

# Front-end

---

JavaScript



# JS. BOM, DOM, API

---

- BOM
- DOM
- API



# JS. BOM, DOM, API

---

## **BOM - Объектная модель браузера**

Объектная модель браузера (BOM от англ. Browser Object Model) позволяет JavaScript "общаться" с браузером.

## **Объектная модель браузера (BOM)**

Не существует каких-либо официальных стандартов для Объектной модели браузера (BOM).

Так как современные браузеры реализуют (почти) одни и те же методы и свойства для JavaScript интерактивно, их часто относят к методам и свойствам BOM.



# JS. BOM, DOM, API

---

Веб-страницы бывают статическими и динамическими, последние отличаются тем, что в них используются сценарии (программы) на языке JavaScript.

В сценариях JavaScript браузер веб-разработчику предоставляет множество "готовых" объектов, с помощью которых он может взаимодействовать с элементами веб-страницы и самим браузером. В совокупности все эти объекты составляют объектную модель браузера (BOM – Browser Object Model).

На самом верху этой модели находится глобальный объект window. Он представляет собой одно из окон или вкладку браузера с его панелями инструментов, меню, строкой состояния, HTML страницей и другими объектами.



# JS. BOM, DOM, API

---

Доступ к этим различным объектам окна браузера осуществляется с помощью следующих основных объектов: `navigator`, `history`, `location`, `screen`, `document` и т.д. Так как данные объекты являются дочерними по отношению к объекту `window`, то обращение к ним происходит как к свойствам объекта `window`.

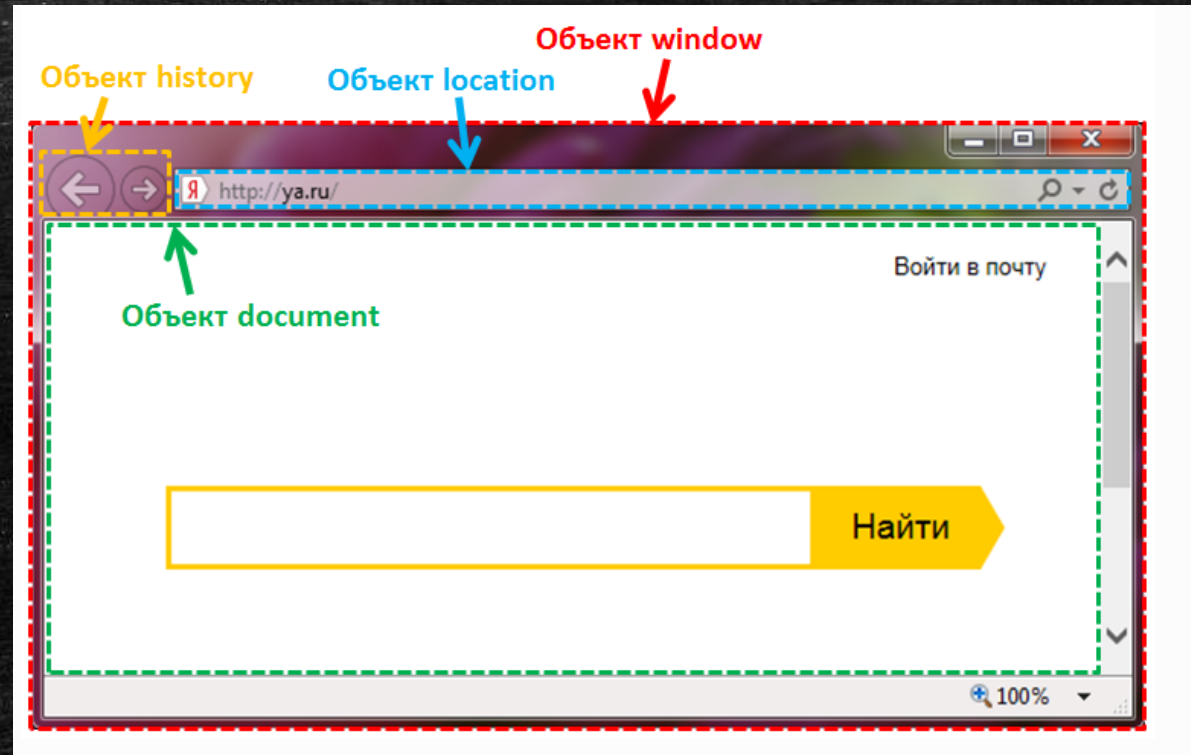
Например, для того чтобы обратиться к объекту `screen`, необходимо использовать следующую конструкцию: `window.screen`. Но если мы работаем с текущим окном, то "`window`." можно опустить. Например, вместо `window.screen` можно использовать просто `screen`.

Из всех этих объектов, наибольший интерес и значимость для разработчика представляет объект `document`, который является корнем объектной модели документа (DOM – Document Object Model). Данная модель в отличие от объектной модели браузера стандартизована в спецификации и поддерживается всеми браузерами.



# JS. BOM, DOM, API

Объект document представляет собой HTML документ, загруженный в окно (вкладку) браузера. С помощью свойств и методов данного объекта Вы можете получить доступ к содержимому HTML-документа, а также изменить его содержимое, структуру и оформление.



Примечание: Объектная модель браузера не стандартизована в спецификации, и поэтому её реализация может отличаться в разных браузерах.



# JS. BOM, DOM, API

---

## Основные объекты BOM

Основные объекты Browser Object Model:

- window
- navigator
- history
- location
- screen
- document



# JS. BOM, DOM, API

---

## Объект window

Объект window поддерживается всеми браузерами. Он представляет окно браузера.

Все глобальные JavaScript объекты, функции и переменные автоматически становятся членами объекта window.

Глобальные переменные являются свойствами объекта window.

Глобальные функции являются методами объекта window.



# JS. BOM, DOM, API

---

## Объект window

window – самый главный объект в браузере, который отвечает за одно из окон (вкладок) браузера. Он является корнем иерархии всех объектов доступных веб-разработчику в сценариях JavaScript. Объект window кроме глобальных объектов (document, screen, location, navigator и др.) имеет собственные свойства и методы, которые предназначены для:

- открытия нового окна (вкладки);
- закрытия окна (вкладки) с помощью метода close();
- распечатывания содержимого окна (вкладки);
- передачи фокуса окну или для его перемещения на задний план (за всеми окнами);
- управления положением и размерами окна, а также для осуществления прокручивания его содержимого;
- изменения содержимого статусной строки браузера;
- взаимодействия с пользователем посредством следующих окон: alert (для вывода сообщений), confirm (для вывода окна, в котором пользователю необходимо подтвердить или отменить действия), prompt (для получения данных от пользователя);
- выполнения определённых действий через определённые промежутки времени и др.



## JS. BOM, DOM, API

---

Если в браузере открыть несколько вкладок (окон), то браузером будет создано столько объектов `window`, сколько открыто этих вкладок (окон). Т.е. каждый раз открывая вкладку (окно), браузер создаёт новый объект `window` связанный с этой вкладкой (окном).



# JS. BOM, DOM, API

---

## Объект navigator

navigator – информационный объект с помощью которого Вы можете получить различные данные, содержащиеся в браузере:

- информацию о самом браузере в виде строки (User Agent);
- внутреннее "кодовое" и официальное имя браузера;
- версию и язык браузера;
- информацию о сетевом соединении и местоположении устройства пользователя;
- информацию об операционной системе и многое другое.



# JS. BOM, DOM, API

---

## Объект navigator

Объект navigator предназначен для предоставления подробной информации о браузере, который пользователь использует для доступа к сайту или веб-приложению. Кроме данных о браузере, в нём ещё содержится сведения о операционной системе, сетевом соединении и др.

Объект navigator – это свойство window:

```
const navigatorObj = window.navigator;
```

```
// или без указания window
```

```
// const navigatorObj = navigator;
```



# JS. BOM, DOM, API

---

## Объект navigator

Свойства и методы объекта navigator

Объект navigator имеет свойства и методы. Очень часто они используются для того чтобы узнать, какие функции поддерживаются браузером, а какие нет.

- appName – кодовое имя браузера;
- appName – имя браузера;
- appVersion — версия браузера;
- cookieEnabled - позволяет определить включены ли cookie в браузере;
- geolocation - используется для определения местоположения пользователя;
- language - язык браузера;
- online - имеет значение true или false в зависимости от того находится ли браузер в сети или нет;
- platform - название платформы, для которой скомпилирован браузер;
- product - имя движка браузера;
- userAgent - возвращает заголовок user agent, который браузер посылает на сервер.



# JS. BOM, DOM, API

---

## Объект navigator

Методы объекта navigator:

javaEnabled – позволяет узнать, включён ли в браузере Java;

sendBeacon - предназначен для отправки небольшого количества информации на веб-сервер без ожидания ответа.



# JS. BOM, DOM, API

---

## Объект navigator

Обнаружение браузера с помощью userAgent

userAgent - это строка, содержащая информацию о браузере, которую он посылает в составе заголовка запроса на сервер.

Пример содержания строки userAgent в браузере Google Chrome:

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/90.0.4430.93 Safari/537.36
```

Она содержит сведения об операционной системе, браузере, версиях, платформах и т.д.



# JS. BOM, DOM, API

---

## Объект navigator

Зачем это нужно? Например, для того, чтобы запускать некоторые скрипты или функции только в определенном браузере.

Но при использовании navigator.userAgent следует иметь в виду, что эта информация не является 100% достоверной, поскольку она может быть изменена пользователем.

Вообще не существует 100% надежных способов идентификации браузера. Поэтому лучше проверять доступность необходимой функции или свойства, и если этой поддержки нет, то написать обходной код для реализации этого функционала или вообще его не предоставлять для этих браузеров.



# JS. BOM, DOM, API

**Объект navigator** - определение мобильного устройства посредством userAgent

Самый простой способ обнаружить мобильные устройства - это найти слово mobile в пользовательском агенте (userAgent):

```
const isMobile = navigator.userAgent.toLowerCase().match(/mobile/i);  
if (isMobile) {  
  // для мобильных устройств  
} else {  
  // для не мобильных устройств  
}
```

Более подробный вариант идентификации мобильного браузера:

```
const isMobile = navigator.userAgent.toLowerCase().match(/mobile/i);  
const isTablet = navigator.userAgent.toLowerCase().match(/tablet/i);  
const isAndroid = navigator.userAgent.toLowerCase().match(/android/i);  
const isiPhone = navigator.userAgent.toLowerCase().match(/iphone/i);  
const isiPad = navigator.userAgent.toLowerCase().match(/ipad/i);
```



# JS. BOM, DOM, API

---

## Объект navigator - объект geolocation

Объект geolocation предназначен для определения местоположения устройства. Доступ к этому объекту осуществляется через свойство «navigator.geolocation»:

```
const geo = navigator.geolocation;
```

// или так

```
const geo = window.navigator.geolocation;
```

Узнать поддерживает ли браузер геолокацию, можно посредством проверки существования свойства geolocation в объекте navigator:

```
if (!navigator.geolocation) {  
    console.error('Ваш браузер не поддерживает геолокацию!');  
}
```

При этом, когда сайт или веб-приложение пытается получить доступ к местонахождению пользователя, браузер из соображений конфиденциальности предоставит его только в том случае если это разрешит пользователь.



# JS. BOM, DOM, API

---

## Объект navigator - объект geolocation

Получение текущего местоположения пользователя осуществляется через метод `getCurrentPosition`.

// аргумент success - обязательный

```
navigator.geolocation.getCurrentPosition(success[, error[, options]]);
```

Этот метод посылает асинхронный запрос. В случае успеха мы можем получить местоположение устройства, в противном случае – нет.

Метод `getCurrentPosition` принимает 3 аргумента:

- success - функцию обратного вызова, которая будет вызвана при успешном получении геоданных (т.е. когда пользователь разрешил доступ сайту или веб-приложению к Geolocation API и данный API определил местоположение пользователя);
- error - функцию обратного вызова, которая вызывается при ошибке (т.е. когда пользователь не разрешил доступ к Geolocation API, или данный API не смог определить местонахождение пользователя, или истекло время ожидания timeout);
- options - объект, содержащий настройки.



# JS. BOM, DOM, API

---

## Объект navigator - объект geolocation

В options можно установить:

- maximumAge - следует ли информацию о местонахождении пользователя кэшировать (в миллисекундах) или пытаться всегда получать реальное значение (значение 0 - по умолчанию);
- timeout - максимальное время в миллисекундах в течении которого нужно ждать ответ (данные о местоположении); если ответ за указанное время не пришёл, то вызывать функцию обратного вызова error (по умолчанию имеет значение infinity, т.е. ждать бесконечно);
- enableHighAccuracy - при значении true будет пытаться получить наилучшие результаты, т.е. более точное местоположение (для этого может понадобиться задействовать GPS), что в свою очередь может привести к более длительному времени отклика или увеличению энергопотребления; по умолчанию - false.



# JS. BOM, DOM, API

---

## Объект navigator - объект geolocation

В функцию success передаётся в качестве первого аргумента объект GeolocationPosition. Он содержит информацию о местоположении устройства (coords) и времени, когда оно было получено (timestamp).

Объект coords содержит следующие свойства:

- latitude - широта (в градусах);
- longitude - долгота (в градусах);
- altitude - высота над уровнем моря (в метрах);
- speed - скорость устройства в метрах в секунду; это значение может быть null.



# JS. BOM, DOM, API

## Объект navigator - объект geolocation

Пример получения местоположения устройства:

```
// при успешном получении сведений о местонахождении
function success(position) {
    // position - объект GeolocationPosition, содержащий информацию о местонахождении
    const latitude = position.coords.latitude;
    const longitude = position.coords.longitude;
    const altitude = position.coords.altitude;
    const speed = position.coords.speed;
    // выведем значения в консоль
    console.log(`Широта: ${latitude}°`);
    console.log(`Долгота: ${longitude}°`);
    console.log(`Высота над уровнем моря: ${altitude}м`);
    console.log(`Скорость: ${speed}м/с`);
}
function error() {
    console.log('Произошла ошибка при определении местоположения!');
}
if (!navigator.geolocation) {
    // получаем текущее местоположение пользователя
    navigator.geolocation.getCurrentPosition(success, error);
}
```



# JS. BOM, DOM, API

---

## Объект navigator - объект geolocation

Методы **watchPosition** и **clearWatch**

Метод watchPosition используется когда нужно получать данные о местоположении каждый раз, когда оно меняется. Метод возвращает целое число, являющееся идентификатором задачи.

```
navigator.geolocation.watchPosition(success[, error[, options]])
```

Метод clearWatch предназначен для удаления задачи по её идентификатору, которую вы создали посредством watchPosition.

```
// создаём задачу и сохраняем её идентификатор в watchId
```

```
let watchId = navigator.geolocation.watchPosition(success, error, options);
```

```
// удаляем задачу по её идентификатору
```

```
clearWatch(watchId);
```



# JS. BOM, DOM, API

---

## Объект history

**history** – объект, который позволяет получить историю переходов пользователя по ссылкам в пределах одного окна (вкладки) браузера.

Данный объект отвечает за кнопки forward (вперёд) и back (назад). С помощью методов объекта history можно имитировать нажатие на эти кнопки, а также переходить на определённое количество ссылок в истории вперёд или назад.

Кроме этого, с появлением HTML5 History API веб-разработчику стали доступны методы для добавления и изменения записей в истории, а также событие, с помощью которого Вы можете обрабатывать нажатие кнопок forward (вперёд) и back (назад).



# JS. BOM, DOM, API

---

## Объект location

**location** – объект, который отвечает за адресную строку браузера. Данный объект содержит свойства и методы, которые позволяют: получить текущий адрес страницы браузера, перейти по указанному URL, перезагрузить страницу и т.п.



# JS. BOM, DOM, API

---

## Объект location

Объект location - это один из дочерних объектов window, который отвечает за адресную строку окна или вкладки браузера. Доступ к данному объекту осуществляется как к свойству объекта window, т.е. через точку.

window.location

Объект location содержит свойства и методы, с помощью которых Вы можете не только получить текущий адрес страницы (URL или его части: имя хоста, номер порта, протокол и т.д.), но и изменить его.



# JS. BOM, DOM, API

---

## Объект location

Свойства объекта location:

- hash - устанавливает или возвращает якорную часть (#) URL;
- host - устанавливает или возвращает имя хоста и номер порта URL;
- hostname - устанавливает или возвращает имя хоста URL;
- href - устанавливает или возвращает содержимое URL;
- origin - возвращает протокол, имя хоста и номер порта URL;
- pathname - устанавливает или возвращает часть URL, содержащей путь;
- port - устанавливает или возвращает номер порта URL;
- protocol - устанавливает или возвращает протокол URL;
- search - устанавливает или возвращает часть URL, содержащей строку с параметрами (?параметр1=значение1&параметр2=значение2&...);



# JS. BOM, DOM, API


## Объект location

`http://itchief.ru/search?q=bootstrap#part2`

1 2 3 4 5

location.href


- 1 – location.protocol
- 2 – location.hostname
- 3 – pathname.hostname
- 4 – search.hostname
- 5 – hash.hostname


 **window.location** CHEATSHEET


PROPERTIES

protocol	hostname	pathname	search	hash
https:	//	www.samanthaming.com	/tidbits/	?filter=JS #2

METHODS

assign  
  
history

replace  


reload  


toString  
"url"

Примечание: Изменение URL или какой либо её части с помощью свойств объекта location приводит к немедленному переходу к этому URL в текущем окне, или в том окне или вкладке браузера, для которого этот объект был вызван.



# JS. BOM, DOM, API

---

## Объект location

Методы объекта location:

- **assign()** - загружает новый документ в текущее окно (вкладку) браузера или в то окно для которого этот метод был вызван.
- **reload()** - перезагружает текущий документ. Метод reload() имеет один необязательный параметр булевского типа. Если в качестве параметра указать значение true, то страница будет принудительно обновлена с сервера. А если параметр не указывать или использовать в качестве параметра значение false, то браузер может обновить страницу, используя данные своего кэша. Метод reload() "имитирует" нажатие кнопки обновить в браузере.
- **replace()** - заменяет текущий документ с помощью нового документа, URL которого указан в качестве параметра.



# JS. BOM, DOM, API

---

## Объект screen

**screen** – объект, который предоставляет информацию об экране пользователя: разрешение экрана, максимальную ширину и высоту, которую может иметь окно браузера, глубина цвета и т.д.

Объект screen предназначен для получения информации об экране, на котором отображается текущее окно браузера.

Обратиться к нему можно как к свойству объекта window:

```
const screenObj = window.screen;
```

Также к объекту screen можно получить доступ без указания window, т.е. так:

```
const screenObj = screen;
```



# JS. BOM, DOM, API

---

## Объект screen

С помощью объекта screen мы можем получить следующую информацию об экране:

- width – ширина экрана в пикселях;
- height – высота экрана в пикселях;
- availWidth – доступная ширина экрана;
- availHeight – доступная высота экрана (например, в Windows, эта высота равна общей высоте из которой нужно вычесть высоту панели задач);
- availLeft – x-координата первого доступного пикселя;
- availTop – y-координата первого доступного пикселя;
- colorDepth – глубину цвета в битах;
- pixelDepth – глубину цвета на пиксель экрана в битах;
- orientation – ориентация экрана.



# JS. BOM, DOM, API

---

## Объект screen

В screen.orientation:

- type – тип ориентации (одно из следующих значений: «portrait-primary», «portrait-secondary», «landscape-primary» или «landscape-secondary»);
- angle - угол ориентации документа.

При изменении ориентации экрана возникает событие change.



# JS. BOM, DOM, API

---

## Объект document

Наибольший интерес среди всех этих объектов предоставляет именно объект document, т.к. он отвечает за документ, загруженный в окно или вкладку браузера. Он даёт начало объектной модели документа (DOM - Document Object Model), которая стандартизована в спецификации и поддерживается всеми браузерами.

Некоторые свойства и методы объекта document, т.е. такие которые особого отношения к объектной модели документа не имеют.



# JS. BOM, DOM, API

---

## Объект document

**document** – HTML документ, загруженный в окно (вкладку) браузера. Он является корневым узлом HTML документа и "владельцем" всех других узлов: элементов, текстовых узлов, атрибутов и комментариев. Объект document содержит свойства и методы для доступа ко всем узловым объектам. document как и другие объекты, является частью объекта window и, следовательно, он может быть доступен как window.document.



# JS. BOM, DOM, API

---

## Объект document

Объект document содержит следующие "общие" свойства и методы:

Свойство document.readyState - возвращает строку, содержащую статус текущего документа. Данное свойство доступно только для чтения.

В процессе загрузки документ последовательно проходит следующие состояния:

- uninitialized - процесс загрузки ещё не начался;
- loading - идёт процесс загрузки;
- loaded - загрузка HTML кода завершена;
- interactive - документ достаточно загружен для того, чтобы пользователь мог взаимодействовать с ним. Код JavaScript может начать выполняться только на этом этапе;
- complete - документ полностью загружен.

В большинстве случаев, код JavaScript обычно выполняется и вызывается после того, как документ полностью загружен, т.е. когда он находится в состоянии complete.



# JS. BOM, DOM, API

---

## Объект document

Свойство **document.referrer** - возвращает строку, содержащую адрес (URL) страницы, с которой пользователь пришёл на эту страницу. Если текущий документ не был открыт через ссылку (например, с помощью закладки или прямого ввода адреса в адресную строку), то данное свойство вернёт пустую строку.



# JS. BOM, DOM, API

---

## Объект document

Свойство **document.URL** - возвращает строку, содержащую полный URL адрес текущего HTML документа.

Свойство **document.domain** - возвращает строку, содержащую доменное имя сервера, с которого загружен текущий документ. Если домен текущего документа не может быть определён, то данное свойство вернёт значение null.



# JS. BOM, DOM, API

---

## Объект document

Метод **document.write()** - предназначен для вывода в документ строки, указанной в качестве параметра данного метода. Если данный метод вызывается в процессе загрузки документа, то он выводит строку в текущем месте.

В том случае, если данный метод вызывается после загрузки документа, то он приводит к полной очистке этого документа и вывода строки.



# JS. BOM, DOM, API

---

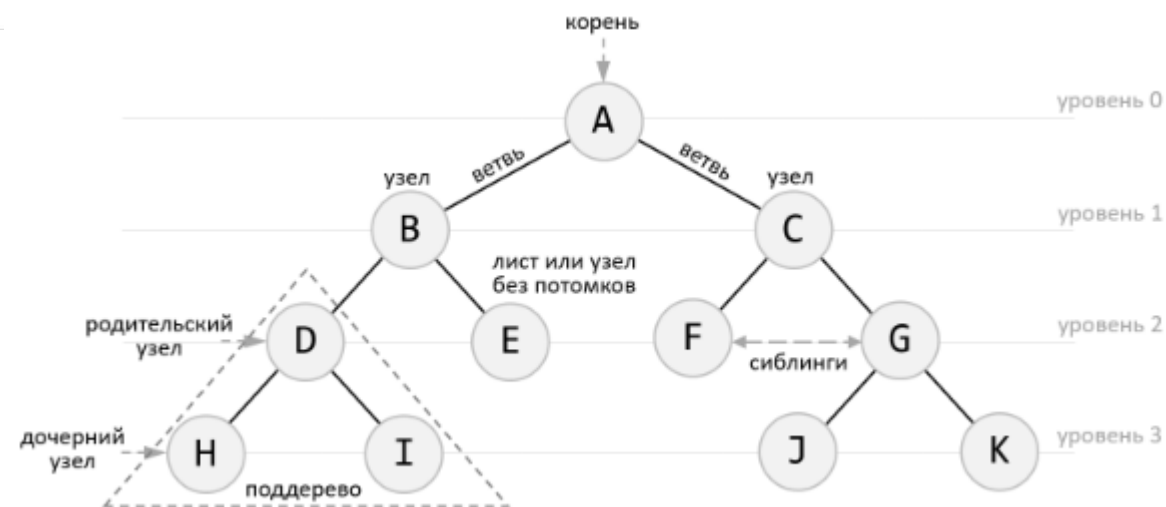
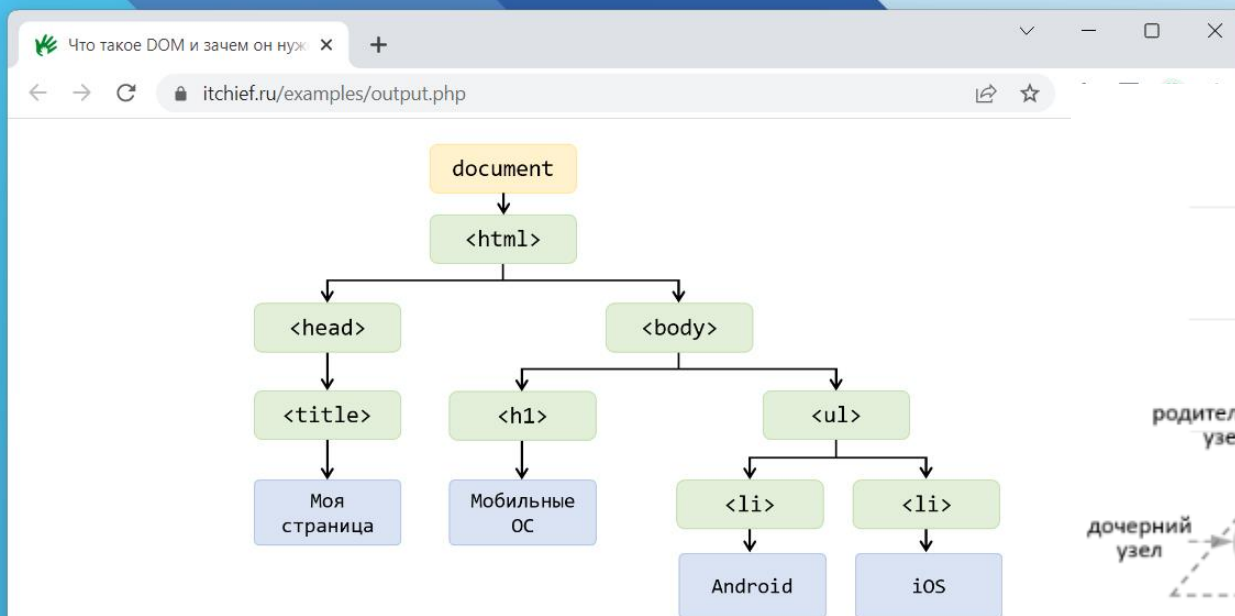
## DOM

Браузер, перед тем как показать вам запрашиваемую страницу, выполняет большое количество различных действий. Самое важное здесь понять, что браузер не работает с HTML-страницей напрямую как с текстом, а строит для этого DOM.

DOM – это объектная модель документа (Document Object Model). Представляет она собой древовидную структуру страницы, состоящую из узлов. Каждый узел в ней – это объект, который может иметь определённые свойства и методы. Иными словами, можно сказать, что DOM – это набор иерархически связанных между собой объектов.



# JS. BOM, DOM, API





# JS. BOM, DOM, API

---

## DOM

**DOM (Document Object Model)** — это специальная древовидная структура, которая позволяет управлять HTML-разметкой из JavaScript-кода. Управление обычно состоит из добавления и удаления элементов, изменения их стилей и содержимого.

Браузер создаёт DOM при загрузке страницы, складывает его в переменную `document` и сообщает, что DOM создан, с помощью события `DOMContentLoaded`. С переменной `document` начинается любая работа с HTML-разметкой в JavaScript.

Как пишется

Объект `document` содержит большое количество свойств и методов, которые позволяют работать с HTML. Чаще всего используются методы, позволяющие найти элементы страницы.



# JS. BOM, DOM, API

---

## DOM

### Свойства

**title** — заголовок документа. Браузер обычно показывает его на вкладке.

Установить свой заголовок можно простым присваиванием:

```
document.title = 'Мое название документа'
```

```
console.log(document.title) // => Мое название документа
```



# JS. BOM, DOM, API

---

## DOM

### Свойства

**forms** — получить список форм на странице. Свойство только для чтения, напрямую перезаписать его нельзя.

**body** — получить <body> элемент страницы.

**head** — получить <head> элемент страницы.



# JS. BOM, DOM, API

---

## DOM

Дерево состоит из обычных и текстовых узлов. Обычные узлы — это HTML-теги, а текстовые узлы — текст внутри тегов.

Обычный узел называется `Element`, и он содержит в себе описание тега, атрибутов тега и обработчиков. Если изменить описание — изменится и HTML-код этого элемента (возможно что-то даже изменится на экране. Например, если поменять цвет шрифта). В статье `Element` мы разбираем всё необходимое для работы с элементами.

У любого узла есть один родительский узел и дочерние. Родительский узел — элемент, в который вложен текущий узел, он может быть только один. Дочерние — узлы, которые вложены в текущий узел.

Это правило не работает только в двух случаях:

корневой узел — у такого узла нет родителя;

текстовый узел — у таких узлов нет дочерних узлов, только родитель. Последний уровень любого DOM-дерева состоит из текстовых узлов.



# JS. BOM, DOM, API

---

## DOM

Чтобы перемещаться по узлам DOM-дерева нужно знать какие они имеют отношения. Зная их можно будет выбирать правильные свойства и методы. Связи между узлами, определяются их вложенностью. Каждый узел в DOM может иметь следующие виды отношений:

- **родитель** – это узел, в котором он непосредственно расположен; при этом родитель у узла может быть только один; также узел может не иметь родителя, в данном примере им является document;
- **дети или дочерние узлы** – это все узлы, которые расположены непосредственно в нём; например, узел <ul> имеет 2 детей;
- **соседи или сиблинги** – это узлы, которые имеют такого же родителя что и этот узел;
- **предки** – это его родитель, родитель его родителя и так далее;
- **потомки** – это все узлы, которые расположены в нем, то есть это его дети, а также дети его детей и так далее.



# JS. BOM, DOM, API

---

## DOM

### Методы

**getElementById** — поиск элемента по идентификатору;

**getElementsByClassName** — поиск элементов по названию класса;

**getElementsByTagName** — поиск элементов по названию тега;

**querySelector** — поиск первого элемента, подходящего под CSS-селектор;

**querySelectorAll** — поиск всех элементов подходящих под CSS-селектор.



# JS. BOM, DOM, API

---

## DOM: Методы

Работа с веб-страницей так или иначе связана с манипулированием HTML-элементами. Но перед тем, как над ними выполнить некоторые действия (например, добавить стили), их сначала нужно получить.

Выбор элементов в основном выполняется с помощью этих методов:

- **querySelector**
- **querySelectorAll**

Они позволяют выполнить поиск HTML-элементов по CSS-селектору. При этом **querySelector** выбирает один элемент, а **querySelectorAll** – все.

Кроме них имеются ещё:

- **getElementById**
- **getElementsByClassName**
- **getElementsByTagName**
- **getElementsByName**

Но они сейчас применяются довольно редко. В основном используется либо **querySelector**, либо **querySelectorAll**.



# JS. BOM, DOM, API

---

## DOM: Методы

### querySelectorAll

Метод `querySelectorAll` применяется для выбора всех HTML-элементов, подходящих под указанный CSS-селектор. Он позволяет искать элементы как по всей странице, так и внутри определённого элемента:

```
const items = document.querySelectorAll('.item'); // выберем элементы по классу item во всем документе
```

```
const buttons = document.querySelector('#slider').querySelectorAll('.btn'); // выберем .btn внутри #slider
```

Здесь на первой строчке мы нашли все элементы с классом `item`. На следующей строчке мы сначала выбрали элемент с `id="slider"`, а затем в нём все HTML-элементы с классом `btn`

Метод `querySelectorAll` принимает в качестве аргумента CSS-селектор в формате строки, который соответственно и определяет искомые элементы. В качестве результата `querySelectorAll` возвращает объект класса `NodeList`. Он содержит все найденные элементы.



# JS. BOM, DOM, API

---

## DOM: Методы

Узнать количество найденных элементов можно с помощью свойства **length**:

```
const submits = document.querySelectorAll('[type="submit"]'); // выберем элементы с атрибутом type="submit"
```

```
const countSubmits = submits.length; // получим количество найденных элементов
```

Обращение к определённому HTML-элементу коллекции выполняется также как к элементу массива, то есть по индексу. Индексы начинаются с 0:

```
const elFirst = submits[0]; // получим первый элемент
```

```
const elSecond = submits[1]; // получим второй элемент
```

Здесь в качестве результата мы получаем HTML-элемент или `undefined`, если элемента с таким индексом в наборе `NodeList` нет.



# JS. BOM, DOM, API

---

## DOM: Методы

Перебор коллекции HTML-элементов

Перебор **NodeList** обычно осуществляется с помощью `forEach`:

```
// получим все <p> на странице
const elsP = document.querySelectorAll('p');

// переберём выбранные элементы
elsP.forEach((el) => {
  // установим каждому элементу background-color="yellow"
  el.style.backgroundColor = 'yellow';
});
```

Также перебрать набор выбранных элементов можно с помощью цикла `for` или `for...of`



# JS. BOM, DOM, API

---

## DOM: Методы

### querySelector

Метод `querySelector` также как и `querySelectorAll` выполняет поиск по CSS-селектору. Но в отличие от него, он ищет только один HTML-элемент:

```
// ищем #title во всём документе
const elTitle = document.querySelector('#title');

// ищем footer в <body>
const elFooter = document.body.querySelector('footer');
```

На первой строчке мы выбираем HTML-элемент, имеющий в качестве id значение `title`. На второй мы ищем в `<body>` HTML-элемент по тегу `footer`.

В качестве результата этот метод возвращает найденный HTML-элемент или `null`, если он не был найден.

`querySelector` всегда возвращает один HTML-элемент, даже если под указанный CSS-селектор подходят несколько



# JS. BOM, DOM, API

---

## DOM: Методы

Методы **getElement(s)By\*** для выбора HTML-элементов

Здесь мы рассмотрим методы, которые сейчас применяются довольно редко для поиска HTML-элементов. Но в некоторых случаях они могут быть очень полезны. Это:

**getElementById** — получает один элемент по id;

**getElementsByClassName** — позволяет найти все элементы с указанным классом или классами;

**getElementsByTagName** — выбирает элементы по тегу;

**getElementsByName** — получает все элементы с указанным значением атрибута name.



# JS. BOM, DOM, API

---

## DOM: Методы

Метод **getElementById** позволяет найти HTML-элемент на странице по значению id

В качестве результата `getElementById` возвращает объект класса `HTMLElement` или значение `null`, если элемент не был найден. Этот метод имеется только у объекта `document`.

Указывать значение id необходимо с учётом регистра. Так например, `document.getElementById('aside')` и `document.getElementById('ASIDE')` ищут элементы с разным id.

Обратите внимание, что в соответствии со стандартом в документе не может быть несколько тегов с одинаковым id, так как значение идентификатора на странице должно быть уникальным.



# JS. BOM, DOM, API

---

## DOM: Методы

Метод **getElementsByClassName** позволяет найти все элементы с заданным классом или классами. Его можно применить для поиска элементов как во всём документе, так и внутри указанного. В первом случае его нужно будет вызывать как метод объекта `document`, а во втором – как метод соответствующего HTML-элемента:

```
// найдем элементы с классом control в документе
```

```
const elsControl = document.getElementsByClassName('control');
```

```
// выберем элементы внутри другого элемента, в данном случае внутри формы с id="myform"
```

```
const elsFormControl = document.forms.myform.getElementsByClassName('form-control');
```

В качестве результата он возвращает живую HTML-коллекцию найденных элементов.



# JS. BOM, DOM, API

---

## DOM: Методы

Метод **getElementsByTagName** предназначен для получения коллекции элементов по имени тега:

```
// найдем все <a> в документе
const anchors = document.getElementsByTagName('a');
// найдем все <li> внутри #list
const elsLi = document.getElementById('list').getElementsByTagName('li');
```

На первой строчке мы выбрали все <a> в документе и присвоили полученную HTMLCollection переменной anchors. На второй – мы сначала получили #list, а затем в нём нашли все <li>



# JS. BOM, DOM, API

---

## DOM: Методы

В JavaScript `getElementsByName` можно использовать для выбора элементов, имеющих определенное значение атрибута `name`:

```
// получим все элементы с name="phone"
```

```
const elsPhone = document.getElementsByName('phone');
```



# JS. BOM, DOM, API

---

## DOM

### **matches, closest и contains**

В JavaScript имеются очень полезные методы:

**matches** — позволяет проверить соответствует ли HTML-элемент указанному CSS-селектору;

**closest** — позволяет найти для HTML-элемента его ближайшего предка, подходящего под указанный CSS-селектор (поиск начинается с самого элемента);

**contains** — позволяет проверить содержит ли данный узел другой в качестве потомка (проверка начинается с самого этого узла).



# JS. BOM, DOM, API

---

## DOM

Метод **closest** очень часто используется в коде. Он позволяет найти ближайшего предка, подходящего под указанный CSS-селектор. При этом поиск начинается с самого элемента, для которого данный метод вызывается. Если этот элемент будет ему соответствовать, то `closest` вернёт его.

```
<div class="level-1">  
  <div class="level-2">  
    <div class="level-3"></div>  
  </div>  
</div>
```

```
const el = document.querySelector('.level-3');  
const elAncestor = el.closest('.level-1');  
console.log(elAncestor);
```



# JS. BOM, DOM, API

---

## DOM

В JavaScript **closest** очень часто используется в обработчиках событий. Это связано с тем, чтобы события всплывают и нам нужно, например, узнать кликнул ли пользователь в рамках какого-то элемента



# JS. BOM, DOM, API

---

## DOM

Метод **contains** позволяет проверить содержит ли некоторый узел другой в качестве потомка. При этом проверка начинается с самого этого узла, для которого этот метод вызывается. Если узел соответствует тому для которого мы вызываем данный метод или является его потомком, то contains в качестве результата возвращает логическое значение true. В противном случае false

```
<div id="div-1">  
  <div id="div-2">  
    <div id="div-3">...</div>  
  </div>  
</div>
```

```
<div id="div-4">...</div>  
const elDiv1 = document.querySelector('#div-1');  
elDiv1.contains(elDiv1); // true  
const elDiv3 = document.querySelector('#div-3');  
elDiv1.contains(elDiv3); // true  
const elDiv4 = document.querySelector('#div-4');  
elDiv1.contains(elDiv4); // false
```



# JS. BOM, DOM, API

---

## DOM

**Элемент** — это кусочек HTML в DOM-дереве. Браузер создаёт DOM для взаимодействия между JavaScript и HTML. Каждый HTML-тег при этом превращается в элемент DOM. Ещё такие элементы называют узлами.

Из DOM можно получить элемент и изменить его. Браузер заметит изменения и отобразит их на странице.



# JS. BOM, DOM, API

---

## DOM

HTML-элементы содержат свойства, которые можно разделить на группы:

- свойства, связанные с HTML-атрибутами: id, классы, стили и так далее;
- свойства и методы связанные с обходом DOM: получение дочерних элементов, родителя, соседей;
- информация о содержимом;
- специфические свойства элемента.

Первые три группы свойств есть у всех элементов. Специфические свойства отличаются в зависимости от типа элемента. Например, у полей ввода есть свойства, которых нет у параграфов и блоков: value, type и другие.



# JS. BOM, DOM, API

---

## DOM

Свойства, связанные с HTML-атрибутами

Читать и записывать значения HTML-атрибутов можно при помощи свойств элемента. Название обычно совпадает с названием атрибута:

**id** — получить идентификатор элемента.

**className** — список классов в HTML-атрибуте class

**style** — добавить стили. Стили добавляются так же с помощью свойств. Свойства именуются по аналогии с CSS-свойствами:



# JS. BOM, DOM, API

---

## DOM

Свойства и методы, связанные с DOM Скопировать ссылку "💡 Свойства и методы, связанные с DOM"

**children** — список дочерних элементов;

**parentElement** — получить родительский элемент;

**nextElementSibling** и **previousElementSibling** — получить следующий/предыдущий узел-сосед



# JS. BOM, DOM, API

---

## DOM

Свойства с информацией о содержимом Скопировать ссылку "💡 Свойства с информацией о содержимом"

- **innerHTML** — это свойство возвращает HTML-код всего, что вложено в текущий элемент. При записи в это свойство, предыдущее содержимое будет затёрто. Страница отобразит новое содержимое
- **outerHTML** — это свойство возвращает HTML-код текущего элемента и всего, что в него вложено. При записи в это свойство, предыдущее содержимое будет затёрто.
- **textContent** — свойство, возвращает текст всех вложенных узлов без HTML-тегов.