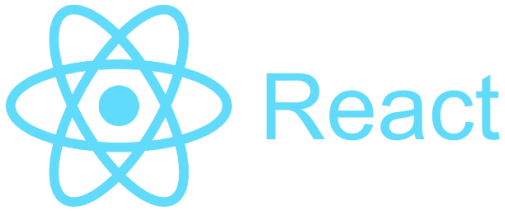


React Essential

JSX в React

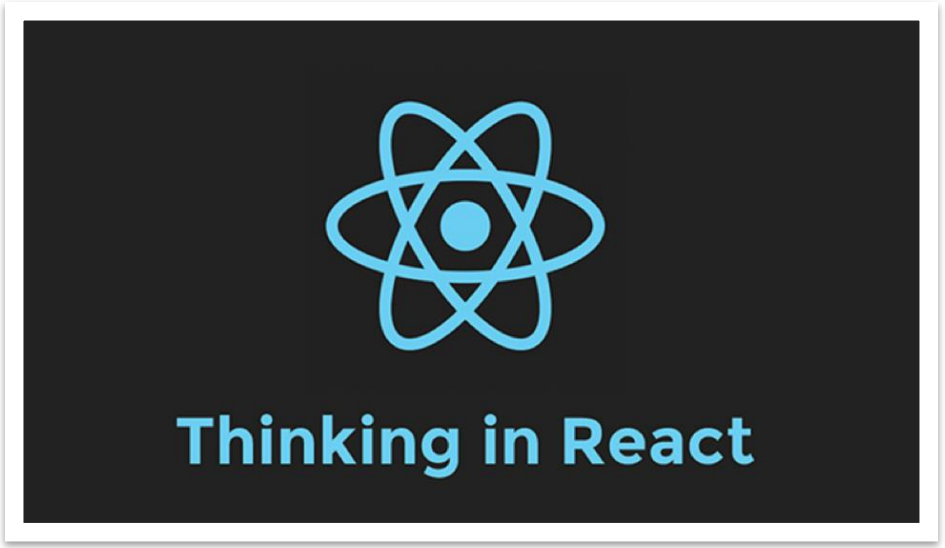
Introduction

Муляк Дмитрий

Front-end developer at GlobalTechMakers

dmitriymuliak

dmitriymuliak



Тема

JSX в React

JSX в React

1. Что такое JSX и как он работает
2. Создание простого компонента
3. Передача параметров (props)
4. Передача контента
5. Работа с условными операторами

Что такое JSX и как он работает

```
// JSX
var dropdown =
  <Dropdown>
    A dropdown list
    <Menu>
      <MenuItem>Do Something</MenuItem>
      { doFun ?
        <MenuItem>Do Something Fun!</MenuItem>
        :
        <MenuItem>Do Something Else</MenuItem>
      }
    </Menu>
</Dropdown>;
```

https://ru.reactjs.org/docs/introducing-jsx

JSX — синтаксический сахар, расширение языка JavaScript. Напоминает язык шаблонов, наделённый силой JavaScript. JSX производит «элементы» React.

JSX код:

```
<MyButton color="blue" shadowSize={2}>
  Click Me
</MyButton>
```

компилируется в:

```
React.createElement(
  MyButton,
  {color: 'blue', shadowSize: 2},
  'Click Me'
)
```

Встраивание выражений в JSX

Для встраивания выражения в JSX нужно обернуть его в фигурные скобки.

JSX допускает использование любых корректных JavaScript-выражений внутри фигурных скобок.

Например, 2 + 2, user.firstName и formatName(user) являются допустимыми выражениями, будучи обрамленными фигурными скобками:

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Марья',
  lastName: 'Моревна'
};

const element = (
  <h1>
    Здравствуй, {formatName(user)}!
  </h1>
);
```

JSX это тоже выражение

После компиляции каждое JSX-выражение становится обычным вызовом JavaScript-функции, результат которого — объект JavaScript.

Из этого следует, что JSX можно использовать внутри выражений if и циклов for, присваивать переменным, передавать функции в качестве аргумента и возвращать из функции.

```
function getGreeting(user) {
  if (user) {
    return <h1>Здравствуй, {formatName(user)}!</h1>;
  }
  return <h1>Здравствуй, незнакомец.</h1>;
}
```

Использование атрибутов JSX

Чтобы использовать строковый литерал в качестве атрибута, используются кавычки:

```
const element = <div tabIndex="0"></div>;
```

Если же в атрибут требуется указать JavaScript-выражение, то на помощь приходят фигурные скобки:

```
const element = <img src={user.avatarUrl}></img>;
```

Поскольку JSX ближе к JavaScript чем к HTML, React DOM использует стиль именования camelCase для свойств, вместо обычных имён HTML-атрибутов.

Например, class становится className в JSX, а tabindex становится tabIndex.

JSX представляет собой объекты

Babel компилирует JSX в вызовы React.createElement().
Следующие два примера кода эквивалентны между собой:

```
const element = (
  <h1 className="greeting">
    Привет, мир!
  </h1>
);
```

React.createElement() проводит некоторые проверки с целью выявить баги в коде, но главное — создаёт объект похожий на такой:

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Привет, мир!'
);
```

Эти объекты называются React-элементами. Можно сказать, что они описывают результат, который мы хотим увидеть на экране. React читает эти объекты и использует их, чтобы конструировать и поддерживать DOM.

```
// Примечание: этот код несколько упрощён.
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Привет, мир!'
  }
};
```

Рендеринг элементов

В отличие от DOM-элементов, элементы React — это простые объекты, не отнимающие много ресурсов. React DOM обновляет DOM, чтобы он соответствовал переданным React-элементам.

```
const element = <h1>Hello, world</h1>;
```

Элементы — это то, «из чего сделаны» компоненты.

Для рендеринга React-элемента в корневой узел DOM, вызовите ReactDOM.render() с React-элементом и корневым DOM узлом в качестве аргументов:

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

React обновляет только то, что необходимо.

https://ru.reactjs.org/docs/rendering-elements

React DOM сравнивает элемент и его дочернее дерево с предыдущей версией и вносит в DOM только минимально необходимые изменения.

Создание простого компонента

```
function Welcome(props) {
  return <h1>Привет, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Алиса" />
      <Welcome name="Базилио" />
      <Welcome name="Буратино" />
    </div>
  );
}

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

Компоненты позволяют разбить интерфейс на независимые части, про которые легко думать в отдельности. Их можно складывать вместе и использовать несколько раз.

Эта функция — компонент, потому что она получает данные в одном объекте («пропсы») в качестве параметра и возвращает React-элемент. Мы будем называть такие компоненты «функциональными», так как они буквально являются функциями.

https://ru.reactjs.org/docs/components-and-props

Передача параметров (props)

Пока что мы только встречали React-элементы, представляющие собой DOM-теги:

```
const element = <div />;
```

Но элементы могут описывать и наши собственные компоненты:

```
const element = <Welcome name="Алиса" />;
```

Когда React встречает подобный элемент, он собирает все JSX-атрибуты и дочерние элементы в один объект и передаёт их нашему компоненту. Этот объект называется «пропсы» (props).

Пропсы можно только читать. Компонент никогда не должен что-то записывать в свои пропсы — вне зависимости от того, функциональный он или классовый.

React-компоненты обязаны вести себя как чистые функции по отношению к своим пропсам.

Передача контета

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <Avatar user={props.author} />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

В компоненты можно передавать другие компоненты или данные с помощью атрибутов, к которым потом можно получить доступ от объекта props.

Компоненты можно вкладывать друг в друга.

Также можно передать данные в середину компонента, поместив их между открывающим и закрывающим тегом.

Получить доступ к данным можно с помощью свойства children в объекте props.

Работа с условными операторами

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  let button;

  if (isLoggedIn) {
    button = <LogoutButton onClick={this.handleLogoutClick} />;
  } else {
    button = <LoginButton onClick={this.handleLoginClick} />;
  }

  return (
    <div>
      <Greeting isLoggedIn={isLoggedIn} />
      {button}
    </div>
  );
}
```

Условный рендеринг в React работает так же, как условные выражения работают в JavaScript. Бывает нужно объяснить React, как состояние влияет на то, какие компоненты требуется скрыть, а какие — отрендерить, и как именно.

В таких ситуациях используйте условный оператор JavaScript или выражения, подобные if.

Работа с условными операторами

```
return (
  <div>
    <h1>Здравствуйте!</h1>
    {unreadMessages.length > 0 &&
      <h2>
        У вас {unreadMessages.length} непрочитанных сообщений.
      </h2>
    }
  </div>
);
```

Вы можете внедрить любое выражение в JSX, заключив его в фигурные скобки. Это правило распространяется и на логический оператор && языка JavaScript, которым можно удобно вставить элемент, в зависимости от условия.

```
return (
  <div>
    Пользователь <b>{isLoggedIn ? 'сейчас' : 'не'}</b> на сайте.
  </div>
);
```

Есть ещё один способ писать условия прямо в JSX. Вы можете воспользоваться тернарным оператором condition ? true : false.

```
if (!props.warn) {
  return null;
}
```

В редких случаях может потребоваться позволить компоненту спрятать себя, хотя он уже и отрендерен другим компонентом. Чтобы этого добиться, верните null вместо того, что обычно возвращается на рендеринг.

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.

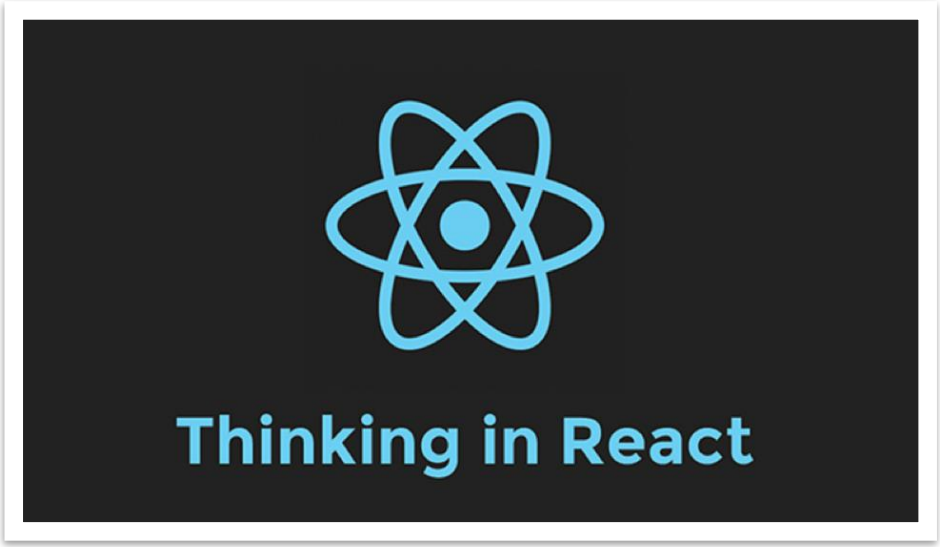
Проверьте как Вы усвоили данный материал на TestProvider.com

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