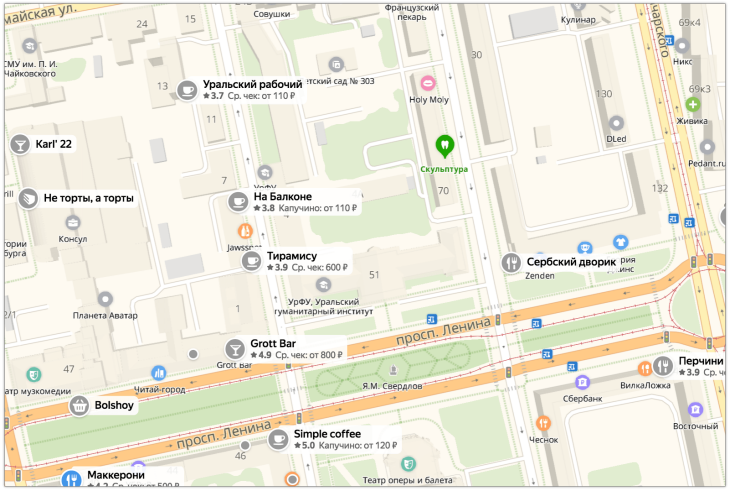
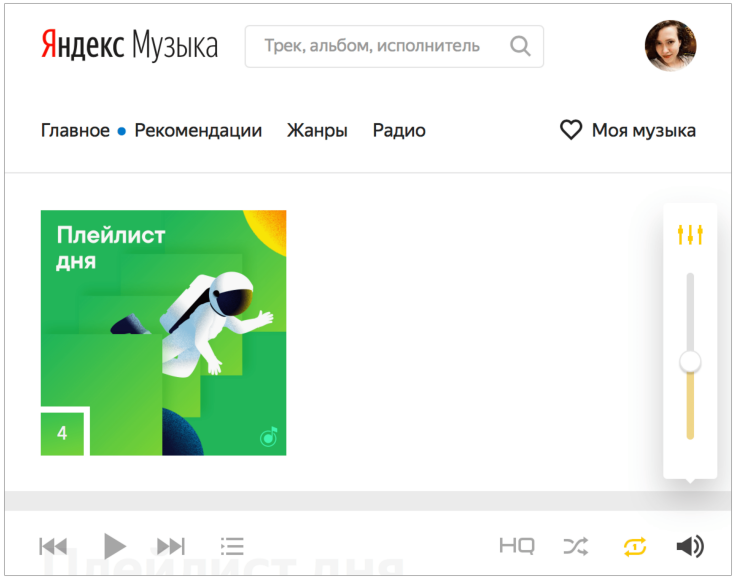
# Web Api

Cookies, Storages, IndexedDB, History APi, PaymentRequest Api

Кристина Ильенко

В данной лекции обсуждается браузерное API


майская ул.
Совушки
Французский пекарь
Кулинар
Уральский рабочий
★3.7 Ср. чек: от 110 ₽
Hoiy Moiy
Karl' 22
Скульптура
Живика
DLed
Pedant.ru
Не торты, а торты
На Балконе
★3.8 Капучино: от 110 ₽
Консул
Тирамису
★3.9 Ср. чек: 600 ₽
Сербский дворик
Zenden
УрФУ, Уральский гуманитарный институт
Планета Аватар
Grott Bar
★4.9 Ср. чек: от 800 ₽
просп. Ленина
Читай-город
Я.М. Свердлов
Сбербанк
ВилкаЛожка
Восточный
Чеснок
Bolshoy
Simple coffee
★5.0 Капучино: от 120 ₽
Театр оперы и балета
Маккерони


Яндекс Музыка
Трек, альбом, исполнитель
Главное
Рекомендации
Жанры
Радио
Моя музыка
Плейлист дня
4
HQ


COOL STORY BRO
NOW, GIVE ME COOKIE.

# Создание

```
document.cookie = 'subject=Song';

document.cookie = 'subject=Beautiful-song';

document.cookie = 'volume=5';

document.cookie = 'subject=' + encodeURIComponent('Песня');
// %D0%9F%D0%B5%D1%81%D0%BD%D1%8F
```

uniq key = name + path + domain

```
document.cookie = 'subject=Song; path=/users; domain=yandex.ru';
```

path=/

domain=music.yandex.ru;

path=/users/playlists;

path=/usersplaylists; domain=yandex.ru;

domain=facebook.com;

# expires

```
subject=Song; expires=Tue, 19 Apr 2019 00:00:00 GMT
```

Чтобы удалить, устанавливаем дату устаревания в прошлом

```
subject=Song; expires=Tue, 19 Apr 1970 00:00:00 GMT
```

# Чтение

```
document.cookie = 'subject=Song; path=/';

console.log(document.cookie);
// subject=Song
```

# Чтение

```
document.cookie = 'subject=Song; path=/';
document.cookie = 'subject=Song; path=/users';

console.log(document.cookie);
```

Раньше будет выведена cookie с заданным path=/users или с path=/?

Если подходят несколько – доступны все в порядке от наиболее специфичной к наименее

# Чтение

```
document.cookie = 'subject=Song; path=/';
document.cookie = 'subject=Song; path=/users';

console.log(document.cookie);

subject=Song;
subject=Song;
```

# js-cookie

```
Cookies.set('subject', 'Song', { expires: 7, path: '' });

Cookies.get('subject');

Cookies.remove('subject');
```

# Отправка на сервер

```
GET / HTTP/1.1
        Host: music.yandex.ru
        Cookie: subject=Song; playlist=1
```

```
const express = require('express')
const app = express();
const cookieParser = require('cookie-parser');

app.use(cookieParser());

app.use((req, res) => {
    console.log(req.cookies);
    // { subject: 'Song' }
});
```

# Отправка на клиент

```
var express = require('express')
var app = express();

app.use((req, res) => {
    res.cookie('subject', 'Song', { path: '/users' });
});
```

```
HTTP/1.1 200 OK
Set-Cookie: subject=Song; path=/users
```

# HTTP-only

```
res.cookie('subject', 'Song', {
    path: '/users',

    httpOnly: true
});

HTTP/1.1 200 OK
Set-Cookie: subject=Song; path=/users; HttpOnly
```

Не доступны в js-скриптах на клиенте

# Secure

```
res.cookie('subject', 'Song', {
    path: '/users',

    secure: true
});

HTTP/1.1 200 OK
Set-Cookie: subject=Song; path=/users; secure
```

Не доступны по HTTP

# Устаревание

# Доступ с сервера

4Kb

Передаются с каждым запросом в заголовке, который не сжимается

Для статики используйте
cookieless домены (CDN)

В cookie храните id, по которому можно на сервере получить полные данные

Кодируйте 01100101

# Using the Same-Site Cookie Attribute to Prevent CSRF Attacks

HTTP cookie

# WebStorage

SessionStorage – хранит данные до окончания сессии (до закрытия вкладки/окна)

LocalStorage – хранит данные перманентно, пока скрипт или пользователь не удалит их

Не передаёт данные на сервер

Ограничение в 10Mb

# Интерфейс

```
localStorage.setItem('volume', 8);

localStorage.getItem('volume'); // "8"

localStorage.removeItem('volume')

localStorage.clear();

localStorage.setItem('repeat', 1);
// QUOTA_EXCEEDED_ERROR
```

# Хранит строки, а не объекты

```
localStorage.setItem('options', { volume: 8 });


localStorage.getItem('options'); // "[object Object]"

localStorage.setItem('options', JSON.stringify({ volume: 8 }));
```

# Событие

```
window.addEventListener('storage', event => {
    console.log(event);
});

{
    key: 'volume',
    oldValue: '8',
    newValue: '6'
}
```

# Приватный режим

```
try {
    localStorage.setItem('options', '8');
} catch (error) {
    console.error(error);
    // SecurityError: The operation is insecure
}
```

Хранение настроек

Хранение промежуточных данных

Строго ограничено источником (origin)

# Синхронный интерфейс

Web Storage API

Html5 Local Storage How-To

Client-side Data Storage

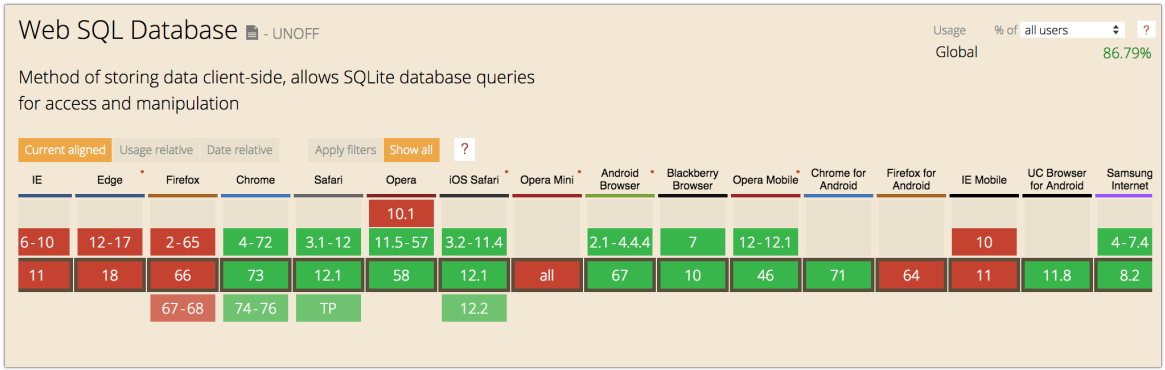
# WebSQL

# Асинхронный интерфейс к SQLite базе

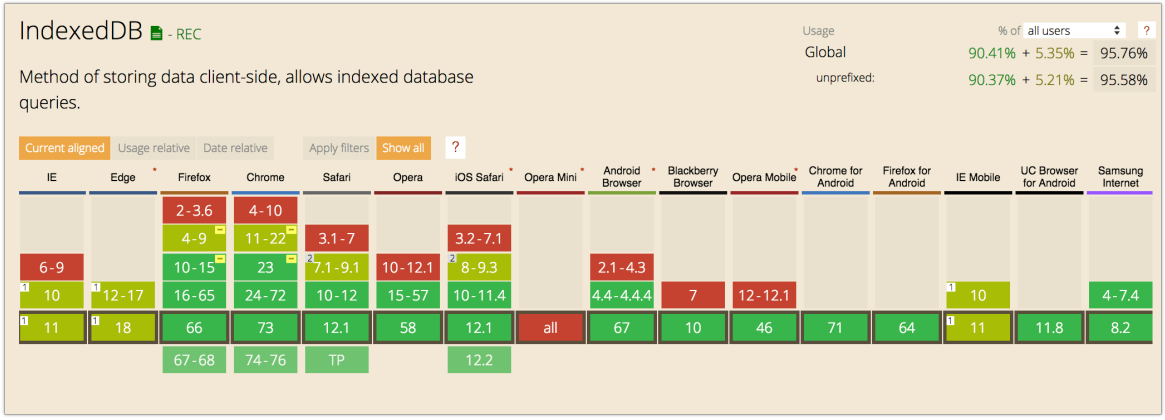
```
const db = openDatabase('my-app', '1.0', null, 1024 * 1024);

db.transaction(tr => {
    tr.executeSql(`
        create table if not exists notes(
            name TEXT
        )
    `);

    tr.executeSql(`
        insert into notes(name)
        values("films")
    `);
}, console.error);
```


Web SQL Database - UNOFF
Usage
% of all users
Global
86.79%
Method of storing data client-side, allows SQLite database queries for access and manipulation
Current aligned
Usage relative
Date relative
Apply filters
Show all
IE
Edge
Firefox
Chrome
Safari
Opera
iOS Safari
Opera Mini
Android Browser
Blackberry Browser
Opera Mobile
Chrome for Android
Firefox for Android
IE Mobile
UC Browser for Android
Samsung Internet
10.1
6-10
12-17
2-65
4-72
3.1-12
11.5-57
3.2-11.4
2.1-4.4.4
7
12-12.1
10
4-7.4
11
18
66
73
12.1
58
12.1
all
67
10
46
71
64
11
11.8
8.2
67-68
74-76
TP
12.2

# IndexedDB


IndexedDB - REC
Method of storing data client-side, allows indexed database queries.
Usage
% of all users
Global
90.41% + 5.35% = 95.76%
unprefixed:
90.37% + 5.21% = 95.58%
Current aligned
Usage relative
Date relative
Apply filters
Show all
IE
Edge
Firefox
Chrome
Safari
Opera
iOS Safari
Opera Mini
Android Browser
Blackberry Browser
Opera Mobile
Chrome for Android
Firefox for Android
IE Mobile
UC Browser for Android
Samsung Internet
6-9
10
11
12-17
18
2-3.6
4-9
10-15
16-65
66
67-68
4-10
11-22
23
24-72
73
74-76
3.1-7
7.1-9.1
10-12
12.1
TP
10-12.1
15-57
58
3.2-7.1
8-9.3
10-11.4
12.1
12.2
all
2.1-4.3
4.4-4.4.4
67
7
10
12-12.1
46
71
64
10
11
11.8
4-7.4
8.2

Строго ограничено источником (origin)

Нет ограничений на размер*

Асинхронное

Не реляционное, а object storage

He SQL, a API

Хранилище объектов всегда сортирует значения по ключам внутри

Поэтому запросы, возвращающие много значений, всегда возвращают их в отсортированном порядке

# Создание

```
// Указываем название и версию
const requestDb = indexedDB.open('my-app', 1);
// Метод open возвращает объект IDBOpenDBRequest
// с тремя обработчиками:
// onerror, onsuccess, onupgradeneeded
requestDb.onerror = event => {
    console.log(event.target.errorCode);
};

requestDb.onsuccess = event => {
    // Экземпляр открытой базы напрямую не отдает
    // Получаем доступ к базе
    const db = event.target.result;
};
```

# Обновление

```
// Будет вызван в первый раз, и когда сменилась версия
requestDb.onupgradeneeded = event => {
    const db = event.target.result;
    const oldVersion = event.oldVersion;
    // Инструкции к миграции на вторую версию
    if (oldVersion < 2) {
        // Создаём хранилище для заметок
        // IndexedDB оперирует не таблицами,
        // а хранилищами объектов: ObjectStore
        db.createObjectStore('notes', {
            keyPath: 'id', // Имя ключевого поля
            autoIncrement: true
```

Можем добавлять/удалять/обновлять данные вне обработчика onupgradeneeded

Можем создавать/измененять хранилище объектов только при обновлении версии БД внутри обработчика onupgradeneeded

# Любые операции с данными в IndexedDB происходят в рамках транзакции

Это обеспечивает целостность базы данных (в случае сбоя операции транзакция откатывается)

# Добавление объекта

```
// Открываем транзакцию
// Указываем, к каким Store будет иметь доступ транзакция
const transaction = db.transaction('notes', 'readwrite');
// В рамках транзакции получаем ссылку на объект Store
const store = transaction.objectStore('notes');
// Выполняем запрос в хранилище
// Добавляем заметку
const request = store.add({
    id: 'films',
    name: 'films'
});
```

Транзакция остаётся “живой“, если есть активные запросы к концу event loop цикла

Может быть несколько паралелльных readonly транзакций, но только одна readwrite

readwrite транзакция “блокирует” хранилище для записи

# Получение объекта

```
// Открываем транзакцию
// Указываем, к каким Store будет иметь доступ транзакция
const transaction = db.transaction(['notes'], 'readonly');
// В рамках транзакции получаем ссылку на объект Store
const store = transaction.objectStore('notes')
// Выполняем запрос в хранилище
// Получаем данные, используя значения ключа (id)
const request = store.get('films')

request.onsuccess = note => console.log(note)
```

Метод get удобно использовать, если знаем ключ, по которому хотим получить данные

А если хотим пройти через все записи в ObjectStore?

## Можно воспользоваться курсором

Курсор – особый объект, который пересекает ObjectStore с заданным запросом и возвращает по одному ключу/значению за раз, таким образом экономя память

# Чтобы получить все объекты, используем курсор

```
const requestCursor = db.transaction(['notes'], 'readonly')
    .objectStore('notes').openCursor();
requestCursor.onsuccess = event => {
    const cursor = event.target.result;
    if (cursor) {
        console.log(cursor.key, cursor.value);
        cursor.continue();
    } else {
        console.log("No more notes");
    }
};
```

Основное отличие курсора в том, что request.onsuccess срабатывает несколько раз: по одному разу для каждого результата

Что использовать, если хотим искать по условию?

Например, нужно найти Заметки с именем Films?

## Использовать индексы

Индекс – это “надстройка“ к хранилищу, отслеживающая заданное поле объекта.

Для каждого значения этого поля хранится список ключей для объектов, имеющих это значение.

# Индексы

```
const db = event.target.result;

const store = db.createObjectStore('notes', {
    keyPath: 'id',
    autoIncrement: true
});

store.createIndex(
    'nameIdx',          // Название
    'name',             // Поле или массив ['name', 'keywords'],
                    //     по которому будем искать
```

# Поиск по условию

```
const transaction = db.transaction(['notes'], 'readonly');
const store = transaction.objectStore('notes')

const requestCursor = store
    .index('nameIdx')
    .openCursor(IDBKeyRange.only(['films', true]));

// ...
```

# dexie.org

```
const db = new Dexie('MyDatabase');


db
    .version(1)
    .stores({
        notes: 'name'
    });

db
    .open()
    .catch(error => console.error(error));
```

# dexie.org

```
db
    .notes

    .where('name')
    .equals(['films'])
    .each(note => {
        console.log(note.name);
    });
```

# Using IndexedDB

SPA

Приложение быстро загружается и работает

Необходимые приложению статические ресурсы подгружаются один раз, и затем контент формируется динамически без перезагрузки страницы

Не требует написания серверной части

Предоставляет пользовательский опыт близкий к нативным приложениям

Тяжело делать SEO оптимизацию

Изначально технология предоставляла возможность только менять содержимое страницы без её перезагрузки

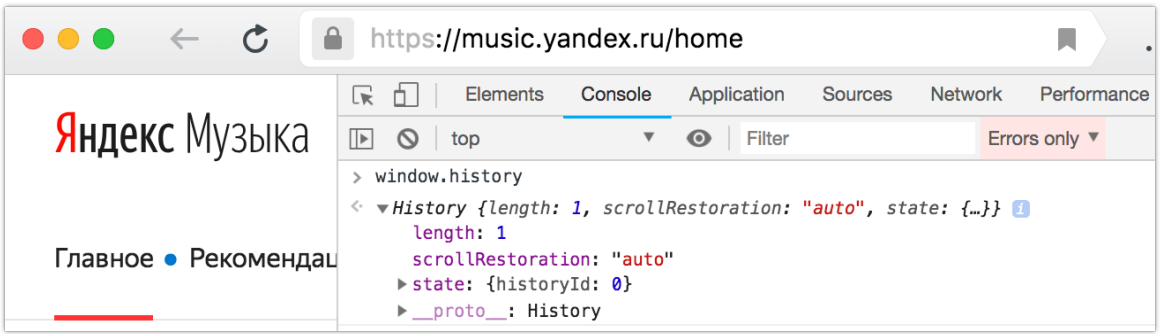
Разработчики научились загружать отдельные страницы с помощью JavaScript так, чтобы одновременно:

- менялось содержание
- генерировался новый адрес страницы

# History API

History API опирается на один DOM

интерфейс — объект History. Он доступен через window.history



# Основные методы и свойства объекта History

window.history.length

window.history.state

window.history.go(n)

window.history.back()

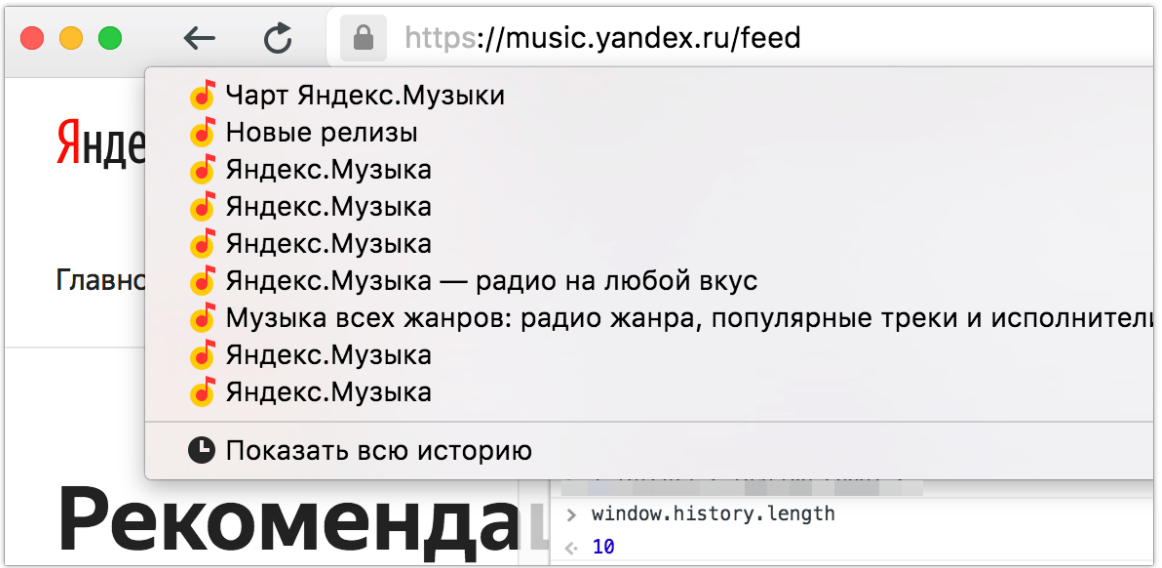
window.history.forward()

window.history.pushState(data, title [, url])

window.history.replaceState(data, title [, url])

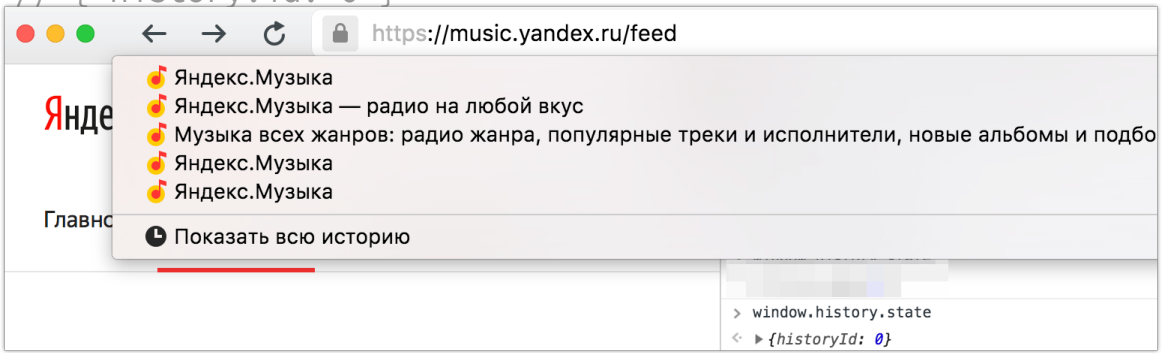
window.history.length – свойство length хранит количество записей в текущей сессии истории

```
window.history.length
// 10
```



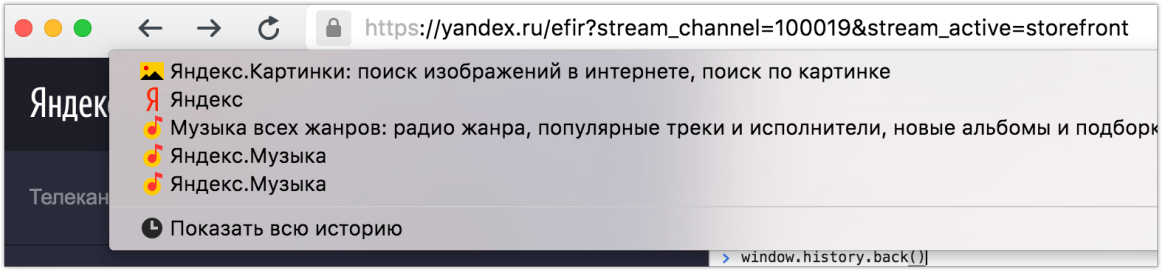
window.history.state – свойство state хранит текущий объект истории

```
window.history.state
// { history.id: 0 }
```



window.history.go(n) – метод, позволяющий гулять по истории. В качестве аргумента передается смещение относительно текущей позиции

window.history.back() – метод, позволяющий гулять по истории. Идентичен вызову window.history.go(-1)




```
window.history.back()
```

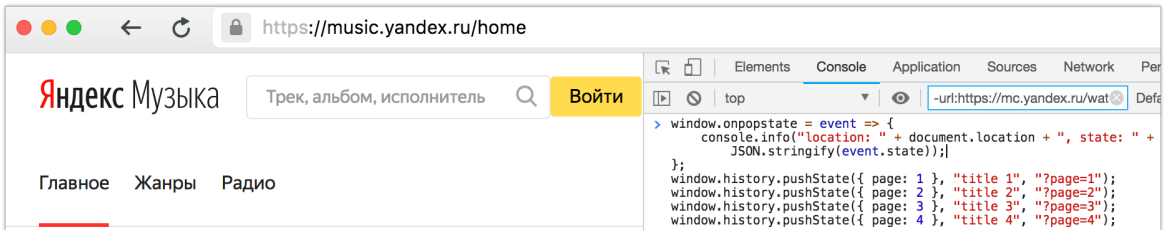
window.history.forward() – метод, позволяющий гулять по истории. Идентичен вызову window.history.go(1)

window.history.pushState(state, title [, url]) – метод, добавляющий новое состояние в историю браузера

state – любой валидный тип в JavaScript, который можно сериализовать

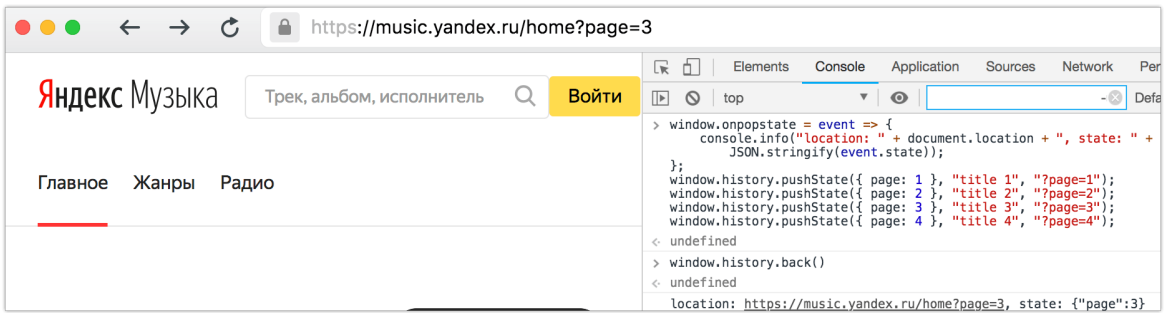
title – все современные браузеры игнорируют этот параметр

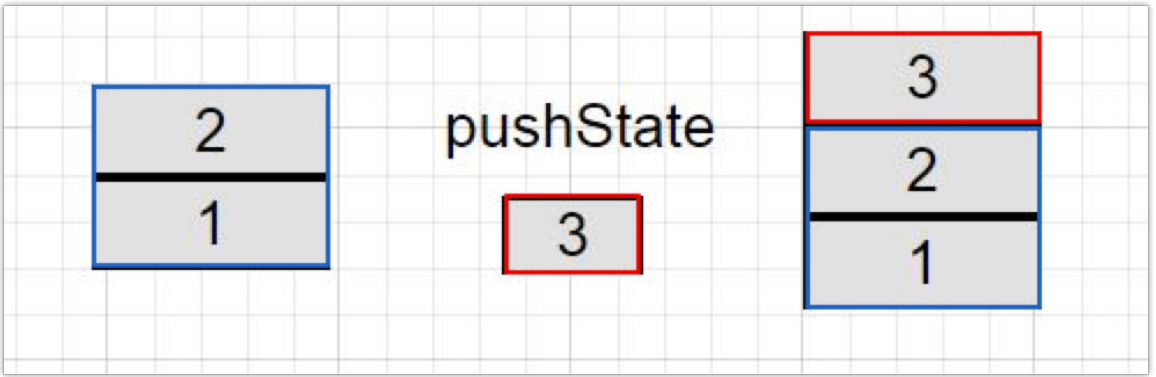
url – относительный/абсолютный URL новой записи в истории браузера




```
window.onpopstate = event => {
    console.info("location: " + document.location + ", state: " +
        JSON.stringify(event.state));
```

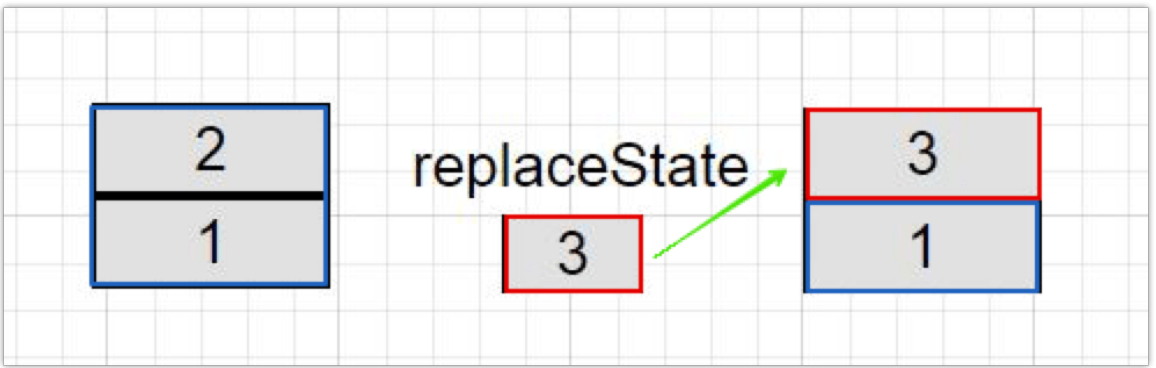
```
window.history.back();
```




2
1
pushState
3
3
2
1

window.history.replaceState(state, title [, url]) – метод,

обновляющий текущее состояние истории браузера



# Событие popstate

## Не вызывает событие

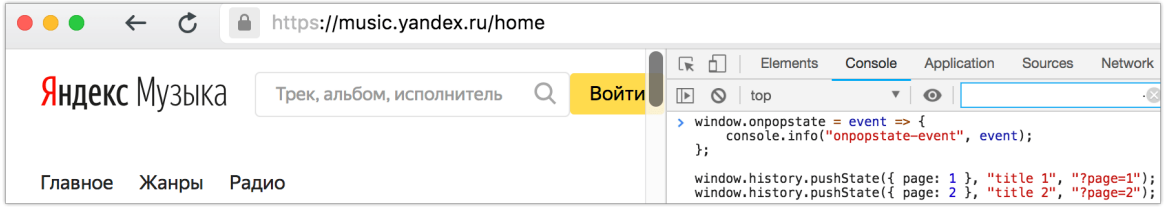
```
window.history.pushState()
window.history.popState()
```

## Вызывает событие

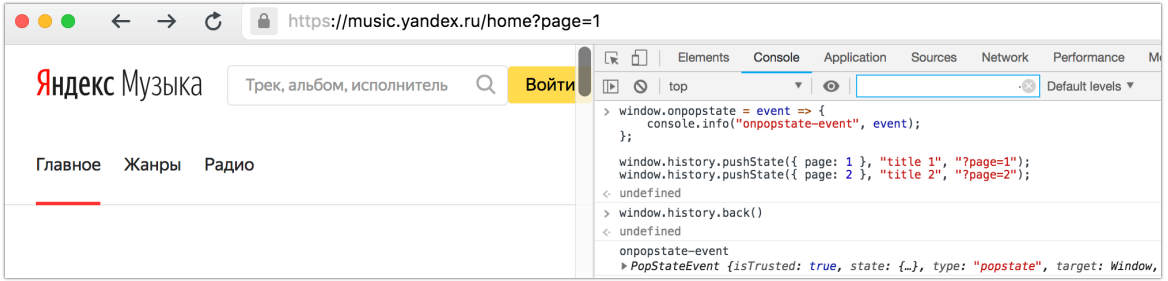
```
window.history.back()
window.history.forward()
Совершение действий в браузере
(нажатие стрелок для движения по истории)
```

```
window.onpopstate = event => {
    console.info("onpopstate-event", event);
};

window.history.pushState({ page: 1 }, "title 1", "?page=1");
window.history.pushState({ page: 2 }, "title 2", "?page=2");
```

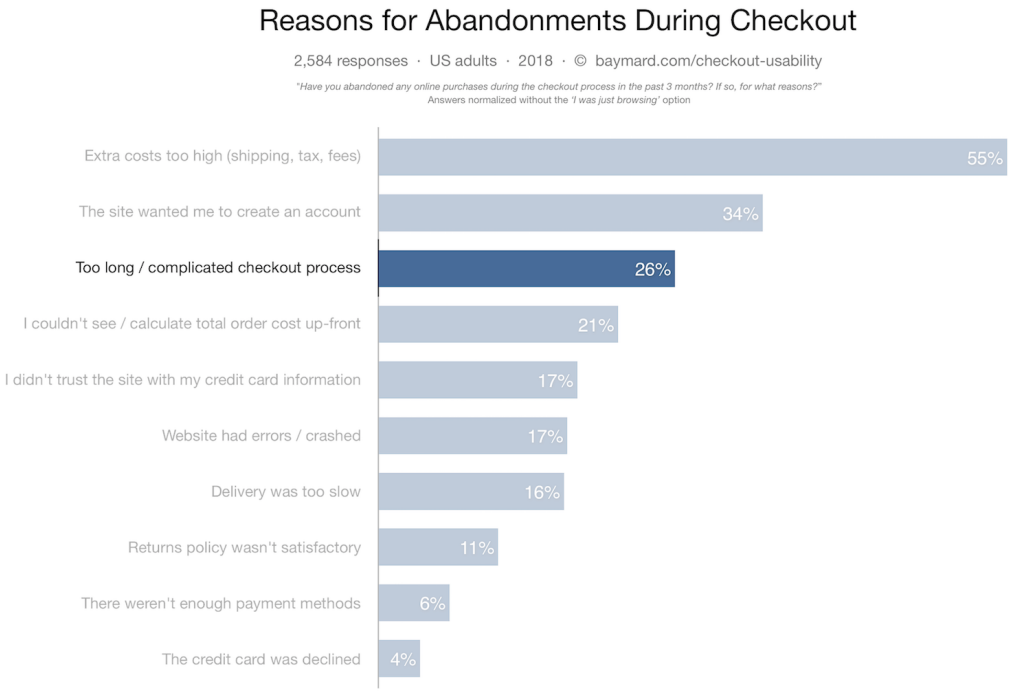


```
window.history.back();
```



# Payment Request API


Reasons For Online Shopping Cart Abandonment
56%
PRESENTED WITH UNEXPECTED COSTS
37%
I WAS JUST BROWSING
36%
FOUND A BETTER PRICE ELSEWHERE
32%
OVERALL PRICE TOO EXPENSIVE
26%
DECIDED AGAINST BUYING
25%
WEBSITE NAVIGATION TOO COMPLICATED
24%
WEBSITE CRASHED
21%
PROCESS WAS TAKING TOO LONG
18%
EXCESSIVE PAYMENT SECURITY CHECKS
17%
CONCERNS ABOUT PAYMENT SECURITY
16%
DELIVERY OPTIONS WERE UNSUITABLE
15%
WEBSITE TIME OUT
13%
PRICE PRESENTED IN A FOREIGN CURRENCY
11%
PAYMENT WAS DECLINED
% of Respondents
Respondents:
19k consumers
153 Senior Retailers
(Jan-Feb 2012)
SOURCE: WorldPay


Reasons for Abandonments During Checkout
2,584 responses · US adults · 2018 · © baymard.com/checkout-usability
"Have you abandoned any online purchases during the checkout process in the past 3 months? If so, for what reasons?"
Answers normalized without the 'I was just browsing' option
Extra costs too high (shipping, tax, fees) 55%
The site wanted me to create an account 34%
Too long / complicated checkout process 26%
I couldn't see / calculate total order cost up-front 21%
I didn't trust the site with my credit card information 17%
Website had errors / crashed 17%
Delivery was too slow 16%
Returns policy wasn't satisfactory 11%
There weren't enough payment methods 6%
The credit card was declined 4%

# Зачем, если есть PayPal/AmazonPay/...?

Добавляет возможность оплаты «одной кнопкой» на многих сайтах

Сохраняет данные карты у себя

Транзакции PayPal обходятся ритейлерам до 1,9% от выручки

Страницы оформления заказа у ритейлеров практически не стандартизированы

Адрес доставки и выставления счетов должны быть введены отдельно в разных форматах в конце процесса

# Преимущества PR API

PR API позволяет хранить данные в браузере

Транзакции бесплатны для ритейлеров

Страницы оформления заказа стандартизованы

# Цель

Помочь пользователям оплатить так, как им хочется

Сделать это быстро и эффективно

# Как?

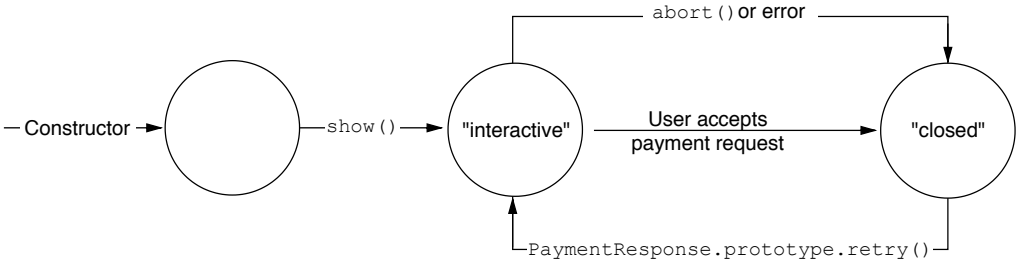
Создать простой и удобный способ оформления заказа

Не вводить вручную платёжные данные при каждой покупке, а хранить реквизиты банковских карточек в браузере и вписывать их в форму автоматически

# Создание экземпляра PaymentRequest

```
const paymentRequest = new PaymentRequest(
    supportedPaymentMethods, // способы оплаты
    orderDetails, // детали заказа
    paymentOptions  // [данные об оплате]
);
```

# Состояния экземпляра PaymentRequest




Конструктор устанавливает начальное состояние "Created"

Метод show() меняет на "Interactive"

Метод abort() или любая ошибка приводят к состоянию "Closed"

Пользователь принимает или отклоняет платеж – также "Closed"

# Поддерживаемые способы оплаты

- Стандартизированные способы оплаты
- Способы оплаты на основе URL

# Стандартизированные способы оплаты

Имеют реестр способов оплаты

- basic-card

- basic-credit-transfer

- tokenized-card

- sepamail

# Способы оплаты на основе URL

Связаны с определенным платежным приложением

Не имеют реестра способов оплаты


# Поддерживаемые способы оплаты

```
const paymentRequest = new PaymentRequest(
    paymentMethods, orderDetails, paymentOptions
);


const paymentMethods = [
    {
        supportedMethods: ['basic-card'],
        data: {
            supportedNetworks: [ // "бренды" поддерживаемых карт
                'visa', 'mastercard', 'unionpay'
            ],
            supportedTypes: [ // типы поддерживаемых карт
                'debit', 'credit'
            ]
        }
    },
    { supportedMethods: "https://apple.com/apple-pay" }
];
```

# Информация о заказе

```
const paymentRequest = new PaymentRequest(
    paymentMethods, orderDetails, paymentOptions
);


const orderDetails = {
    displayItems: [{
        label: '1 x Футболка ДММ',
        amount: { currency: 'RUB', value: '650.00' }
    }],
    total: {
        label: 'ДММ мерч',
        amount: { currency: 'RUB', value: '650.00' }
    }
};
```

# 2 + 2 = 5? Возможно

PR API не выполняет арифметическую проверку

Разработчик приложения отвечает за то, что значение в total будет соответствовать значениям в displayItems

# Поддерживаемые методы оплаты

```
const paymentRequest = new PaymentRequest(
    paymentMethods, orderDetails, paymentOptions
);

// Пользователю будет предложено указать
// адрес электронной почты, имя, номер телефона, адрес доставки
// и тип доставки, например:
const paymentOptions = {
    requestPayerEmail: true,
    requestPayerPhone: true,
    requestShipping: true
};
```

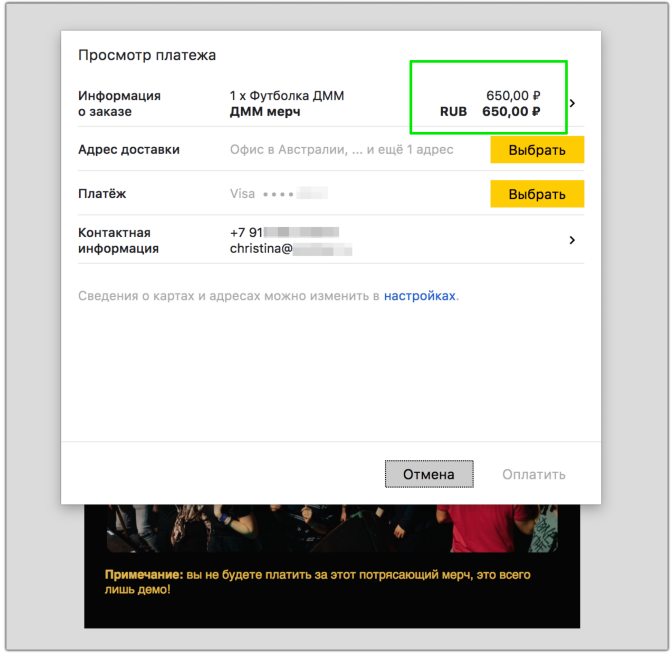
# Создание экземпляра PaymentRequest

```
const paymentRequest = new PaymentRequest(
    paymentMethods, // способы оплаты
    orderDetails, // детали заказа
    paymentOptions  // [данные об оплате]
);
```

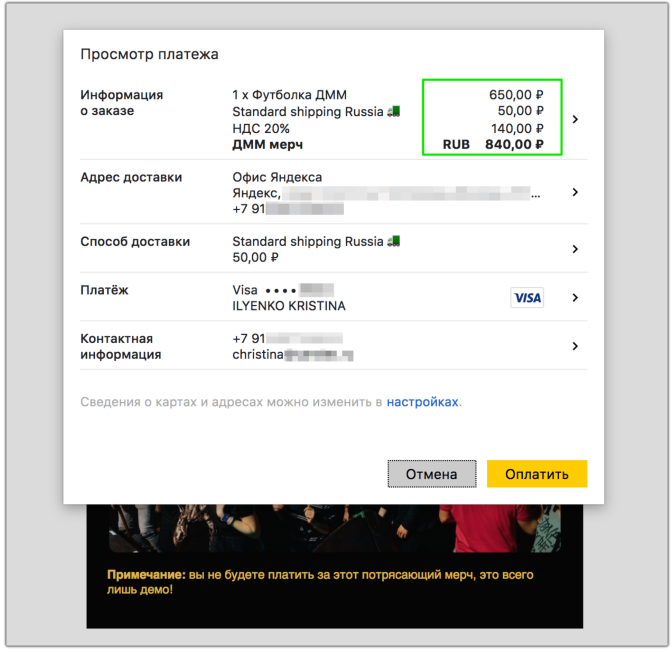
# Обработка транзакции

```
paymentRequest
    .show()
    .then(paymentResponse => {
        const paymentData = { ... };
        return fetch('/paymentGatewayAddress', paymentData)
            .then(response => paymentResponse.complete(...)})
            .then(() => successPanel.innerHTML = 'Order complete!')
            .catch(() => paymentResponse.complete('fail'));
        })
    .catch(() => {
        failPanel.innerHTML = 'Order failed';
    });
```

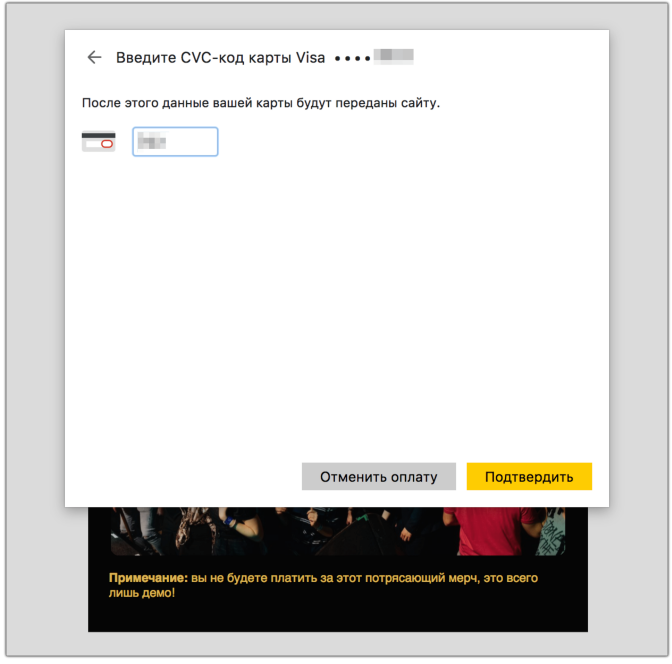
```
paymentRequest.show()
```



`paymentRequest.show()`



`paymentRequest.show()`



```
paymentRequest
    .show().then(paymentResponse => console.info(paymentResponse));
// {
//    methodName: 'basic-card',
//    details: {
//        billingAddress: {
//            city: 'Москва',
//            country: 'RU',
//            organization: 'Яндекс',
//            phone: '+79991233212',
//            ...
//        },
//        cardNumber: '0000000000000000',
//        cardSecurityCode: '123',
//        cardholderName: 'ILYENKO KRISTINA',
//        expiryMonth: '06',
//        expiryYear: '2020'
//    }
//    ...
// }
```

```
paymentRequest
    .show()
    .then(paymentResponse => {
        const paymentInfo = {

            details:    paymentResponse.details,
            methodName: paymentResponse.methodName
        };

        const paymentData = {
            body: JSON.stringify(paymentInfo),
            credentials: 'include',
            headers: { 'Content-Type': 'application/json' },
            method: 'POST'
        };

        return fetch('/paymentGatewayAddress', paymentData)
            .then(...)
            .catch(...);
    });
```

```
paymentRequest
    .show()
    .then(paymentResponse => {
        const paymentData = { ... };

        return fetch('/paymentGatewayAddress', paymentData)
            .then(response => {
                if (response.status === 200) {
                    return paymentResponse.complete('success');
                } else {
                    return paymentResponse.complete('fail');
                }
            })
            .then(() => document.getElementById('status')
                .innerHTML = 'Order complete!')
            .catch(() => paymentResponse.complete('fail'));
    })
    .catch(() => document.getElementById('status')
        .innerHTML = 'Order failed!';);
```

# Payment Request API Demo

Ой, а тут покупают ДММ-мерч

Футболка ДММ

RUB 650.00

Купить Order complete!

**Примечание:** вы не будете платить за этот потрясающий мерч, это всего лишь демо!