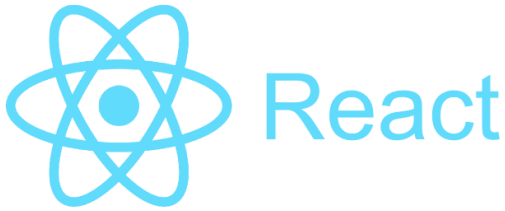


React Essential

Основы React

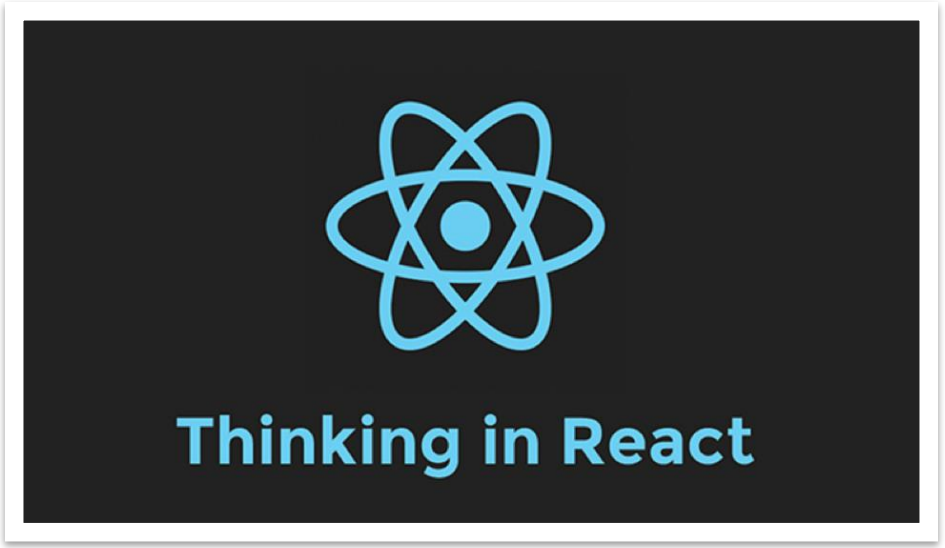
Introduction

Муляк Дмитрий

Front-end developer at GlobalTechMakers

dmitriymuliak

dmitriymuliak



Тема

Основы в React

Основы React

1. Виды компонентов (stateless, stateful)
2. Знакомство с State
3. Добавление событий
4. Работа со списками

Виды компонентов (stateless, stateful)

```jsx
const Button = props =>
  <button onClick={props.onClick}>
    {props.text}
  </button>
```

```jsx
class ButtonCounter extends React.Component {
  constructor() {
    super()
    this.state = { clicks: 0 }
    this.handleClick = this.handleClick.bind(this)
  }

  handleClick() {
    this.setState({ clicks: ++this.state.clicks })
  }

  render() {
    return (
      <Button
        onClick={this.handleClick}
        text={`You've clicked me ${this.state.clicks} times!`}
      />
    )
  }
}
```

Компоненты в React, содержащие внутреннее состояние (state), называются - **Stateful**. Без него — **Stateless**.

Многие заблуждаются, думая, что Stateless компоненты могут быть только в виде function. Единственное, что отличает Stateless от Stateful компонентов — это отсутствие внутреннего состояния.

В React версии 16 нет разницы, используйте вы компонент как class или как function, производительность от этого не изменится (как это было раньше).

https://ru.reactjs.org/docs/state-and-lifecycle

Знакомство с State

1) Создаём ES6-класс, указываем React.Component в качестве родительского класса.
2) Описываем конструктор и вызываем базовый конструктор с аргументом props - super(props).
3) Определяем свойство state.
4) Добавляем в класс метод render().

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  render() {
    return (
      <div>
        <h1>Привет, мир!</h1>
        <h2>Сейчас {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

Классовые компоненты всегда должны вызывать базовый конструктор с аргументом props.

- Чтобы получить доступ к state нужно вызвать this.state
- Чтобы изменить state нужно вызвать метод this.setState

Добавление событий

Обработка событий в React-элементах очень похожа на обработку событий в DOM-элементах. Но есть несколько синтаксических отличий:

- События в React именуются в стиле camelCase вместо нижнего регистра.

- С JSX вы передаёте функцию как обработчик события вместо строки.

- Ещё одно отличие — в React нельзя предотвратить обработчик события по умолчанию, вернув false. Нужно явно вызвать preventDefault.

Например, в HTML:

```
<button onclick="activateLasers()">
  Активировать лазеры
</button>
```

В React немного иначе:

```
<button onClick={activateLasers}>
  Активировать лазеры
</button>
```

Синтетические события

```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('По ссылке кликнули.');
  }

  return (
    <a href="#" onClick={handleClick}>
      Нажми на меня
    </a>
  );
}
```

В приведённом коде e — это синтетическое событие. React определяет синтетические события в соответствии со спецификацией W3C, поэтому не волнуйтесь о кроссбраузерности.

Посмотрите руководство о SyntheticEvent, чтобы узнать о них больше.

При использовании React обычно не нужно вызывать addEventListener, чтобы добавить обработчики в DOM-элемент после его создания. Вместо этого добавьте обработчик сразу после того, как элемент отрендерился.

События в классовых компонентах

В компоненте, определённом с помощью ES6-класса, в качестве обработчика события обычно выступает один из методов класса.

Например, этот компонент Toggle рендерит кнопку, которая позволяет пользователю переключать состояния между «Включено» и «Выключено»:

При обращении к this в JSX-колбэках необходимо учитывать, что методы класса в JavaScript по умолчанию не привязаны к контексту.

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // Эта привязка обязательна для работы `this` в колбэке.
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(state => ({
      isToggleOn: !state.isToggleOn
    }));
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'Включено' : 'Выключено'}
      </button>
    );
  }
}

ReactDOM.render(
  <Toggle />,
  document.getElementById('root')
);
```

Привязка методов

Если вы забудете привязать метод this.handleClick и передать его в onClick, значение this будет undefined в момент вызова функции.

Дело не в работе React, это часть того, [как работают функции в JavaScript](). Обычно, если ссылаться на метод без () после него, например, onClick={this.handleClick}, этот метод нужно привязать.

Если вам не по душе bind, существует два других способа. Если вы пользуетесь экспериментальным [синтаксисом общедоступных полей классов](), вы можете использовать его, чтобы правильно привязать колбэки.

```
class LoggingButton extends React.Component {
  // Такой синтаксис гарантирует, что `this` привязан к handleClick.
  // Предупреждение: это экспериментальный синтаксис
  handleClick = () => {
    console.log('значение this:', this);
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        Нажми на меня
      </button>
    );
  }
}
```

```
class LoggingButton extends React.Component {
  handleClick() {
    console.log('значение this:', this);
  }

  render() {
    // Такой синтаксис гарантирует, что `this` привязан к handleClick.
    return (
      <button onClick={(e) => this.handleClick(e)}>
        Нажми на меня
      </button>
    );
  }
}
```

Передача аргументов в обработчики событий

Внутри цикла часто нужно передать дополнительный аргумент в обработчик события. Например, если id — это идентификатор строки, можно использовать следующие варианты:

```
<button onClick={(e) => this.deleteRow(id, e)}>Удалить строку</button>
<button onClick={this.deleteRow.bind(this, id)}>Удалить строку</button>
```

Две строки выше — эквивалентны, и используют стрелочные функции и Function.prototype.bind соответственно.

В обоих случаях аргумент e, представляющий событие React, будет передан как второй аргумент после идентификатора. Используя стрелочную функцию, необходимо передавать аргумент явно, но с bind любые последующие аргументы передаются автоматически.

Работа со списками

Рендер нескольких компонентов

Вы можете создать коллекцию элементов и встроить её в JSX с помощью фигурных скобок {}.

К примеру, пройдём по массиву numbers, используя функцию JavaScript map(), и вернём элемент <li> в каждой итерации. Получившийся массив элементов сохраним в listItems:

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li>{number}</li>
);
```

Теперь мы включим массив listItems внутрь элемента <ul> и отрендерим его в DOM:

```
ReactDOM.render(
  <ul>{listItems}</ul>,
  document.getElementById('root')
);
```

Простой компонент-список

Как правило, вы будете рендерить списки внутри какого-нибудь компонента.

Мы можем отрефакторить предыдущий пример с использованием компонента, который принимает массив numbers и выводит список элементов. Когда вы запустите данный код, то увидите предупреждение о том, что у каждого элемента массива должен быть ключ (key).

«Ключ» — это специальный строковый атрибут, который нужно указывать при создании списка элементов.

Чтобы исправить проблему с неуказанными ключами, добавим каждому элементу в списке атрибут key.

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li>{number}</li>
  );
  return (
    <ul>{listItems}</ul>
  );
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

Ключи

Ключи помогают React определять, какие элементы были изменены, добавлены или удалены. Их необходимо указывать, чтобы React мог сопоставлять элементы массива с течением времени.

Лучший способ выбрать ключ — это использовать строку, которая будет явно отличать элемент списка от его соседей. Чаще всего вы будете использовать ID из ваших данных как ключи.

Мы не рекомендуем использовать индексы как ключи, если порядок элементов может поменяться. Это негативно скажется на производительности и может вызвать проблемы с состоянием компонента.

Если вы опустите ключ для элемента в списке, то React по умолчанию будет использовать индексы как ключи.

Ключи нужно определять непосредственно внутри массивов. Как правило, элементам внутри map() нужны ключи.

Ключи внутри массива должны быть уникальными только среди своих соседних элементов. Им не нужно быть уникальными глобально. Можно использовать один и тот же ключ в двух разных массивах.

```
const todoItems = todos.map((todo) =>
  <li key={todo.id}>
    {todo.text}
  </li>
);
```

```
const todoItems = todos.map((todo, index) =>
  // Делайте так, только если у элементов массива нет заданного ID
  <li key={index}>
    {todo.text}
  </li>
);
```

Проверка знаний

TestProvider.com

TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

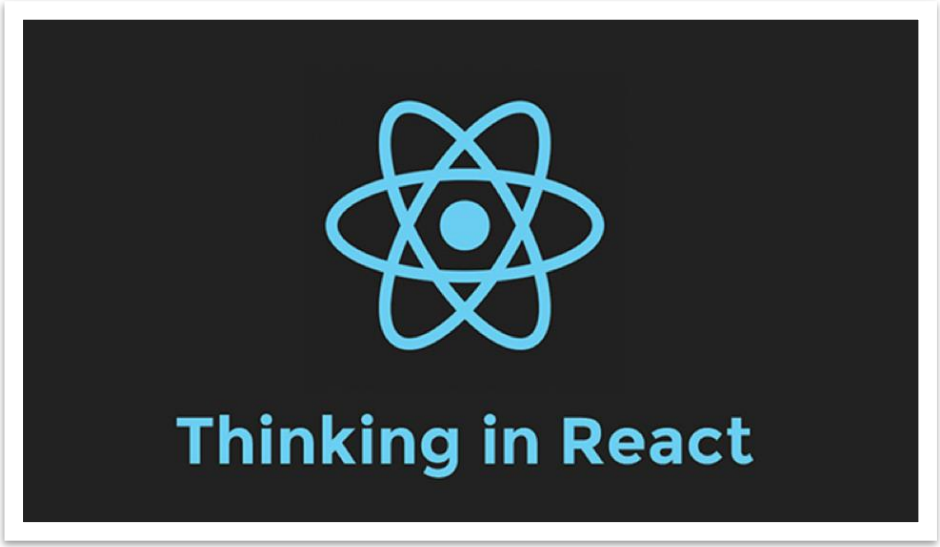
Проверьте как Вы усвоили данный материал на TestProvider.com

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.

Спасибо за внимание! До новых встреч!

Муляк Дмитрий

Front-end Developer at GTM



Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Silver Learning Windows Azure Cloud Partner EBA European Business Association