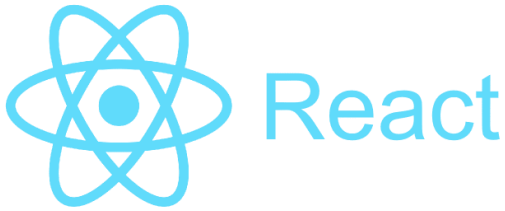


React Essential

Формы

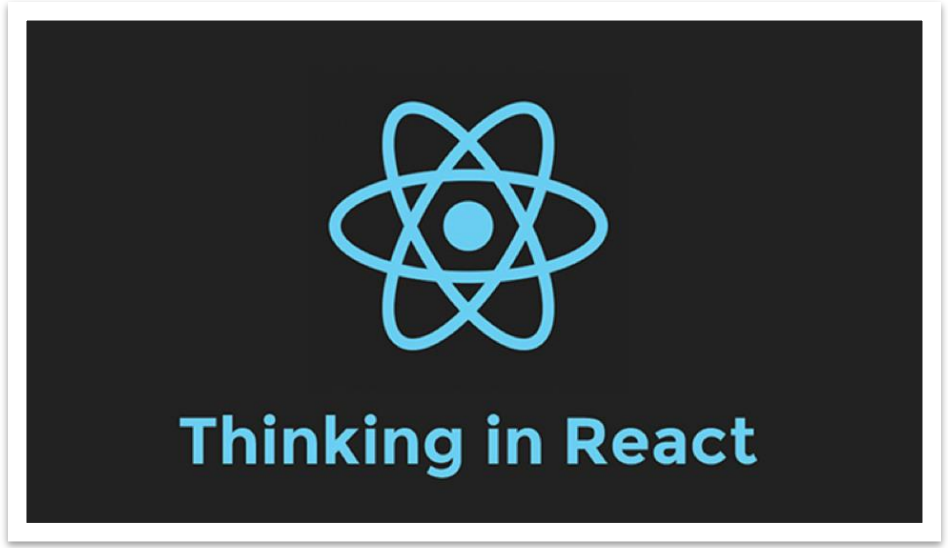
Introduction

Муляк Дмитрий

Front-end developer at GlobalTechMakers

dmitriymuliak

dmitriymuliak



Тема

Формы

Формы

1. Контролируемые и не контролируемые компоненты
2. Работа с Input
3. Подьем состояния
4. Валидация параметров с PropTypes
5. Мышление в стиле React

Контролируемые и не контролируемые компоненты

В React формы работают как обычно. Однако, чаще всего форму удобнее обрабатывать с помощью JavaScript-функции, у которой есть доступ к введённым данным. Стандартный способ реализации такого поведения называется «управляемые компоненты».

В React мутабельное состояние обычно содержится в свойстве компонентов state и обновляется только через вызов setState()

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('Отправленное имя: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Имя:
          <input type="text" value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Отправить" />
      </form>
    );
  }
}
```

Обработка 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.

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

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

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

Добавление обработчика события

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.

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

```
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>
    );
  }
}
```

Привязка контекста «this»

При обращении к this в JSX-колбэках необходимо учитывать, что методы класса в JavaScript по умолчанию не привязаны к контексту. Если вы забудете привязать метод this.handleClick и передать его в onClick, значение this будет undefined в момент вызова функции.

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>
    );
  }
}
```

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.

Тег textarea

HTML-элемент <textarea> в качестве текста отображает дочерний элемент.

В React <textarea> использует атрибут value. Таким образом, форму с <textarea> можно написать почти тем же способом, что и форму с однострочным <input>:

```
class EssayForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: 'Будьте любезны, напишите сочинение о вашем любимом DOM-элементе.'
    };

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('Сочинение отправлено: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Сочинение:
          <textarea value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Отправить" />
      </form>
    );
  }
}
```

Тег select

В HTML <select> создаёт выпадающий список. HTML-код в этом примере создаёт выпадающий список вкусов:

```
<select>
  <option value="grapefruit">Грейпфрут</option>
  <option value="lime">Лайм</option>
  <option selected value="coconut">Кокос</option>
  <option value="mango">Манго</option>
</select>
```

Пункт списка «Кокос» выбран по умолчанию из-за установленного атрибута selected. React вместо этого атрибута использует value в корневом теге select. В управляемом компоненте так удобнее, потому что обновлять значение нужно только в одном месте (state).

```
constructor(props) {
  super(props);
  this.state = {value: 'coconut'};

  this.handleChange = this.handleChange.bind(this);
  this.handleSubmit = this.handleSubmit.bind(this);
}
```

В атрибут value можно передать массив, что позволит выбрать несколько опций в теге select.

```
<select value={this.state.value} onChange={this.handleChange}>
  <option value="grapefruit">Грейпфрут</option>
  <option value="lime">Лайм</option>
  <option value="coconut">Кокос</option>
  <option value="mango">Манго</option>
</select>
```

Загрузка файла

В HTML <input type="file"> позволяет пользователю выбрать один или несколько файлов для загрузки с устройства на сервер или управлять им через JavaScript с помощью File API.

В React <input type="file"> всегда является неуправляемым компонентом, потому что его значение может быть установлено только пользователем, а не программным путём.

Для взаимодействия с файлами следует использовать File API. В следующем примере показано, как создать реф на DOM-узел, чтобы затем получить доступ к файлам в обработчике отправки формы:

```
class FileInput extends React.Component {
  constructor(props) {
    super(props);
    this.handleSubmit = this.handleSubmit.bind(this);
    this.fileInput = React.createRef();
  }
  handleSubmit(event) {
    event.preventDefault();
    alert(
      `Selected file - ${this.fileInput.current.files[0].name}`
    );
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Upload file:
          <input type="file" ref={this.fileInput} />
        </label>
        <br />
        <button type="submit">Submit</button>
      </form>
    );
  }
}
```

Подъем состояния

В React, совместное использование состояния достигается перемещением его до ближайшего предка компонентов, которым оно требуется. Это называется «подъём состояния».

Например, мы хотим, чтобы два поля ввода синхронизировались друг с другом. Когда мы обновляем поле ввода градусов по Цельсию, поле ввода градусов по Фаренгейту должно отражать сконвертированную температуру и наоборот.

https://ru.reactjs.org/docs/lifting-state-up

```
class Calculator extends React.Component {
  constructor(props) {
    super(props);
    this.handleCelsiusChange = this.handleCelsiusChange.bind(this);
    this.handleFahrenheitChange = this.handleFahrenheitChange.bind(this);
    this.state = {temperature: '', scale: 'c'};
  }

  handleCelsiusChange(temperature) {
    this.setState({scale: 'c', temperature});
  }

  handleFahrenheitChange(temperature) {
    this.setState({scale: 'f', temperature});
  }
```

```
return (
  <div>
    <TemperatureInput
      scale="c"
      temperature={celsius}
      onTemperatureChange={this.handleCelsiusChange} />
    <TemperatureInput
      scale="f"
      temperature={fahrenheit}
      onTemperatureChange={this.handleFahrenheitChange} />
    <BoilingVerdict
      celsius={parseFloat(celsius)} />
  </div>
);
```

Валидация параметров с PropTypes

По мере роста вашего приложения вы можете отловить много ошибок с помощью проверки типов. Для этого можно использовать расширения JavaScript вроде Flow и TypeScript.

Но, даже если вы ими не пользуетесь, React предоставляет встроенные возможности для проверки типов. Для запуска этой проверки на пропсах компонента вам нужно использовать специальное свойство propTypes:

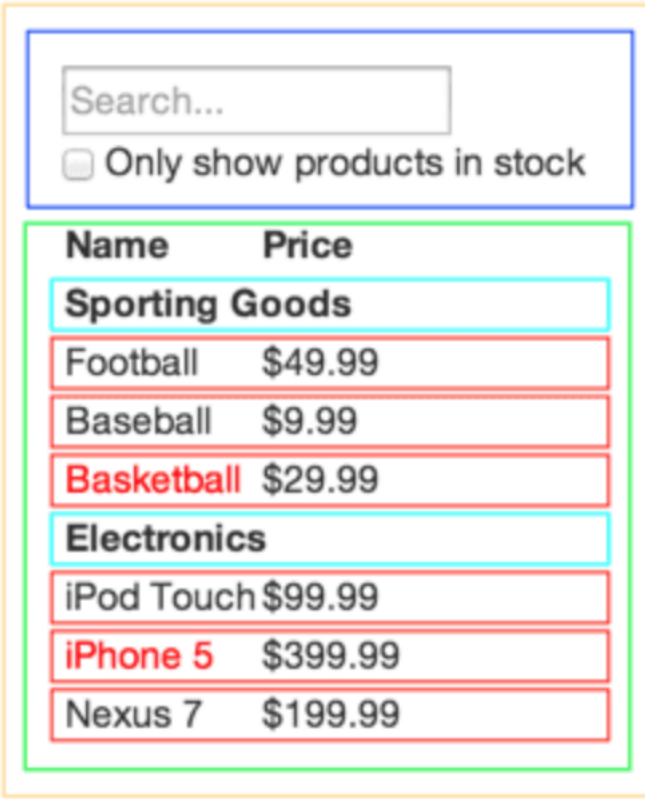
```
import PropTypes from 'prop-types';

class Greeting extends React.Component {
  render() {
    return (
      <h1>Привет, {this.props.name}</h1>
    );
  }
}

Greeting.propTypes = {
  name: PropTypes.string
};
```

Мышление в стиле React

Одна из особенностей React — это предлагаемый им процесс мышления при создании приложений.

Вот несколько шагов, которые помогут вам при разработке React приложений и компонентов :

1. Разбейте интерфейс на составляющие.
2. Создайте статическое приложение (компонент) в React.
3. Определите минимальное (но полноценное) отображение состояния интерфейса. (DRY)
4. Определите, где должно находиться ваше состояние.
5. Добавьте обратный поток данных.



Проверка знаний

TestProvider.com

Проверьте как Вы усвоили данный материал на TestProvider.com

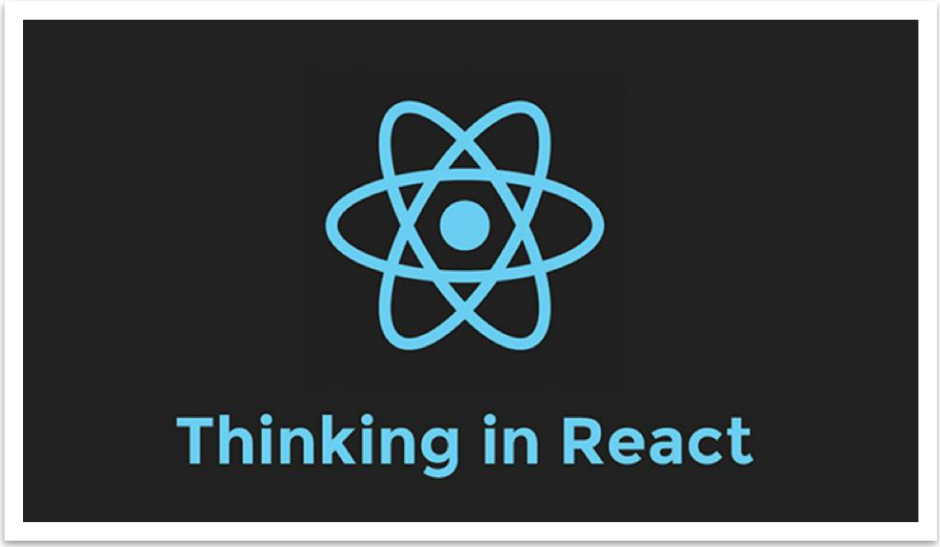
TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

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