

[РАЗРАБОТКА ВЕБ-САЙТОВ\\*](#), [REACTJS\\*](#), [БЛОГ КОМПАНИИ RUVDS.COM](#)

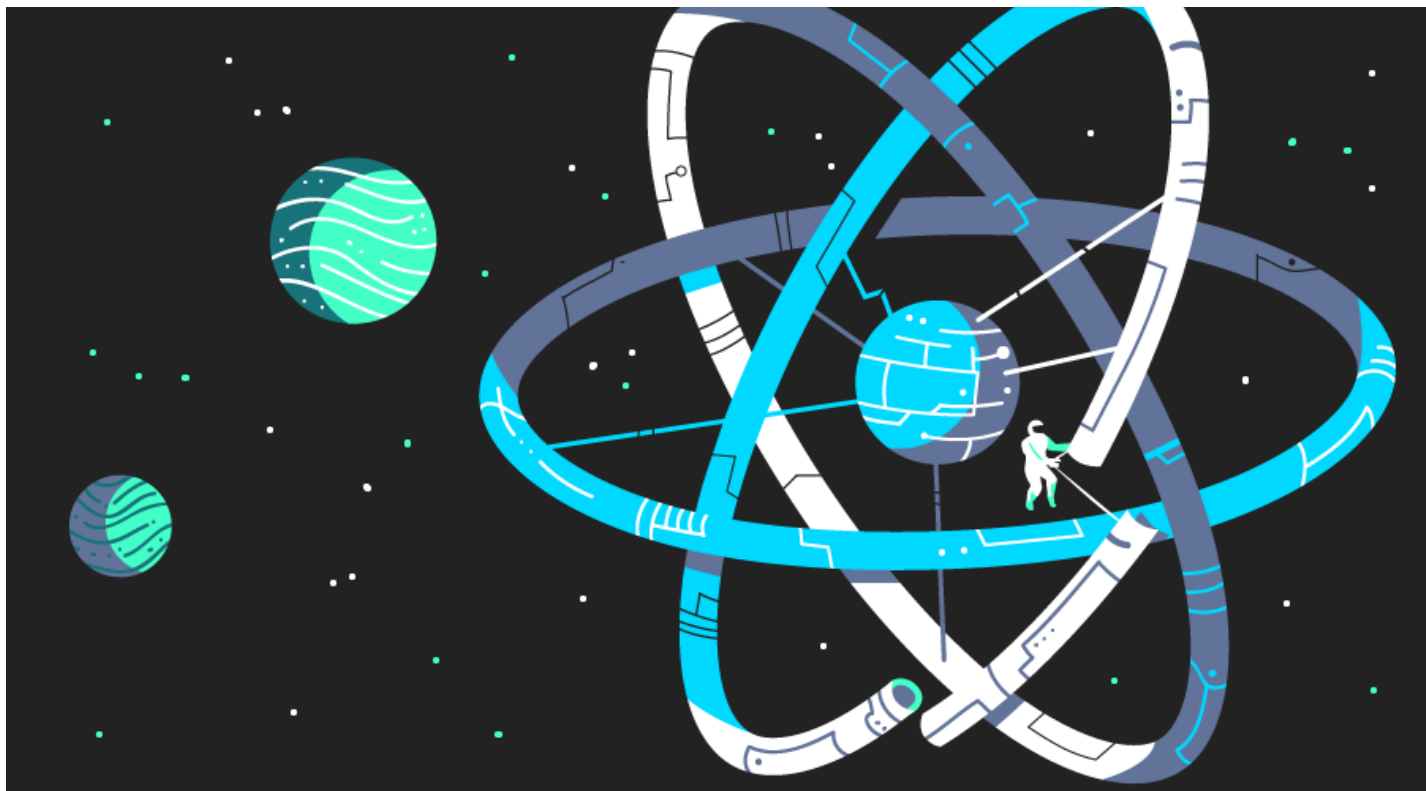
## Новшества React 16.3(.0-alpha)

**ПЕРЕВОД**

ru\_vds 12 февраля в 12:02 👁 9k

Оригинал: [Bartosz Szczeciński](#)

React 16.3-alpha опубликован в [npm](#), его уже можно загружать и использовать. Сегодня мы поговорим о самых крупных и интересных нововведениях этого релиза. В частности, речь пойдёт об API Context, о новых методах жизненного цикла, о статическом методе `getDerivedStateFromProps`, о компонентах `StrictMode` и `AsyncMode`, а также об обновлённых инструментах разработчика React.



# Новое API Context

API Context всегда выглядело как нечто таинственное. С одной стороны — это официальное, документированное API React, но, с другой стороны, разработчиков предупреждали о том, что использовать его не следует, так как оно может со временем измениться, и документация по нему, намеренно, была неполной. Теперь время этого API настало, [фаза RFC](#) пройдена, и новое API, которое, определённо, стало более дружелюбным, доступно разработчикам. Оно может облегчить вам жизнь, если всё, что вам нужно — простое управление состоянием без «излишеств» Redux или MobX.

Новое API доступно в виде `React.createContext()`, оно даёт в наше распоряжение два компонента:

```
import { createContext } from "react";
const ThemeContext = createContext({
  background: 'yellow',
  color: 'white'
});
```

*Создание нового контекста командой `React.createContext`*

Тут, вызвав фабричную функцию, мы получаем объект, в котором имеются элементы `Provider` и `Consumer`.

`Provider` — это особый компонент, цель существования которого заключается в том, чтобы предоставлять данные всем компонентам в собственном поддереве. Например, это может

выглядеть так:

```
class Application extends React.Component {  
  render() {  
    return (  
      <ThemeContext.Provider value={{background: 'black', color: 'white'}}>  
        <Header />  
        <Main />  
        <Footer />  
      </ThemeContext.Provider>  
    )  
  }  
}
```

### *Использование нового API Context — Context.Provider*

Тут выбрано поддерево (в данном случае — всё дерево), которому мы хотим передать контекст `theme`, и установлены значения, которые нужно передать. Значения, конечно, могут быть динамическими (то есть, основанными на `this.state`).

Следующий шаг заключается в использовании элемента `Consumer`:

```
const Header = () => (  
  <ThemeContext.Consumer>  
    {(context) => {  
      return (  
        <div style={{background: context.background, color: context.color}}>  
          Welcome!  
        </div>  
      );  
    }}  
  </ThemeContext.Consumer>  
);
```

### *Использование нового API Context — Context.Consumer*

Если случилось так, что вы рендерите `Consumer` не внедряя его в соответствующий `Provider`, будут использованы значения по умолчанию, объявленные в `createContext`.

Обратите внимание на следующее:

- У компонента `Consumer` должен быть доступ к уже используемому компоненту `Context`. Если будет создан новый контекст, с теми же входными параметрами, это равносильно созданию нового компонента `Context`, в результате те данные, которые были в исходном компоненте `Context`, компоненту `Consumer` переданы не будут. По этой причине к `Context` стоит относиться как к компоненту — его нужно создавать один раз, а затем экспортировать или импортировать там, где это нужно.
- В новом синтаксисе используется шаблон React «функция как потомок» (иногда это называют `render prop`). Если вы с этим шаблоном не знакомы — взгляните на [данный материал](#).
- При работе с новыми конструкциями, если планируется использовать новое API `Context`, больше не нужно использовать `prop-types` для указания `contextProps`.

Данные из контекста, передаваемые функции, соответствуют свойству `value`, установленному в провайдерах компонента `Context.provider`, и изменение данных в компоненте `Provider` приводит к перерисовке всех потребителей этих данных.

## Новые методы жизненного цикла

Ещё одно новшество, которое, из стадии [RFC](#), перешло в

рассматриваемый альфа-релиз React, касается признания устаревшими некоторых методов жизненного цикла и введения нескольких новых методов.

Это изменение нацелено на внедрение рекомендованных подходов к разработке (вот [материал](#) о том, почему с этими функциями может быть непросто обращаться), что обретёт особую важность после того, как будет полностью активирован асинхронный режим рендеринга (что является одной из важнейших целей архитектуры Fiber, появившейся в React 16).

Вот функции, которые через некоторое время признают устаревшими:

- Функция `componentWillMount`, вместо которой предлагается использовать `componentDidMount`.
- Функция `componentWillUpdate`, которую заменит `componentDidUpdate`.
- Функция `componentWillReceiveProps`. В качестве заменителя этой функции предлагается новая статическая функция `getDerivedStateFromProps`.

В свете вышесказанного не стоит впадать в панику, так как эти функции всё ещё можно использовать. Уведомления о том, что данные функции устарели, появятся в React 16.4, а их удаление запланировано в 17-й версии React.

There will be NO breaking changes in React 16.3.0 (or anywhere in the 16.x release cycle). React respects semver (and always has).

*Паника в стане React-разработчиков. Дэн Абрамов предлагает не беспокоиться, но его слова действуют не на всех*

Уведомления будут выводиться только при использовании новых компонентов `StrictMode` или `AsyncMode`, но и в этих случаях их можно отключить, используя следующие методы:

- `UNSAFE_componentWillMount`
- `UNSAFE_componentWillReceiveProps`
- `UNSAFE_componentWillUpdate`

## Статический метод `getDerivedStateFromProps`

Так как `componentWillReceiveProps` собираются убрать, нужны какие-то механизмы для обновления состояния на основании изменения свойств. Для решения этой задачи было решено представить новый статический метод.

Что такое статический метод? Это — метод, который существует в классе, а не в его экземплярах. Пожалуй, легче всего описать этот метод, как такой, у которого нет доступа к `this` и имеется ключевое слово `static` в его объявлении.

Однако тут возникает вопрос. Если у функции нет доступа к `this`, как же вызвать `this.setState`? Ответ заключается в том, что

ничего такого вызывать и не нужно. Функция лишь должна вернуть обновлённые данные состояния, или, если обновление не требуется, `null`:

```
static getDerivedStateFromProps(nextProps, prevState) {  
  if (nextProps.currentRow === prevState.lastRow) {  
    return null;  
  }  
  return {  
    lastRow: nextProps.currentRow,  
    isScrollingDown: nextProps.currentRow > prevState.lastRow  
  };  
}
```

*Использование `getDerivedStateFromProps` для обновления состояния*

Возвращённое значение ведёт себя точно так же, как текущее значение `setState` — нужно лишь вернуть изменённую часть состояния, все остальные значения останутся такими же, как были.

## Полезные советы по использованию `getDerivedStateFromProps`

### ■ Не забудьте инициализировать состояние

✖ ▶Warning: Demo: Did not properly initialize state during construction. Expected state to be an object, but it was undefined. [VM482 bundle.js:1193](#)

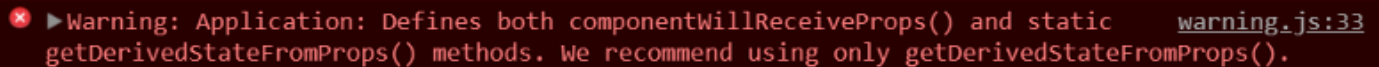
*Помните о необходимости инициализации состояния*

Инициализацию исходного состояния компонента никто не отменял. Делать это надо либо в конструкторе, либо путём настройки поля класса.

## ■ Особенности вызова `getDerivedStateFromProps`

Рассматриваемая функция вызывается и при первоначальном монтировании, и при перерисовке компонента, поэтому вы можете использовать её вместо создания в конструкторе состояния, основанного на свойствах.

## ■ Ошибка при совместном использовании `getDerivedStateFromProps` и `componentWillReceiveProps`

A screenshot of a warning message from the React DevTools console. The message is displayed on a dark red background with white text. It reads: "Warning: Application: Defines both componentWillMountReceiveProps() and static getDerivedStateFromProps() methods. We recommend using only getDerivedStateFromProps().". To the right of the text, it says "warning.js:33".

```
Warning: Application: Defines both componentWillMountReceiveProps() and static getDerivedStateFromProps() methods. We recommend using only getDerivedStateFromProps(). warning.js:33
```

*Предупреждение, в котором рекомендуется использовать только `getDerivedStateFromProps`*

Если вы объявите и `getDerivedStateFromProps`, и `componentWillReceiveProps`, будет вызвана лишь функция `getDerivedStateFromProps`, а в консоли появится предупреждение.

## ■ О коллбэках и `componentDidUpdate`

Если обычно вы применяете коллбэки для того, чтобы обеспечить выполнение некоего кода после успешного обновления состояния,



теперь, вместо коллбэков, используйте `componentDidUpdate`.

## О ключевом слове `static`

Если вы предпочитаете не использовать ключевое слово `static`, можете использовать альтернативную форму записи:

```
ComponentName.getDerivedStateFromProps = (nextProps, prevState) => {  
  
}
```

*Объявление `getDerivedStateFromProps` без использования ключевого слова `static`*

## Новый компонент `StrictMode`

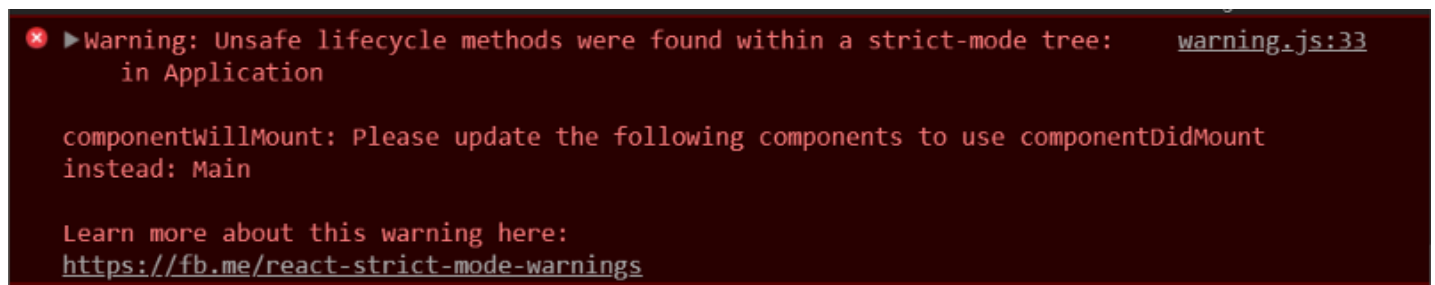
Строгий режим React представлен компонентом, который доступен по адресу `React.StrictMode`. Его можно добавить в дерево или поддерево приложения:

```
import { StrictMode } from "react";  
  
class Application extends React.Component {  
  render() {  
    return (  
      <StrictMode>  
        <Context.Provider value={{background: 'black', color: 'white'}}>  
          <Header />  
          <Main />  
          <Footer />  
        </Context.Provider>  
      </StrictMode>  
    )  
  }  
}
```

*Использование use strict... Ох, не то. Мы, конечно, имеем в виду компонент StrictMode*

Смысл использования StrictMode заключается в том, что благодаря наличию этого компонента система помогает обеспечивать соответствие кода рекомендованным подходам к разработке.

Если один из дочерних компонентов, выведенных в поддереве StrictMode, использует некоторые из методов, упомянутых выше (вроде componentWillMount), вы увидите сообщение об ошибке в консоли браузера при запуске проекта в режиме разработки:



*Ошибка, сообщающая об использовании небезопасных методов жизненного цикла в поддереве StrictMode*

Сейчас сообщение об ошибке [указывает](#) на RFC, где говорится об удалении методов жизненного цикла.

## Новый компонент AsyncMode

Пока ещё неактивный механизм поддержки асинхронных компонентов был переименован так, чтобы его имя

соответствовало имени компонента `StrictMode`. Теперь этот компонент можно найти по адресу `React.unsafe_AsyncMode`. Использование `AsyncMode` также активирует уведомления, характерные для `StrictMode`.

Если вы хотите узнать подробности об асинхронных компонентах React, загляните [сюда](#) и [сюда](#).

## Новая версия инструментов разработчика React

В дополнение к вышеописанным новшествам, в релизе React, о котором мы говорим, была представлена новая версия инструментов разработчика, которая поддерживает отладку новых компонентов.

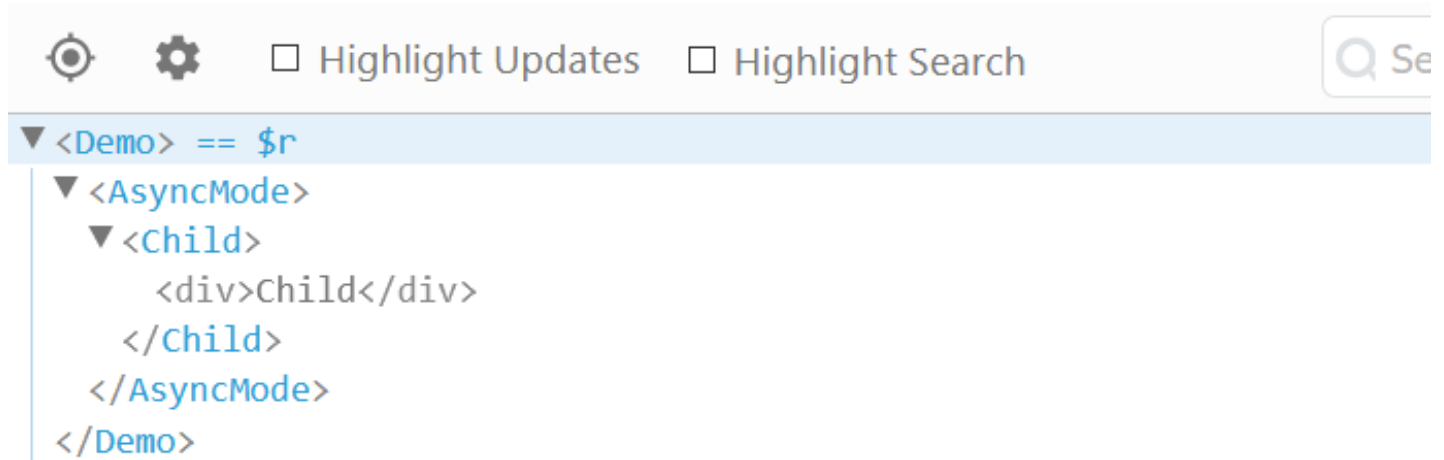
Если вы используете Chrome, то вам понадобится немного подождать, так как соответствующее расширение в Интернет-магазине Chrome пока не обновлено, а попытки отлаживать старыми инструментами новый код приводят к... интересным результатам:

```
▼ <Demo> == $r
  ▼ <TODO_NOT_IMPLEMENTED_YET $$typeof=Symbol(react.element) type=Child() key=null...>
    ▶ <Child>...</Child>
    </TODO_NOT_IMPLEMENTED_YET>
  </Demo>
```

*Рекорд*

*React.\_\_SECRET\_INTERNALS\_DO\_NOT\_USE\_OR\_YOU\_WILL\_BE\_FIRED пока ещё не побит*

А вот пользователи Firefox уже могут отлаживать новые конструкции:



*Новый асинхронный компонент можно видеть при отладке в Firefox*

## Итоги: самое интересное впереди

Тут мы рассмотрели новшества альфа-релиза React 16.3, однако, стабильная версия этого релиза может содержать в себе другой набор изменений. Если ориентироваться на слова Дэна Абрамова по этому поводу, то React 16.3.0 должен выйти совсем скоро.



**Dan Abramov**

@dan\_abramov

Following



Replying to @kentcdodds @acdlite

I think 16.3.0 is likely to come out next week, with a blog post.

4:19 PM - 2 Feb 2018

3 Retweets 33 Likes



1



3



33



*На прошлой неделе говорили о следующей неделе, которая уже наступила*

**Уважаемые читатели!** Что вы думаете о новых возможностях React?

**Habrahabr10**

Промо-код для скидки в 10% на наши виртуальные сервера

Проголосовать:



**+27**



Поделиться:



Сохранить:



## Похожие публикации

### React, встроенные функции и производительность

ПЕРЕВОД

ru\_vds • 13 октября 2017 в 13:48

23

### Новшества серверного рендеринга в React 16

ПЕРЕВОД

ru\_vds • 2 октября 2017 в 15:21

18

### Добротный риалтайм на React и Socket.io

ПЕРЕВОД

ru\_vds • 18 июля 2017 в 16:12

8

## Популярное за сутки

### Наташа — библиотека для извлечения структурированной информации из текстов на русском языке

alexkuku • вчера в 16:12

14

### Unit-тестирование скриншотами: преодолеваем звуковой барьер. Расшифровка доклада

lahmatiy • вчера в 13:05

4

**Люди не хотят чего-то действительно нового — они хотят привычное, но сделанное иначе**

ПЕРЕВОД

Smileek • вчера в 10:32

25

**Руководство по SEO JavaScript-сайтов. Часть 2. Проблемы, эксперименты и рекомендации**

ПЕРЕВОД

ru\_vds • вчера в 12:04

2

**Как адаптировать игру на Unity под iPhone X к апрелю**

P1CACHU • вчера в 16:13

0

**Лучшее на Geektimes**

**Стивен Хокинг, автор «Краткой истории времени», умер на 77 году жизни**

HostingManager • вчера в 13:49

33

**Обзор рынка моноколес 2018**

lozga • вчера в 06:58

70

**«Битва за Telegram»: 35 пользователей подали в суд на ФСБ**

alizar • вчера в 15:14

40

## Стивен Хокинг и его работа — что дал ученый человечеству?

marks • вчера в 14:46

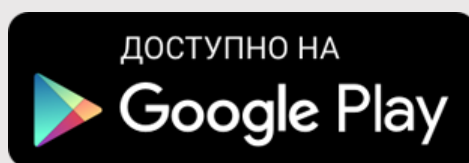
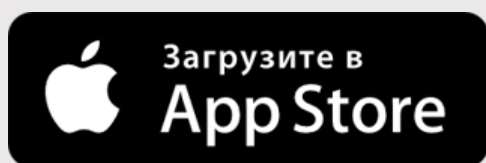
8

## Sunlike — светодиодный свет нового поколения

AlexeyNadezhin • вчера в 20:32

17

Мобильное приложение



Полная версия

2006 – 2018 © TM