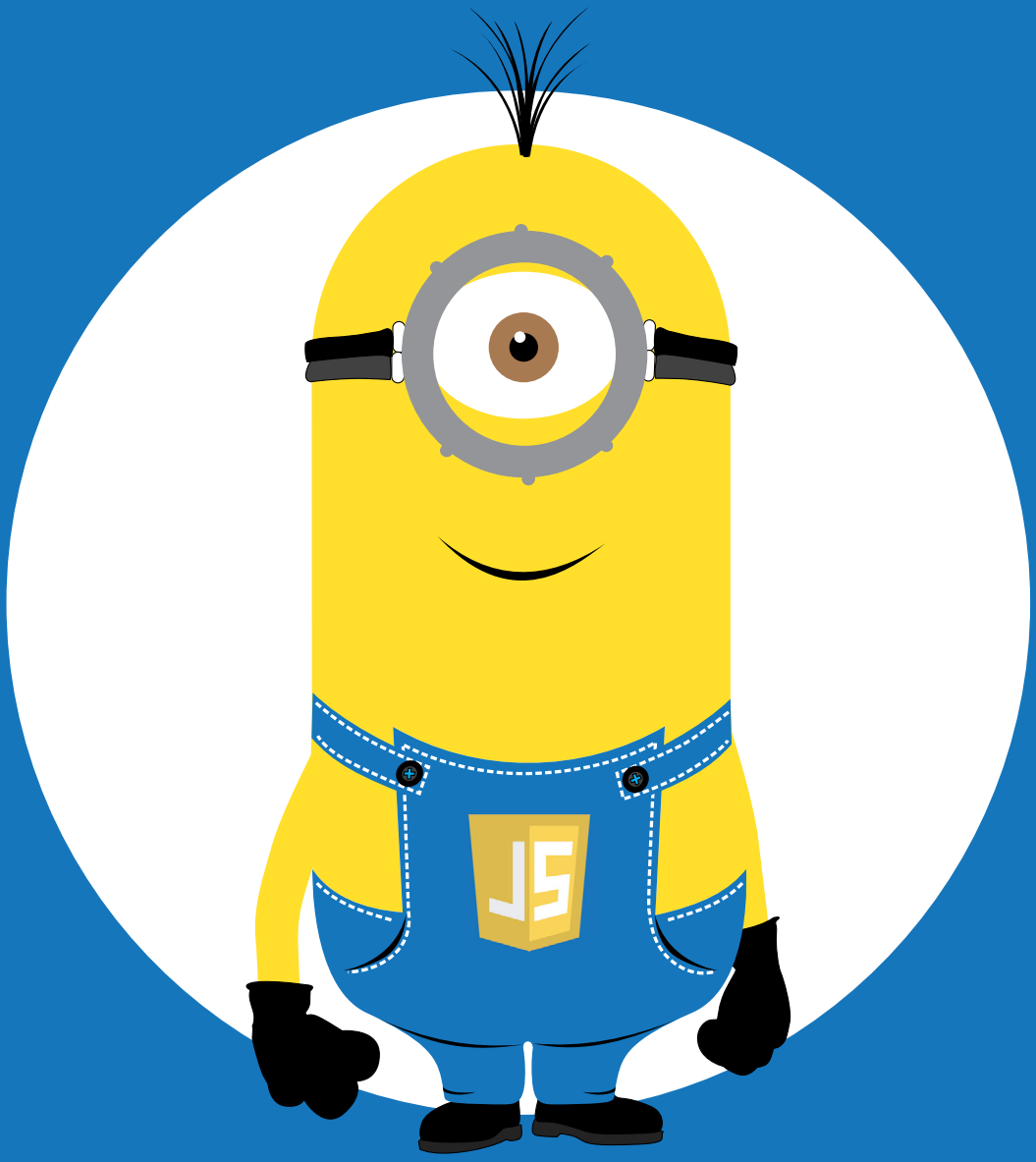


Основы Javascript #2

LoftSchool
от мыслителя к создателю

Содержание

1

Объектная Модель HTML Документа

DOM расшифровывается как **Document Object Model** (Объектная Модель HTML-Документа).

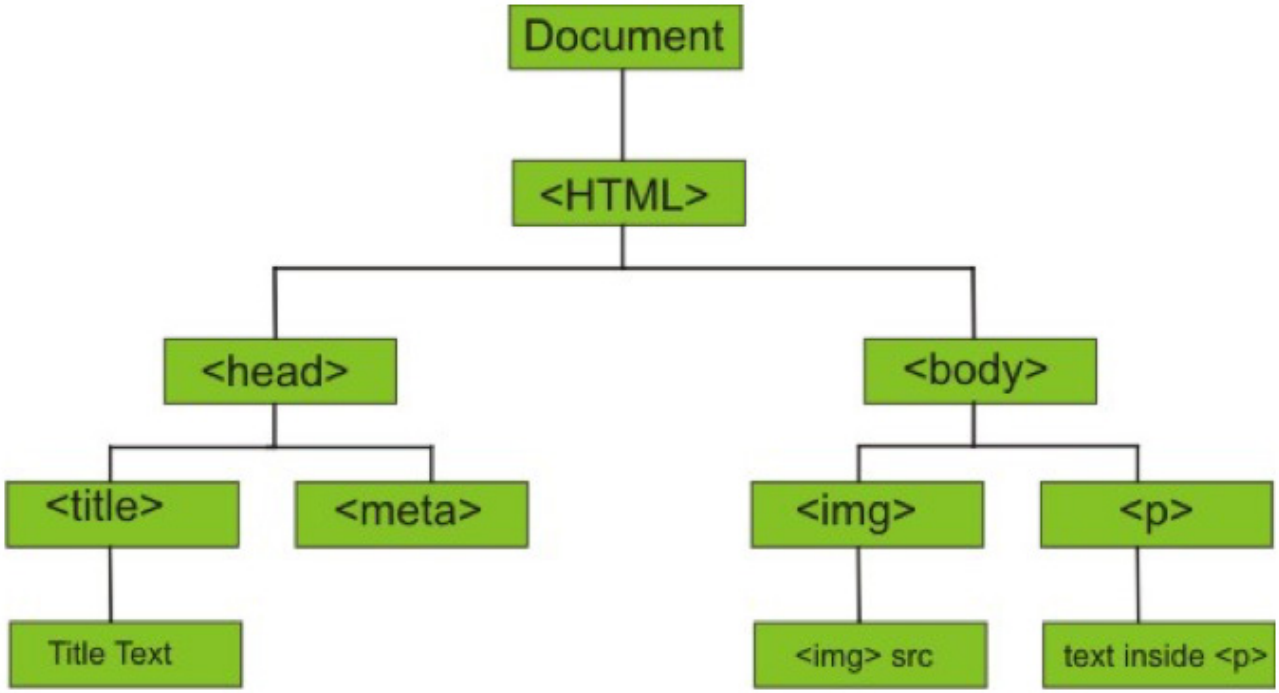
При открытии любого HTML-документа браузер предварительно производит разбор его содержимого и на основе этого разбора создаёт объектную модель HTML-документа — DOM.

DOM состоит из вложенных друг в друга иерархически расположенных объектов, которые называются узлами. Это значит, что HTML-документ представляет собой набор узлов (**Nodes**). Каждый узел в структуре представляет располагающийся на странице HTML-элемент.

Различают 3 основных типа узлов:

1. Элемент (в основном это теги).
2. Текст (текстовые узлы).
3. Комментарии.

Для составления документа узлы вкладывают друг в друга, образуя дерево из узлов.



Используя DOM, вы можете взаимодействовать (считывать, изменять, удалять) с содержимым HTML-документов из скриптов.

Любые узлы могут быть вложены только в элементы. Другими словами, можно вложить узел типа «текст» в узел типа «элемент», но не наоборот! Из чего состоит элемент?

- Открывающий тег (и его атрибуты).
- Дочерние элементы.
- Закрывающий тег.

```
<div id =”...” class =”....”>
      <p>текст</p>
</div>
```

У каждого элемента в открывающем теге могут быть атрибуты. Атрибуты — это произвольный набор свойств (имя=значение). Часть атрибутов являются служебными и влияют на поведение/отображение элемента в браузере. Остальные можно использовать в своих целях. У некоторых элементов может быть только открывающий тег.

Основные служебные атрибуты

id

Уникальный идентификатор элемента (элементы с одинаковым id создавать не рекомендуется).

name

Имя элемента (элементы с одинаковым name создавать не рекомендуется, за одним исключением: когда создаются радиокнопки).

class

Произвольная метка элемента. У каждого элемента может быть несколько таких меток, разделённых пробелом. Разрешено использовать одинаковые метки для разных элементов. В основном используется для объединения элементов в группы. Например, для изменения визуальных стилей у всей группы элементов.

Узлы в объектной структуре связаны друг с другом. Существует несколько специальных терминов для описания отношений между узлами:

Родительский узел (parent node)

Родительским узлом по отношению к рассматриваемому объекту является узел, в который вложен рассматриваемый объект. На нашей схеме по отношению к узлам <img> и <p> **<body>** является родительским. Для узла <title> родительским является узел **<head>**.

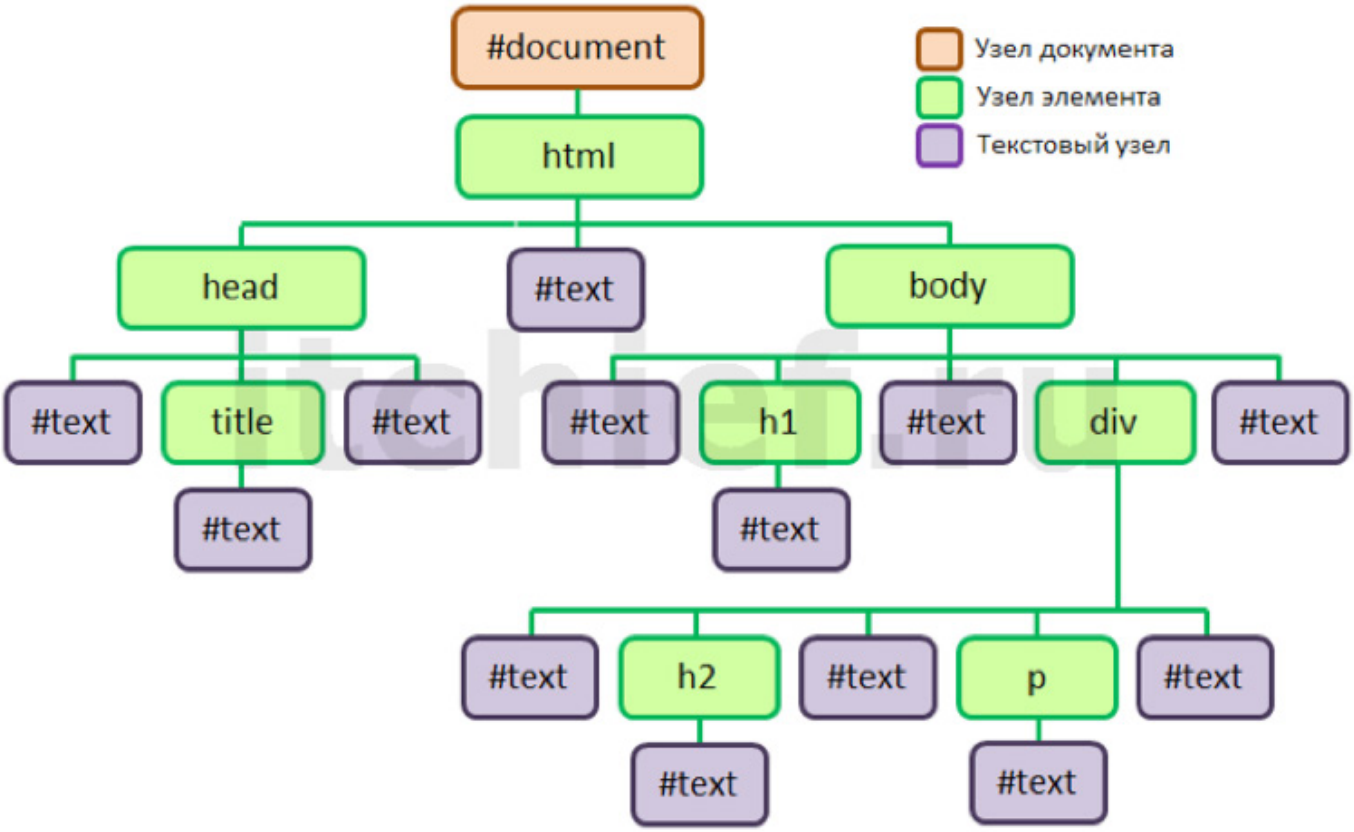
Узлы-потомки (child node)

Узлом-потомком по отношению к рассматриваемому объекту является узел, который вложен в рассматриваемый объект. На нашей схеме по отношению к узлу <body> **<img>** и **<p>** являются потомками. Для узла <head> потомком является **<title>**.

Узлы-братья (sibling node)

Узлы, находящиеся на одинаковом уровне вложенности по отношению к их родительскому узлу. На нашей схеме узлами-братьями являются **<body>** и **<head>, <p>** и **<img>**.

Давайте закрепим полученные знания следующей картинкой. Эти знания нам пригодятся далее, в манипуляциях с DOM-элементами. Например, нам нужно будет создать новый узел, удалить, поменять местами, добавить класс и т. д.


#document
html
head
#text
body
#text
title
#text
#text
#text
h1
#text
div
#text
#text
#text
h2
#text
p
#text
#text
#text
Узел документа
Узел элемента
Текстовый узел

2

DOM выборки и манипуляции

Все функции для выборки и поиска элементов хранятся внутри объекта **document**.

getElementById(id) — классический метод для поиска элементов по его id-атрибуту.

Следующие методы возвращают коллекцию элементов. Node-коллекции или NodeList — это некое подобие массивов, но не массив, а объект. Общим у node-коллекций с массивами является только свойство **length**, и то, что к их содержимому также можно обращаться по индексу. Но таких методов, как **pop, push** у node-коллекций нет.

getElementsByName(name)	найти элемент по его name-атрибуту
getElementsByClassName(className)	найти элемент по его классу
getElementsByTagName(tagName)	найти элемент по имени его тега

Пример:

```
document.getElementsByClassName('box');
```

*Найти все элементы в HTML-документе с классом **box***

Есть более универсальные методы — **querySelector** и **querySelectorAll**. Им в качестве параметра можно передать любой css-селектор. Разница между ними только в том, что первый метод вернёт только один элемент, удовлетворяющий селектору, а второй метод вернёт все элементы, удовлетворяющие селектору.

querySelector(selector)	найти первый элемент, подходящий под css-селектор
querySelectorAll(selector)	найти все элементы, подходящие под css-селектор

Также стоит отметить, что методы querySelector и querySelectorAll можно использовать не только на всём документе, но и на отдельных элементах, то есть выборку вы можете делать в пределах одного родительского элемента. Каждый элементный узел имеет набор предопределённых свойств и методов. С помощью этих свойств и методов вы можете производить различные манипуляции над содержимым элементных узлов объектной структуры.

Свойства и методы Node (узла)

childNodes	все прямые потомки узла
firstChild	первый прямой потомок узла
lastChild	последний прямой потомок узла
nextSibling	соседний узел, стоящий за текущим
previousSibling	соседний узел, стоящий перед текущим
nodeType	тип узла (текст, элемент, комментарий и т.д.)
parentNode	родительский узел
nodeValue	для узлов типа «текст» и «комментарии» хранит их содержимое
textContent	все дочерние текстовые узлы, объединённые вместе
appendChild(child)	добавить потомка child в узел (в конец узла)
insertBefore(what, before)	добавить потомка what в узел, перед другим потомком before внутри узла
removeChild(child)	удалить дочерний узел

Пример. Узнаем значение свойства первого потомка элемента с **id=par** и сохраним результат в переменной x:

```
var x = document.getElementById("par").childNodes[0].nodeValue;
```

Свойства и методы Element / HTMLElement (элемента)

id	содержит значение атрибута id
children	все прямые потомки узла, имеющие тип «элемент»
className	содержит значение атрибута class
firstElementChild	первый прямой потомок узла, имеющий тип «элемент»
lastElementChild	последний прямой потомок узла, имеющий тип «элемент»
nextElementSibling	соседний «элемент», стоящий за текущим
previousElementSibling	соседний «элемент», стоящий перед текущим
innerHTML	html-код содержимого текущего элемента
outerHTML	html-код текущего элемента и его содержимого
style	позволяет добавлять/удалять/просматривать css-стили (inline) элемента
tagName	имя тега узла
getAttribute(name)	получить значение атрибута по его имени
setAttribute(name, value)	установить значение атрибута value по его имени name
removeAttribute(name)	удалить атрибут по его имени
remove()	удалить элемент (вместе со всем его содержимым) из его родителя

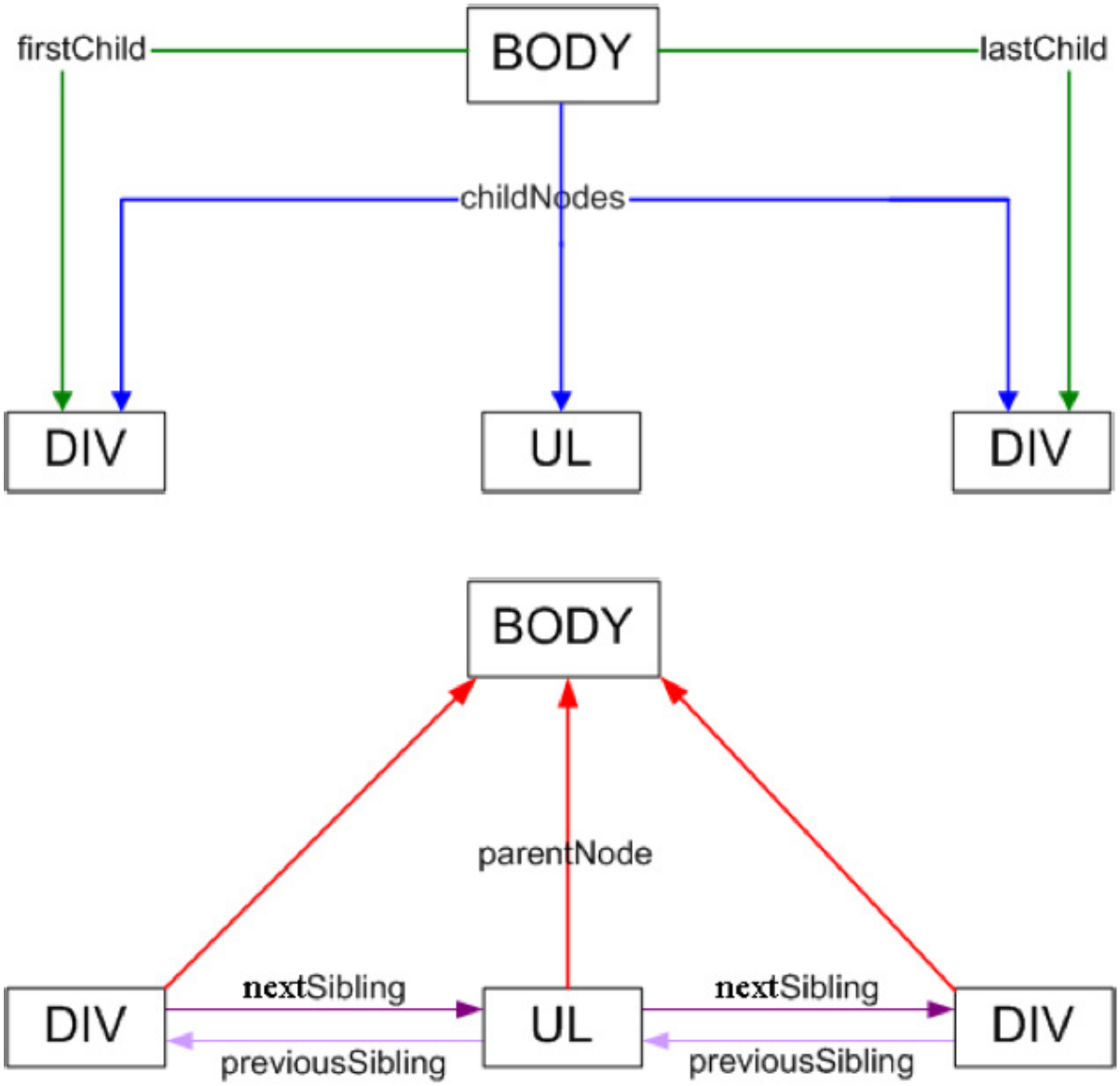
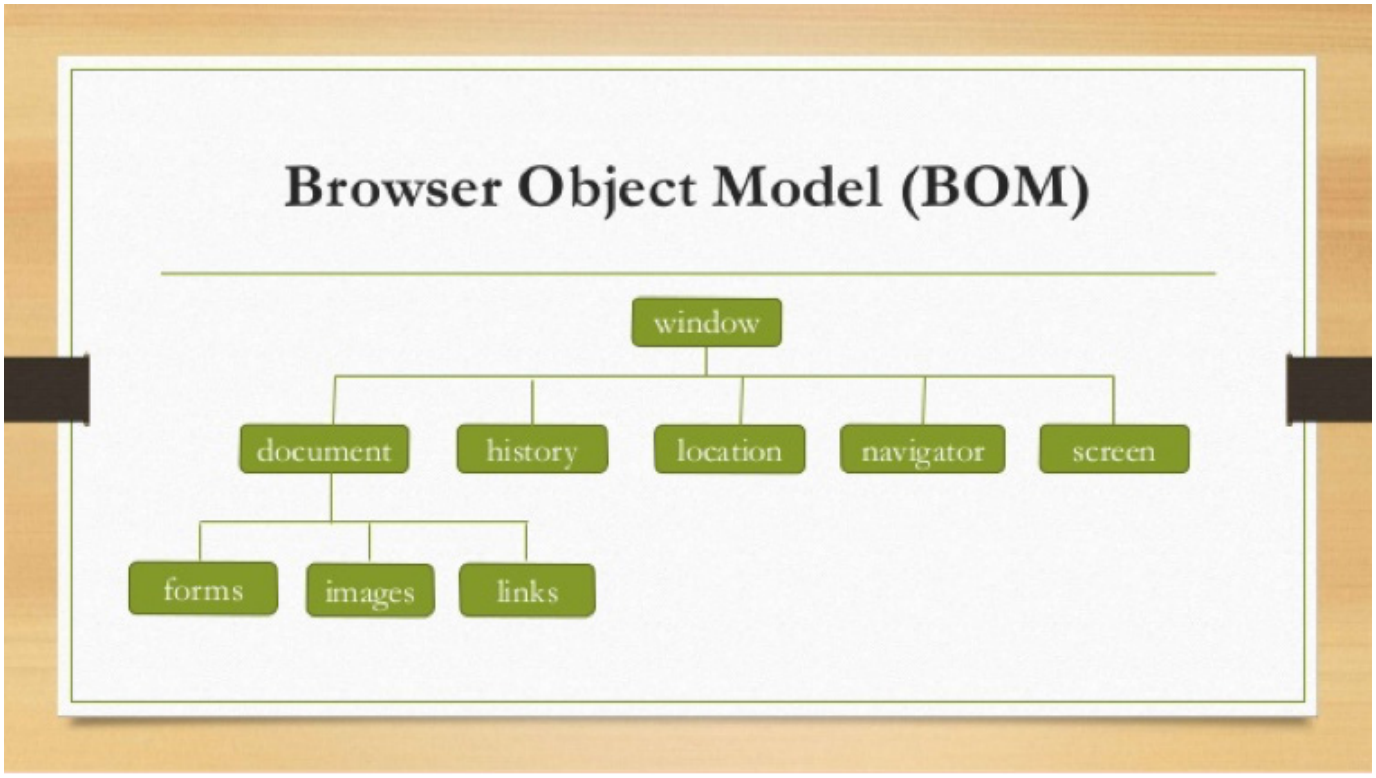


Схема основных свойств узлов для перехода между DOM-элементами

3

BOM объекты

С помощью объектной модели браузера (**BOM**) возможно управление поведением браузера из JavaScript. **BOM** включает в себя несколько объектов.



Объект **window** является корневым объектом JavaScript. Все объекты JavaScript, а также переменные и функции, определяемые пользователем, хранятся в объекте **window**.

Если вы знаете имя другого открытого окна, можете обращаться к объектам, созданным в другом окне.

Писать **"window."** при обращении к объектам и переменным необязательно, так как JavaScript подставляет его автоматически.

Объект window

С помощью метода ***alert()*** вы можете вывести окно оповещения.

Пример:

```
//Выведем окно оповещения
alert('Это окно оповещения');
```

С помощью метода ***confirm(question)*** можно показать окно с вопросом. Вернёт **true** или **false**, в зависимости от того, что нажал пользователь.

С помощью метода ***prompt(question, default)*** можно показать окно с полем для ввода текста. Если указан параметр **default**, то по умолчанию в текстовом поле будет именно это значение. Вернёт введённое значение или **null**, в случае отмены ввода.

С помощью метода ***open()*** вы можете открыть новое окно.

Пример:

```
//Откроем новое пустое окно
var nw = open();
//Выведем сообщение в новое окно
nw.document.write('Этот текст был выведен с помощью
JavaScript.');
```

С помощью метода ***close()*** вы можете закрыть окно.

Объект navigator

С помощью объекта **navigator** вы можете определить, какой браузер использует пользователь.

Так же с помощью **navigator** вы можете проверить, может ли пользователь принимать **cookie**.

С помощью свойства ***userAgent*** вы можете узнать всю информацию о браузере пользователя.

Пример:

```
//Выведем информацию о вашем браузере на страницу
document.write('Информация о вашем браузере: ' +
navigator.userAgent);
```

С помощью свойства ***cookieEnabled*** вы можете узнать, включена ли возможность использования **cookie** в браузере пользователя. Свойство возвращает **true**, если возможность использования **cookie** включена и **false** — если нет.

С помощью свойства ***appVersion*** вы можете узнать версию браузера.

Пример:

```
//Узнаем версию вашего браузера и выведем ее на экран
document.write(navigator.appVersion);
```

С помощью свойства ***appName*** вы можете узнать название браузера, а с помощью ***appCodeName*** кодовое название браузера.

С помощью свойства ***platform*** вы можете узнать платформу, т. е. узнать ОС, которую использует пользователь.

Объект Screen.

Объект **Screen** содержит информацию об экране пользователя.

С помощью свойств данного объекта вы можете узнать, какое разрешение, а также — какая глубина цвета установлена на экране пользователя.

Свойство ***width*** определяет ширину экрана пользователя, а свойство ***height*** — высоту.

Пример:

```
//Узнаем разрешение вашего экрана и выведем его
//на страницу
document.write('Разрешение экрана: <b>' + screen.width +
'X' + screen.height + '</b>');
```

С помощью свойства ***colorDepth*** вы можете узнать глубину цвета (измеряется в битах на пиксель), установленную у пользователя.

Объект History

Объект **History** содержит список URL, которые были посещены в данном окне браузера.

С помощью свойства ***length*** вы можете узнать количество посещённых URL, хранящихся в списке.

Пример:

```
//Выведем количество посещенных в окне браузера URL

document.write(history.length);
```

Вы можете перемещаться по списку посещённых URL с помощью метода ***go(смещение)***. Например, с помощью **go(2)** Вы можете переместиться на два URL вперёд, а **go(-3)** переместит Вас на три URL назад.

back()	вернуться на предыдущую страницу истории переходов
forward()	перейти на следующую страницу истории переходов

Объект Location

С помощью объекта **location** вы можете узнать любую информацию об URL текущего документа.

Свойства

hash	возвращает часть URL, начиная со знака **#**
host	возвращает часть URL, содержащую домен сайта
href	возвращает URL целиком
pathname	возвращает часть URL, содержащую путь к загруженному документу
protocol	возвращает часть URL, содержащую название протокола
search	возвращает часть URL, начиная со знака **?**

Методы

assign()	загружает новый документ в том же окне браузера
reload()	перезагружает документ

4

Таймеры

Установка таймера

Метод ***setTimeout*** выполняет код через заданный (в миллисекундах) промежуток времени. **setTimeout** имеет два параметра: **функцию**, выполнение которой должно быть отложено и **количество миллисекунд**, на которые нужно отложить выполнение функции.

Функция, указанная первым аргументом, выполнится через количество миллисекунд, указанное вторым аргументом.

Функция, передаваемая в **setTimeout**, называется **callback-функцией**.

callback-функция — это функция, которая будет вызвана в будущем, когда произойдет какое-то событие. В случае с **setTimeout** — когда подойдет время таймера.

Пример:

```
function start()
{
   console.log('Прошло три секунды');
}
setTimeout(start,3000);
```

Отмена таймера

Функция **setTimeout** возвращает числовой идентификатор таймера **timerId**, который можно передать в функцию ***clearTimeout*** для отмены действия.

```
var timerId = setTimeout(function() {
//....
}, 1000);
clearTimeout(timerId);
```

Интервалы

setInterval запускает выполнение функции не один раз, а регулярно повторяет её через указанный интервал времени.

Следующий пример при запуске станет выводить сообщение каждые две секунды:

```
// Начать повторы с интервалом 2 сек
var timerId = setInterval(function() {
  alert("шаг");
}, 2000);
```

Остановить исполнение можно вызовом **clearInterval(timerId)**.

Рекурсивный setTimeout

Альтернатива **setInterval** — рекурсивный **setTimeout**. Рекурсивный **setTimeout** — более гибкий метод тайминга, чем **setInterval**, так как время до следующего выполнения можно запланировать по-разному, в зависимости от результатов текущего:

```
var timerId = setTimeout(function step() {
  alert("step");
  timerId = setTimeout(tick, 4000);
}, 2000);
```

Резюме:

setTimeout(fn, milliseconds) — выполнить функцию один раз, через определённый промежуток времени (возвращает идентификатор таймера).

clearTimeout(timerId) — удалить таймер, созданный с помощью функции **setTimeout**.

setInterval(fn, milliseconds) — выполнять функцию с указанным интервалом, пока таймер не будет удалён (возвращает идентификатор таймера).

clearInterval(timerId) — удалить таймер, созданный с помощью функции **setInterval**.

5

События (DOM Events)

Механизм DOM-события призван помочь разработчику перехватить и обработать различные действия пользователей (клики мышкой по элементам, нажатия клавиш и прочее).

Почти все события состоят из двух фаз:

- **capturing** (захват)
- **bubbling** (всплытие)

Когда происходит событие, информация о нем спускается от корневого элемента DOM-дерева вниз, до того элемента, на котором произошло событие (например, если кликнули по ссылке, то событие спускается от корня дерева до ссылки). Этот процесс называется фазой захвата.

После этого процесс начинается в обратном порядке. То есть информация о событии поднимается от элемента, на котором произошло событие до корневого элемента DOM-дерева. Этот процесс называется фазой всплытия.

После того, как событие «всплывёт», происходит действие по умолчанию. Например, для ссылок действием по умолчанию является переход на сайт, заданный в атрибуте **href**.

Для того, чтобы добавить обработчик события на элемент, есть несколько способов.

Первый — добавить атрибут **on*** (вместо звездочки подставить имя события).

```
<button onclick="alert('click'">click me</button>
```

Здесь мы добавили обработчик кликов мышкой по элементу. Таким образом, при клике мышкой по элементу сработает код, который указан в значении атрибута **onclick** (в нашем случае это **alert('click')**). Мы не рекомендуем добавлять таким образом обработчики события, т. к. это считается плохим

стилем и влечёт за собой путаницу между логикой и ее представлением.

Второй способ — присвоить в свойство **on*** элемента функцию, которая отработает при наступлении события. Например:

```
<button>click me</button>
<script>
   var button = document.querySelector('button');
   // находим элемент
   button.onclick = function() {
       alert('click');
   }
</script>
```

Этот вариант лучше первого, но существует недостаток — мы не можем добавить несколько обработчиков событий.

Третий вариант, который мы рассмотрим — установка обработчиков через метод **addEventListener**.

Это наиболее предпочтительный вариант. Мы рекомендуем использовать именно его. При помощи данного метода вы можете задавать несколько обработчиков события.

```
var button = document.querySelector('button');
button.addEventListener('click', function() {
   alert('click');
})
button.addEventListener('click', function() {
   alert('second');
})
```

При клике на кнопку сработают оба обработчика. В том порядке, в котором

они были установлены. Первым параметром необходимо передать имя события, на которое нужно установить обработчик. Вторым параметром — функцию, которая будет выполнять роль обработчика.

Для того, чтобы удалить обработчик события с элемента, необходимо воспользоваться методом **removeEventListener**:

```
button.removeEventListener('click', handler);
```

Когда браузер вызывает обработчик события, он передаёт в эту функцию первым параметром специальный объект с описанием события. Вот основные свойства этого объекта:

type	имя события
target	элемент, для которого изначально было предназначено событие
currentTarget	элемент, который перехватил событие в данный момент
eventPhase	фаза события (захват, выполнение, всплытие)

```
button.addEventListener('click', function(e){
   console.log(e);
});
```

У данного объекта так же есть метод **preventDefault**. У html-элементов есть действие по умолчанию. Как мы уже упоминали, при клике на элементы **<a>** происходит переход по ссылке из атрибута href. При нажатии клавиш в текстовом поле — вывод введённых символов.

Механизм событий позволяет отменить действие по умолчанию при помощи вызова метода **preventDefault** из объекта с информацией о событии.

Например, вот как можно запретить переход по ссылке:

```
<a href="http://ya.ru" id="link">link</a>
<script>
   link.addEventListener('click', function(e){
       e.preventDefault();
   });
</script>
```

Теперь при клике по ссылке перехода осуществлено не будет.

Рассмотрим основные события JavaScript.

События мыши

mousedown	нажата кнопка мыши
mouseup	отпущена кнопка мыши
click	клик кнопкой мыши
dblclick	двойной клик кнопкой мыши
mouseover	курсор вошёл в рамки элемента
mouseenter	курсор вошёл в рамки элемента (событие не всплывает)
mousemove	курсор двигается над поверхностью элемента
mouseout	курсор вышел за рамки элемента
mouseleave	курсор вышел за рамки элемента (событие не всплывает)
contextmenu	вызов контекстного меню правой кнопкой мыши
wheele	вращение колеса мыши (имеет разные реализации в браузерах)

Наверное, одно из самых распространённых событий — это клик «мышкой» по элементу. Пример:

```
<button>Click me</button>
<script>
   var button = document.querySelector('button');
   // отобрали кнопку
   button.addEventListener('click', function() {
        // вешаем обработчик события на клик
        alert('click'); // выводим сообщение по клику
   });
</script>
```

События клавиатуры

keydown	была нажата клавиша
keypress	отрабатывает по нажатию на клавишу. В отличии от **keydown**, событие **keypress** срабатывает только на те клавиши, которые печатают символы. То есть такие клавиши, как **ctrl, alt, shift** не входят в эту категорию, так как не печатают символы.
keyup	была отпущена клавиша

Пример:

```
<input type="text" />
<script>
   var input = document.querySelector('input');
   input.addEventListener('keydown', function() {
   // по нажатию на клавишу в текстовом поле отработает alert
        alert('key');
   })
</script>
```

Некоторые свойства, характерные для событий:

keyCode	содержит код клавиши (в случае с **keypress** — не знает код клавиши, но знает ASCII-код символа)
altKey	была ли в момент события зажата клавиша **alt**
shiftKey	была ли в момент события зажата клавиша **shift**
ctrlKey	была ли в момент события зажата клавиша **ctrl**

Загрузка страницы. Загрузка документа делится на два этапа:

- загрузка html и скриптов;

- загрузка внешних ресурсов.

Каждый из этих этапов можно перехватить и обработать.

DOMContentLoaded — событие, которое сработает после того, как весь html будет загружен браузером и js в документе будет выполнен. То есть все DOM-элементы будут созданы и доступны через JS. Данное событие срабатывает на объекте **document**:

```
document.addEventListener("DOMContentLoaded", function()
{
   console.log("html структура загружена");
});
```

load — событие, которое сработает, когда браузер загрузит все внешние ресурсы (картинки, css) и страница будет полностью готова к работе. Данное событие срабатывает на объекте **window**:

```
window.addEventListener('load', function() {
   console.log("страница полностью готова");
});
```

Список всех событий JavaScript можно будет найти по этой ссылке.

6

Список рекомендуемых источников

- Современный учебник JavaScript
- ОСНОВЫ JAVASCRIPT
- ПРОДВИНУТЫЙ JAVASCRIPT
- Javascript-джедай
- JS Учебник