|  |
| --- |
| Министерство образования Республики Беларусь  УО «Полоцкий государственный университет им. Ефросинии Полоцкой» |

|  |  |
| --- | --- |
|  | Факультет информационных технологий  Кафедра технологий программирования |

|  |
| --- |
| РЕФЕРАТ |
| **React**  **Дисциплина**: «Программирование для Интернет» |

|  |  |
| --- | --- |
| Выполнили: | Студенты группы 19-ИТ-2  Гилимович А.Н., Суховило Н. |

|  |  |
| --- | --- |
| Проверила: | Преподователь  Деканова М.В. |

|  |
| --- |
| Полоцк, 2022 г. |

Что такое React

React (иногда React.js или ReactJS) — JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов.

React разрабатывается и поддерживается Facebook, Instagram и сообществом отдельных разработчиков и корпораций.

React может использоваться для разработки одностраничных и мобильных приложений. Его цель — предоставить высокую скорость разработки, простоту и масштабируемость. В качестве библиотеки для разработки пользовательских интерфейсов React часто используется с другими библиотеками, такими как MobX, Redux и GraphQL.

React был создан Джорданом Валке, разработчиком программного обеспечения из Facebook. На него оказал влияние XHP — компонентный HTML-фреймворк для PHP. Впервые React использовался в новостной ленте Facebook в 2011 году и позже в ленте Instagram в 2012 году. Исходный код React открыт в мае 2013 года на конференции «JSConf US».

React Native анонсирован на конференции Facebook «React.js Conf» в феврале 2015 года, а исходный код открыт в марте 2015 года. Он позволяет разрабатывать нативные Android-, iOS- и UWP-приложения с использованием React.

18 апреля 2017 года Facebook анонсировал React Fiber, переписанную и оптимизированную версию React. React Fiber станет основой разработки всех будущих функций и улучшений.

Особенности

1. ***Однонаправленная передача данных***

Свойства передаются от родительских компонентов к дочерним. Компоненты получают свойства как множество неизменяемых (англ. immutable) значений, поэтому компонент не может напрямую изменять свойства, но может вызывать изменения через callback-функции. Такой механизм называют «свойства вниз, события наверх».

1. **Виртуальный DOM**

React использует виртуальный DOM (англ. virtual DOM). React создаёт кэш-структуру в памяти, что позволяет вычислять разницу между предыдущим и текущим состояниями интерфейса для оптимального обновления DOM браузера. Таким образом программист может работать со страницей, считая, что она обновляется вся, но библиотека самостоятельно решает, какие компоненты страницы необходимо обновить.

1. ***Redux***

Часто React используют в связке с Redux для управления состояниями компонентов

1. ***JSX***

JavaScript XML (JSX) — это расширение синтаксиса JavaScript, которое позволяет использовать HTML-подобный синтаксис для описания структуры интерфейса. Как правило, компоненты написаны с использованием JSX, но также есть возможность использования обычного JavaScript. JSX напоминает другой язык, созданный в компании Фейсбук для расширения PHP, XHP.

1. ***Методы жизненного цикла***

Методы жизненного цикла позволяют разработчику запускать код на разных стадиях жизненного цикла компонента. Например:

* **shouldComponentUpdate** — позволяет предотвратить перерисовку компонента с помощью возврата false, если перерисовка не нужна.
* **componentDidMount** — вызывается после первой отрисовки компонента. Часто используется для запуска получения данных с удалённого источника через API.
* **render** — важнейший метод жизненного цикла. Каждый компонент должен иметь этот метод. Обычно вызывается при изменении данных компонента для перерисовки данных в интерфейсе.

1. ***Не только отрисовка HTML в браузере***

React используется не только для отрисовки HTML в браузере. Например, Facebook имеет динамические графики, которые отрисовываются в теге <canvas>. Netflix и PayPal используют изоморфные загрузки для отрисовки идентичного HTML на сервере и клиенте.

1. ***React Hooks***

Хуки позволяют использовать состояние и другие возможности React без написания классов.

Построение пользовательских хуков позволяет помещать логику компонента в повторно используемые функции.

Привет, мир

Самый маленький пример на React выглядит так:

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<h1>Привет, мир!</h1>);

На странице появится заголовок «Привет, мир!».

Сегодня мы рассмотрим строительные блоки React-приложений: элементы и компоненты. Освоив их, вы сможете создавать сложные приложения из маленьких повторно используемых частей.

Знакомство с JSX

**JSX** — расширение языка JavaScript. Когда требуется объяснить React, как должен выглядеть UI реклмендуется использовать JSX. JSX напоминает язык шаблонов, наделённый силой JavaScript. JSX производит «элементы» React.

React исходит из принципа, что логика рендеринга неразрывно связана с прочей логикой UI: с тем, как обрабатываются события, как состояние изменяется во времени и как данные готовятся к отображению.

Вместо того, чтобы искусственно разделить технологии, помещая разметку и логику в разные файлы, React разделяет ответственность с помощью слабо связанных единиц, называемых «*компоненты*», которые содержат и разметку, и логику.

React можно использовать и без JSX, но большинство людей ценит его за наглядность при работе с UI, живущем в JavaScript-коде. Помимо этого, JSX помогает React делать сообщения об ошибках и предупреждениях понятнее.

JSX допускает использование любых корректных JavaScript-выражений внутри фигурных скобок. Например, *2 + 2*, *user.firstName* и *formatName(user)* являются допустимыми выражениями.

Рендеринг элементов

**Элементы** — мельчайшие кирпичики React-приложений. лемент описывает то, что вы хотите увидеть на экране.

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

**Рендеринг элемента в DOM**

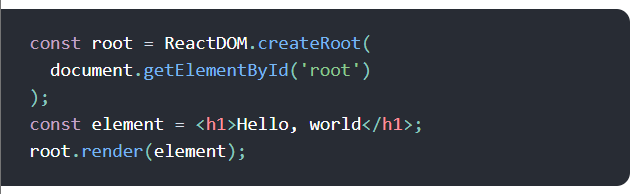
Допустим, в вашем HTML-файле есть <div>:



Такой тег будет называться «корневым» узлом DOM, так как React DOM будет управлять его содержимым.

Обычно в приложениях, написанных полностью на React, есть только один корневой элемент. При встраивании React в существующее приложение вы можете рендерить во столько независимых корневых элементов, во сколько посчитаете нужным.

Для рендеринга React-элемента, сперва передайте DOM-элемент в ReactDOM.createRoot(), далее передайте с React-элементом в *root.render()*:

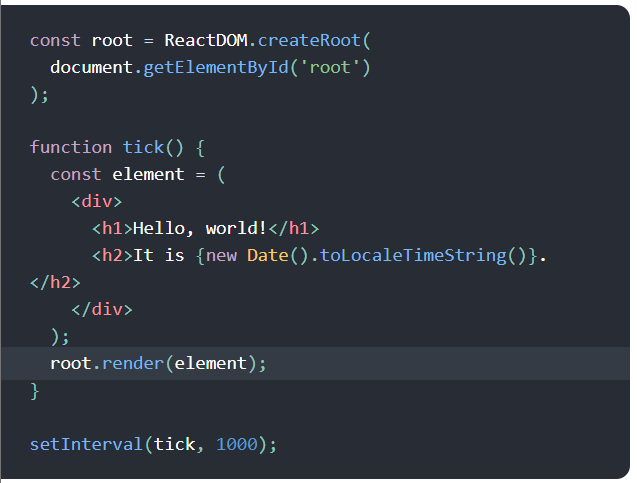


**Обновление элементов на странице**

Элементы React иммутабельны. После создания элемента нельзя изменить его потомков или атрибуты. Элемент похож на кадр в фильме: он отражает состояние интерфейса в конкретный момент времени.

Пока что, мы знаем только один способ обновить интерфейс — это создать новый элемент и передать его в root.render().

Рассмотрим пример с часами:



В этом примере root.render() вызывается каждую секунду с помощью колбэка setInterval().

React обновляет только то, что необходимою. React DOM сравнивает элемент и его дочернее дерево с предыдущей версией и вносит в DOM только минимально необходимые изменения.

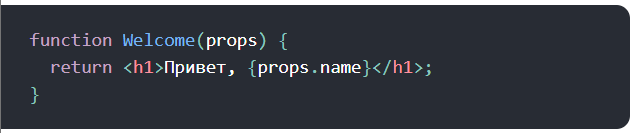
Компоненты и пропсы

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

Во многом компоненты ведут себя как обычные функции JavaScript. Они принимают произвольные входные данные (так называемые «пропсы») и возвращают React-элементы, описывающие, что мы хотим увидеть на экране.

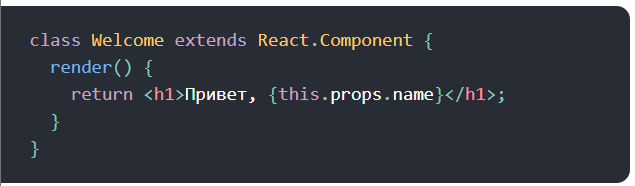
**Функциональные и классовые компоненты**

Проще всего объявить React-компонент как функцию:



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

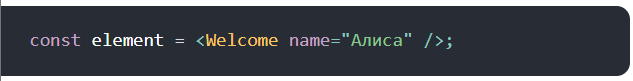
Ещё компоненты можно определять как классы:



С точки зрения React, эти два компонента эквивалентны. Функциональным и классовым компонентам доступны дополнительные возможности.

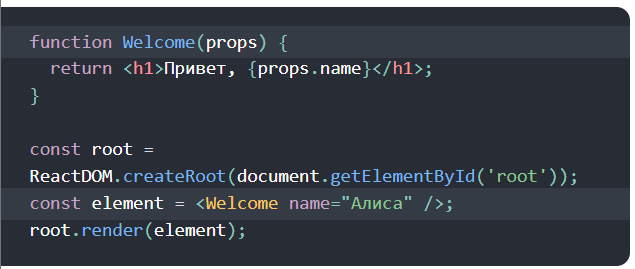
**Как отрендерить компонент**

Элементы могут описывать и наши собственные компоненты:



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).

Например, этот компонент выведет «Привет, Алиса» на страницу:



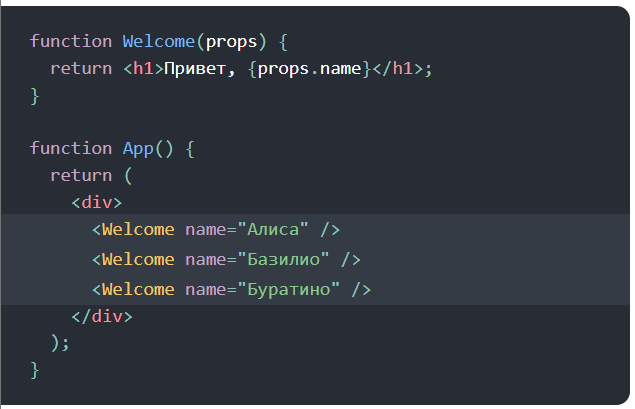
1. Мы вызываем root.render() c React-элементом <Welcome name="Алиса" />.
2. React вызывает наш компонент Welcome с пропсами {name: 'Алиса'}.
3. Наш компонент Welcome возвращает элемент <h1>Привет, Алиса</h1> в качестве результата.
4. React DOM делает минимальные изменения в DOM, чтобы получилось <h1>Привет, Алиса</h1>.

Всегда называйте компоненты с заглавной буквы.

Если компонент начинается с маленькой буквы, React принимает его за DOM-тег. Например, <div /> это div-тег из HTML, а <Welcome /> это уже наш компонент Welcome, который должен быть в области видимости.

Компоненты могут ссылаться на другие компоненты в возвращённом ими дереве. Это позволяет нам использовать одну и ту же абстракцию — компоненты — на любом уровне нашего приложения. Неважно, пишем ли мы кнопку, форму или целый экран: все они, как правило, представляют собой компоненты в React-приложениях.

Например, компонент App может отрендерить компонент Welcome несколько раз



**Пропсы можно только читать**

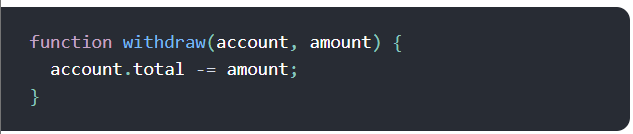
Компонент никогда не должен что-то записывать в свои пропсы — вне зависимости от того, функциональный он или классовый.

Возьмём для примера функцию sum:



Такие функции называют «чистыми», потому что они не меняют свои входные данные и предсказуемо возвращают один и тот же результат для одинаковых аргументов.

А вот пример нечистой функции — она записывает данные в свои же аргументы:



React достаточно гибкий, но есть одно правило, которое нельзя нарушать:

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

Состояние

https://metanit.com/web/react/2.4.php

Объект state описывает внутреннее состояние компонента, он похож на props за тем исключением, что состояние определяется внутри компонента и доступно только из компонента.

Если props представляет входные данные, которые передаются в компонент извне, то состояние хранит такие объекты, которые создаются в компоненте и полностью зависят от компонента.

Также в отличие от props значения в state можно изменять.

И еще важный момент - значения из state должны использоваться при рендеринге. Если какой-то объект не используется в рендерниге компонента, то нет смысла сохранять его в state.

Нередко state описывает какие-то визуальные свойства элемента, которые могут изменяться при взаимодействии с пользователем. Например, кнопку нажали, и соответственно можно изменить ее состояние - придать ей какой-то другой цвет, тень и так далее. Кнопку нажали повторно - можно вернуть исходное состояние.

Стоит отметить, что традиционно объект state применялся только в классах-компонентах. В функциональных же компонентах для управления состоянием применяется другая архитектура, основанная на хуках.

При использовании класса-компонента единственное место, где можно установить объект state — это конструктор класса:

Для обновления состояния вызывается функция setState():

this.setState({welcome: "Привет React"});

Изменение состояния вызовет повторный рендеринг компонента, в соответствии с чем веб-страница будет обновлена.

В то же время не стоит изменять свойства состояния напрямую, например:

this.state.welcome = "Привет React";

В данном случае изменения повторного рендеринга компонента происходить не будет.

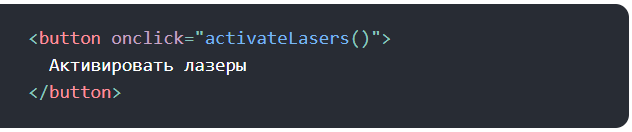
При этом нам не обязательно обновлять все его значения. В процессе работы программы мы можем обновить только некоторые свойства. Тогда необновленные свойства будут сохранять старые значения.

Обработка событий

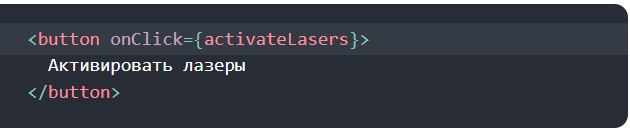
Обработка событий в React-элементах очень похожа на обработку событий в DOM-элементах. Но есть несколько синтаксических отличий:

* События в React именуются в стиле camelCase вместо нижнего регистра.
* С JSX вы передаёте функцию как обработчик события вместо строки.

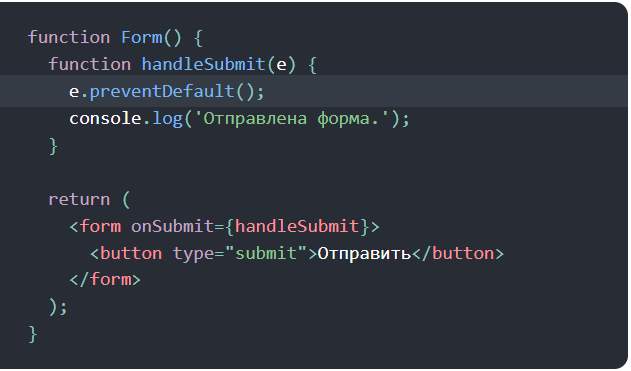
Например, в HTML:



В React немного иначе:



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



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 класса, в качестве обработчика события обычно выступает один из методов класса.

Условный рендеринг

React позволяет разделить логику на независимые компоненты. Эти компоненты можно показывать или прятать в зависимости от текущего состояния.

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.

Можно создать компонент Greeting, который отражает один из этих компонентов в зависимости от того, выполнен ли вход на сайт:



В этом примере рендерится различное приветствие в зависимости от значения пропа isLoggedIn.

Элементы React можно сохранять в переменных. Это может быть удобно, когда какое-то условие определяет, надо ли рендерить одну часть компонента или нет, а другая часть компонента остаётся неизменной.

Предотвращение рендеринга компонента

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.

Сам факт возврата null из метода render компонента никак не влияет на срабатывание методов жизненного цикла компонента.

Формы