Передовые инженерные школы

МИНОБРНАУКИ РОССИИ

УНИВЕРСИТЕТ ИННОПОЛИС

Занятие 17

PropsTypes | SyntheticEvent | Ref

План занятия

ПЕРЕДОВАЯ
ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

1. Props Types
2. Synthetic Event
3. Ref

Props Types

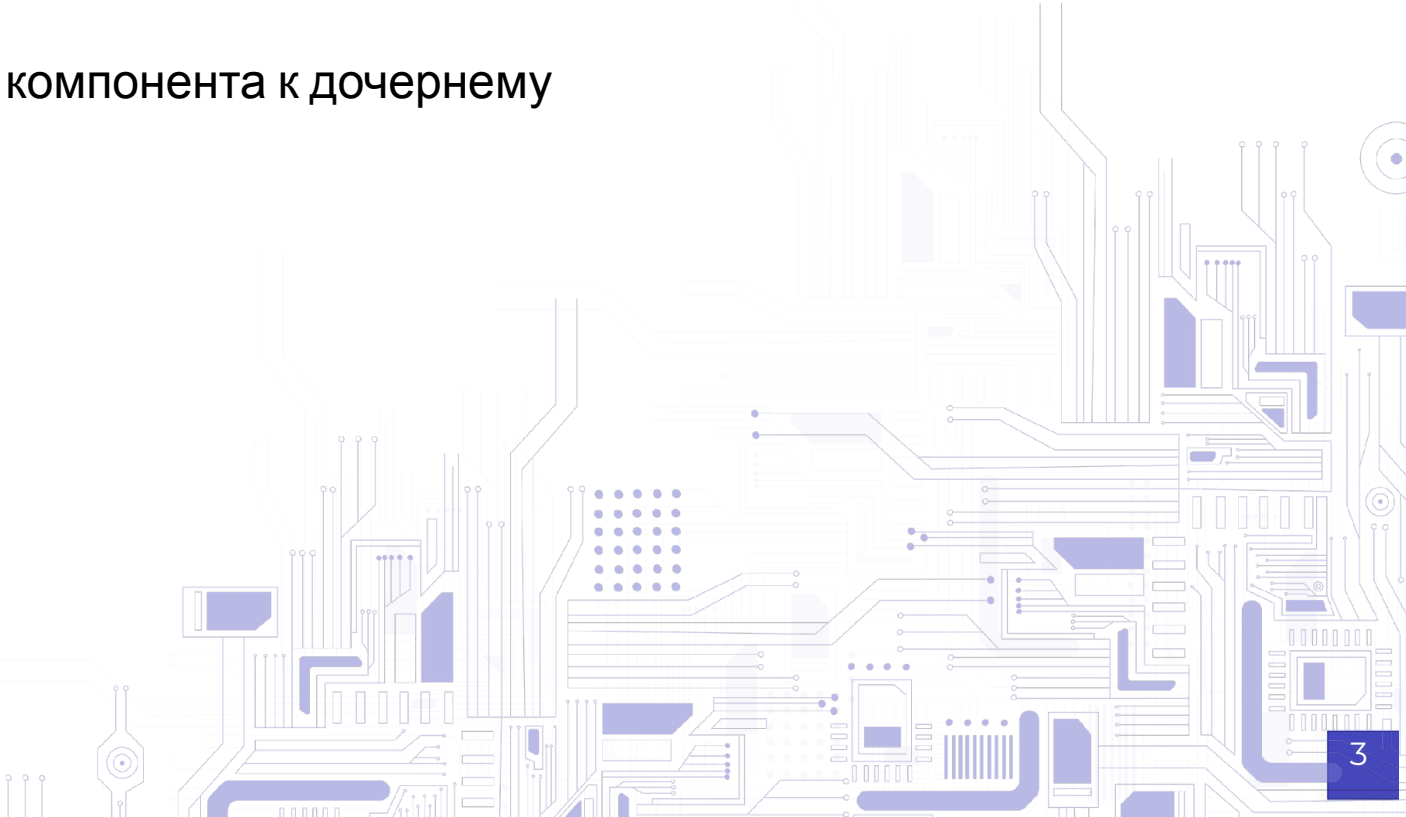
ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

PropTypes - хорошая защита, когда дело доходит до отладки приложений

Разберем концепцию props перед PropTypes

Props - это свойства, доступные только для чтения, которые совместно используются компонентами для придания динамичности однонаправленному потоку React.

Props в основном передаются от родительского компонента к дочернему компоненту, но возможно и обратное



Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

PropTypes - это просто механизм, который гарантирует, что передаваемое значение имеет правильный тип данных.

PropsTypes гарантирует, что консоль не выдаст ошибку в момент выполнения приложения, с которой может быть нелегко справиться

- Не нужно использовать PropsTypes в малых приложениях, таких как одностроничники
- Для крупных проектов, использование часто является мудрым выбором и хорошей практикой

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Как использовать PropTypes

До выпуска React 15.5.0 PropTypes были доступны в пакете React, но теперь нужно добавить библиотеку prop-types в проект в ручную

Для этого, выполнить следующую команду в терминале:

```
npm install prop-types --save
```

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Используем PropTypes для проверки любых данных, которые получаем от props

Но перед использованием в начале импортируем пакет prop-types в приложение:

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

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

PropTypes часто используются после описания компонента и начинаются с имени компонента, как показано:

```
import React from 'react';
import { PropTypes } from "prop-types";

const Count = (props) => {
  return (
    <>
      ..........
    </>
  )
};

Count.propTypes = {

}

export default Count;
```

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

PropTypes - это объекты с пара ключ и значение

где:

- key - это имя prop
- значение представляет тип или класс, с помощью которого они определены

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Основные типы PropTypes

Тип prop, - подпадает под категорию примитивных типов в JavaScript, таких как логическое значение, строка, объект и так далее

Тип	Класс	Пример
Строка	propType.string	“черт возьми”
Объект	propType.object	{имя: “Rohit”}
Число	propType.number	10
Логическое	propType.bool	true / false
Функция	propType.func	const say = {console.log(“привет”)}
Символ	propType.symbol	Символ (“m”)

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

В коде

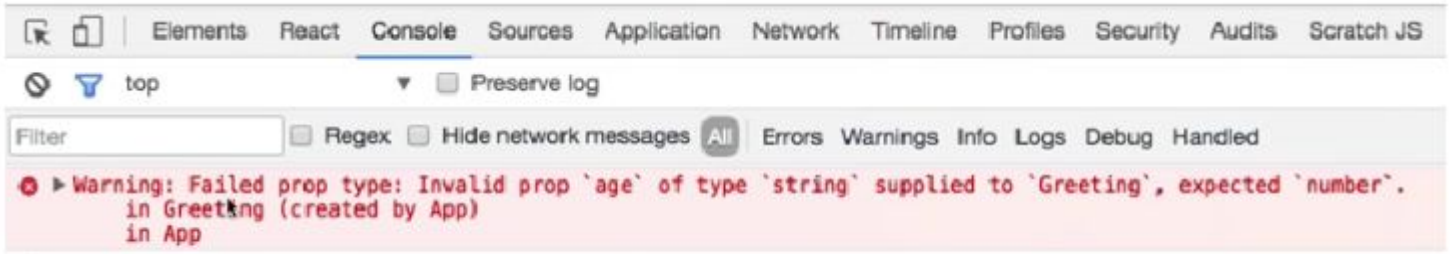
```
Count.propTypes = {
  name: PropTypes.string,
  age: PropTypes.number,
  address: PropTypes.object,
  friends: PropTypes.array,
};
```

В приведенном коде ожидается, что name prop будет иметь значение, представляющее собой строку, age - число, address - объект, а friends - массив

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Если какое-либо значение, отличное от использовании в описании props, что и значение, в консоли отобразится ошибка, подобная этой:



Ошибка консоли из-за неправильных PropTypes

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Чтобы поле из вышеперечисленных было обязательным добавим isRequired, чтобы убедиться, что отображается предупреждение, если prop не предоставлен

```
Count.propTypes = {
  basicObject: PropTypes.object,
  numbers: PropTypes.objectOf(PropTypes.numbers),
  messages: PropTypes.instanceOf(Message),
  contactList: PropTypes.shape({
    name: PropTypes.string.isRequired,
    phone: PropTypes.string.isRequired,
  }),
};
```

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Collective Type

Типы массивов
обсудим все варианты, которые могут быть сформированы с помощью массива, на их примерах

Тип	Класс	Пример
Массив	PropTypes.array	[]
Массив чисел	PropTypes.ArrayOf([тип])	[1,2,3]
Массив строк	PropTypes.oneOf([arr])	[“Красный”, “синий”]
Массив объектов	PropTypes.oneOfType([типы])	PropTypes.string, PropTypes.instanceOf(заголовок)

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Collective Type

Типы массивов

```
Count.propTypes = {
  counts: PropTypes.array,
  users: PropTypes.arrayOf(PropTypes.object),
  alarmColor: PropTypes.oneOf(['red', 'blue']),
  description: PropTypes.oneOfType([
  PropTypes.string,
  PropTypes.instanceOf(Title)
  ]),
}
```

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Collective Type

Типы объектов
Похоже на типы массивов:

Тип	Класс	Пример
Объект	PropTypes.object	{имя: “Anu”}
Числовой объект	PropTypes.objectOf()	{возраст: 25}
Форма объекта	PropTypes.shape()	{имя: PropTypes.string, телефон: PropTypes.string}
Экземпляр	PropTypes.objectOf()	Новое сообщение()

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Collective Type

Типы объектов

```
Count.propTypes = {
  basicObject: PropTypes.object,
  numbers: PropTypes.objectOf(PropTypes.numbers),
  messages: PropTypes.instanceOf(Message),
  contactList: PropTypes.shape({
    name: PropTypes.string,
    phone: PropTypes.string,
  }),
};
```

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Advance Type Checking - Предварительная проверка типов

Как проверить наличие компонента React

Нужно проверить, является ли prop компонентом React, можно использовать **PropTypes.element**

Полезно для определения того, чтобы у компонента всегда был только один дочерний компонент

Тип	Класс	Пример
Элемент	PropTypes.элемент	<Название />

```
Count.propTypes = {
  displayEle: PropTypes.element,
};
```

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Advance Type Checking - Предварительная проверка типов

Как проверить наличие имени компонента React

Проверить, является ли prop именем компонента React, используем **PropTypes.ElementType**

```
Component.propTypes = {
  as: PropTypes.elementType
}
```

```
<AnotherComponent as={Component} />
```

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Пользовательские типы

Можно создать пользовательский валидатор или проверку типов на наличие реквизитов, но для этого требуется объект error, если проверка завершается неудачей

Можно использовать как для массивов, так и для объектов, но объект error будет вызываться для каждого ключа в массиве или объекте

Первые два аргумента средства проверки - это сам массив или объект и ключ текущего элемента

Тип	Класс	Пример
Пользовательский	функция (props, propName, ComponentName) {}	“привет”
Пользовательский массив	PropTypes.ArrayOf(функция(props, propName, ComponentName) {})	[“привет”]

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

```
Count.propTypes = {  // normal functionn
  customProp: function (props, propName, componentName) {
    if (!/matchme/.test(props[propName])) {
      return new Error(
        "Invalid prop `" +
          propName +
          "` supplied to" +
          " `" +
          componentName +
          "`. Validation failed."
      );
    }
  },
};
```

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

```
Count.propTypes = { // array function
  customArrayProp: PropTypes.arrayOf(function (
    propValue,
    key,
    componentName,
    location,
    propFullName
  ) {
    if (!/matchme/.test(propValue[key])) {
      return new Error(
        "Invalid prop `" +
          propFullName +
          "` supplied to" +
          " `" +
          componentName +
          "`. Validation failed."
      );
    }
  }),
};
```

Props Types

Props по умолчанию

```
Header.defaultProp = {
  title: "GitHub Users",
};
```


Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

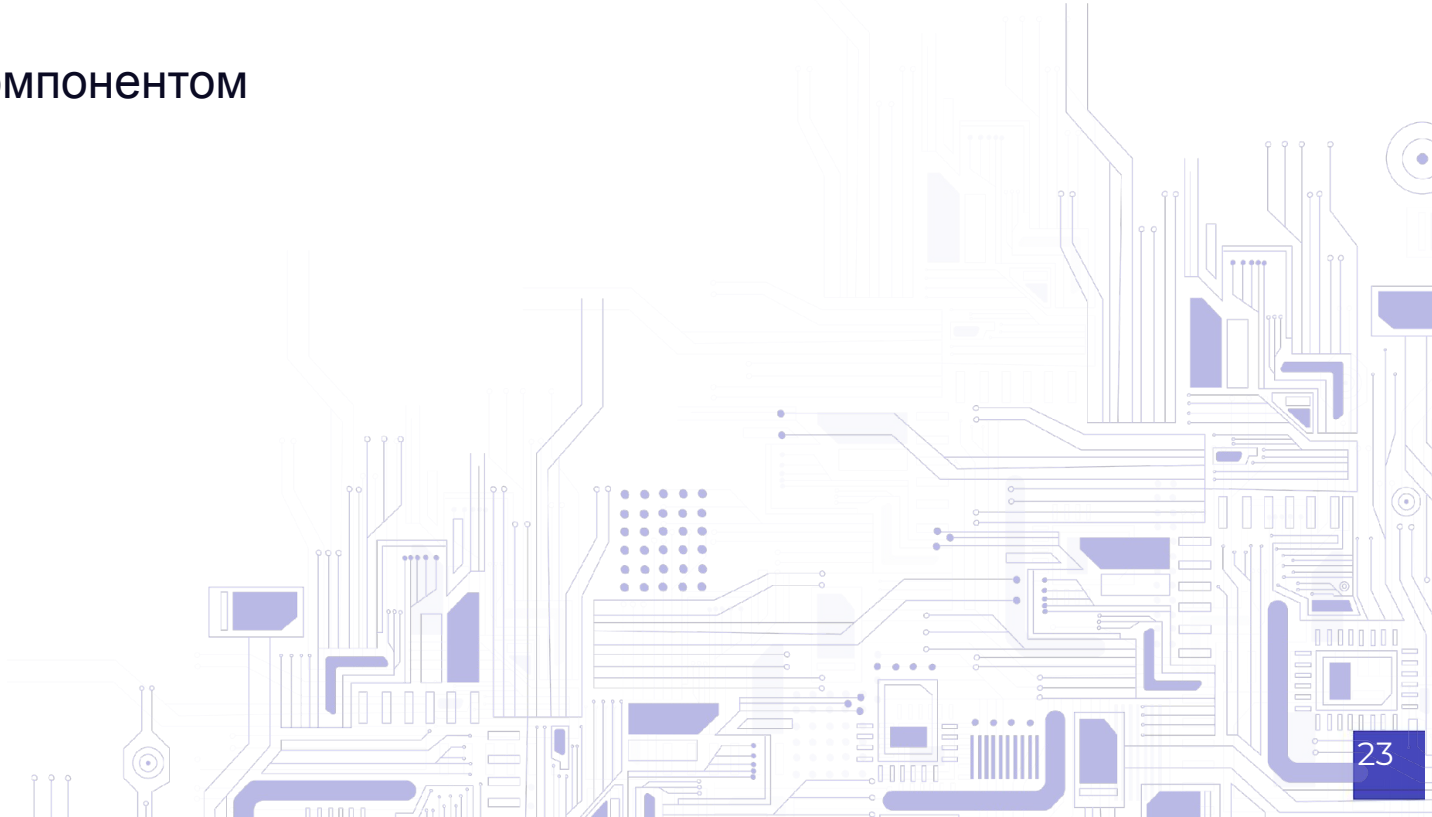
Пример времени

Как все это работает в коде React

Создадим два повторно используемых компонента:

- **About.js**
- **Count.js**.

- компонент **About** является родительским компонентом
- компонент **Count** - дочерним компонентом



Props Types

JS About.js M

src > JS About.js > ...

```jsx
import React from "react";
import Count from "./Count";
const About = () => {
  return (
    <>
      <div className="app">
        <Count name="Ateev"
        age={25} />
      </div>
    </>
  );
};

export default About;
```

JS Count.js M

src > JS Count.js > Count

```jsx
import React from "react";
import PropTypes from "prop-types";
const Count = (props) => {
  return (
    <div className="bg-dark text-white">
      <h1 className="px-5 pt-5">
        My name is {props.name} of  
        {props.age}.
      </h1>
      <p className="fs-5 px-5 pb-5">I am a
      front-end developer.</p>
    </div>
  );
};
Count.propTypes = {
  name: PropTypes.string,
  age: PropTypes.number,
};
export default Count;
```

My name is Ateev of 25.

I am a front-end developer.

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Изменим значение параметра age с числа на строку без изменения его типа (PropTypes)

```
import React from "react";
import Count from "./Count";
const About = () => {
  return (
    <>
      <div className="app">
        <Count name="Ateev" age="25" />
      </div>
    </>
  );
};

export default About;
```

Props Types

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Получим сообщение об ошибке в консоли с сообщением об этом:

В нем четко указано, что переданное значение параметра age не соответствует ожидаемому значению (PropTypes)

PropTypes не такие типобезопасные, как TypeScript, но их намного проще настроить и работать с ними

Synthetic Event

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Сначала давайте разберемся, как React работает с событиями

React прослушивает каждое событие на уровне документа, после получения события из браузера React оборачивает это событие оболочкой, которая имеет тот же интерфейс, что и собственное событие браузера, что означает, что мы все еще можем использовать такие методы, как preventDefault()

Так зачем нужна эта оболочка ?!!😉

Synthetic Event

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Представьте ситуацию, когда одно и то же событие имеет разные имена в разных браузерах

Представьте событие, которое срабатывает, когда пользователь подмигивает😉, это событие в Chrome называется **A**, в Safari - **B**

В таком случае нам нужно будет создавать разные реализации для каждого браузера😵

Synthetic Event

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Что делает оболочка?

Регистрирует все разные имена для одного и того же эффекта события, ***подмигивая*** *в нашем случае*, только с одним именем, поэтому в случае, когда мы хотим прослушать наш эффект подмигивания вместо того, чтобы прослушивать **A** для Chrome и **B** для Safari, мы просто используем **onWink**, оболочку, которую react создает вокруг реального события

Поэтому всякий раз, когда запускаем событие в компоненте React, фактически имеем дело не с реальным событием DOM, а с пользовательским типом события React, **синтетическим событием**.

Synthetic Event

Все onClick (ы), onBlur (ы), onChange (ы), которые когда-либо использовали в компонентах react, это ненастоящие, это синтетические события react

Больше не нужно думать о разных реализациях браузеров, react делает создание кроссбраузерных приложений намного менее болезненным, что означает, что можно больше сосредоточиться на коде и логике приложения

Synthetic Event

Еще одним заметным преимуществом синтетических событий является то, что React повторно использует эти объекты events путем их объединения, что приводит к увеличению производительности

Как только вызывается обработчик событий

обработчик событий - это метод, выполняемый при запуске события

все свойства этого объекта event будут аннулированы, *им будут установлены их пустые состояния по умолчанию*, чтобы быть готовыми к повторному использованию снова

И можно увидеть

предупреждение: это синтетическое событие повторно используется по соображениям производительности в браузере.

Synthetic Event

Представьте

```
import React, { useState } from "react"

const ExampleComponent = (() => {
  const [counter, setCounter] = useState()

  function handelArrowBtn(event) {
    if (event.keyCode === 40) { //down arrow button
      setCounter(counter - 1)
    } else if (event.keyCode === 38) { // up arrow button
      setCounter(counter + 1)
    }
  }

  return (
    <div>
      <input
        type="number"
        value={counter}
        onKeyDown={handelArrowBtn}
       />
    </div>
  )
})

export default ExampleComponent
```

Synthetic Event

Этот счетчик не будет ни увеличен, ни уменьшен

И прекрасное красное предупреждение будет напечатано в консоли браузера

Synthetic Event

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Давайте посмотрим, что здесь происходит...

После вызова функции обработчика событий, handelArrowBtn(), объект синтетического события, onKeyDown, обнуляется, старые значения ключей внутри этого объекта больше не существуют, объект события возвращается в исходное состояние, чтобы быть готовым к повторному использованию, а поскольку это объект, то у нашей handelArrowBtn() есть доступ к нему по ссылке, что означает, что наша функция теперь имеет доступ к объекту события с его исходным состоянием (обнуленной версией)

Synthetic Event

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Как мы можем это решить?!😮

Сохраняем нужное нам свойство event

```
function handelArrowBtn(event) {
  let keyCode = event.keyCode
  if (keyCode === 40) {
    setCounter(counter - 1)
  } else if (keyCode === 38) {
    setCounter(counter + 1)
  }
}
```

или

Synthetic Event

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

можем передавать нужные свойства в качестве аргументов функции-обработчику событий вместо прямого доступа к ней из функции

```
return (
    <div>
      <input
        type="number"
        value={counter}
        onKeyDown={(e) => handelArrowBtn(e.keyCode)}
      />
    </div>
  )
```

Synthetic Event

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

А еще

Использование event.persist() который удалит синтетическое событие из пула, что позволит получить доступ к свойствам объекта event в коде

```
function handelArrowBtn(event) {
   event.persist()
   if (event.keyCode === 40) {
      setCount(count - 1)
    } else if (event.keyCode === 38) {
      setCount(count + 1)
    }
  }
```

Ref

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

React автоматически обновляет DOM в соответствии с выводом на экран, поэтому компонентами не часто требуется манипулировать напрямую

Иногда может понадобиться доступ к элементам DOM, управляемым React - например, для фокусировки узла, прокрутки к нему или измерения его размера и положения

Встроенного способа сделать это в React нет, поэтому понадобится *ref(т.е. ссылка на элемент)* на узел DOM

Ref

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Получение ссылки на узел

Чтобы получить доступ к узлу DOM, управляемому React, сначала импортируйте хук useRef:

```
import { useRef } from 'react’;
```

Затем используйте его для объявления ссылки внутри вашего компонента:

```
const myRef = useRef(null);
```

Передайте его узлу DOM в качестве атрибута ref:

```
<div ref={myRef}>
```

Ref

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Хук useRef возвращает объект с единственным свойством current

Изначально myRef.current будет null

Когда React создаст DOM-узел для этого , React поместит ссылку на этот узел в myRef.current

Затем можно обращаться к этому узлу DOM из обработчиков событий и использовать встроенные API браузера

```
myRef.current.scrollIntoView();
```

Ref


```jsx
import { useRef } from 'react';

export default function Form() {
    const inputRef = useRef(null);

    function handleClick() {
        inputRef.current.focus();
    }

    return (
        <>
            <input ref={inputRef} />
            <button onClick={handleClick}>
                Focus the input
            </button>
        </>
    );
}
```

Ref

Манипуляции с DOM являются наиболее распространенным случаем использования ссылок.

Но хук useRef можно использовать для хранения других вещей вне React, например, идентификаторов таймеров.

Аналогично состоянию, ссылки остаются между рендерами.

Ссылки похожи на переменные состояния, которые не вызывают повторных рендеров, когда вы их устанавливаете.

Ref

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Доступ к узлам DOM другого компонента

Когда помещаете ссылку на встроенный компонент, который выводит элемент <input />,

React установит свойство current этой ссылки на соответствующий узел DOM (фактический <input /> в браузере)

Однако если попытаетесь поместить ссылку на свой собственный компонент, например <MyInput />, по умолчанию вы получите null

Чтобы помочь заметить проблему, React выводит ошибку в консоль:

```
Warning: Function components cannot be given refs.
Attempts to access this ref will fail.
Did you mean to use React.forwardRef()?
```

Ref

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Происходит потому, что по умолчанию React не позволяет компоненту обращаться к узлам DOM других компонентов. Даже своим собственным детям! Это намеренно.

Ссылки - это аварийный люк, который следует использовать очень редко. Ручное манипулирование DOM-узлами другого компонента делает ваш код еще более хрупким

Вместо этого, компоненты, узлы DOM которых нужно передать, должны оптировать такое поведение

В компонент можно указать, что он "переадресует" свою ссылку одному из своих дочерних компонентов

MyInput может использовать API forwardRef:

```
const MyInput = forwardRef((props, ref) => {
    return <input {...props} ref={ref} />;
});
```

Ref

это работает:

- <MyInput *ref*={inputRef} /> говорит React поместить соответствующий узел DOM в inputRef.current. Компонент MyInput нужно настроить - по умолчанию он этого не делает

- Компонент MyInput объявлен с использованием forwardRef. Это позволяет ему получать inputRef сверху в качестве второго аргумента ref, который объявляется после props

- Сам MyInput передает полученный ref в <input> внутри него

Ref

- Обычно шаблоном для низкоуровневых компонентов, таких как кнопки, входы и т.д., является передача ссылок на их узлы DOM

- Высокоуровневые компоненты, такие как формы, списки или разделы страницы, обычно не передают свои узлы DOM, чтобы избежать случайных зависимостей от структуры DOM

Ref

Когда React присоединяет рефы

В React каждое обновление делится на две фазы:

- Во время render, React вызывает компоненты, чтобы выяснить, что должно быть на экране.
- Во время commit, React применяет изменения в DOM

Во время первого рендеринга узлы DOM еще не были созданы, поэтому ref.current будет null. А во время рендеринга обновлений, узлы DOM еще не были обновлены. Поэтому читать их еще рано.

React устанавливает ref.current во время фиксации. Перед обновлением DOM, React устанавливает затронутые значения ref.current в null. После обновления DOM, React немедленно прикрепляет их к соответствующим узлам DOM

Обращаетесь к рефкам принято из обработчиков событий
Чтобы сделать с рефкой действие, но нет конкретного события, в котором можно сделать манипулирование и нужен useEffect.

Ref

Избегайте изменения узлов DOM, управляемых React. Изменение, добавление дочерних элементов или удаление дочерних элементов из элементов, управляемых React, может привести к непоследовательным визуальным результатам или сбоям

Однако это не означает, что этого нельзя делать вообще. Это требует осторожности. Можно безопасно изменять части DOM, которые у React нет причин обновлять.

Например, если какой-то всегда пуст в JSX, у React не будет причин трогать его список детей. Поэтому безопасно вручную добавлять или удалять там элементы.

Передовые инженерные школы

МИНОБРНАУКИ РОССИИ

УНИВЕРСИТЕТ ИННОПОЛИС

Спасибо за внимание