

[КАК СТАТЬ АВТОРОМ](#)[Неделя тестировщиков](#)[Как разработчикам и дизайнерам найти...](#)**728.55**

Рейтинг

## Timeweb Cloud

Облачная платформа для разработчиков и бизнеса



aio350 8 февраля в 15:45

## JavaScript: делаем селфи с помощью браузера

Блог компании Timeweb Cloud, Разработка веб-сайтов\*, JavaScript\*

# JAVASCRIPT

Привет, друзья!

В этой статье я покажу вам, как делать **селфи** в браузере.

Мы разработаем простое приложение со следующим функционалом:

- при инициализации приложение запрашивает у пользователя разрешение на захват медиапотока (далее также — поток) из видеокамеры его устройства;
- захваченный поток передается в элемент `video` ;
- из потока извлекается видеотрек (далее также — трек), который передается в интерфейс для захвата изображений;
- из экземпляра интерфейса извлекается список поддерживаемых возможностей (capabilities) и

настроек (settings) для фото;

- из трека также извлекается список поддерживаемых возможностей и настроек;
- формируется список диапазоновых полей ( `<input type="range">` ) для установки настроек для фото;
- пользователь имеет возможность снимать фото (take photos) и захватывать фреймы (grab frames);
- фото выводится в элемент `img` , генерируется ссылка для его скачивания;
- фрейм инвертируется и отрисовывается на холсте ( `canvas` ), генерируется ссылка для его скачивания.

Репозиторий.

Приложение будет разработано на чистом JavaScript .

Для создания шаблона приложения будет использован [Vite](#) .

Если вам это интересно, прошу под кат.

Данная статья является четвертой в серии статей, посвященных работе с медиа в браузере:

- [статья о создании аудиозаписей](#);
- [статья о записи экрана](#);
- [статья о сведении аудио и видеотреков](#).

При разработке приложения мы будем опираться на следующие спецификации:

- [MediaStream Image Capture](#) — захват изображения;
- [Media Capture and Streams](#) — захват потока;
- [File API](#) — работа с файлами.

Ссылки на соответствующие разделы [MDN](#) будут приводиться по мере необходимости.

*Обратите внимание:* основная технология, которую мы будем рассматривать, является экспериментальной. На сегодняшний день она поддерживается только [71%](#) браузеров. Поэтому говорить о возможности ее использования в продакшне рано. Можно сказать, что эта технология относится к категории "Веб завтрашнего дня".

---

Создаем шаблон приложения:

```
# image-capture - название проекта
# --template vanilla - используемый шаблон
yarn create vite image-capture --template vanilla
# или
npm init vite ...
```

Пока создается шаблон, поговорим об интерфейсах и методах, которые мы будем использовать.

Для захвата потока из устройств пользователя используется метод `getUserMedia` расширения `MediaDevices` интерфейса `Navigator`:

```
// ограничения или требования к потоку
// https://w3c.github.io/mediacapture-main/#constrainable-interface
// https://developer.mozilla.org/en-US/docs/Web/API/MediaTrackConstraints
const constraints = { video: true }
// поток
// https://w3c.github.io/mediacapture-main/#mediastream
// https://developer.mozilla.org/en-US/docs/Web/API/MediaStream
const stream = await navigator.mediaDevices.getUserMedia(constraints)
```

Поток состоит из треков (одного и более). Для получения треков используются методы `getTrackById`, `getTracks`, `getAudioTracks` и `getVideoTracks`. Мы будем использовать только последний:

```
// треки
// https://w3c.github.io/mediacapture-main/#mediastreamtrack
// https://developer.mozilla.org/en-US/docs/Web/API/MediaStreamTrack
const videoTracks = stream.getVideoTracks()
// мы будем извлекать первый (наиболее подходящий с точки зрения соответствия ограни
const videoTrack = stream.getVideoTracks()[0]
```

Для получения списка возможностей и настроек трека используются методы `getCapabilities` и `getSettings`, соответственно:

```
// https://w3c.github.io/mediacapture-main/#dom-mediastreamtrack-getcapabilities
const trackCapabilities = videoTrack.getCapabilities()
// https://w3c.github.io/mediacapture-main/#dom-mediastreamtrack-getsettings
const trackSettings = videoTrack.getSettings()
```

Для применения к треку (дополнительных или продвинутых) ограничений используется метод `applyConstraints`:

```
// https://w3c.github.io/mediacapture-main/#dictionary-mediastreamtrack-constraints-members
const advancedConstraints = {
  contrast: 75,
  sharpness: 75
}
// https://w3c.github.io/mediacapture-main/#dom-mediastreamtrack-applyconstraints
await applyConstraints({ advanced: [advancedConstraints] })
```

Для захвата изображений и фреймов из видеотрека используется интерфейс `ImageCapture` :

```
// https://www.w3.org/TR/image-capture/#imagecaptureapi
const imageCapture = new ImageCapture(videoTrack)
```

Для получения списка возможностей и настроек для фото используются методы `getPhotoCapabilities` и `getPhotoSettings` , соответственно:

```
// https://www.w3.org/TR/image-capture/#dom-imagecapture-getphotocapabilities
const photoCapabilities = await imageCapture.getPhotoCapabilities()
// https://www.w3.org/TR/image-capture/#dom-imagecapture-getphotosettings
const photoSettings = await imageCapture.getPhotoSettings()
```

Для получения снимка используется метод `takePhoto` . Данный метод возвращает `Blob` :

```
// https://www.w3.org/TR/image-capture/#dom-imagecapture-takephoto
// метод принимает опциональный параметр - настройки для фото
// https://www.w3.org/TR/image-capture/#photosettings-section
// https://developer.mozilla.org/en-US/docs/Web/API/ImageCapture/takePhoto#parameter
const blob = await imageCapture.takePhoto(photoSettings)
```

Для создания ссылки на `blob` используется метод `createObjectUrl` интерфейса `URL` :

```
// https://w3c.github.io/FileAPI/#dfn-createObjectURL
const src = URL.createObjectUrl(blob)
```

Для освобождения (release) ссылки на `blob` используется метод `revokeObjectUrl` :

```
// https://w3c.github.io/FileAPI/#dfn-revokeObjectURL
URL.revokeObjectUrl(src)
```

Для захвата фрейма используется метод `grabFrame`. Данный метод возвращает `ImageBitmap`:

```
// https://www.w3.org/TR/image-capture/#dom-imagecapture-grabframe
const imageBitmap = await imageCapture.grabFrame()
```

Для рендеринга `imageBitmap` на холсте используется метод `drawImage`:

```
const canvas$ = document.querySelector('canvas')
const ctx = canvas$.getContext('2d')

ctx.drawImage(imageBitmap, 0, 0)
```

Для извлечения пиксельных данных (pixel data) из холста используется метод `getImageData`. Данный метод возвращает `ImageData`:

```
// пиксели, которые мы будем инвертировать, содержатся в свойстве `imageData.data`
const imageData = ctx.getImageData(0, 0, canvas$.width, canvas$.height)
```

Наконец, для рендеринга `ImageData` на холсте используется метод `putImageData`:

```
ctx.putImageData(imageData, 0, 0)
```

Шаблон давно готов и с нетерпением ждет, когда мы приступим к разработке приложения.

Структура приложения будет следующей:

```
- index.html
- styles
- loader.css
- main.css
- scripts
- utils
  - getMedia.js
  - takePhoto.js
  - grabFrame.js
- globals.js
- main.js
- ...
```

Начнем с разметки ( index.html ):

```
<head>
  <link rel="stylesheet" href="styles/main.css" />
</head>
<body>
  <!-- индикатор загрузки -->
  <div class="lds-roller">
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
  </div>
  <!-- приложение -->
  <div id="app">
    <div>
      <!-- контейнер для потока -->
      <video autoplay muted></video>
      <!-- результат -->
      <div class="result">
        <!-- контейнер для фото -->
        <img />
        <!-- ссылка для скачивания фото -->
        <a class="photo-link">Download</a>
        <!-- контейнер для фрейма -->
        <canvas width="0" height="0"></canvas>
        <!-- ссылка для скачивания фрейма -->
        <a class="canvas-link">Download</a>
      </div>
    </div>
    <!-- настройки -->
    <form class="settings">
      <!-- кнопки для -->
      <div class="controls">
        <!-- получения снимка -->
        <button class="take-photo">Take photo</button>
        <!-- удаления снимка -->
        <button class="remove-photo" type="button">Remove photo</button>
        <!-- получения фрейма -->
        <button class="grab-frame" type="button">Grab frame</button>
        <!-- очистки холста -->
        <button class="clear-canvas" type="button">Clear canvas</button>
      </div>
    </form>
  </div>
```

```
<!-- скрипты -->
<!-- глобальные утилиты и переменные -->
<script src="scripts/globals.js"></script>
<!-- основной модуль -->
<script type="module" src="scripts/main.js"></script>
</body>
```

Индикатор загрузки вместе со стилями ( styles/loader.css ) я взял [отсюда](#).

Добавим немного красоты ( styles/main.css ):

```
@import url('https://fonts.googleapis.com/css2?family=Montserrat:wght@200;400;600&di
@import './loader.css';

*,
*::before,
*::after {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: 'Montserrat', sans-serif;
  font-size: 1rem;
  color: #292b2c;
}

:root {
  --primary: #0275d8;
}

body {
  display: flex;
  justify-content: center;
}

#app {
  padding: 1rem;
  display: none;
}

video,
img,
canvas {
  max-width: 320px;
  border-radius: 4px;
}

.settings {
  padding: 0 1rem;
}
```

```
.field {
  margin-bottom: 0.75rem;
  flex-grow: 1;
  display: grid;
  align-items: center;
  grid-template-columns: repeat(3, 1fr);
  grid-gap: 1rem;
}

.controls {
  margin-bottom: 1rem;
  display: flex;
}

button {
  padding: 0.3rem 0.5rem;
  background: var(--primary);
  border: none;
  border-radius: 4px;
  outline: none;
  color: #f7f7f7;
  font-size: 0.88rem;
  box-shadow: 0 1px 2px rgba(0, 0, 0, 0.4);
  transition: 0.3s;
  cursor: pointer;
  user-select: none;
}

button:active {
  box-shadow: none;
}

button[disabled] {
  filter: grayscale(50);
  opacity: 0.8;
  cursor: not-allowed;
}

.result {
  display: flex;
  flex-direction: column;
  align-items: center;
}

a {
  margin: 0.5rem 0;
  text-decoration: none;
  font-size: 0.8rem;
  display: none;
}
```



```
a[href] {  
  color: var(--primary);  
  border-bottom: 1px dashed var(--primary);  
}
```

Переходим к скриптам.

---

Начнем с определения глобальных утилит и переменных ( `scripts/globals.js` ).

Определяем утилиту для получения ссылок на элементы:

```
// привет, jQuery  
const $ = (selector, parent = document) => parent.querySelector(selector)
```

Определяем утилиту для создания элементов. Мы могли бы сделать это так:

```
// привет, React  
const create$ = (tag, attrs, children) => {  
  if (typeof tag !== 'string') return  
  const $ = document.createElement(tag)  
  
  if (attrs) {  
    if (typeof attrs !== 'object') return  
  
    for (const k in attrs) {  
      const v = attrs[k]  
  
      if (k === 'class') {  
        $.className = v  
        continue  
      }  
  
      if (k === 'style' && typeof v === 'object') {  
        Object.assign($.style, v)  
        continue  
      }  
  
      if (k === 'text') {  
        $.textContent = v  
        continue  
      }  
  
      $[k] = v  
    }  
  }  
}
```

```
}

if (Array.isArray(children) && children.length > 0) {
  children.forEach(([tag, attrs, children]) => {
    $.append(create$(tag, attrs, children))
  })
}

return $
}
```

Но лучше сделаем так:

```
const create$ = (template) => {
  if (typeof template !== 'string') return
  return new Range().createContextualFragment(template).children[0]
}
```

Определяем функцию для рендеринга элементов:

```
const render$ = (parent, child, place = 'beforeend') => {
  parent.insertAdjacentElement(place, child)
}
```

Мы будем довольно много манипулировать атрибутами элементов, поэтому имеет смысл определить для этого соответствующие утилиты:

```
// добавляем атрибуты
// `attrs` - объект
// ключи объекта - названия атрибутов
// значения ключей - значения атрибутов
const setAttrs = ($, attrs) => {
  if (attrs && (typeof attrs !== 'object' || Array.isArray(attrs))) return

  Object.keys(attrs).forEach((key) => {
    $.setAttribute(key, attrs[key])
  })
}

// удаляем атрибуты
// `attrs` - массив
// элементы массива - названия атрибутов
const removeAttrs = ($, attrs) => {
  if (!Array.isArray(attrs)) return
```

```
attrs.forEach((name) => {  
  $.removeAttribute(name)  
})  
}
```

Получаем ссылки на DOM-элементы:

```
const loader$ = $('.lds-roller')  
const app$ = $('#app')  
const video$ = $('.video')  
const image$ = $('.result img')  
const photoLink$ = $('.photo-link')  
const canvas$ = $('.result canvas')  
const canvasLink$ = $('.canvas-link')  
const controls$ = $('.controls')  
const grabFrame$ = $('.grab-frame')  
const removePhoto$ = $('.remove-photo')  
const clearCanvas$ = $('.clear-canvas')  
const settings$ = $('.settings')
```

Создаем еще несколько глобальных переменных:

```
// хранилище для инпутов  
const inputs$ = []  
// контекст рисования  
const ctx = canvas$.getContext('2d')  
// видеотрек, экземпляр `ImageCapture` и источник фото  
let videoTrack  
let imageCapture  
let photoSrc
```

И определяем (продвинутые) настройки для трека:

```
const settingDictionary = {  
  brightness: 'Яркость',  
  colorTemperature: 'Цветовая температура',  
  contrast: 'Контрастность',  
  saturation: 'Насыщенность',  
  sharpness: 'Резкость',  
  pan: 'Панорамирование',  
  tilt: 'Наклон',  
  zoom: 'Масштаб'  
}
```

Мы берем настройки, которые можно использовать в диапазонных полях. С полным списком настроек для трека можно ознакомиться [здесь](#).

Рассмотрим утилиту для захвата видеопотока и подготовки инпутов ( `scripts/utils/getMedia.js` ).

Определяем дефолтные ограничения потока:

```
const defaultConstraints = {
  // эти ограничения должны быть определены явно
  video: {
    pan: true,
    tilt: true,
    zoom: true
  }
}

async function getMedia(
  constraints = defaultConstraints
) {
  try {
    // TODO
  } catch (e) {
    console.error(e)
  }
}

export default getMedia
```

Захватываем видеопоток, передаем его в элемент `video` с помощью `srcObject`, извлекаем видеотрек и создаем экземпляр `ImageCapture`:

```
const stream = await navigator.mediaDevices.getUserMedia(constraints)
video$.srcObject = stream
// глобальные переменные
videoTrack = stream.getVideoTracks()[0]
imageCapture = new ImageCapture(videoTrack)
```

Получаем списки возможностей и настроек фото и трека:

```
const photoCapabilities = await imageCapture.getPhotoCapabilities()
console.log('*** Photo capabilities', photoCapabilities)
```

```
const photoSettings = await imageCapture.getPhotoSettings()
console.log('*** Photo settings', photoSettings)

const trackCapabilities = videoTrack.getCapabilities()
console.log('*** Track capabilities', trackCapabilities)

const trackSettings = videoTrack.getSettings()
console.log('*** Track settings', trackSettings)
```

Перебираем ключи "словаря" с настройками:

```
Object.keys(settingDictionary).forEach((key) => {
  // TODO
})
```

Если трек поддерживает настройку:

```
if (key in trackSettings) {
  // TODO
}
```

Далее мы делаем следующее:

- формируем шаблон поля для установки конкретной настройки с помощью возможностей и настроек трека — `fieldTemplate`;
- создаем элемент поля — `field$`;
- извлекаем из него инпут и контейнер для вывода текущего значения инпута — `input$` и `info$`, соответственно;
- регистрируем обработчик изменения значения инпута — `oninput`;
- помещаем инпут в глобальный массив инпутов — `inputs$`;
- рендерим элемент поля.

```
const fieldTemplate = `
<div class="field">
  <label for="${key}">
    ${settingDictionary[key]}
  </label>
  <input
    type="range"
    id=${key}
    name=${key}
```

```
    min=${trackCapabilities[key].min}
    max=${trackCapabilities[key].max}
    step=${trackCapabilities[key].step}
    value=${trackSettings[key]}
  />
  <p>${trackSettings[key]}</p>
</div>
`

const field$ = create$(fieldTemplate)
// первый элемент - это `label`, который нас не интересует
const [, input$, info$] = field$.children

input$.oninput = ({ target: { value } }) => {
  info$.textContent = value
}

inputs$.push(input$)

render$(settings$, field$)
```

Наконец, скрываем индикатор загрузки и отображаем приложение:

```
loader$.style.display = 'none'
app$.style.display = 'flex'
```

#### ► Полный код этой утилиты:

Рассмотрим утилиту для получения снимка ( `scripts/utls/takePhoto.js` ).

Определяем дефолтные настройки для фото:

```
const defaultSettings = {
  imageHeight: 480,
  imageWidth: 640
}

async function takePhoto(settings = defaultSettings) {
  // TODO
}
```

Применяем настройки, установленные пользователем:

```
const advancedConstraints = {}
for (const { name, value } of inputs$) {
  advancedConstraints[name] = value
}

try {
  await videoTrack.applyConstraints({
    advanced: [advancedConstraints]
  })
} catch (e) {
  console.error(e)
}
```

Получаем blob с помощью метода `takePhoto`, формируем на него ссылку и передаем ее в элементы `img` и `a`:

```
try {
  const blob = await imageCapture.takePhoto(settings)
  // глобальная переменная
  photoSrc = URL.createObjectURL(blob)

  image$.src = photoSrc
  setAttrs(photoLink$, {
    href: photoSrc,
    download: `my-photo-${Date.now()}.png`,
    style: 'display: block;'
  })
} catch (e) {
  console.error(e)
}
```

---

Рассмотрим утилиту для захвата фрейма ( `scripts/utils/grabFrame.js` ):

```
async function grabFrame() {
  try {
    // TODO
  } catch(e) {
    console.error(e)
  }
}
```

Получаем `ImageBitmap` с помощью метода `grabPhoto`, определяем ширину и высоту холста и рендерим фрейм:

```
const imageBitmap = await imageCapture.grabFrame()

canvas$.width = imageBitmap.width
canvas$.height = imageBitmap.height

// рендерим фрейм
ctx.drawImage(imageBitmap, 0, 0)
```

Читаем пиксели и инвертируем цвета:

```
const imageData = ctx.getImageData(0, 0, canvas$.width, canvas$.height)
const { data } = imageData
for (let i = 0; i < data.length; i += 4) {
  data[i] ^= 255 // красный
  data[i + 1] ^= 255 // зеленый
  data[i + 2] ^= 255 // синий
}
```

Перерисовываем изображение и генерируем ссылку для скачивания фрейма:

```
ctx.putImageData(imageData, 0, 0)

setAttrs(canvasLink$, {
  href: canvas$.toDataURL(),
  download: `my-frame-${Date.now()}.png`,
  style: 'display: block;'
})
```

Для формирования ссылки на фрейм мы используем метод `toDataURL`, возвращающий URL с префиксом `data:`, который позволяет встраивать небольшие файлы прямо в документ, например, изображение, преобразованное в строку `base64`.

Осталось разобраться с основным модулем приложения (`scripts/main.js`).

Импортируем утилиты и запускаем захват медиа:

```
import getMedia from './utils/getMedia'
import takePhoto from './utils/takePhoto'
import grabFrame from './utils/grabFrame'
```



```
getMedia()
```

Регистрируем обработчик отправки формы — применения пользовательских настроек и получение снимка:

```
settings$.onsubmit = (e) => {  
  e.preventDefault()  
  // получаем снимок  
  takePhoto()  
  // снимаем блокировку с соответствующей кнопки  
  removePhoto$.disabled = false  
}
```

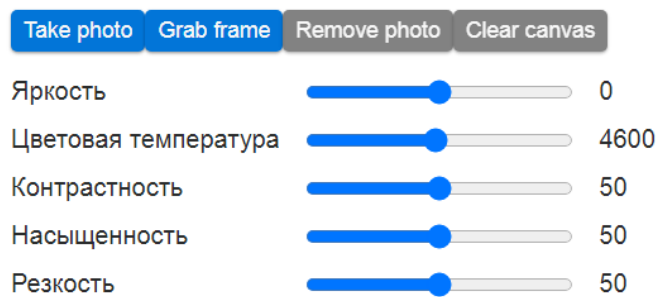
Регистрируем обработчик нажатия кнопок:

```
controls$.onclick = ({ target }) => {  
  // нас интересует только клик по кнопке  
  if (target.localName !== 'button') return  
  // выполняемая операция зависит от цели клика  
  switch (target) {  
    // цель клика - кнопка для удаления фото  
    case removePhoto$:  
      // освобождаем ссылку на `blob`  
      URL.revokeObjectURL(photoSrc)  
      // удаляем источник фото  
      image$.removeAttribute('src')  
      // удаляем ссылку на фото  
      removeAttrs(photoLink$, ['href', 'download', 'style'])  
      // блокируем соответствующую кнопку  
      removePhoto$.disabled = true  
      break  
    // цель клика - кнопка для захвата фрейма  
    case grabFrame$:  
      // захватываем фрейм  
      grabFrame()  
      // снимаем блокировку с соответствующей кнопки  
      clearCanvas$.disabled = false  
      break  
    // цель клика - кнопка для очистки холста  
    case clearCanvas$:  
      // очищаем холст  
      ctx.clearRect(0, 0, canvas$.width, canvas$.height)  
      // сжимаем холст  
      setAttrs(canvas$, { width: 0, height: 0 })  
      // удаляем ссылку на фрейм  
      removeAttrs(canvasLink$, ['href', 'download', 'style'])  
      // блокируем соответствующую кнопку
```

```
clearCanvas$.disabled = true
break
default:
return
}
}
```

Протестируем работоспособность нашего приложения.

Запускаем сервер для разработки с помощью команды `yarn dev` или `npm run dev` и открываем вкладку браузера по адресу `http://localhost:3000` :



Нажимаем на кнопку `Take photo` :



Take photo	Grab frame	Remove photo	Clear canvas
Яркость	<input type="range"/>	0	
Цветовая температура	<input type="range"/>	4600	
Контрастность	<input type="range"/>	50	
Насыщенность	<input type="range"/>	50	
Резкость	<input type="range"/>	50	

[Download](#)

Получаем селфи и ссылку для его скачивания.

Удаляем фото, нажав на кнопку `Remove photo`.

Меняем настройки и снова нажимаем `Take photo`:



Take photo Grab frame Remove photo Clear canvas

Яркость	<input type="range"/>	-20
Цветовая температура	<input type="range"/>	5300
Контрастность	<input type="range"/>	70
Насыщенность	<input type="range"/>	30
Резкость	<input type="range"/>	60

[Download](#)

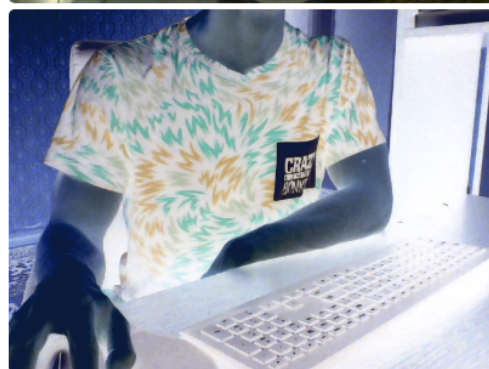
*Обратите внимание:* применение к треку продвинутых настроек повлияло также на видео, поскольку речь идет об одном и том же треке.

Удаляем фото, возвращаем настройки в исходное состояние и нажимаем на кнопку Grab frame :



Take photo Grab frame Remove photo Clear canvas

Яркость	<input type="range"/>	0
Цветовая температура	<input type="range"/>	4600
Контрастность	<input type="range"/>	50
Насыщенность	<input type="range"/>	50
Резкость	<input type="range"/>	50

[Download](#)

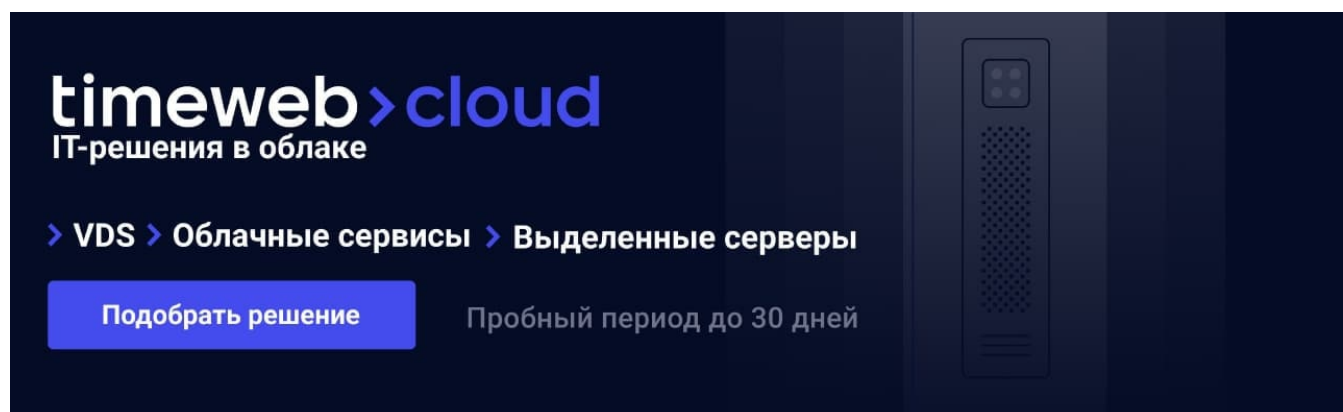
Получаем инвертированное изображение и ссылку для его скачивания.

Очищаем холст, нажав на кнопку `Clear canvas`.

Отлично! Наше приложение работает, как ожидается.

Пожалуй, это все, чем я хотел поделиться с вами в данной статье.

Благодарю за внимание и happy coding!



**Теги:** javascript, image capture api, takephoto, grabframe, mediastream, videotrack, photo, frame, захват изображения, получение снимка, захват фрейма, медиапоток, видеотрек

**Хабы:** Блог компании Timeweb Cloud, Разработка веб-сайтов, JavaScript

◆ +10

👁 5.8K

🔖 61



## Редакторский дайджест

Присылаем лучшие статьи раз в месяц



Электронная почта



Timeweb Cloud

Облачная платформа для разработчиков и бизнеса



120

Карма

51.6

Рейтинг

Igor Agapov @aio350

JavaScript Developer

## Комментарии



Здесь пока нет ни одного комментария, вы можете стать первым!

Только полноправные пользователи могут оставлять комментарии. [Войдите](#), пожалуйста.

## ПОХОЖИЕ ПУБЛИКАЦИИ

## Ваш аккаунт

Войти  
Регистрация

## Разделы

Публикации  
Новости  
Хабы  
Компании  
Авторы  
Песочница

## Информация

Устройство сайта  
Для авторов  
Для компаний  
Документы  
Соглашение  
Конфиденциальность

## Услуги

Корпоративный блог  
Медийная реклама  
Нативные проекты  
Образовательные  
программы  
Мегапроекты



Настройка языка

Техническая поддержка

Вернуться на старую версию

© 2006–2022, Habr

♦ +31

👁 4.1K

📖 14

💬 10 +10

вчера в 21:20

## Личный опыт выгорания

♦ +26

👁 8.3K

📖 31

💬 25 +25

---

вчера в 17:00

### DIY квантовые вычисления: как я начал собирать квантовые схемы

 +20

 8.4K

 42

 25 +25

---

вчера в 21:00

### Антиматерия и бариогенезис. Три причины, почему нет антивещества, но есть мы

 +15

 2.9K

 21

 1 +1

































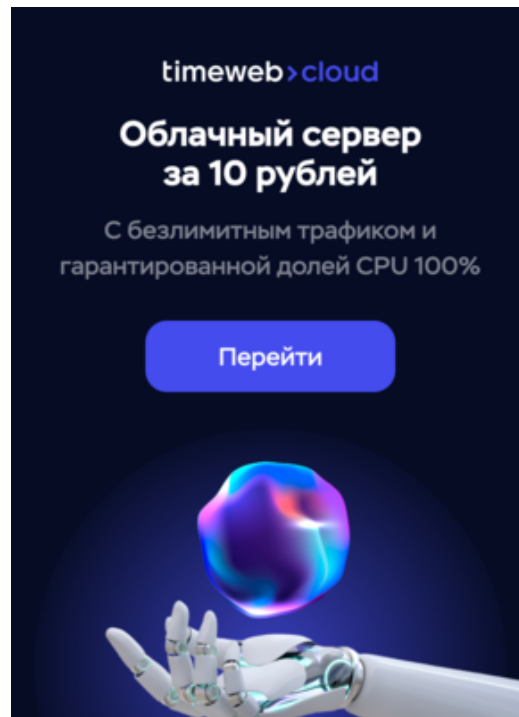


## ИНФОРМАЦИЯ

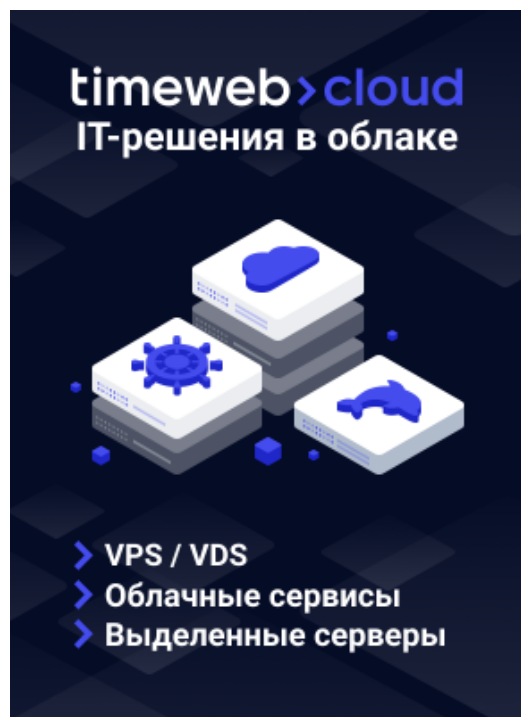
---

Дата основания	25 мая 2006
Местоположение	Россия
Сайт	<a href="https://cloud.timeweb.com">cloud.timeweb.com</a>
Численность	201–500 человек
Дата регистрации	11 августа 2011

## ВИДЖЕТ



#### ВИДЖЕТ



#### ССЫЛКИ

Серверы под любые задачи  
[cloud.timeweb.com](https://cloud.timeweb.com)

Облачные базы данных (DBaaS)  
[cloud.timeweb.com](https://cloud.timeweb.com)

Объектное S3-хранилище  
[cloud.timeweb.com](https://cloud.timeweb.com)



[cloud.timeweb.com](https://cloud.timeweb.com)

Партнерская программа

[timeweb.com](https://timeweb.com)

Timeweb News — актуальные новости и скидки

[t.me](https://t.me)

«Релиз в пятницу» — подкаст от команды Timeweb Cloud

[www.youtube.com](https://www.youtube.com)

Craftum — конструктор сайтов

[craftum.com](https://craftum.com)

## НОВОСТИ

---

Развертывание приложений Python с помощью Gunicorn

30 июня

Копирование файлов по SSH

29 июня

Обзор ZeroTier One: работа с программно-конфигурируемой сетью и создание VPN

27 июня

Как установить Docker на Ubuntu

27 июня

MySQL: для чего нужна, как устроена, основные преимущества и недостатки

24 июня

RDP-протокол: что это такое, для чего используется и как работает

24 июня

Установка NextCloud на Ubuntu 20.04

24 июня

Проекты и новая база знаний

23 июня

Установка NextCloud на Ubuntu 20.04

23 июня

Как настроить Laravel, Nginx и MySQL с помощью Docker Compose

22 июня

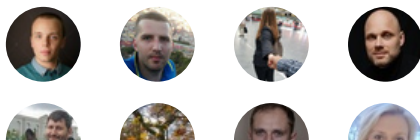
## В КОНТАКТЕ

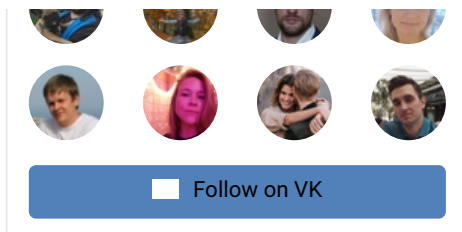
---



Timeweb: всё про хостинг, IT ...

54,856 followers





## ПРИЛОЖЕНИЯ

---



### Приложение для VDS Evo

Управляйте VDS Evo со своего мобильного в удобное для вас время.

Информация о работе ваших VDS и консультация со службой поддержки теперь доступны на мобильном устройстве.

[Android](#) [iOS](#)



### Приложение для хостинга

Управляйте виртуальным хостингом прямо со смартфона. Все данные о хостинге и возможность связаться со Службой поддержки внутри одного приложения для Android и iOS.



[Android](#) [iOS](#)

## БЛОГ НА ХАБРЕ

---



2 июля в 16:06

### Непромокаемый компьютер из 1960 года

 7.2K  16 **+16**

1 июля в 19:52

### Как турецкий муниципальный район перешёл на Linux

 4.9K  18 **+18**

30 июня в 18:33

### Разрабатываем десктопное приложение для заметок с помощью Tauri (React + Rust)

 2.4K  3 **+3**

29 июня в 17:36

### Первый высокопроизводительный пластиковый процессор стоимостью в 1 цент

 32K  104 **+104**

28 июня в 19:40

### Cisco UCS C220 (Fabric Interconnection 6842) + Dell EMC VNX 5300

 1.6K  2 **+2**