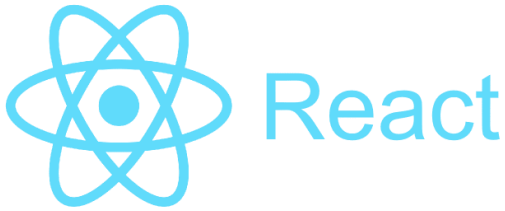


React Essential

Создание Модального окна

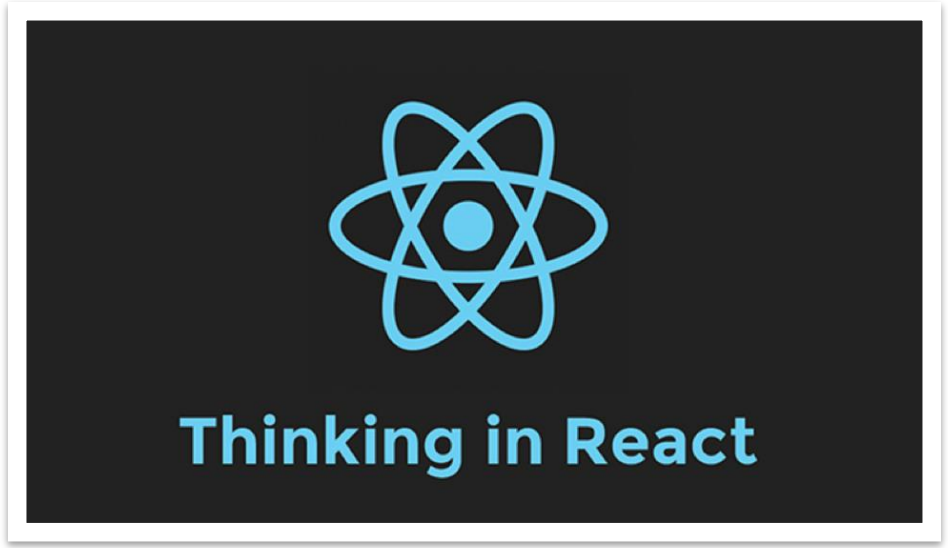
Introduction

Муляк Дмитрий

Front-end developer at GlobalTechMakers

dmitriymuliak

dmitriymuliak



Тема

Создание Модального окна

Создание Модального окна

1. Рефы и DOM
2. Context API
3. Рендер-пропсы

Рефы и DOM

Рефы дают возможность получить доступ к DOM-узлам или React-элементам, созданным в рендер-методе.

Создание рефов.

Рефы создаются с помощью React.createRef() и прикрепляются к React-элементам через ref атрибут. Обычно рефы присваиваются свойству экземпляра класса в конструкторе, чтобы на них можно было ссылаться из любой части компонента. (но можно использовать useRef() hook).

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.myRef = React.createRef();
  }
  render() {
    return <div ref={this.myRef} />;
  }
}
```

Рефы и DOM

Работа с рефами

Доступ к рефам.

Когда реф передаётся элементу в методе render, ссылка на данный узел доступна через свойство рефа current.

```
const node = this.myRef.current;
```

Не злоупотребляйте рефами.

Когда использовать рефы:

- Управление фокусом, выделение текста или воспроизведение медиа.
- Императивный вызов анимаций.
- Интеграция со сторонними DOM-библиотеками.

Избегайте использования рефов в ситуациях, когда задачу можно решить декларативным способом.

Значение рефа отличается в зависимости от типа узла.

- Когда атрибут ref используется с HTML-элементом, свойство current созданного рефа в конструкторе с помощью React.createRef() получает соответствующий DOM-элемент.
- Когда атрибут ref используется с классовым компонентом, свойство current объекта-рефа получает экземпляр смонтированного компонента.
- Нельзя использовать ref атрибут с функциональными компонентами, потому что для них не создаётся экземпляров.

Добавление рефа к DOM-элементу

В представленном ниже примере ref используется для хранения ссылки на DOM-элемент.

React присвоит DOM-элемент свойству current при монтировании компонента и присвоит обратно значение null при размонтировании.

Обновление свойства ref происходит перед вызовом методов componentDidMount и componentDidUpdate.

```
class CustomTextInput extends React.Component {
  constructor(props) {
    super(props);
    // создадим реф в поле `textInput` для хранения DOM-элемента
    this.textInput = React.createRef();
    this.focusTextInput = this.focusTextInput.bind(this);
  }

  focusTextInput() {
    // Установим фокус на текстовое поле с помощью чистого DOM API
    // Примечание: обращаемся к "current", чтобы получить DOM-узел
    this.textInput.current.focus();
  }

  render() {
    // описываем, что мы хотим связать реф <input>
    // с `textInput` созданным в конструкторе
    return (
      <div>
        <input
          type="text"
          ref={this.textInput} />
        <input
          type="button"
          value="Фокус на текстовом поле"
          onClick={this.focusTextInput}
        />
      </div>
    );
  }
}
```

Добавление рефа к классовому компоненту

Для того чтобы произвести имитацию клика по CustomTextInput из прошлого примера сразу же после монтирования, можно использовать реф, чтобы получить доступ к пользовательскому <input> и явно вызвать его метод focusTextInput:

Кроме того, React поддерживает другой способ определения рефов, который называется «колбэк-рефы».

Для добавления рефа к функциональному компоненту нужно использовать - forwardRef(), useImperativeHandle().

```
class AutoFocusTextInput extends React.Component {
  constructor(props) {
    super(props);
    this.textInput = React.createRef();
  }

  componentDidMount() {
    this.textInput.current.focusTextInput();
  }

  render() {
    return (
      <CustomTextInput ref={this.textInput} />
    );
  }
}
```

Context API

В типичном React-приложении данные передаются сверху вниз (от родителя к дочернему компоненту) с помощью пропсов. Однако, этот способ может быть чересчур громоздким для некоторых типов пропсов (например, выбранный язык, UI-тема), которые необходимо передавать во многие компоненты в приложении.

Когда использовать контекст:

Контекст разработан для передачи данных, которые можно назвать «глобальными» для всего дерева React-компонентов (например, текущий аутентифицированный пользователь, UI-тема или выбранный язык).

Context API

React.createContext:

```
const MyContext = React.createContext(defaultValue);
```

Context.Provider:

```
<MyContext.Provider value={/* некоторое значение */}>
```

Context.Consumer:

```
<MyContext.Consumer>
  {value => /* отрендерить что-то, используя значение
</MyContext.Consumer>
```

Рендер-пропсы

Термин «рендер-проп» относится к возможности компонентов React разделять код между собой с помощью пропа, значение которого является функцией.

```
<DataProvider render={data => (
  <h1>Привет, {data.target}</h1>
)}/>
```

Компонент с рендер-пропом берёт функцию, которая возвращает React-элемент, и вызывает её вместо реализации собственного рендера.

```
render() {
  return (
    <div style={{ height: '100vh' }} onMouseMove={this.handleMouseMove}>
      {this.props.render(this.state)}
    </div>
  );
}
```

Такой подход, в частности, применяется в библиотеках React Router, Downshift и Formik.

Проверка знаний

TestProvider.com

Проверьте как Вы усвоили данный материал на TestProvider.com

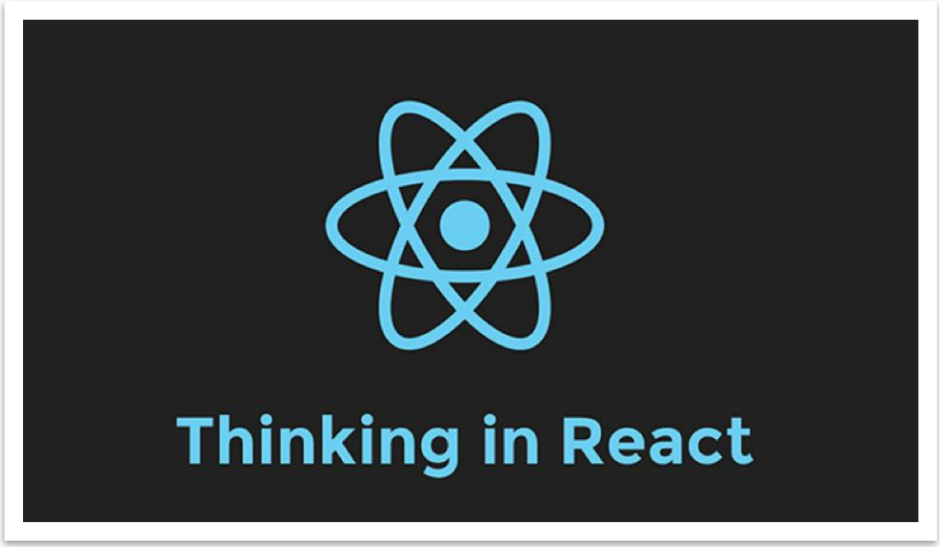
TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

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