



# Web-технологии

Использование библиотеки React

# Содержание

- Библиотека Immutable.JS
- Понятие JSX
- Компоненты и их иерархия, props, state
- Жизненный цикл компонентов, события
- Работа с формами в React
- Маршрутизация, хуки
- Redux – создание единого состояния приложения
  - действия, редьюсеры, хранилище

<https://facebook.github.io/immutable-js/>

<https://ru.reactjs.org/>

<https://metanit.com/web/react/>

<https://rajdee.gitbooks.io/redux-in-russian/>

<https://github.com/facebook/react>

# Неизменяемые объекты и коллекции



<https://immutable-js.com/>

Не нужны  
блокировки в  
многопоточных  
приложениях

Хранение проще

Копирование за  
константное  
время

Оптимизация  
сравнения  
значений

- Иммутабельность
  - после создания нельзя изменить!
- × Видимые мутации
  - контролируются наблюдателями через API
- × Невидимые мутации
  - side-эффекты

# Преимущества неизменяемых объектов и коллекций

- Нет side-эффектов
- Чистые функции
  - используют только то, что передано в качестве параметров
- Ссылочная прозрачность
  - можно заменить любую функцию, будучи уверенным, что это не повлияет на остальные
- Персистентность
  - возможность хранения старых версий данных
- Ленивые вычисления
  - не нужно беспокоиться, что данные будут изменены
- Композиции
  - прозрачность последовательного применения функций
- Использование памяти
  - нет пиковых значений использования памяти (упрощается работа сборщика мусора)

# Примеры ImmutableJS (1)

5

## Пример с Map

npm install immutable

```
const { Map } = require('immutable')
const map1 = Map({ a: 1, b: 2, c: 3 })
const map2 = map1.set('b', 47)
console.log(map1.get('b') + " vs. " + map2.get('b')) // 2 vs. 47
```

## Пример с List #1

```
const { List } = require('immutable')
const originalList = List([ 0 ]);
console.log(originalList) // List [ 0 ]
console.log( // List [ 0, 1 ]
  originalList.set(1, 1))
console.log( // List [ "overwritten" ]
  originalList.set(0, 'overwritten'))
console.log( // List [ 0, undefined, 2 ]
  originalList.set(2, 2))
```

## Пример с List #2

```
const { List } = require('immutable')
const originalList = List([ 1, 2, 3, 5, 8 ]);
console.log(originalList) // List [ 1, 2, 3, 5, 8 ]
console.log( // 19
  originalList.reduce((accum, curValue, index)=> accum + curValue))
console.log( // List [ 1, 102, 203, 305, 408 ]
  originalList.map((currValue, index) => currValue + index * 100))
console.log( // List [ 8, 5, 3, 2, 1 ]
  originalList.reverse())
console.log(originalList) // List [ 1, 2, 3, 5, 8 ]
```

# Примеры ImmutableJS (2)

6

**Seq** предназначен для «ленивых» операций. Не создаёт дополнительные коллекции при выполнении методов, применимых к коллекции

```
const { Seq } = require('immutable')
const sum = (collection) =>
  collection.reduce((sum, x) => sum + x, 0)
console.log(
  Seq([ 1, 2, 3 ])
    .map(x => x + 1)
    .filter(x => x % 2 === 0)
    .reduce((accum, curValue, index) => accum + curValue))
// 6
```

**Range** возвращает числа в заданном интервале с заданным шагом

```
const { Range } = require('immutable')
console.log( // [ 0, 1, 2, 3, ... ]
  Range())
console.log( // [ 10, 11, 12, 13, ... ]
  Range(10))
console.log( // [ 10, 11, 12, 13, 15 ]
  Range(10, 15))
console.log( // [ 10, 15, 20, 25, 30 ]
  Range(10, 30, 5))
console.log( // [ 30, 25, 20, 15, 10 ]
  Range(30, 10, 5))
console.log( // []
  Range(30, 30, 5))
```

- Range { \_start: 0, \_end: Infinity, \_step: 1, size: Infinity }
- Range { \_start: 10, \_end: Infinity, \_step: 1, size: Infinity }
- Range { \_start: 10, \_end: 15, \_step: 1, size: 5 }
- Range { \_start: 10, \_end: 30, \_step: 5, size: 4 }
- Range { \_start: 30, \_end: 10, \_step: -5, size: 4 }
- Range { \_start: 30, \_end: 30, \_step: 5, size: 0 }

# Основные объекты и функции Immutable.JS

List

Map

OrderedMap

Set

OrderedSet

Stack

Range

Repeat

Record

Seq

Collection

ValueObject

fromJS

is

hash

isImmutable

isCollection

isKeyed

isIndexed

isAssociative

isOrdered

isValueObject

get

has

remove

set

update

getIn

hasIn

removeIn

setIn

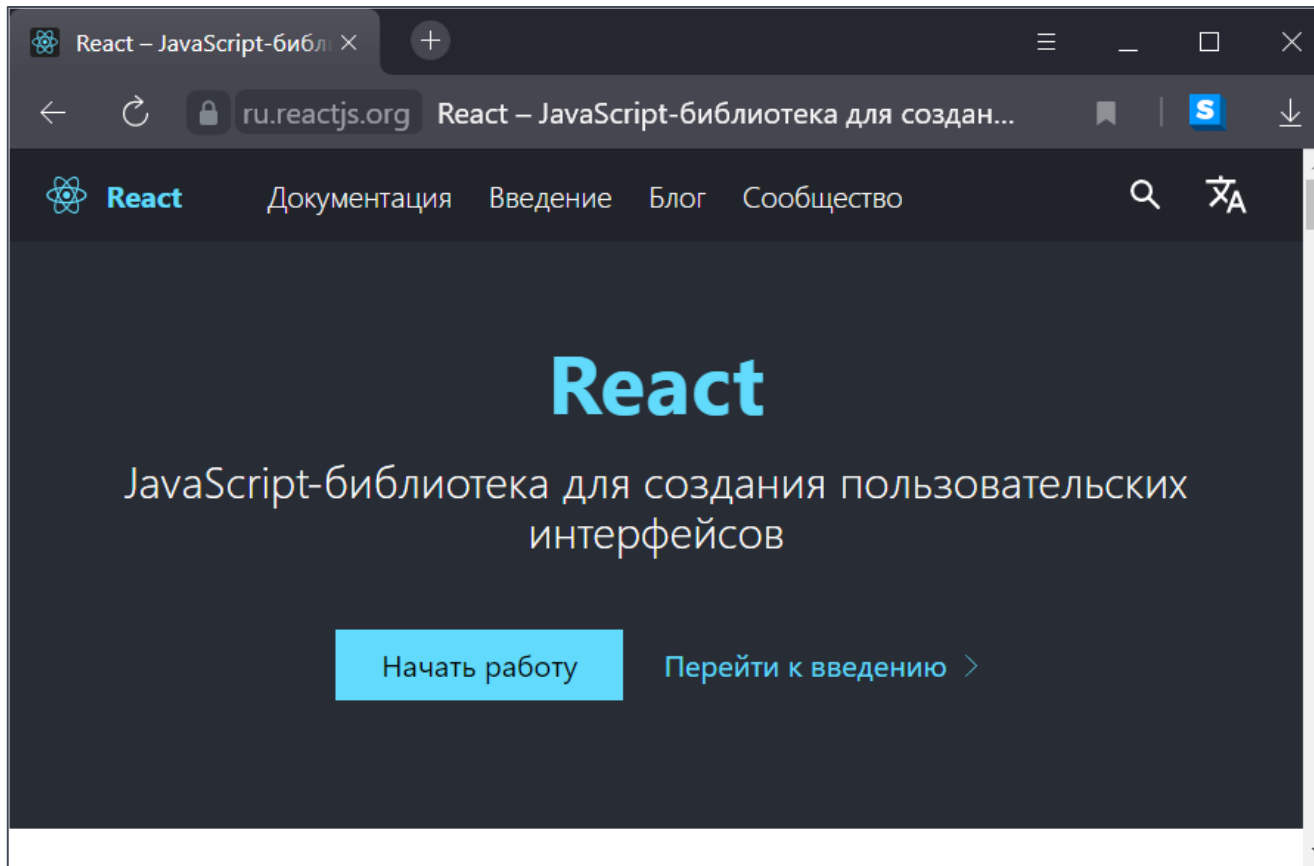
updateIn

merge

mergeWith

mergeDeep

mergeDeepWith



<https://ru.reactjs.org/>

**my-app**

Установятся **react**, **react-dom**, **react-scripts**

### Вариант 1

- **npm** init react-app my-app

### Вариант 2

- **npm** create-react-app my-app
- **cd** my-app
- **npm** start



# Среда разработки React после применения Create React App

- Поддержка синтаксиса **JSX**, **ES6**, **TypeScript**, **Flow**
- Поддержка **JavaScript** стандарта не ниже **ES6**
- Обработка **CSS**
- Настроенные **модульные тесты** с поддержкой отчётов о покрытии
- «**Живой**» сервер разработки с предупреждениями о типовых ошибках
- Скрипт сборки в «**bundle**» **JS**, **CSS**, рисунков для «**production**» с **hash** и **sourcemaps**

**create-react-app** – для одностраничных приложений, статических сайтов, разработки компонент и изучения.

Поддержка динамических сайтов с использованием, например:

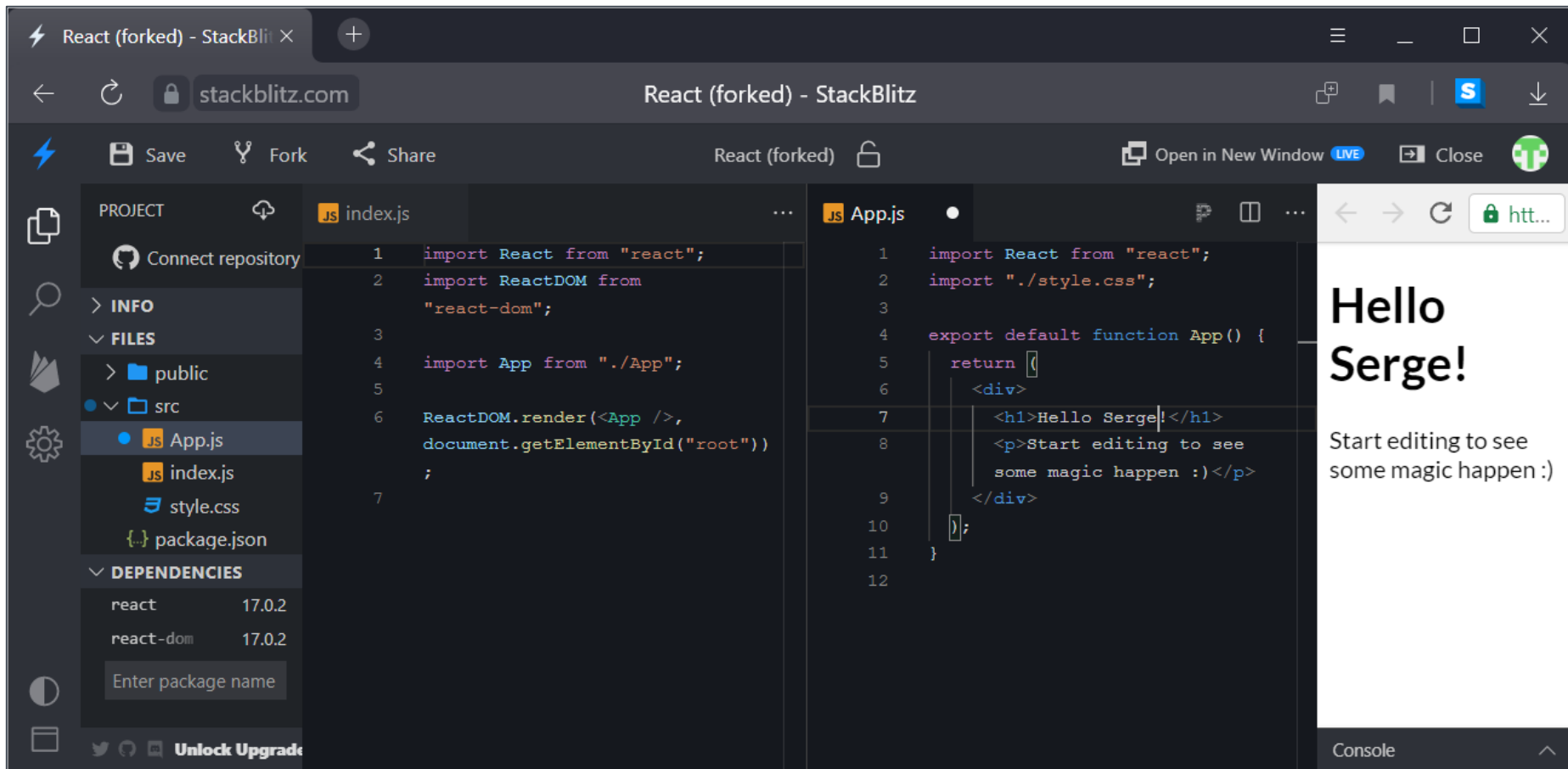
- **nwb** – <https://github.com/insin/nwb>
- **Neutrino** – <https://neutrinojs.org/>

Статические сайты с пререндерингом – **Gatsby** – <https://www.gatsbyjs.com/>

<https://create-react-app.dev/>

# Online-редакторы React (1)

10



<https://stackblitz.com/>

# Online-редакторы React (2)

11

The screenshot displays the CodeSandbox web interface. The browser tab is titled "React - CodeSandbox" and the address bar shows "codesandbox.io". The main header includes the user "bserge" with a "Free" badge, the project name "React", and a "Template" dropdown. Action buttons for "Share", "Create Sandbox", and a notification bell are also present.

The left sidebar contains a "Template Info" section for a "React" starter project, showing statistics (1.8k likes, 64.4M views, 2.4M forks) and a "Bookmark Template" button. Below this is a user profile for "Compulves" and a "Files" section showing a "public" folder.

The central editor area has two tabs: "App.js" and "index.js". The "App.js" tab is active, showing the following code:

```
1 import "../styles.css";
2
3 export default function App() {
4   return (
5     <div className="App">
6       <h1>Hello Serge</h1>
7       <p>Start editing to see some magic happen!</p>
8     </div>
9   );
10 }
```

The "index.js" tab shows the main application logic:

```
1 import { StrictMode } from "react";
2 import ReactDOM from "react-dom";
3
4 import App from "./App";
5
6 const rootElement = document.getElementById('root');
7 ReactDOM.render(
8   <StrictMode>
9     <App />
10   </StrictMode>,
11   rootElement
12 );
```

The right sidebar features a "Browser" window showing the rendered application. It displays the text "Hello Serge" in a large font, followed by "Start editing to see some magic happen!". Below the browser is a "Console" window with a red badge indicating 2 messages.

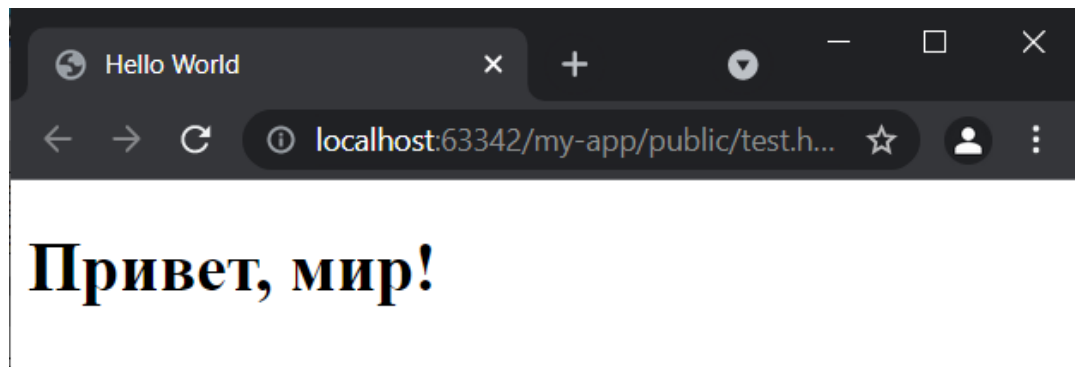
The bottom status bar shows the file path "bb2167d81", the cursor position "Ln 1, Col 1", and the file encoding "UTF-8" and "JavaScript".

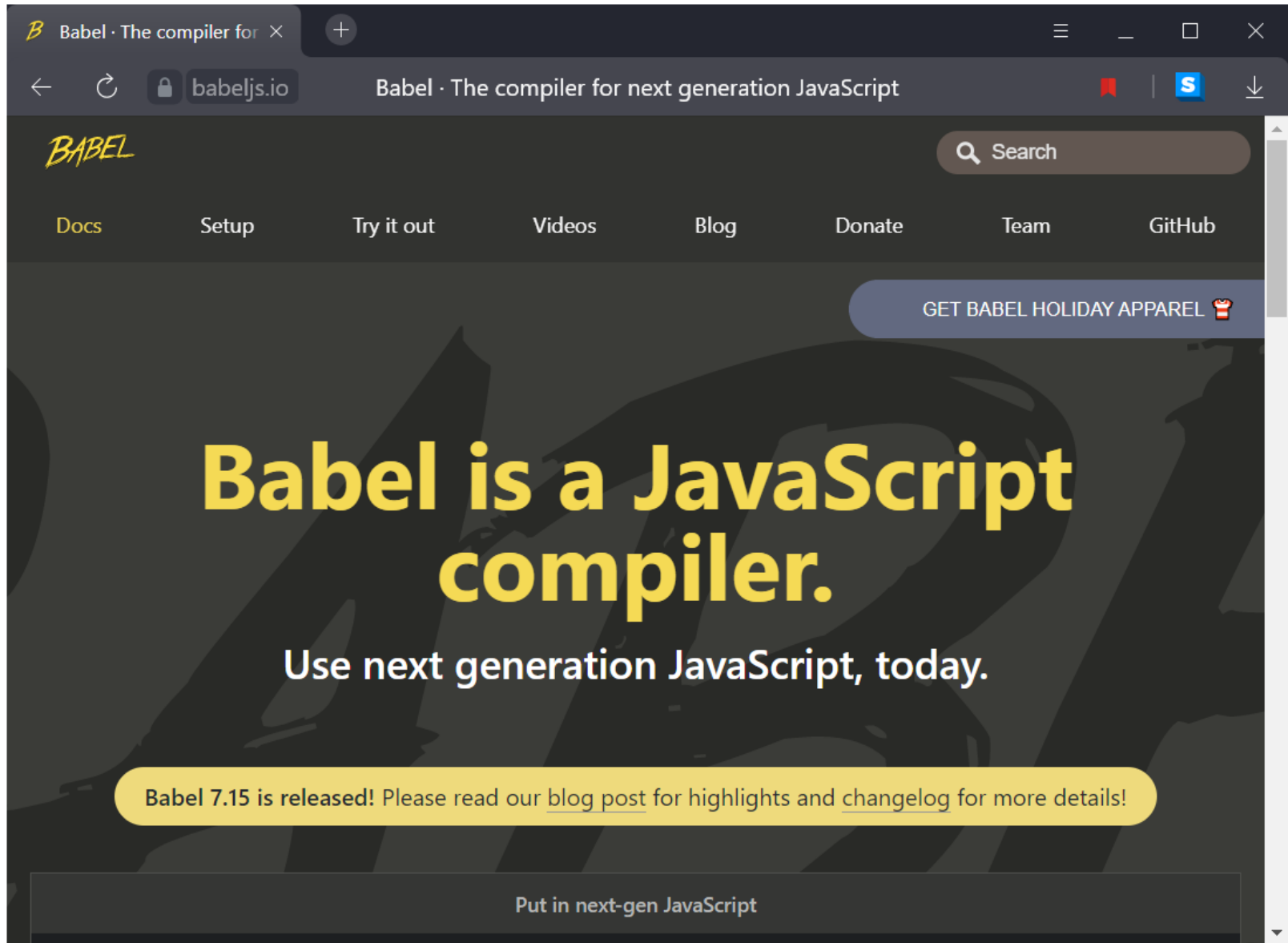
<https://codesandbox.io/>

# Hello World (не для production!)

12

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8" />
  <title>Hello World</title>
  <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
  <!-- Don't use this in production: -->
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
<div id="root"></div>
<script type="text/babel">
  ReactDOM.render(
    <h1>Привет, мир!</h1>,
    document.getElementById('root')
  );
</script>
</body>
</html>
```





<https://babeljs.io/>

# Hello World

14

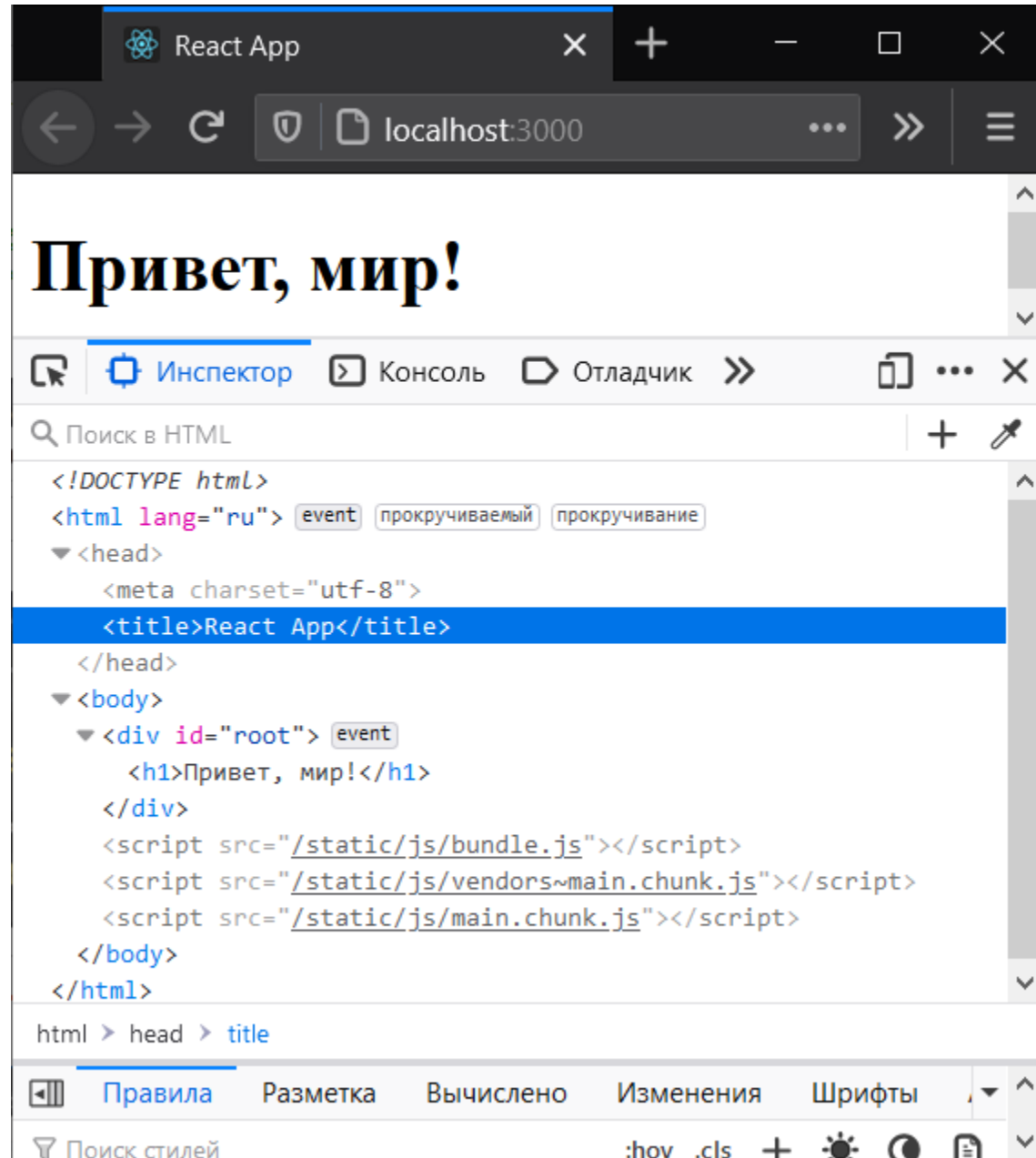
## index.html

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8" />
    <title>React App</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

## index.js

```
import ReactDOM from 'react-dom';
```

```
ReactDOM.render(
  <h1>Привет, мир!</h1>,
  document.getElementById('root')
);
```



# Отладка: React Developer Tools

15


The image shows a Chrome browser window with the React Developer Tools extension page. The address bar shows 'chrome.google.com' and the page title is 'React Developer Tools - Интернет-магазин Chrome'. The page content includes the React logo, the title 'React Developer Tools', the author 'Автор: Facebook', and a rating of 4.5 stars from 1,342 reviews. A blue 'Установить' (Install) button is visible. Below the main content, there are tabs for 'Обзор' (Overview), 'Меры по обеспечению конфиденциальности' (Privacy measures), 'Отзывы' (Reviews), 'Поддержка' (Support), and 'Похожие' (Similar). A preview of the React Developer Tools interface is shown at the bottom, displaying a 'todos' application with a 'Try React DevTools' overlay. The interface includes tabs for Elements, Sources, Console, Components, Profiler, Performance, and Network. The Components tab is active, showing a tree view with 'App', 'Header', 'MainSection', and 'TodoList'. The 'TodoList' component is selected, and its props are displayed on the right: 'newTodo: [checked]', 'onSave: onSave()', and 'placeholder: What needs to be done?'.

React Developer Tools - Интернет-магазин Chrome

интернет-магазин chrome

@gmail.com

Разные > Расширения > React Developer Tools

 **React Developer Tools**

Автор: Facebook

★★★★★ 1 342 | [Инструменты разработчика](#) | Пользователей: 3 000 000+

Установить

Обзор Меры по обеспечению конфиденциальности Отзывы Поддержка Похожие

todos

Try React DevTools

TodoTextInput 55Age 45Age

No items left Add Active Completed Clear completed

Elements Sources Console Components Profiler Performance Network

Search (text or /regex/)

App

- Header
- TodoTextInput**
- MainSection
- TodoList

TodoTextInput

props

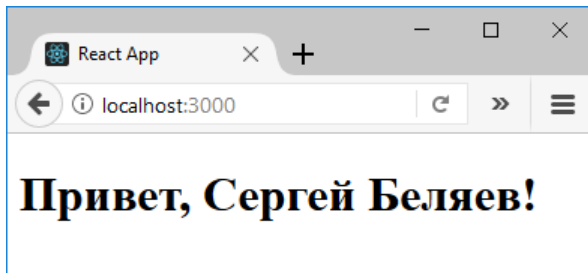
- newTodo: ☒
- onSave: onSave()
- placeholder: What needs to be done?

<https://github.com/facebook/react/tree/main/packages/react-devtools>

# JSX – Hello World / HTML, JS, {}

16

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



В фигурных скобках  
используется любое  
JS-выражение

index.js

```
import React from 'react';  
import ReactDOM from 'react-dom';  
  
function formatName(user) {  
  return user.firstName + ' ' + user.lastName;  
}  
  
const user = {  
  firstName: 'Сергей',  
  lastName: 'Беляев'  
};  
  
const element = (  
  <h1>  
    Привет, {formatName(user)}!  
  </h1>  
)  
;  
  
ReactDOM.render(  
  element,  
  document.getElementById('root')  
)
```



# JSX – варианты размещения

17

index.js

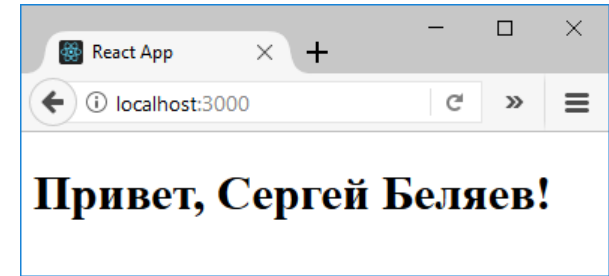
```
import React from 'react';
import ReactDOM from 'react-dom';

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

function getGreeting(user) {
  if (user) {
    return <h1>Привет, {formatName(user)}!</h1>;
  }
  return <h1>Hello, stranger.</h1>;
}

const user = {
  firstName: 'Сергей',
  lastName: 'Беляев'
};

ReactDOM.render(
  getGreeting(user),
  document.getElementById('root')
);
```



JSX-выражения могут  
использоваться в любом  
«месте» JS

# JSX представляет собой объекты / `React.createElement()`

## Определения идентичны

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

const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);

// Note: this structure is simplified
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, world'
  }
};
```

JSX предотвращает атаки,  
основанные на инъекции  
кода

**React DOM** обновляет **DOM**,  
чтобы он соответствовал  
переданным **React-**  
**элементам**

## Работа с атрибутами

```
// Простой атрибут
const element1 = <div tabIndex="0"></div>;
// Атрибут - JS
const element2 = <img src={userData}></img>;
```

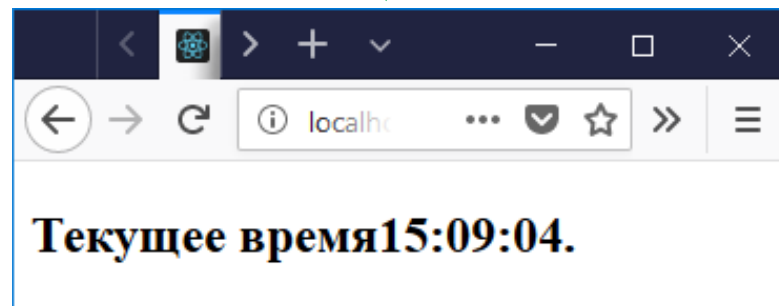
**Элементы React иммутабельны!**

# JSX – обновление DOM / ReactDOM.render()

## Обновление данных на странице

```
import React from 'react';
import ReactDOM from 'react-dom';
function tick() {
  const element = (
    <div>
      <h2>Текущее время
        {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  ReactDOM.render(
    element,
    document.getElementById('root')
  );
}
setInterval(tick, 1000);
```

Обновление только  
изменений в DOM



**ReactDOM.render()** создаёт новый элемент!

# Компоненты в React – использование

20

## Функциональные и классовые компоненты

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

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

Объявления  
идентичны

Входные  
данные –  
«пропсы»

## Функциональные компоненты:

- на вход единственный объект
- на выходе – элемент React
- входные параметры не изменяются!

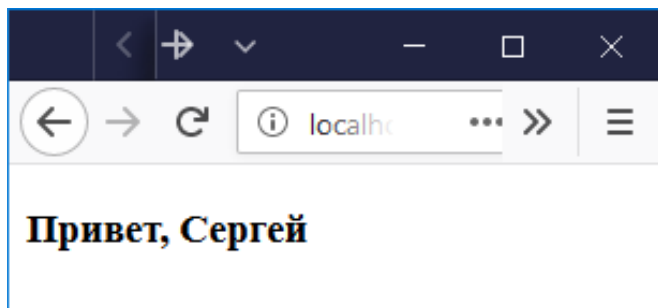
**У классов есть доп.возможности**

## Отображение компонента

```
import React from 'react';  
import ReactDOM from 'react-dom';
```

```
function Welcome(props) {  
  return <h3>Привет, {props.name}</h3>;  
}
```

```
const element = <Welcome name="Сергей" />;  
ReactDOM.render(  
  element,  
  document.getElementById('root')  
) ;
```



# Компоненты в React – композиция

21

## Композиция компонентов

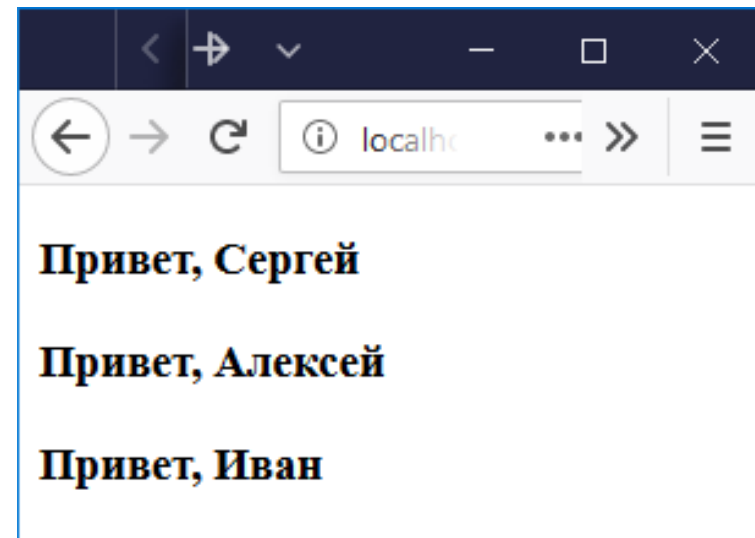
```
import React from 'react';
import ReactDOM from 'react-dom';

function Welcome(props) {
  return <h3>Привет, {props.name}</h3>;
}

function App() {
  return (
    <div>
      <Welcome name="Сергей" />
      <Welcome name="Алексей" />
      <Welcome name="Иван" />
    </div>
  );
}

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

Всегда называйте  
компоненты с заглавной  
буквы



# Компоненты в React – вложенность

22

```
import React from 'react';
import ReactDOM from 'react-dom';
function formatDate(date) {
  return date.toLocaleDateString();
}
function Avatar(props) {
  return (
    <img
      className="Avatar"
      src={props.user.avatarUrl} />
  );
}
```

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

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

```
const comment = {
  date: new Date(),
  text: 'Передаваемый текст',
  author: {
    name: 'Имя автора',
    avatarUrl: 'http://pl.com/g/64',
  },
};
```

```
ReactDOM.render(
  <Comment
    date={comment.date}
    text={comment.text}
    author={comment.author}
  />,
  document.getElementById('root')
);
```

**Пропсы можно только читать!**

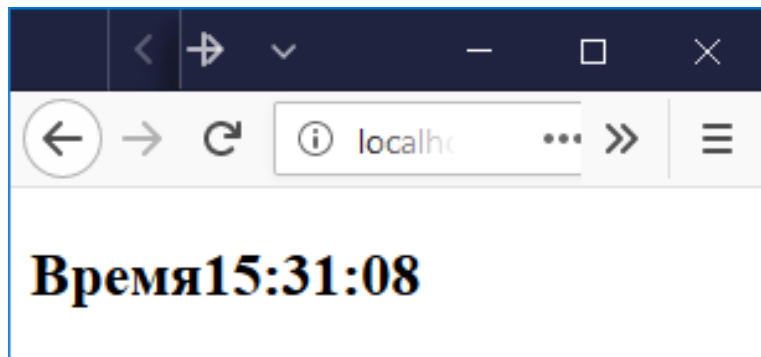
React-компоненты обязаны вести себя как чистые функции по отношению к своим пропсам

# Компонент, показывающий время

23

## Функция

```
import React from 'react';
import ReactDOM from 'react-dom';
function Clock(props) {
  return (
    <div><h2>Время
    {props.date.toLocaleTimeString()}
    </h2></div>
  );
}
function tick() {
  ReactDOM.render(
    <Clock date={new Date()} />,
    document.getElementById('root')
  );
}
setInterval(tick, 1000);
```



## Класс

```
import React from 'react';
import ReactDOM from 'react-dom';
class Clock extends React.Component {
  render() {
    return (
      <div><h2>Время
      {this.props.date.toLocaleTimeString()}
      </h2></div>
    );
  }
}
function tick() {
  ReactDOM.render(
    <Clock date={new Date()} />,
    document.getElementById('root')
  );
}
setInterval(tick, 1000);
```

Проблема в том, что компонент **Clock** не обновляет себя каждую секунду автоматически.

# Компонент с внутренним состоянием

24

## Но время остановилось...

```
import React from 'react';
import ReactDOM from 'react-dom';
class Clock extends React.Component {
  constructor(props) {
    super(props); // Отправить родителю
    this.state = {date: new Date()};
  }
  render() {
    return (
      <div>
        <h2>Время
          {this.state.date.toLocaleTimeString()}</h2>
        </div>
      );
  }
}
ReactDOM.render(
  <Clock />, // Не надо передавать параметры
  document.getElementById('root')
);
```



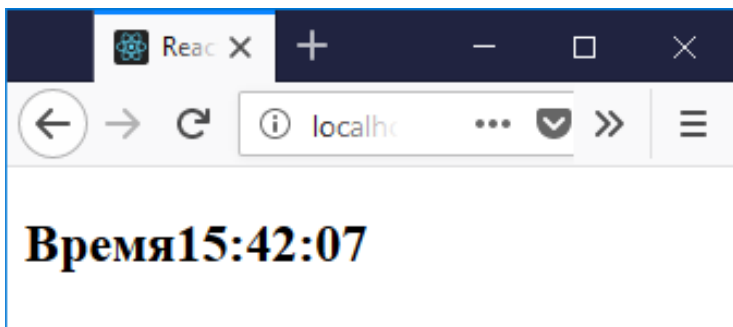
# Методы жизненного цикла / componentDidMount, componentWillUnmount

```
import React from 'react';
import ReactDOM from 'react-dom';
class Clock extends React.Component {
  constructor(props) {
    super(props); // Отправить родителю
    this.state = {date: new Date()};
  }
  componentDidMount() { // Подключение
    this.timerID = setInterval(() => this.tick(), 1000);
  }
  componentWillUnmount() { // Отключение
    clearInterval(this.timerID);
  }
  tick() { // Изменение состояния
    this.setState({
      date: new Date()
    });
  }
}
```

Изменение состояния через  
**this.setState()**

```
render() { // Отображение
  return (
    <div><h2>Время
      {this.state.date.toLocaleTimeString()}
    </h2></div>
  );
}
```

```
ReactDOM.render(
  <Clock />, // Не надо передавать параметры
  document.getElementById('root')
);
```



# Методы жизненного цикла компонента

26

## Монтирование

- constructor, **render**, componentDidMount

## Обновление

- shouldComponentUpdate, **render**, getSnapshotBeforeUpdate, componentDidUpdate

## Размонтирование

- componentWillUnmount

## Обработка ошибок

- componentDidCatch

## Другие методы API

- setState, forceUpdate

## Свойства класса

- defaultProps, displayName

## Свойства экземпляра

- props, state

<https://ru.reactjs.org/docs/react-component.html>

*// НЕЛЬЗЯ - не происходит повторный рендеринг*

```
this.state.comment = 'Hello';
```

*// ПРАВИЛЬНО*

```
this.setState({comment: 'Hello'});
```

*// НЕЛЬЗЯ*

```
this.setState({
```

*// Выполняется асинхронное обновление state & props*

```
  counter: this.state.counter + this.props.increment,
```

```
});
```

*// ПРАВИЛЬНО*

```
this.setState((prevState, props) => ({
```

```
  counter: prevState.counter + props.increment
```

```
}));
```

*// ПРАВИЛЬНО*

```
this.setState(function(prevState, props) {
```

```
  return {
```

```
    counter: prevState.counter + props.increment
```

```
  };
```

```
});
```

Изменения состояния  
объединяются

# Обновление состояния из независимых полей

28

```
import React from 'react';
class DataRow extends React.Component {
  constructor(props) {
    super(props);
    this.state = { // Составное состояние
      posts: [],
      comments: []
    };
  }
  componentDidMount() { // Отдельное обновление
    this.fetchPosts().then(response => {
      this.setState({
        posts: response.posts
      });
    });
    this.fetchComments().then(response => {
      this.setState({
        comments: response.comments
      });
    });
  }
  fetchPosts() { return new Promise(); } // "Заглушки"
  fetchComments() { return new Promise(); } // "Заглушки"
}
```

# Нисходящий (однонаправленный) поток данных

*// Компонент не знает, откуда пришла дата*

```
function FormattedDate(props) {  
  return <h3>Сейчас {props.date.toLocaleTimeString()}.</h3>;  
}
```

```
class DataRow extends React.Component {
```

```
  ...
```

```
  getFormattedDate() {
```

```
    // Передаёт дату из своего состояния
```

```
    return <FormattedDate date={this.state.date}/>
```

```
  }
```

```
  ...
```

```
}
```



# Сравнение событий в React и в HTML

30

## html

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```



## React

```
const element = <button onClick={activateLasers}>
  Activate Lasers
</button>
```

- События в **React** именуются в стиле **camelCase** вместо нижнего регистра
- С **JSX** функция передаётся как обработчик события вместо строки
- В **React** нельзя остановить распространение события, вернув **false**, нужно явно вызвать **preventDefault()**

## html

```
<a href="#"
  onclick="console.log('The link was clicked.');" return false">
  Click me
</a>
```



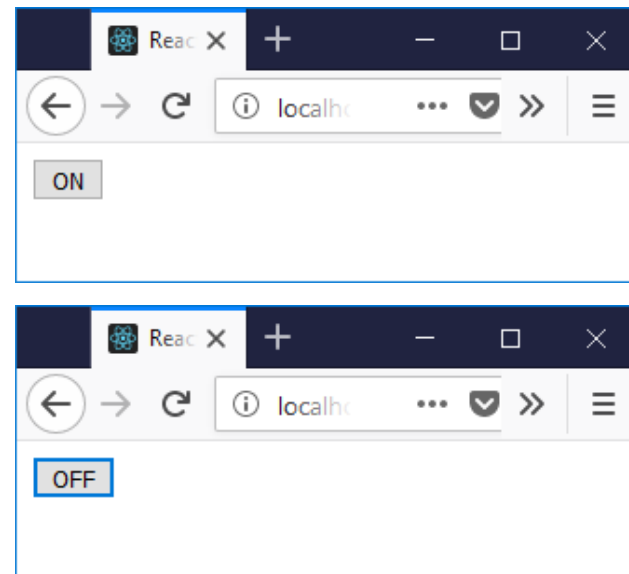
## React

```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('The link was clicked.');
```

```
  }
  return (
    <a href="#" onClick={handleClick}>
      Click me
    </a>
  );
}
```

# Обработчик события (1)

```
import React from 'react';
import ReactDOM from 'react-dom';
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(prevState => ({
      isToggleOn: !prevState.isToggleOn
    }));
  }
  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
ReactDOM.render(
  <Toggle />,
  document.getElementById('root')
)
```



Если ссылаться на метод без () после него, например, **onClick={this.handleClick}**, этот метод нужно привязать

# Обработчик события (2) / ()=>{

32

```
import React from 'react';
import ReactDOM from 'react-dom';
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};
  }
  // Такой синтаксис гарантирует, что this привязан к handleClick
  handleClick = () => {
    this.setState(prevState => ({
      isToggleOn: !prevState.isToggleOn
    }));
  }
  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```

Другие варианты – к пред.реализации без использования bind()

```
const element = <button onClick={ (e) => this.deleteRow(id, e)}>Delete Row</button>
const element = <button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>
```



# «Условный» рендеринг

33

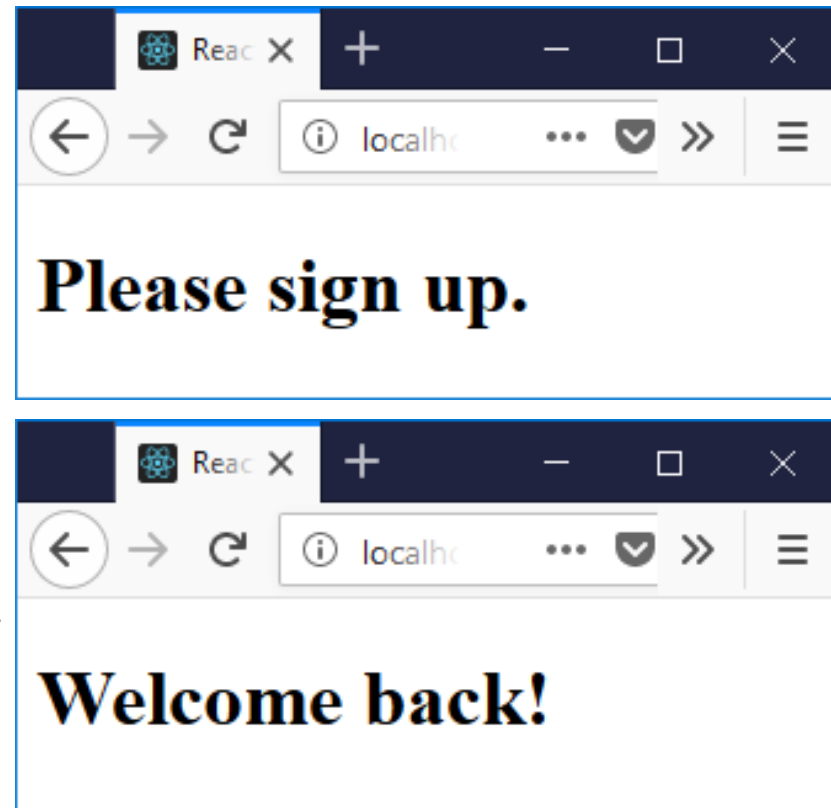
```
import React from 'react';
import ReactDOM from 'react-dom';

function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}

function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}

ReactDOM.render(
  // Попробуйте поменять на isLoggedIn={true}:
  <Greeting isLoggedIn={false} />,
  document.getElementById('root')
);
```



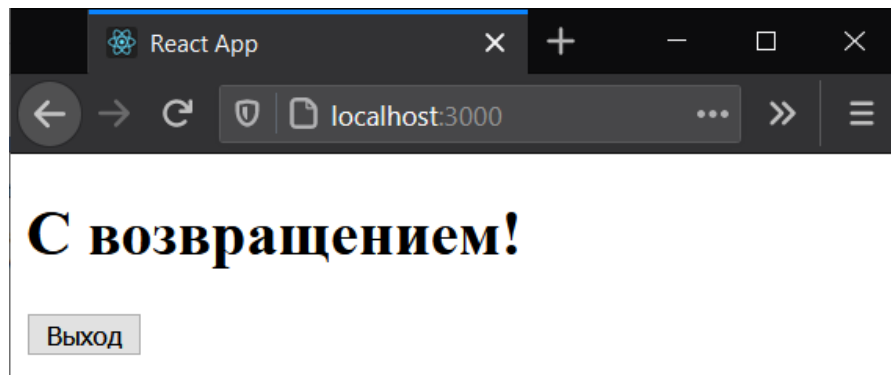
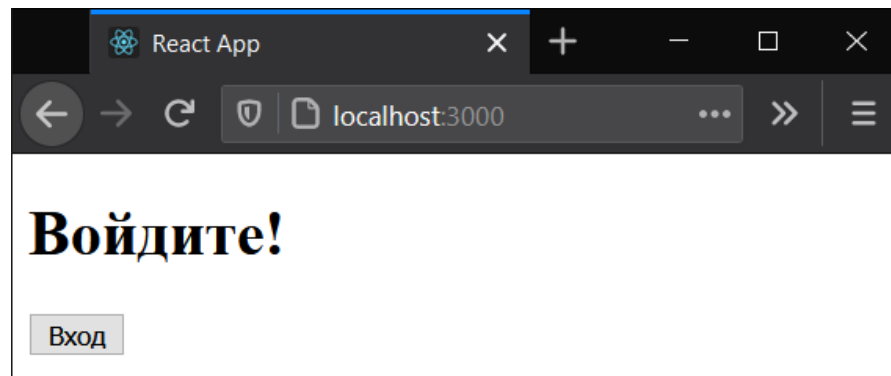
# Условный рендеринг – компоненты <sup>34</sup> (подготовка)

```
import React from 'react';
import ReactDOM from 'react-dom';
function UserGreeting() {
  return <h1>С возвращением!</h1>;
}
function GuestGreeting() {
  return <h1>Войдите!</h1>;
}
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn)
    return <UserGreeting />;
  return <GuestGreeting />;
}
function LoginButton(props) {
  return (
    <button onClick={props.onClick}>
      Вход
    </button>
  );
}
function LogoutButton(props) {
  return (
    <button onClick={props.onClick}>
      Выход
    </button>
  );
}
```

# Условный рендеринг – использование

```
class LoginControl extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isLoggedIn: false};
  }
  handleLogin = () => {
    this.setState({isLoggedIn: true});
  }
  handleLogout = () => {
    this.setState({isLoggedIn: false});
  }
  render() {
    const isLoggedIn = this.state.isLoggedIn;
    let button = null;
    if (isLoggedIn)
      button = <LogoutButton onClick={this.handleLogout} />;
    else
      button = <LoginButton onClick={this.handleLogin} />;
    return (
      <div>
        <Greeting isLoggedIn={isLoggedIn} />
        {button}
      </div>
    );
  }
}

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



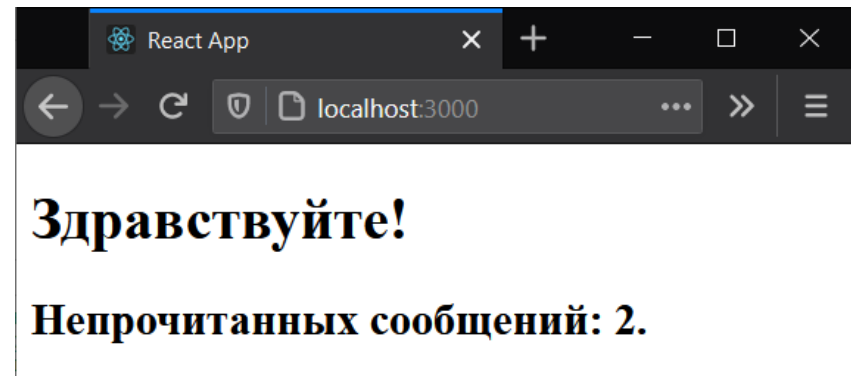
# Встроенные условия if (1)

36

```
import React from 'react';
import ReactDOM from 'react-dom';
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
```

- **true && expression** равно **expression**
- **false && expression** равно **false**  
false проигнорирован React

```
  return (
    <div>
      <h1>Здравствуйте!</h1>
      {unreadMessages.length > 0 &&
        <h2>
          Непрочитанных сообщений: {unreadMessages.length}.
        </h2>
      }
    </div>
  );
}
const messages = ['Re: React', 'Fwd:Re: React'];
ReactDOM.render(
  <Mailbox unreadMessages={messages} />,
  document.getElementById('root')
);
```

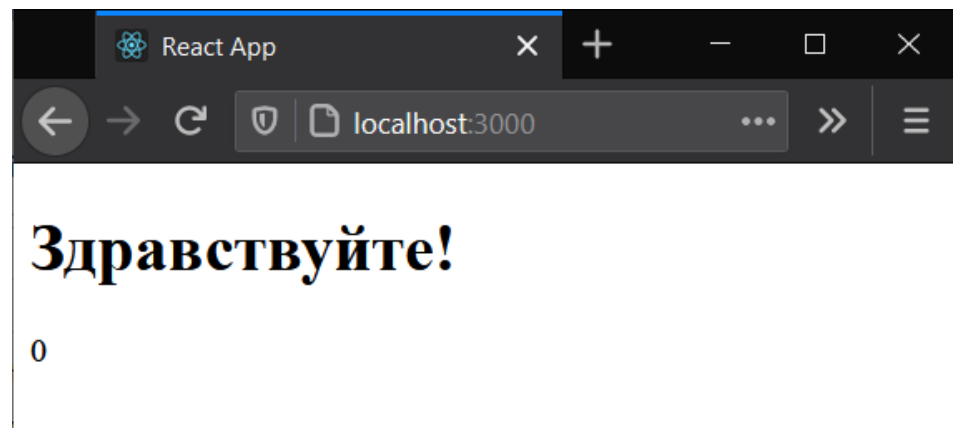
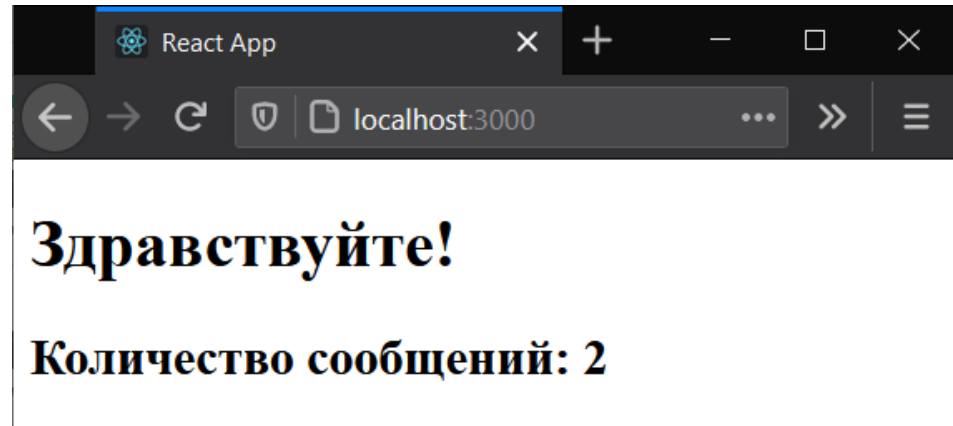


# Встроенные условия if (2)

37

```
import React from 'react';
import ReactDOM from 'react-dom';
function Mailbox(props) {
  const count = props.count;
  return (
    <div>
      <h1>Здравствуйте!</h1>
      { count && <h2>Количество сообщений: {count}</h2>}
    </div>
  );
}
const messages = ['Re: React', 'Fwd:Re: React'];
ReactDOM.render(
  <Mailbox count={messages.length} />,
  document.getElementById('root')
);
```

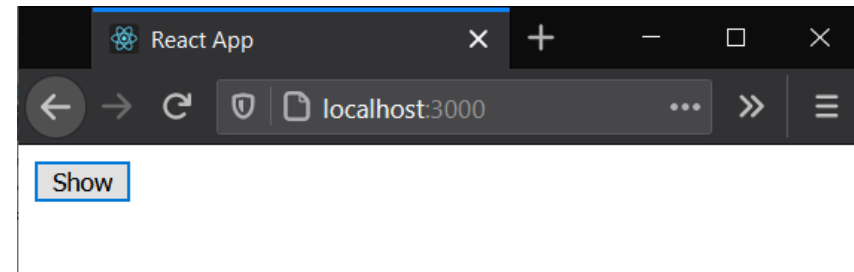
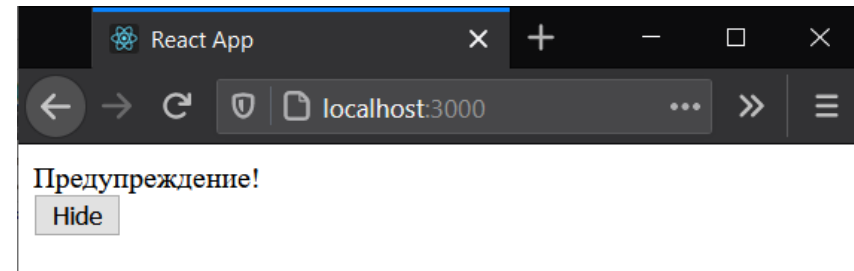
Можно использовать  
тернарный оператор: ?



# Предотвращение рендеринга – null

38

```
import React from 'react';
import ReactDOM from 'react-dom';
function WarningBanner(props) {
  if (!props.warn)
    return null; // Скрытие компонента
  return (<div className="warning">Предупреждение!</div>);
}
class Page extends React.Component {
  constructor(props) {
    super(props);
    this.state = {warning: true}
  }
  handleToggle = () => {
    this.setState(prevState => ({warning: !prevState.warning}));
  }
  render() {
    return (
      <div>
        <WarningBanner warn={this.state.warning} />
        <button onClick={this.handleToggle}>
          {this.state.warning ? 'Hide' : 'Show'}
        </button>
      </div>
    );
  }
}
ReactDOM.render(<Page />, document.getElementById('root'));
```



# Списки – простое использование key<sup>39</sup>

```
import React from 'react';
import ReactDOM from 'react-dom';
```

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li>{number}</li>
  );
  return (
    <ul>{listItems}</ul>
  );
}
```

```
const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

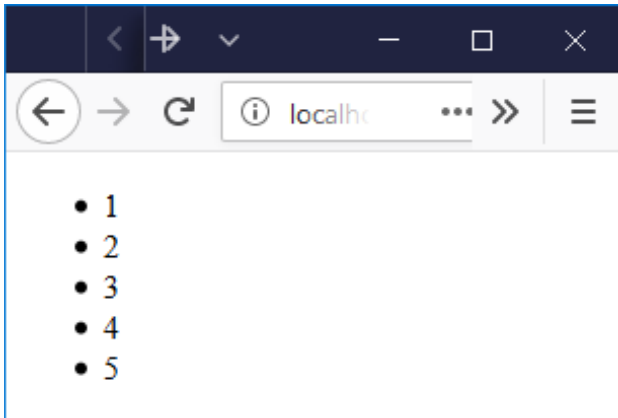


Ключи помогают React определять, какие элементы были изменены, добавлены или удалены

```
import React from 'react';
import ReactDOM from 'react-dom';

function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}
```

```
const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```



# Использование компонентов с ключами

```
import React from 'react';
import ReactDOM from 'react-dom';
```

```
function ListItem(props) {
  // Ключ здесь определять не надо
  return <li>{props.value}</li>;
}
```

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    // Ключ должен определяться внутри массива
    <ListItem key={number.toString()}
      value={number} />
  );
  return <ul>{listItems}</ul>;
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

Ключи должны быть  
уникальны в рамках  
одного списка

Ключи не передаются  
компоненту  
Он НЕ получит **props.key**!

## Альтернатива

```
function NumberList(props) {
  const numbers = props.numbers;
  return (
    <ul>
      {numbers.map((number) =>
        <ListItem key={number.toString()}
          value={number} />
      )}
    </ul>
  );
}
```



# Управляемые компоненты – input

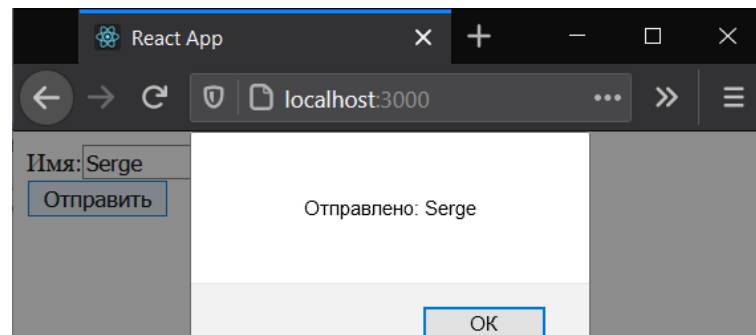
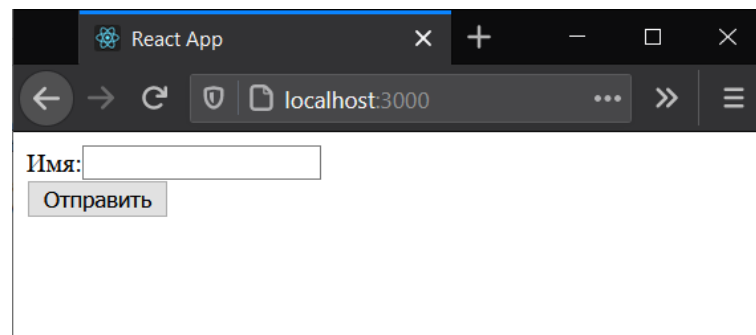
```
import React from 'react';
import ReactDOM from 'react-dom';
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {input: ""};
  }
  handleChange = (event) => {
    this.setState({input: event.target.value});
  }
  handleSubmit = (event) => {
    alert(`Отправлено: ${this.state.input}`);
    event.preventDefault();
  }
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>Имя:
          <input type="text"
            value={this.state.input}
            onChange={this.handleChange} />
        </label><br/>
        <input type="submit" value="Отправить" />
      </form>
    );
  }
}
ReactDOM.render(
  <NameForm />,
  document.getElementById('root')
);
```

Сохранение «ВВОДА»

«Отправка»

## В html

```
<form>
  <label>
    Name:
    <input type="text" name="name" />
  </label>
  <input type="submit" value="Submit" />
</form>
```



# Обновление state / список, по частям, порядок элементов

```
import React from 'react';
import ReactDOM from 'react-dom';
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {input: '', list: []};
  }
  handleChange = (event) => {
    this.setState({input: event.target.value});
  }
  handleSubmit = (event) => {
    this.setState((prevState, props) => ({
      list: [this.state.input, ...prevState.list]
    }));
    event.preventDefault();
  }
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>Имя:
          <input type="text"
            value={this.state.input}
            onChange={this.handleChange} />
        </label>
        <p>{this.state.list.join(",")}</p>
        <input type="submit" value="Отправить" />
      </form>
    );
  }
}
ReactDOM.render(<NameForm />, document.getElementById('root'));
```

Сохранение  
«Ввода»

Добавление  
к списку

Показ  
списка

React App localhost:3000

Имя:

ivan

React App localhost:3000

Имя:

serge,ivan

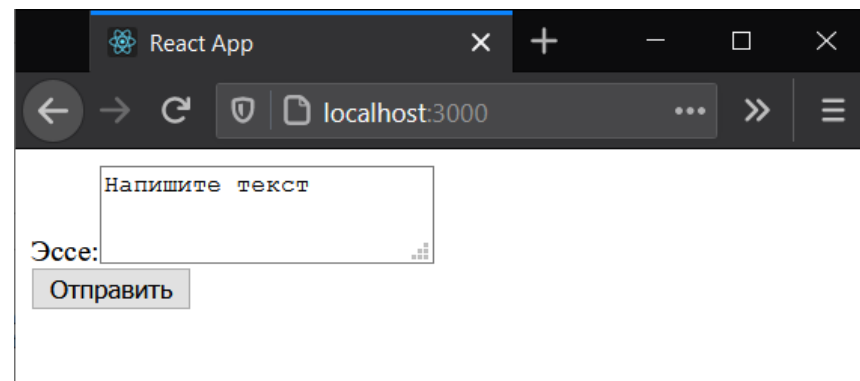
React App localhost:3000

Имя:

kirill,serge,ivan

# Управляемые компоненты – textarea

```
import React from 'react';
import ReactDOM from 'react-dom';
class EssayForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = { essay: 'Напишите текст' };
  }
  handleChange = (event) => {
    this.setState({ essay: event.target.value });
  }
  handleSubmit = (event) => {
    alert('Эссе отправлено: ' + this.state.essay);
    event.preventDefault();
  }
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>Эссе:
          <textarea value={this.state.essay}
            onChange={this.handleChange} />
        </label><br/>
        <input type="submit" value="Отправить" />
      </form>
    );
  }
}
ReactDOM.render(
  <EssayForm />,
  document.getElementById('root')
);
```



Идентично `<input>`:

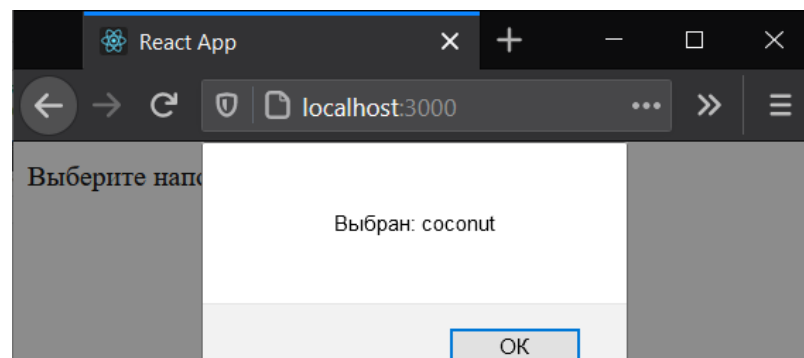
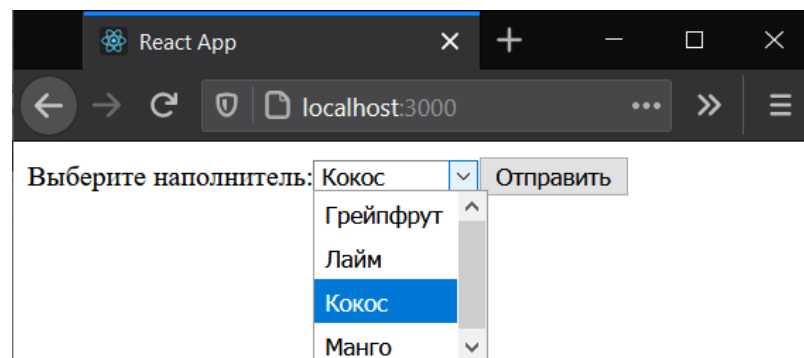
- value
- onChange
- handleChange()
- handleSubmit()

# Управляемые компоненты – select

```
import React from 'react';
import ReactDOM from 'react-dom';
class FlavorForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: 'coconut'};
  }
  handleChange = (event) => {
    this.setState({value: event.target.value});
  }
  handleSubmit = (event) => {
    alert('Выбран: ' + this.state.value);
    event.preventDefault();
  }
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>Выберите наполнитель:</label>
        <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>
        <input type="submit" value="Отправить" />
      </form>
    );
  }
}
ReactDOM.render(<FlavorForm />, document.getElementById('root'));
```

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

`<select multiple={true} value={['Б', 'В']>`



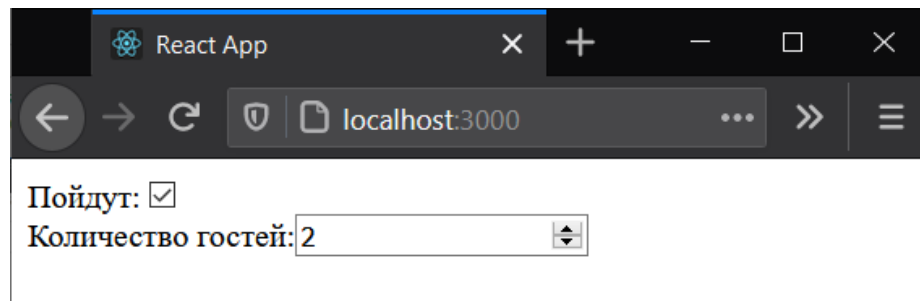
# Работа с несколькими input

```
import React from 'react';
import ReactDOM from 'react-dom';
class Reservation extends React.Component {
  constructor(props) {
    super(props);
    this.state = { isGoing: true, guests: 2 };
  }
  handleInput = (event) => {
    const tgt = event.target;
    const value = tgt.type === 'checkbox' ? tgt.checked : tgt.value;
    const name = tgt.name; // Обработка по имени
    this.setState({ [name]: value });
  }
  render() {
    return (
      <form>
        <label>Пойдут:
          <input name="isGoing" type="checkbox"
            checked={this.state.isGoing}
            onChange={this.handleInput} />
        </label> <br/>
        <label>Количество гостей:
          <input name="guests" type="number"
            value={this.state.guests}
            onChange={this.handleInput} />
        </label>
      </form>
    );
  }
}
```

Сохранить  
с учётом  
имени

Нужно учитывать  
тип, т.к. разные  
input по-разному  
хранят значения

- isGoing – checkbox – checked
- guests – number – value
- this.setState({[name]:value})



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

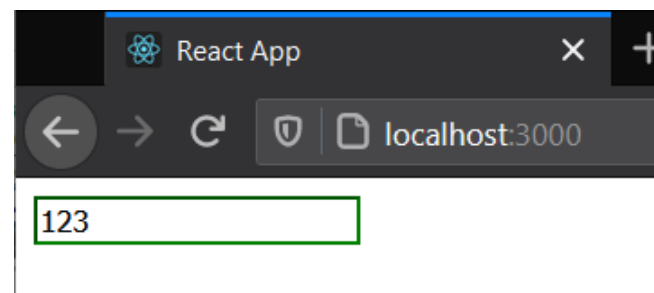
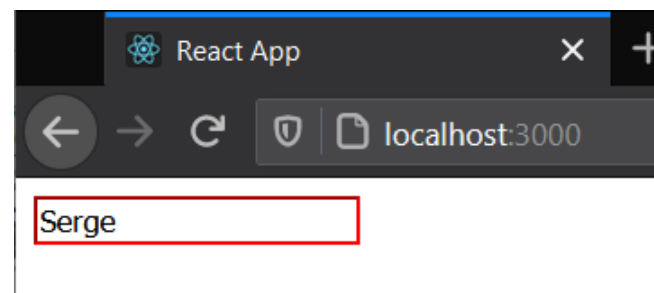
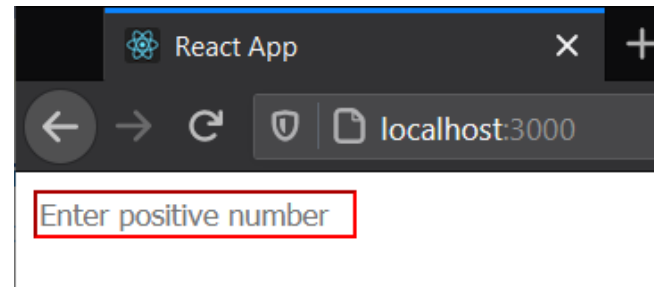
# Валидация ввода / input, style

46

```
import React from 'react';
import ReactDOM from 'react-dom';

class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ""}
  }
  onChange = (e) => {
    this.setState({value: e.target.value})
  }
  render() {
    let value = this.state.value
    let mystyle = parseInt(value) > 0 ? "green" : "red"
    return (
      <input placeholder="Enter positive number"
        value={value}
        onChange={this.onChange}
        style={{borderColor:mystyle}}/>
    )
  }
}

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



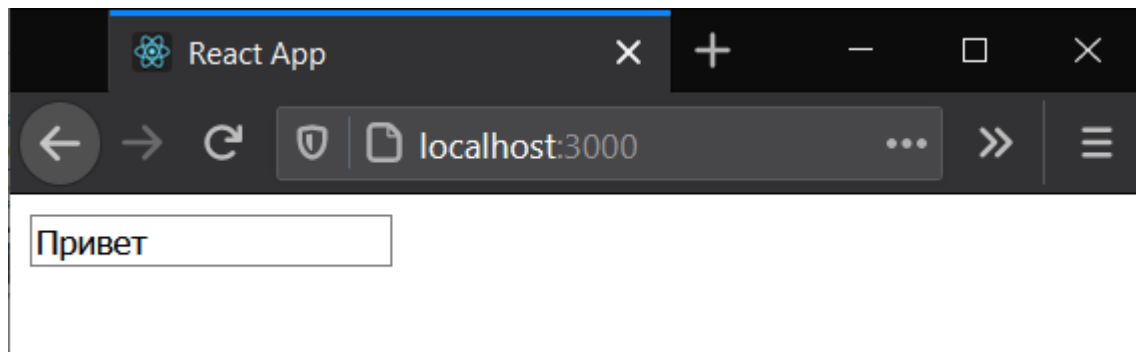
Использование  
camelCase в style

# Запрет изменения значения input / null, undefined

```
import React from 'react';
import ReactDOM from 'react-dom';
// Нельзя редактировать установленный value
ReactDOM.render(<input value="Привет" />,
  document.getElementById('root'));

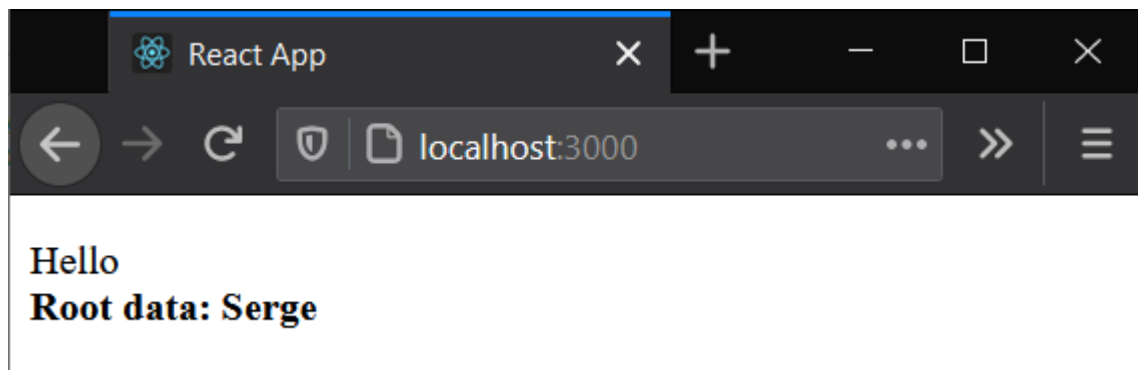
// Можно редактировать value= null || undefined
setTimeout(function() {
  ReactDOM.render(<input value={null} />,
    document.getElementById('root'));
}, 3000);
```

Запрет  
редактирования через  
3 секунды



# Передача данных потомку / props.children

```
import React from 'react';
import ReactDOM from 'react-dom';
function RootElem(props) {
  // Передача данных потомку
  return (
    <ChildElem>
      <b>Root data: {props.value}</b>
    </ChildElem>
  );
}
function ChildElem(props) {
  // Данные от родителя
  return (
    <p>
      Hello<br/>
      {props.children}
    </p>
  );
}
ReactDOM.render(<RootElem value="Serge"/>,
  document.getElementById('root'));
```





# Настройка стилей / импорт .css, .css.js

## index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { ChildElem } from './child';
import './parent.css'

function RootElem(props) {
  return (
    <div>
      <p>Parent paragraph</p>
      <ChildElem>
        <b>Root data: {props.value}</b>
      </ChildElem>
    </div>
  );
}

ReactDOM.render(<RootElem value="Serge"/>,
  document.getElementById('root'));
```

## parent.css

```
p {
  color: red;
}
```

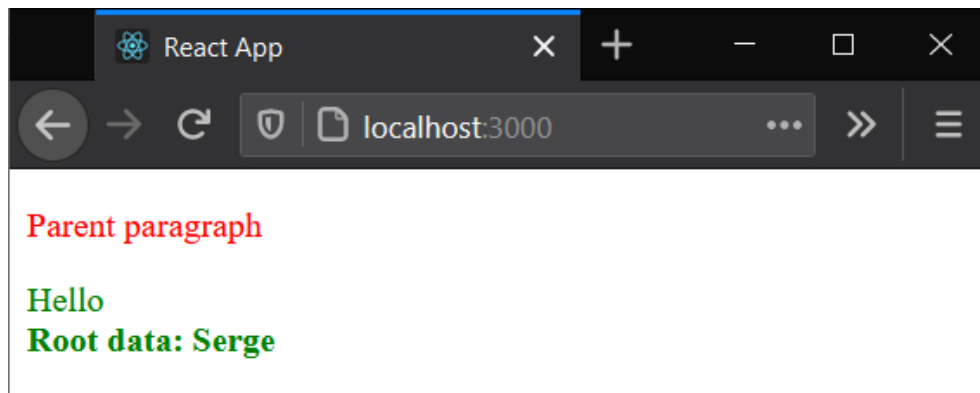
## child.js

```
import mystyles from './child.css.js'
export function ChildElem(props) {
  // Данные от родителя
  return (
    <p style={mystyles.data}>
      Hello<br/>
      {props.children}
    </p>
  );
}
```

## child.css.js

```
const data = {
  color: "green"
}
export default {data}
```

.js



# Передача данных потомку / КОМПОНЕНТЫ

```
import React from 'react';
import ReactDOM from 'react-dom';
function RootElem(props) {
```

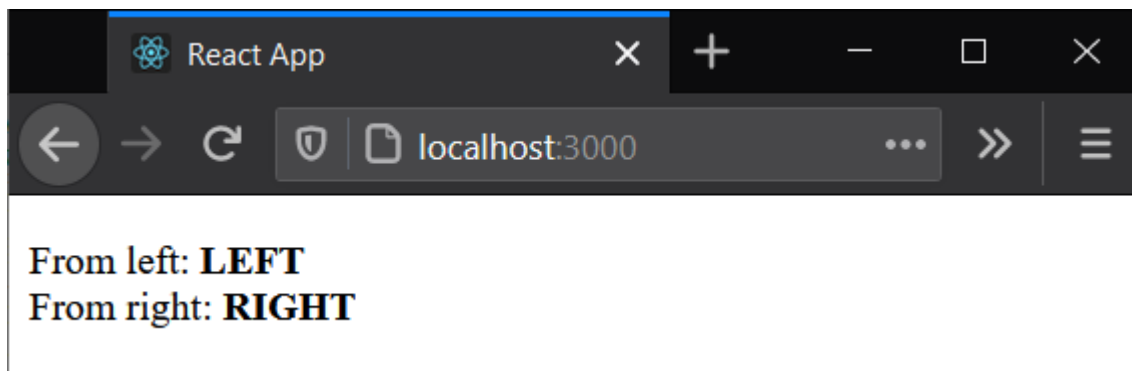
*// Передача данных потомку*

```
  return (
    <ChildElem
      left={<b>LEFT</b>}
      right={<b>RIGHT</b>}
    />
  );
}
```

Могут быть  
любые React-  
компоненты

```
function ChildElem(props) {
  // Данные от родителя
  return (
    <p>
      From left: {props.left}<br/>
      From right: {props.right}
    </p>
  );
}
```

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



- left
- right

# Неуправляемые компоненты / createRef, ref

```
import React from 'react';
import ReactDOM from 'react-dom';

class NameForm extends React.Component {
  constructor(props) {
    super(props);
    // Ссылка на неуправляемый компонент
    this.input = React.createRef();
  }
  handleSubmit = (event) => {
    alert('Отправлено: ' + this.input.current.value);
    event.preventDefault();
  }
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>Имя:
          <input type="text" ref={this.input} />
        </label>
        <input type="submit" value="Отправить" />
      </form>
    );
  }
}

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

Рекомендуется  
использовать  
управляемые  
компоненты

Загрузка файла:

```
<input type="file" ref={this.fileInput} />
```

Это неуправляемый компонент –  
доступен только на чтение:

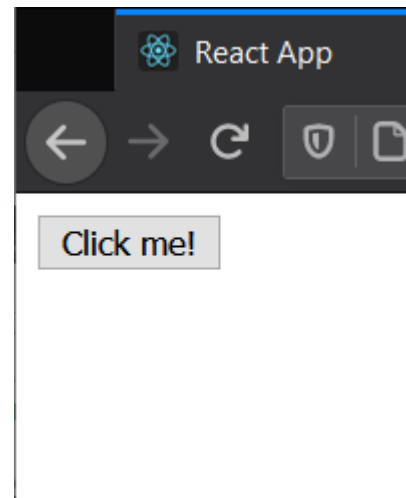
```
// constructor()
this.fileInput = React.createRef();
// handle...()
this.fileInput.current.files[0].name
```

# ref-указатель на элемент DOM

```
import React from 'react';
import ReactDOM from 'react-dom';
const FancyButton = React.forwardRef((props, ref) => (
  <button ref={ref} className="FancyButton">
    {props.children}
  </button>
));
```

*// Теперь ref будет указывать непосредственно на DOM-узел button:*

```
const ref = React.createRef();
ReactDOM.render(<FancyButton ref={ref}>Click me!</FancyButton>,
  document.getElementById('root'));
```



1. Создаём «**ref**», вызвав **React.createRef** и записываем его в переменную **ref**
2. Передаём переменную **ref** в **<FancyButton ref={ref}>**, указывая её в **JSX**-атрибуте
3. React передаёт **ref** в функцию **(props, ref) => ...** внутри **forwardRef** в качестве второго аргумента
4. Передаём аргумент **ref** дальше в **<button ref={ref}>**, указывая его в **JSX**-атрибуте.
5. После привязки рефа **ref.current** будет указывать на **DOM**-узел **<button>**

# Подъём состояния – подготовка

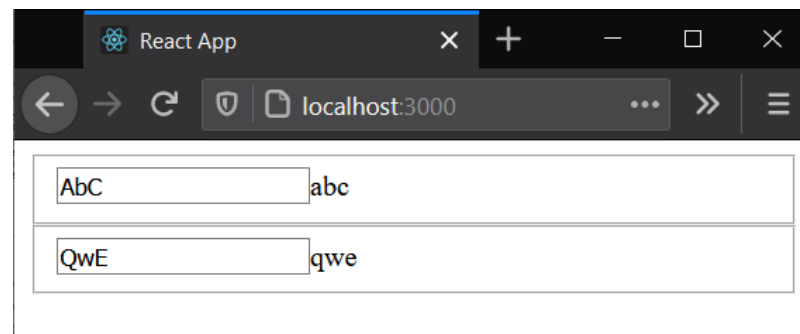
```
import React from 'react';
import ReactDOM from 'react-dom';
class RootElem extends React.Component {
  render() { // Два компонента
    return (
      <div><ChildElem/><ChildElem/></div>
    );
  }
}
class ChildElem extends React.Component {
  constructor(props) {
    super(props); // Внутреннее состояние
    this.state = { myvalue: "" }
  }
  handleChange = (e) => { // Обновление
    this.setState({ myvalue: e.target.value })
  }
  render() { // Отображение
    const myvalue = this.state.myvalue
    return (
      <fieldset>
        <input value={myvalue} onChange={this.handleChange}/>
        <Lowercase value={myvalue}/>
      </fieldset>
    )
  }
}
function Lowercase(props) { // Отображение
  return props.value ? props.value.toLowerCase() : ""
}
ReactDOM.render(<RootElem />, document.getElementById('root'));
```

У каждого компонента

- своё состояние
- отдельный вывод

Результат: независимое изменение

Нет единого «источника истины»



# Подъём состояния – реализация

```

class RootElem extends React.Component {
  constructor(props) {
    super(props);
    this.state = {myvalue: ""} // Общее состояние
  }
  handleChange = (value) => { // Обработчик изменений
    this.setState({myvalue: value})
  }
  render() {
    const value = this.state.myvalue
    // Передать значение и получить событие с изменениями
    return (
      <div>
        <ChildElem myvalue={value} onMyChange={this.handleChange}/>
        <ChildElem myvalue={value} onMyChange={this.handleChange}/>
        <Lowercase value={value}/>
      </div>
    );
  }
}

class ChildElem extends React.Component {
  handleChange = (e) => { // Обновление
    this.props.onMyChange(e.target.value)
  }
  render() { // Отображение
    const myvalue = this.props.myvalue
    return (
      <fieldset>
        <input value={myvalue} onChange={this.handleChange}/>
      </fieldset>
    )
  }
}

function Lowercase(props) { // Отображение
  return props.value ? props.value.toLowerCase() : ""
}

```

Не приведены (см.пред.слайд):

- импорт
- render

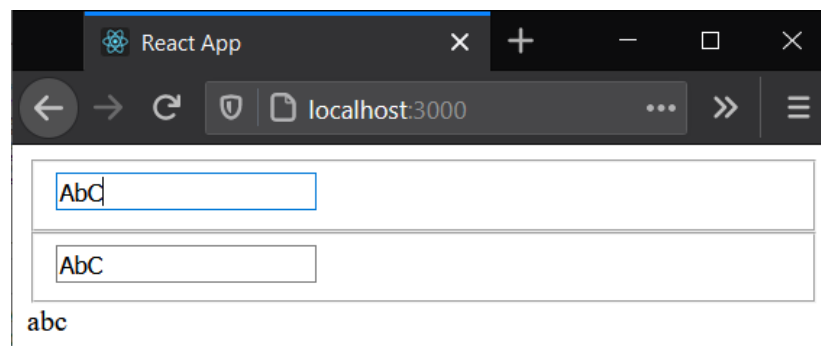
Вниз: **myvalue**

Вверх: **onMyChange**

У потомка

- нет своего **state**
- обращение к **props** на чтение
- вызов **onMyChange** при изменениях

Результат: одновременные изменения



# Специализация компонентов

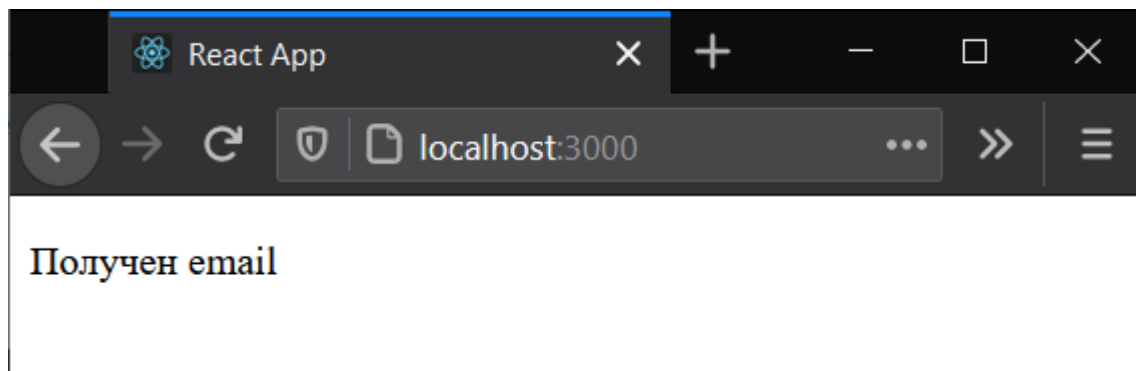
55

```
import React from 'react';
import ReactDOM from 'react-dom';
// Общий компонент
function Received(props) {
  return <p>Получен {props.type}</p>
}
// Специализированный компонент
function ReceivedEmail(props) {
  // Задание значений свойства
  return <Received type="email"></Received>
}
ReactDOM.render(<ReceivedEmail />,
  document.getElementById('root'));
```

**Facebook НЕ рекомендует использовать наследование!**

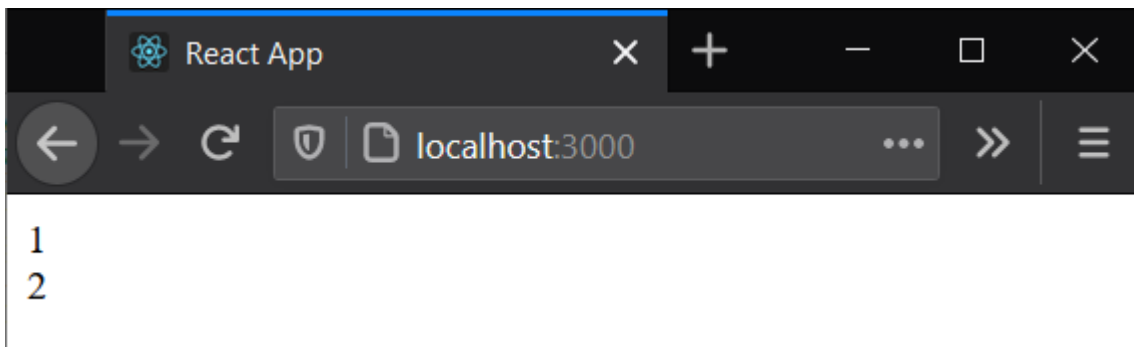
Компоненты могут принимать произвольные пропсы, включая

- примитивные значения,
- React-элементы или
- функции



# Использование фрагментов / React.Fragment

```
import React from 'react';
import ReactDOM from 'react-dom';
function RootElem() {
  return (
    <React.Fragment>
      <ChildElem value="1"></ChildElem>
      <ChildElem value="2"></ChildElem>
    </React.Fragment>
  );
}
function ChildElem(props) {
  return <div>{props.value}</div>
}
ReactDOM.render(<RootElem />,
  document.getElementById('root'));
```





0. Создаём макет интерфейса пользователя
1. Разбиваем интерфейс на иерархию компонентов
  - каждый компонент должен заниматься какой-то одной задачей
2. Строим статическую версию интерфейса на React
  - нужно создать компоненты, которые используют другие компоненты и передают данные через пропсы
3. Определяем минимальное (но полноценное) отображение состояния интерфейса
  - всё остальное вычисляйте при необходимости
4. Определяем, где должно находиться наше состояние
  - в React поток данных односторонний и сходит сверху вниз в иерархическом порядке
5. Добавляем обратный поток данных
  - задача сделать так, чтобы компоненты формы в самом низу иерархии обновляли состояние «родительских» компонентах

# «Бандлинг» (bundle)

- Большинство **React**-приложений «собирают» свои файлы такими инструментами, как **Webpack**, **Rollup** или **Browserify** (последний – вместе с **GULP**)
- Лучший способ внедрить разделение кода в приложение – использовать синтаксис динамического импорта: **import()**

*// Статический импорт*

```
import { add } from './math';  
console.log(add(16, 26));
```

*// Динамический импорт*

```
import("./math").then(math => {  
  console.log(math.add(16, 26));  
});
```

# Ленивая загрузка / React.lazy

59

```
import React, { Suspense } from 'react';
```

```
// Ленивая загрузка
```

```
const OtherComponent = React.lazy(() => import('./OtherComponent'));
```

```
function MyComponent() {
```

```
  // Suspense показывает компонент на время загрузки
```

```
  return (
```

```
    <div>
```

```
      <Suspense fallback={<div>Загрузка...</div>}>
```

```
        <OtherComponent />
```

```
      </Suspense>
```

```
    </div>
```

```
  );
```

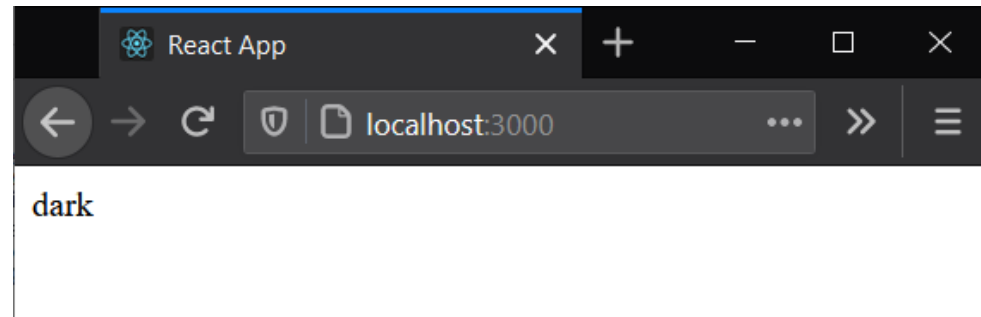
```
}
```

Проп **fallback** принимает любой **React**-элемент, который вы хотите показать, пока происходит загрузка компонента

# Контекст в React

```
import React from 'react';
import ReactDOM from 'react-dom';
// Значение по умолчанию - light
const ThemeContext = React.createContext('light');
class App extends React.Component {
  render() {
    // Компонент Provider используется для передачи вниз
    return (
      <ThemeContext.Provider value="dark">
        <Toolbar/>
      </ThemeContext.Provider>
    );
  }
}
function Toolbar() { // Компонент "посередине"
  return <div><ThemedSpan/></div>
}
class ThemedSpan extends React.Component {
  // Определяем contextType, чтобы получить значение контекста
  static contextType = ThemeContext;
  render() {
    return <span>{this.context}</span>;
  }
}
ReactDOM.render(<App />, document.getElementById('root'));
```

- **React.createContext** – создает Context
- **Context.Provider** – позволяет потомкам подписаться на изменения
- **Class.contextType** – назначает объект контекста для использования с **this.context**
- **Context.Consumer** – подписывается на изменения контекста

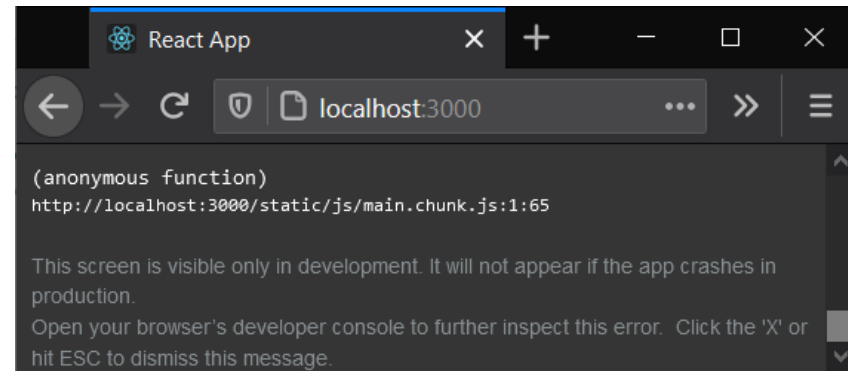
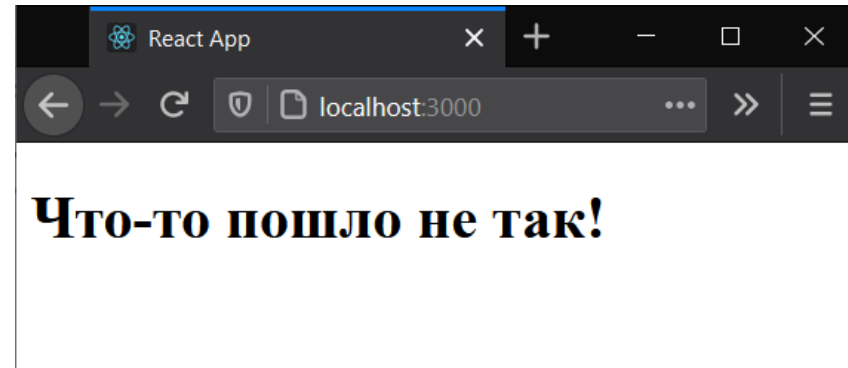


Контекст используется, если необходимо обеспечить доступ данных во многих компонентах на разных уровнях вложенности

# Предохранители

61

```
import React from 'react';
import ReactDOM from 'react-dom';
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }
  static getDerivedStateFromError(error) {
    // Обновление состояния - переход на запасной UI
    return { hasError: true };
  }
  componentDidCatch(error, errorInfo) {
    // Можно также сохранить информацию об ошибке
    console.log(error, errorInfo);
  }
  render() {
    if (this.state.hasError) {
      // Можно отрендерить запасной UI произвольного вида
      return <h1>Что-то пошло не так!</h1>;
    }
    return this.props.children;
  }
}
function MyWidget() {
  throw new Error("My error")
}
ReactDOM.render(<ErrorBoundary><MyWidget/></ErrorBoundary>, document.getElementById('root'));
```



Являются  
декларативным  
аналогом try/catch

- **Основные**

- **useState** – предназначен для управления состоянием компонентов
- **useEffect** – предназначен для перехвата различного рода изменений в компонентах, которые нельзя обработать внутри компонентов
- **useContext** – позволяет подписываться на контекст **React**

- **Дополнительные**

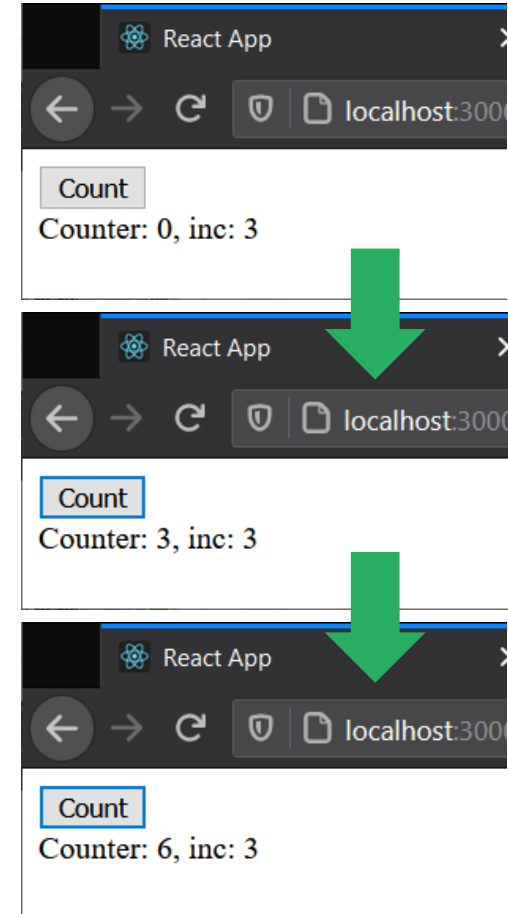
- **useReducer** – позволяет управлять локальным состоянием сложных компонентов
- **useCallback** – позволяет управлять функциями обратного вызова
- **useMemo** – предназначен для управления мемоизированными (кэшированными) значениями
- **useRef** – возвращает некоторое изменяемое значение, например, ссылку на **html**-элементы **DOM**, которой затем можно управлять в коде **JavaScript**
- **useImperativeHandle** – настраивает объект, который передается родительскому компоненту при использовании **ref**
- **useLayoutEffect** – аналогичен хуку **useEffect**, но вызывается синхронно после всех изменений в структуре **DOM**
- **useDebugValue** – предназначен для отображения некоторого значения в целях отладки

# Пример программы (подготовка)

63

```
import React from 'react';
import ReactDOM from 'react-dom';
class ClickButton extends React.Component {
  constructor(props) {
    super(props);
    this.state = {counter: 0};
  }
  press = () => {
    this.setState((prevState, props)=>({
      counter: prevState.counter + props.inc
    }));
  }
  render() {
    return <div>
      <button onClick={this.press}>Count</button>
      <div>Counter: {this.state.counter}, inc: {this.props.inc}</div>
    </div>
  }
}
ReactDOM.render(<ClickButton inc={3} />, document.getElementById("root"))
```

- Реализовано с помощью **класса**
- Имеет **состояние**



# Реализация примера с помощью хука useState

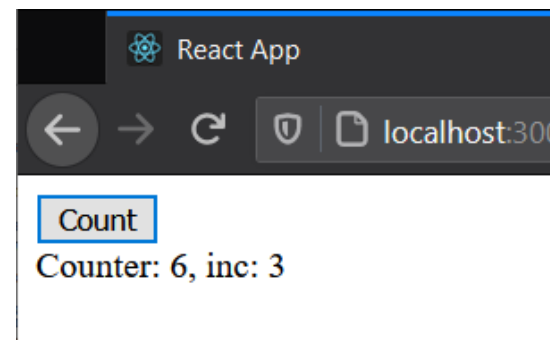
```
import React from 'react';
import ReactDOM from 'react-dom';

function ClickButton(props){
  const [count, setCount] = React.useState(0);
  const press = function(){
    setCount(count + props.inc);
  };
  return (<div>
    <button onClick={press}>Count</button>
    <div>Counter: {count}, inc: {props.inc}</div>
  </div>);
}

ReactDOM.render(<ClickButton inc={3} />, document.getElementById("root"))
```

- **count** – состояние компонента (функционального!!!)
- **setCount** – позволяет изменять значение переменной **count**
- **0** – значение **count** по умолчанию
- **useState** возвращает:
  1. текущее состояние
  2. функцию, обновляющую состояние

- Хуки вызываются **только на верхнем уровне** (top-level) компонента
  - **НЕ вызываются** внутри циклов, условных конструкций, внутри стандартных функций javascript
- Хуки можно вызывать только из **функциональных компонентов** React, либо из других хуков
  - Их **нельзя вызывать** из классов-компонентов





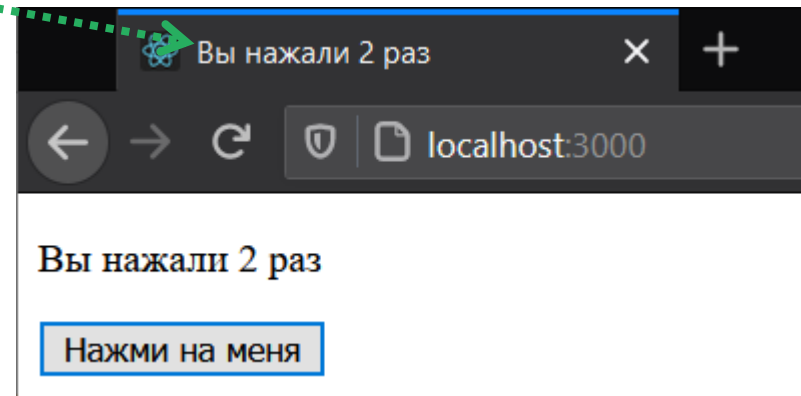
# Хук эффекта / useEffect, {useState}

65

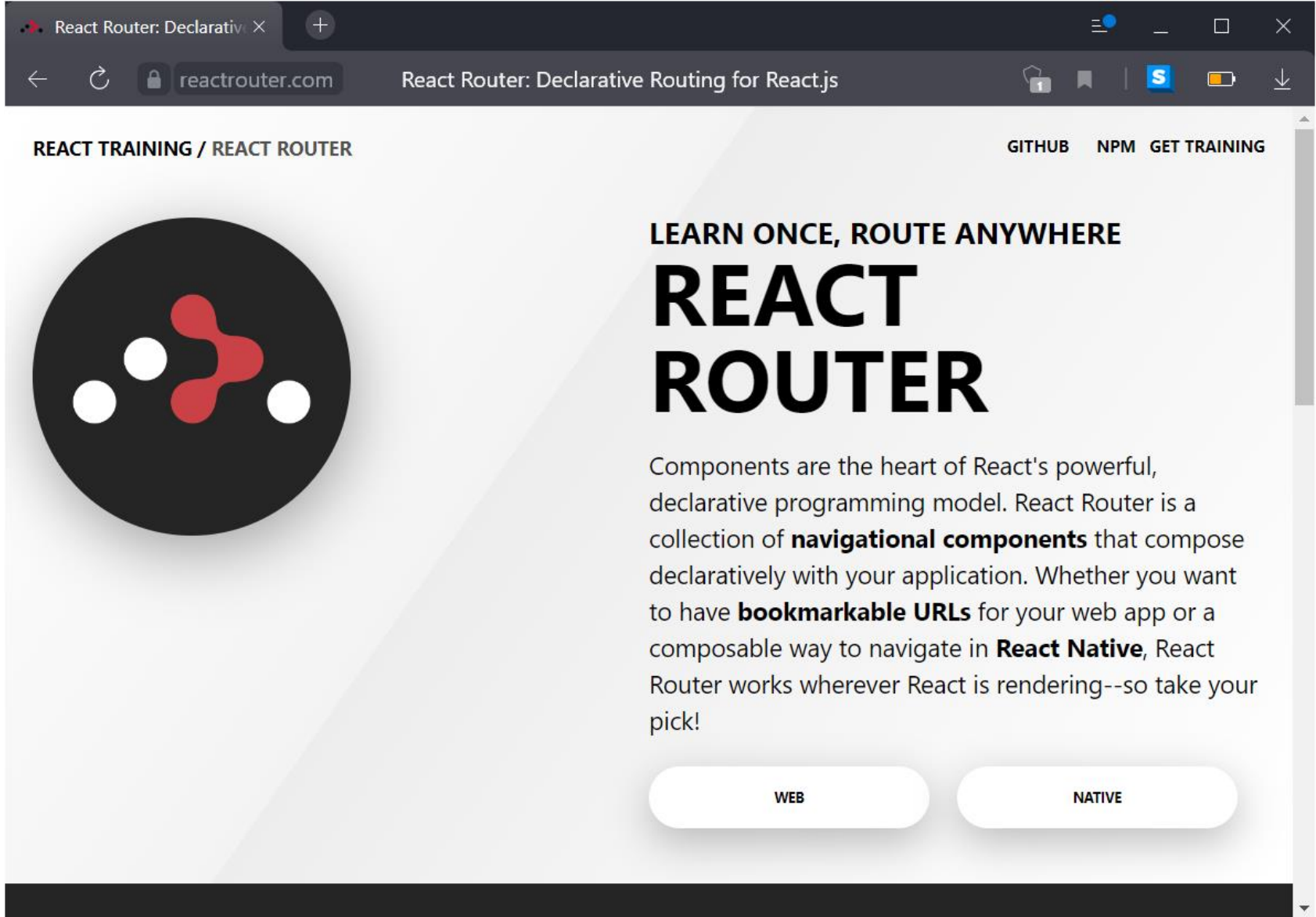
```
import React, { useState, useEffect } from 'react';
import ReactDOM from 'react-dom';

function ClickButton() {
  const [count, setCount] = useState(0);
  // Аналогично componentDidMount и componentDidUpdate:
  useEffect(() => {
    // Обновляем заголовок документа с помощью API браузера
    document.title = `Вы нажали ${count} раз`;
  });
  return (
    <div>
      <p>Вы нажали {count} раз</p>
      <button onClick={() => setCount(count + 1)}>
        Нажми на меня
      </button>
    </div>
  );
}

ReactDOM.render(<ClickButton />, document.getElementById("root"))
```



Используя этот хук, вы говорите **React** сделать что-то после рендера

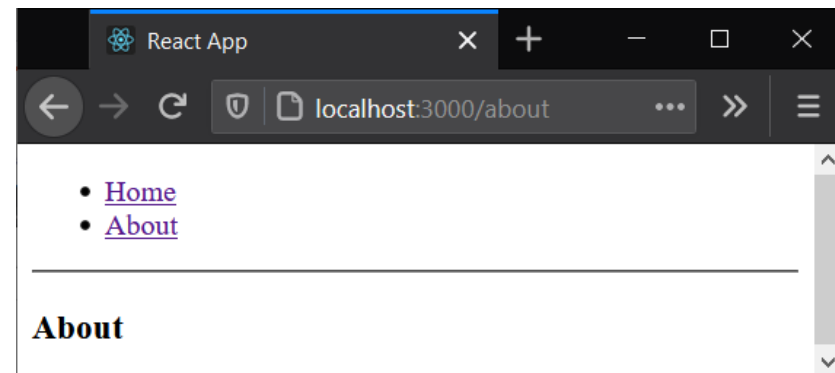
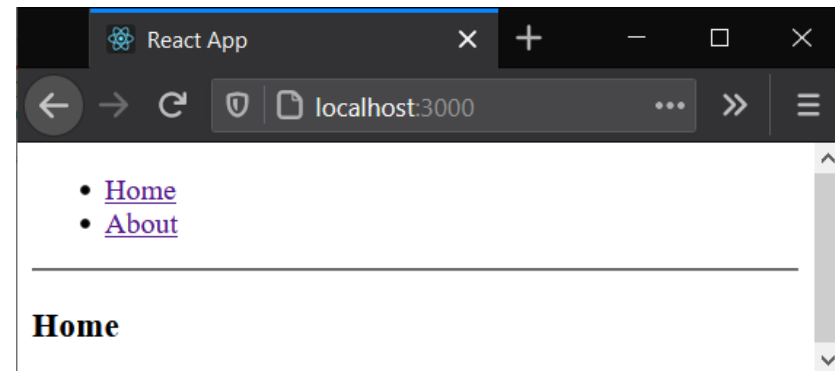


# Простой роутер / Link, path="\*"

67

```
import React from "react";
import ReactDOM from "react-dom";
import {BrowserRouter, Switch, Route, Link} from "react-router-dom";
function BasicExample() {
  return (
    <BrowserRouter>
      <div>
        <ul>
          <li><Link to="/">Home</Link></li>
          <li><Link to="/about">About</Link></li>
        </ul>
        <hr />
        <Switch>
          <Route exact path="/"><Home/></Route>
          <Route path="/about"><About/></Route>
          <Route path="*"><NoMatch/></Route>
        </Switch>
      </div>
    </BrowserRouter>
  );
}
function Home() { return <div><h3>Home</h3></div> }
function About() { return <div><h3>About</h3></div> }
function NoMatch() { return <div><h3>No match!</h3></div> }
ReactDOM.render(<BasicExample />, document.getElementById('root'));
```

Рассматривайте  
подключаемые  
компоненты как  
страницы  
приложения



Альтернативное объявление

`<Route exact path="/" component={Home} />`

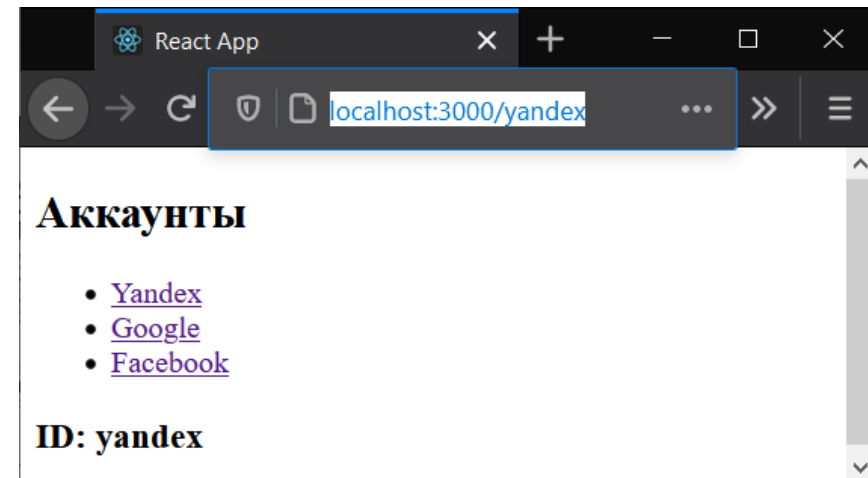
# Анализ параметров URL

68

```
import React from "react";
import ReactDOM from "react-dom";
import {BrowserRouter, Switch, Route, Link, useParams} from "react-router-dom";
function ParamsExample() {
  return (
    <BrowserRouter>
      <div>
        <h2>Аккаунты</h2>
        <ul>
          <li><Link to="/yandex">Yandex</Link></li>
          <li><Link to="/google">Google</Link></li>
          <li><Link to="/facebook">Facebook</Link></li>
        </ul>
        <Switch>
          <Route path="/:id" children={<Child />} />
        </Switch>
      </div>
    </BrowserRouter>
  );
}

function Child() {
  let {id} = useParams(); // Доступ к URL
  return <div><h3>ID: {id}</h3></div>
}

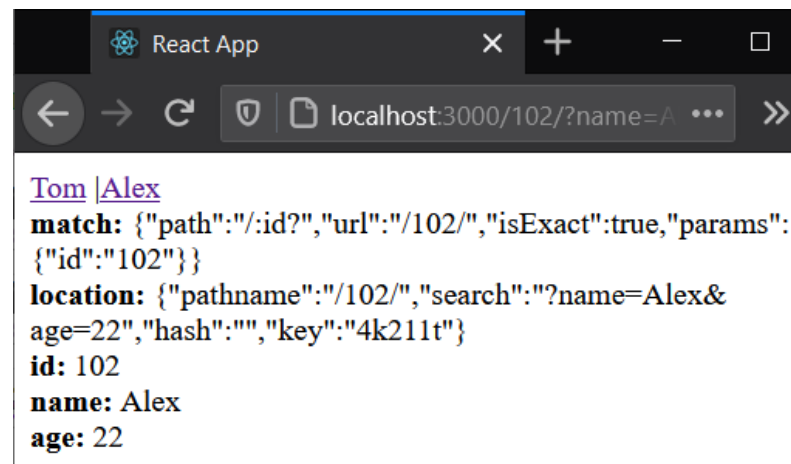
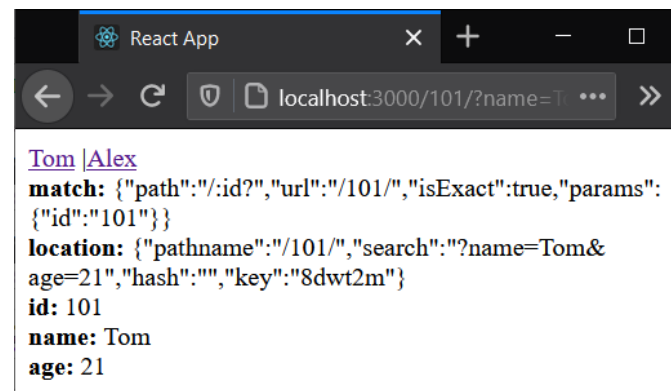
ReactDOM.render(<ParamsExample />, document.getElementById('root'));
```



# Чтение параметров запроса

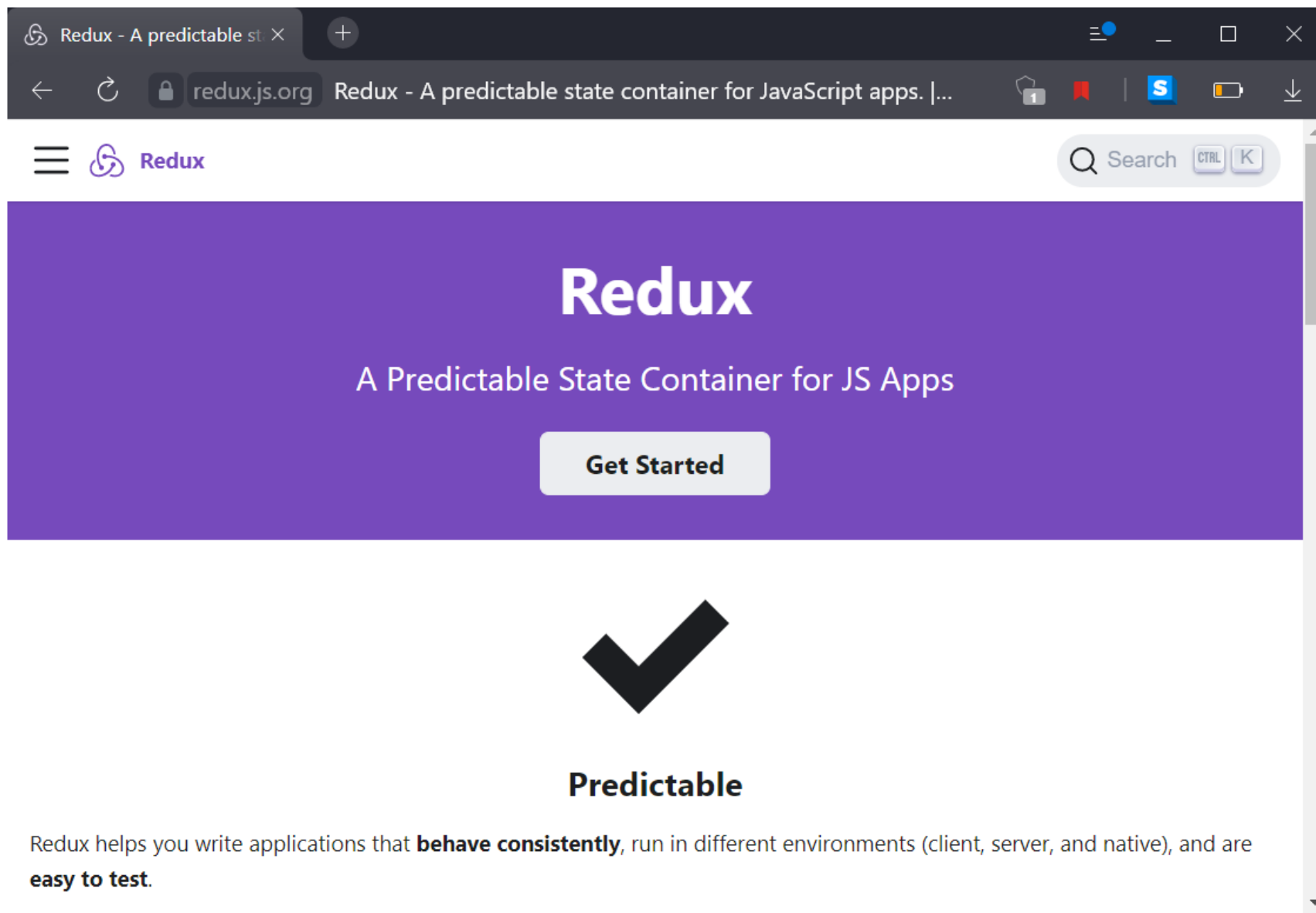
69

```
import React from "react";
import ReactDOM from "react-dom";
import {BrowserRouter, Link, Route, Switch} from "react-router-dom";
class Home extends React.Component{
  render(){
    const match = this.props.match;
    const loc = this.props.location;
    return <div>
      <b>match:</b> {JSON.stringify(match)}<br/>
      <b>location:</b> {JSON.stringify(loc)}<br/>
      <b>id:</b> {match.params.id}<br/>
      <b>name:</b> {new URLSearchParams(loc.search).get("name")}<br/>
      <b>age:</b> {new URLSearchParams(loc.search).get("age")}
    </div>;
  }
}
ReactDOM.render(
  <BrowserRouter>
    <div>
      <nav>
        <Link to="/101/?name=Tom&age=21">Tom</Link> |
        <Link to="/102/?name=Alex&age=22">Alex</Link>
      </nav>
      <Switch>
        <Route path="/:id?" component={Home} />
      </Switch>
    </div>
  </BrowserRouter>,
  document.getElementById("root")
)
```



# Redux – контейнер состояния

70



<https://redux.js.org/>

# Redux

- **Redux** является предсказуемым контейнером состояния для **JavaScript** приложений
- Основные принципы
  - Единое дерево неизменяемого дерева состояний
  - Состояние – только для чтения
  - Используются только чистые функции
- Обработчик события – чистая функция, которая получает на вход предыдущее состояние и событие  
**(state, action) => newState**

```
npm install @reduxjs/toolkit
npm install redux
npx create-react-app my-app --template redux
```



альтернативы в зависимости от задачи

```
// Pure functions
function square(x) {
  return x**2;
}
function squareAll(items) {
  return items.map(square);
}

// Impure functions
function square(x) {
  askDB(x);
  return x**2;
}
function squareAll(items) {
  for (let i = 0; i <
items.length; i++)
    items[i] =
square(items[i]);
}
```

## Хранилище (store)

- хранит состояние приложения

## Действия (actions)

- некоторый набор информации, который исходит от приложения к хранилищу и который указывает, что именно нужно сделать
- для передачи этой информации у хранилища вызывается метод `dispatch()`

## Создатели действий (action creators)

- функции, которые создают действия

## Reducer

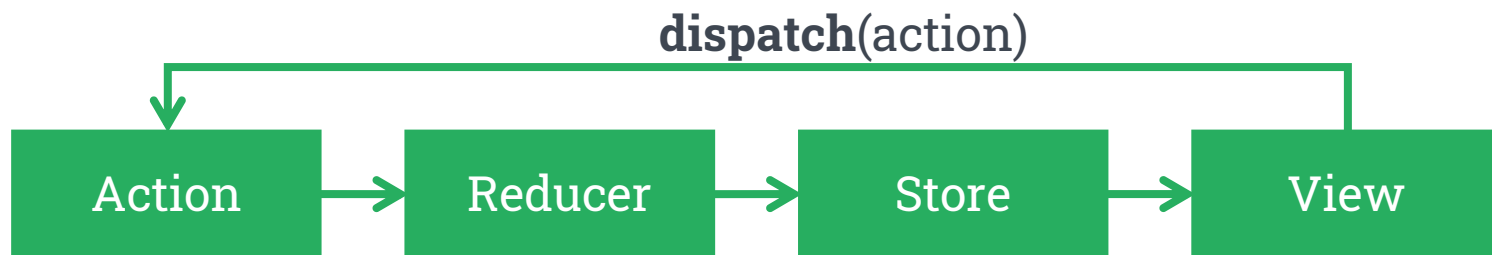
- функция (или несколько функций), которая получает действие и в соответствии с этим действием изменяет состояние хранилища



# Основной поток в Redux

73

1. Из **компонентов React** вызывается действие
2. Его обрабатывает **reducer**
3. Который в соответствии с ним обновляет **хранилище**
4. Компоненты React **применяют** обновленное состояние

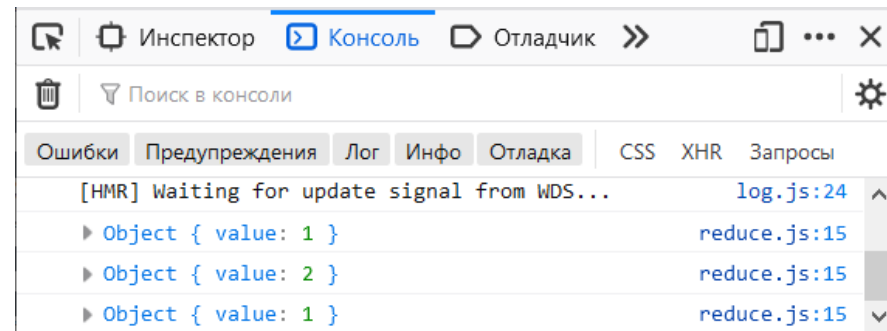


# Простой пример использования Redux

```
import { createStore } from 'redux'
// reducer - применяет действие к текущему состоянию
function counterReducer(state = { value: 0 }, action) {
  switch (action.type) {
    case 'counter/inc': return { value: state.value + 1 }
    case 'counter/dec': return { value: state.value - 1 }
    default: return state
  }
}
// Создание Redux-хранилища
// API: { subscribe, dispatch, getState }.
let store = createStore(counterReducer)
// В случае использовать React будет использоваться React Redux
// Рассмотрим простой пример с subscribe()
store.subscribe(() => console.log(store.getState()))
store.dispatch({ type: 'counter/inc' }) // {value: 1}
store.dispatch({ type: 'counter/inc' }) // {value: 2}
store.dispatch({ type: 'counter/dec' }) // {value: 1}
```

**createStore**  
принимает в качестве параметра **reducer** и создаёт хранилище, которое хранит полное состояние приложения

Здесь «**counter**» - «feature» приложения



# Простой пример использования Redux Toolkit

```
import { createSlice, configureStore } from '@reduxjs/toolkit'
```

```
const counterSlice = createSlice({
  name: 'counter',
  initialState: { value: 0 },
  reducers: {
    inc: state => { // Используется Immutable.JS
      state.value += 1
    },
    dec: state => {
      state.value -= 1
    }
  }
})
```

```
const { inc, dec } = counterSlice.actions
```

```
const store = configureStore({
  reducer: counterSlice.reducer
})
```

// Можно подписаться

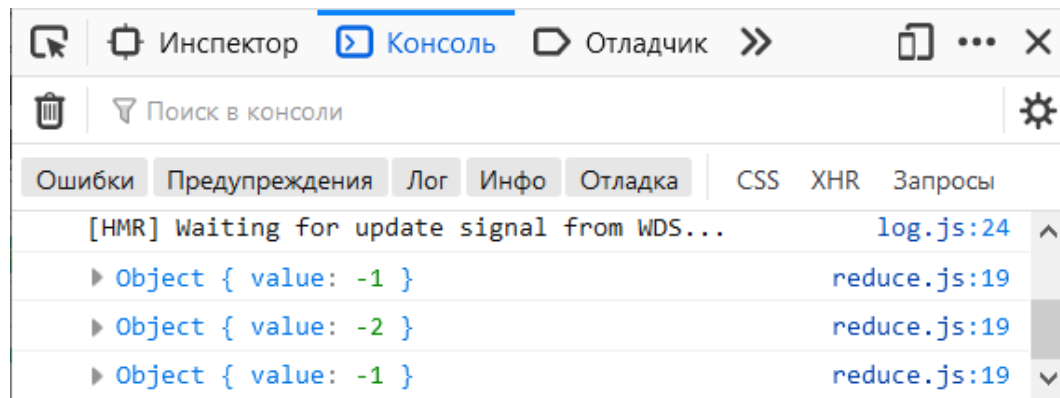
```
store.subscribe(() => console.log(store.getState()))
```

// Отправка действий диспетчеру

```
store.dispatch(dec()) // {value: -1}
```

```
store.dispatch(dec()) // {value: -2}
```

```
store.dispatch(inc()) // {value: -1}
```



- **slice** – коллекция **reducers** и **действий** для отдельной «feature» (обычно описываются в одном файле)
- **createSlice** упрощает логику **reducer** и **действий**

# Основная идея Redux (1) / state, action<sup>76</sup>

## Состояние

```
const state = {  
  todos: [{  
    text: 'Выучить React',  
    completed: true  
  }, {  
    text: 'Сдать Web-технологии',  
    completed: false  
  }],  
  visibilityFilter: 'SHOW_COMPLETED'  
}
```

## Выполняемые действия

```
const seqActions = [  
  { type: 'ADD_TODO', text: 'Выполнить л/р' },  
  { type: 'TOGGLE_TODO', index: 1 },  
  { type: 'SET_VISIBILITY_FILTER', filter: 'SHOW_ALL' }  
]
```

- Действия — это структуры, которые передают данные из вашего приложения в хранилище
- Они являются единственными источниками информации для хранилища
- Действие — объект
- У действия всегда есть тип, описываемый строкой **type**
  - хороший стиль, когда в **type** передаётся константа

# Основная идея Redux (2) / reducers

77

```
function visibilityFilter(state = 'SHOW_ALL', action) {
  if (action.type === 'SET_VISIBILITY_FILTER') {
    return action.filter
  } else {
    return state
  }
}

function todos(state = [], action) {
  switch (action.type) {
    case 'ADD_TODO':
      return state.concat([{ text: action.text, completed: false }])
    case 'TOGGLE_TODO':
      return state.map((todo, index) =>
        action.index === index
          ? { text: todo.text, completed: !todo.completed }
          : todo
      )
    default:
      return state
  }
}

function todoApp(state = {}, action) {
  return {
    todos: todos(state.todos, action),
    visibilityFilter: visibilityFilter(state.visibilityFilter, action)
  }
}
```

- **Reducers** определяют, как в ответ на действие изменяется состояние (state) приложения
- Все состояние приложения хранится в виде единственного объекта
- **Reducer** — это чистая функция, которая принимает предыдущее состояние и действие и возвращает следующее состояние (новую версию предыдущего)
  - (previousState, action) => newState
- **Никогда нельзя** делать в **reducer**:
  1. непосредственно изменять то, что пришло в аргументах функции
  2. выполнять какие-либо сайд-эффекты: обращаться к API или осуществлять переход по роутам
  3. вызывать не чистые функции, например Date.now() или Math.random()

# Уточнённый поток данных Redux

78

1. Вызов **store.dispatch(action)**

- { **type: ADD\_TODO**, **text** }

2. Хранилище вызывает **reducer**, который ему передали

- Параметры: текущее дерево состояния (**current state tree**) и действие (**action**)

3. Главный **reducer** может комбинировать результат работы нескольких **reducers** в дерево состояния приложения

4. Хранилище сохраняет полное дерево состояния, которое возвращает главный **reducer**

- Для прослушивания могут использоваться слушатели: **store.subscribe(listener)**
- Слушатели могут вызывать **store.getState()** для получения текущего состояния приложения

# Описанный пример в CodeSandbox

79

The screenshot displays a CodeSandbox workspace for a project named 'todos'. The browser address bar shows 'codesandbox.io' and the project path 'reduxjs / redux / master / examples / todos'. The file explorer on the left shows a directory structure with 'public', 'src', 'actions', 'components', 'containers', and 'reducers' folders, along with files like '.gitignore', 'README.md', 'package-lock.json', and 'package.json'. The main editor shows the 'index.js' file with the following code:

```
1 import React from 'react'
2 import { render } from 'react-dom'
3 import { createStore } from 'redux'
4 import { Provider } from 'react-redux'
5 import App from '../components/App'
6 import rootReducer from '../reducers'
7
8 const store = createStore(rootReducer)
9
10 render(
11   <Provider store={store}>
12     <App />
13   </Provider>,
14   document.getElementById('root')
15 )
16
```

The right sidebar contains a 'Browser' view showing a simple todo application interface. It includes an input field, an 'Add Todo' button, a list of todos (one of which is 'Выучить React'), and filter buttons for 'All', 'Active', and 'Completed'. The bottom status bar indicates the file is at 'Ln 1, Col 1' with 'Spaces: 2', 'UTF-8' encoding, 'LF' line endings, and 'JavaScript' language.

<https://codesandbox.io/s/github/reduxjs/redux/tree/master/examples/todos>

# Material UI React

80

MUI: The React compon X

mui.com

MUI: The React component library you always wanted

Products Docs Pricing About us

Поиск... Ctrl+K

# The React UI library you always wanted

MUI provides a robust, customizable, and accessible library of foundational and advanced components, enabling you to build your own design system and develop React applications faster.

Get started >

```
$ npm install @mui/material
```

March 25th

Check the docs for getting every component API

Assigned to Michael Scott

60%

Ultraviolet

Basement • Beside Myself

<https://mui.com/ru/>



# Вопросы для самопроверки

81

- Что такое иммутабельность? Зачем нужна?
- Что такое JSX? Какое отношение он имеет к React?
- Что такое React-элемент? Из чего состоит?
- Какие бывают варианты определения компонента в React?
- Что такое пропсы (props)? props.children?
- Что такое state? Как его можно менять? Как нельзя?
- Какие методы жизненного цикла есть в React?
- Чем отличаются управляемые и неуправляемые компоненты? Как их использовать?
- Что такое ключ? И в каком случае он нужен?
- Что такое реф (ref)? Как его использовать?
- Какие особенности есть у событий React? Чем отличаются от HTML?
- Что такое условный рендеринг? Как "отключать" компоненты?
- Как сделать валидацию?
- Как построить иерархию компонентов? Как передавать данные "вверх"? "Вниз"?
- Что такое предохранители? Аналогом чего являются?
- Что такое хуки? Как и в каких случаях можно использовать useState? useEffect? Где применимы? Где нет?
- Как сделать маршрутизацию?
- Как реализовать единое состояние в приложении?