виде методов класса;
- Подведём итоги.

Обзор

Привет! 🙌 В этой части конспекта мы детально разберём подходы объявления пользовательских методов класса, а также плохие и хорошие практики в этом контексте.



К делу! 🙌

Объявление пользовательских методов в виде инициализаторов свойств класса

Следуя подходу написания компонентов в виде класса, удобным паттерном создания пользовательских методов и обработчиков событий является объявления методов класса. Однако, учитывая гибкость языка программирования JavaScript, сделать это можно несколькими способами.

Возьмём знакомый нам генератор магических палочек в качестве примера для рассмотрения.

💻 Пример кода 10.1:


```
1 export default class MagicWandGenerator extends Component {
2   state = {
3     wandsQuantity: 0
4   }
5
6   _generateWands = () => {
7     this.setState(({ wandsQuantity }) => ({
```

```

8         wandsQuantity: wandsQuantity + 1
9     }));
10 }
11
12 render () {
13     const { wandsQuantity } = this.state;
14
15     return (
16         <>
17         <span>Magic wands generated: { wandsQuantity }.</span>
18         <button onClick = { this._generateWands }>Generate!
19     </button>
20         </>
21     );
22 }

```

В примере кода 10.1 обработчик события `onClick` описан на строке кода 6 в виде инициализатора свойства класса. Имя метода дополняет префикс в виде символа нижнего подчёркивания, помогающий визуально отделить пользовательский метод от метода, унаследованного от `React.Component`.

 Хозяйке на заметку:

Такой подход объявления хорош своей лаконичностью, однако синтаксис инициализатора свойства класса на данный момент является предложением ко включению в стандарт ECMAScript, и для его использования необходимо настроить «трайнспайлер» [Babel](https://babeljs.io/). 🧐

Объявление пользовательских методов в виде методов класса

Более консервативным подходом является подход объявления пользовательских методов и обработчиков событий в виде методов класса.

 Пример кода 10.2:

```

1 export default class MagicWandGenerator extends Component {
2     constructor () {
3         super();
4
5         this._generateWands = ::this._generateWands;
6     }
7
8     state = {
9         wandsQuantity: 0
10    }

```

```

11
12   _generateWands () {
13       this.setState(({ wandsQuantity }) => ({
14           wandsQuantity: wandsQuantity + 1
15       }));
16   }
17
18   render () {
19       const { wandsQuantity } = this.state;
20
21       return (
22           <>
23               <span>Magic wands generated: { wandsQuantity }.</span>
24               <button onClick = { this._generateWands }>Generate!
</button>
25           </>
26       );
27   }
28 }

```

В примере кода 10.2 на строке кода 12 объявлен обработчик события в виде метода класса. Следуя данному подходу, стоит проявлять особую осторожность, учитывая динамическую природу ключевого слова `this` в JavaScript и асинхронную модель обработки событий в браузере. ⚠️

Дело в том, что в JavaScript методы класса не имеют «привязки к контексту выполнения» по умолчанию. Поэтому привязку необходимо осуществлять вручную для каждого метода класса.

! Важно:

Если забыть привязать метод класса к контексту выполнения, ссылка по адресу `this.метод-обработчик` получит значение `undefined` в рантайме приложения.

Поэтому в примере кода 10.2 на строке кода 5 осуществлена привязка метода `_generateWands` к контексту данного класса.

🔍 Хозяйке на заметку:

Здесь использован синтаксис «оператора привязки», предложенный в стандарт ECMAScript. Этот синтаксис является более современной и более красивой версией обычной привязки в JavaScript, которая выглядела бы так:

```

1   this._generateWands = this._generateWands.bind(this);

```

Однако для его использования необходимо настроить «трайспайлер» [Babel](#). 🧐

Необходимость привязывать каждый метод в конструкторе класса вручную может утруждать, хотя привязки метода класса можно избежать, воспользовавшись неким трюком.

Пример кода 10.3:

```
1 export default class MagicWandGenerator extends Component {
2   _handleClick () {
3     console.log('Magic wand generator from inside:', this);
4   }
5
6   render () {
7
8     return (
9       <button onClick = { () => this._handleClick() }>
10         Click me to investiagte how MagicWandGenerator looks
11         from inside.
12       </button>
13     );
14   }
15 }
```

В примере кода 10.3 контекст выполнения метода `_handleClick` приобретёт правильное значение за счёт обёртки в виде стрелочной функции на строке кода 9, однако данный подход спорный в своей практичности, и в целом аргументов к его использованию чуть более, чем немного. 🧙

При использовании подхода объявления пользовательских методов в виде методов класса существует еще один подводный камень, достойный рассмотрения. 🧑

❌ Плохая практика:

```
1 export default class MagicWandGenerator extends Component {
2   state = {
3     wandsQuantity: 0
4   }
5
6   _generateWands () {
7     this.setState(({ wandsQuantity }) => ({
8       wandsQuantity: wandsQuantity + 1
9     }));
10  }
11
12  render () {
13    const { wandsQuantity } = this.state;
14
15    return (
16      <section>
17        <span>Magic wands generated: { wandsQuantity }.</span>
18        <button onClick = { ::this._generateWands }>Generate!
19      </button>
20    </section>
21  )
22 }
```

```
20         );  
21     }  
22 }
```

В данном примере на строке кода 18 метод класса `_generateWands` привязан непосредственно в методе `render`. Несмотря на то, что технически данный подход работает, он обладает существенным неявным недостатком: привязка, объявленная на строке кода 18, будет выполнена каждый раз при вызове метода `render`, что является собой не что иное, как неэффективный расход ресурсов CPU, что в свою очередь может привести к ухудшению производительности приложения. Поэтому следование данному подходу не рекомендуется. 🚫

Подведём итоги

Существует несколько подходов объявления пользовательских методов в React-компонентах:

- Пользовательский метод в виде инициализатора свойства класса;
- Пользовательский метод в виде метода класса.

Каждый из подходов имеет свои преимущества и недостатки. Выбор следует осуществлять с учётом вкуса и технических требований.

Спасибо, что остаёшься с нами! 🙌 В следующей части конспекта мы разберём дизайнерские решения организации компонентов. До встречи! 🐼

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