# Основы JavaScript

**JavaScript** создавался как скриптовый язык для браузеров Netscape. Компания Microsoft также признала его потенциал и включила под именем **JScript** в Internet Explorer 3, обеспечив частичную поддержку стандартов языка, что привело в итоге к неразберихе со стандартами и версиями JavaScript. Поэтому Netscape, Microsoft и другие заинтересованные компании обратились в организацию ECMA (Европейская ассоциация производителей компьютеров), где была одобрена первая спецификация языка ECMA-262. В связи с тем, что название «JavaScript» являлось зарегистрированным товарным знаком, для нового стандарта было решено использовать ECMAScript (или сокращенно ES). ECMAScript изначально был разработан для использования в качестве языка сценариев, но позже стал широко использоваться в качестве языка программирования общего назначения.

В основу создания JavaScript была положена идея динамического управления объектами HTML-документов без перезагрузки текущей страницы (так называемые бессерверные сценарии). Со временем возможности языка расширились:

* С помощью JavaScript можно запускать одностраничные приложения на стороне клиента;
* JavaScript используется на стороне сервера с такими технологиями, как Node.js;
* JavaScript помогает создавать настольные приложения с Electron и может использоваться для работы с одноплатными компьютерами типа Raspberry Pi;
* Также, JavaScript используется для обучения моделей машинного обучения в браузере с помощью технологии TensorFlow.js.

#### Что такое JavaScript?

1) JavaScript — язык сценариев, или скриптов. Скрипт представляет собой программный код — набор инструкций, который не требует предварительной обработки (например, компиляции) перед запуском. Код JavaScript интерпретируется движком браузера во время загрузки веб-страницы. Интерпретатор браузера выполняет построчный анализ, обработку и выполнение исходной программы или запроса.

2) JavaScript — объектно-ориентированный язык с прототипным наследованием. Он поддерживает несколько встроенных объектов, а также позволяет создавать или удалять свои собственные (пользовательские) объекты. Объекты могут наследовать свойства непосредственно друг от друга, образуя цепочку объект-прототип.

## JavaScript на веб-страницах

* **СОДЕРЖАНИЕ:**
* [1. Подключение сценариев к html-документу](https://html5book.ru/osnovy-javascript/#part1)
* [2. Типы данных и переменные в JavaScript](https://html5book.ru/osnovy-javascript/#part2)
* [2.1. Переменные в JavaScript](https://html5book.ru/osnovy-javascript/#part3)
* [2.2. Типы данных переменных](https://html5book.ru/osnovy-javascript/#part4)
* [2.3. Глобальные и локальные переменные](https://html5book.ru/osnovy-javascript/#part5)
* [3. Ключевые слова JavaScript](https://html5book.ru/osnovy-javascript/#part6)
* [4. Оформление имен переменных и функций в коде JavaScript](https://html5book.ru/osnovy-javascript/#part7)

### 1. Подключение сценариев к html-документу

Сценарии JavaScript бывают **встроенные**, т.е. их содержимое является частью документа, и **внешние**, хранящиеся в отдельном файле с расширением .js. Сценарии можно внедрить в html-документ следующими способами:

#### 1.1. В виде гиперссылки

Для этого нужно разместить код в отдельном файле и включить ссылку на файл в заголовок

<head>

<script src="script.js"></script>

</head>

HTML

или тело страницы.

<body>

<script src="script.js"></script>

</body>

HTML

Этот способ обычно применяется для сценариев большого размера или сценариев, многократно используемых на разных веб-страницах.

#### 1.2. В виде обработчика события

Каждый html-элемент имеет JavaScript-события, которые срабатывают в определенный момент. Нужно добавить необходимое событие в html-элемент как атрибут, а в качестве значения этого атрибута указать требуемую функцию. Функция, вызываемая в ответ на срабатывание события, является **обработчиком события**. В результате срабатывания события исполнится связанный с ним код. Этот способ применяется в основном для коротких сценариев, например, можно установить смену цвета фона при нажатии на кнопку:

<script>

var colorArray = ["#5A9C6E", "#A8BF5A", "#FAC46E", "#FAD5BB", "#F2FEFF"]; // создаем массив с цветами фона

var i = 0;

function changeColor(){

document.body.style.background = colorArray[i];

i++;

if( i > colorArray.length - 1){

i = 0;

}

}

</script>

<button onclick="changeColor();">Change background</button>

HTML

#### 1.3. Внутрь элемента <script>

Элемент <script> может вставляться в любое место документа. Внутри тега располагается код, который выполняется сразу после прочтения браузером, или содержит описание функции, которая выполняется в момент ее вызова. Описание функции можно располагать в любом месте, главное, чтобы к моменту ее вызова код функции уже был загружен.

Обычно код JavaScript размещается в заголовке документа (элемент <head>) или после открывающего тега <body>. Если скрипт используется после загрузки страницы, например, код счетчика, то его лучше разместить в конце документа:

<footer>

<script>

document.write("Введите свое имя");

</script>

</footer>

</body>

HTML

### 2. Типы данных и переменные в JavaScript

Компьютеры обрабатывают информацию — данные. Данные могут быть представлены в различных формах или типах. Большая часть функциональности JavaScript реализуется за счет простого набора объектов и типов данных. Функциональные возможности, связанные со строками, числами и логикой, базируются на строковых, числовых и логических типах данных. Другая функциональная возможность, включающая регулярные выражения, даты и математические операции, осуществляется с помощью объектов RegExp, Date и Math.

**Литералы** в JavaScript представляют собой особый класс типа данных, фиксированные значения одного из трех типов данных — строкового, числового или логического:

"это строка"

3.14

true

alert("Hellow"); // "Hellow" - это литерал

var myVariable = 15; // 15 - это литерал

JavaScript

**Примитивный тип данных** является экземпляром определенного типа данных, таких как строковый, числовой, логический, null и undefined.

### 2.1. Переменные в JavaScript

Данные, обрабатываемые сценарием JavaScript, являются **переменными**. Переменные представляют собой именованные контейнеры, хранящие данные (значения) в памяти компьютера, которые могут изменяться в процессе выполнения программы. Переменные имеют **имя**, **тип** и **значение**.

Имя переменной, или **идентификатор**, может включать только буквы a-z, A-Z, цифры 0-9 (цифра не может быть первой в имени переменной), символ $ (может быть только первым символом в имени переменной или функции) и символ подчеркивания \_, наличие пробелов не допускается. Длина имени переменной не ограничена. Можно, но не рекомендуется записывать имена переменных буквами русского алфавита, для этого они должны быть записаны в Unicode.

В качестве имени переменной нельзя использовать ключевые слова JavaScript. Имена переменных в JavaScript чувствительные к регистру, что означает, что переменная var message; и var Message; — разные переменные.

Переменная создается (объявляется) с помощью ключевого слова var, за которым следует имя переменной, например, var message;. Объявлять переменную необходимо перед ее использованием.

Переменная **инициализируется** значением с помощью операции присваивания =, например, var message="Hellow";, т.е. создается переменная message и в ней сохраняется ее **первоначальное** значение "Hellow". Переменную можно объявлять без значения, в этом случае ей присваивается значение по умолчанию undefined. Значение переменной может изменяться во время исполнения скрипта. Разные переменные можно объявлять в одной строке, разделив их запятой:

var message="Hellow", number\_msg = 6, time\_msg = 50;

JavaScript

#### 2.2. Типы данных переменных

JavaScript является нетипизированным языком, тип данных для конкретной переменной при ее объявлении указывать не нужно. Тип данных переменной зависит от значений, которые она принимает. Тип переменной может изменяться в процессе совершения операций с данными (**динамическое приведение типов**). Преобразование типов выполняется автоматически в зависимости от того, в каком контексте они используются. Например, в выражениях, включающих числовые и строковые значения с оператором +, JavaScript преобразует числовые значения в строковые:

var message = 10 + " дней до отпуска"; // вернет "10 дней до отпуска"

JavaScript

Получить тип данных, который имеет переменная, можно с помощью оператора typeof. Этот оператор возвращает строку, которая идентифицирует соответствующий тип.

typeof 35; // вернет "number"

typeof "text"; // вернет "string"

typeof true; // вернет "boolean"

typeof [1, 2, 4]; // вернет "object"

typeof undefined; // вернет "undefined"

typeof null; // вернет "object"

JavaScript

Все типы данных в JavaScript делятся на две группы — **простые** типы данных (primitive data types) и **составные** типы данных (composite data types).

К **простым** типам данных относят строковый, числовой, логический, null и underfined.

##### 2.2.1. Строковый тип (string)

Используется для хранения строки символов, заключенных в двойные или одинарные кавычки. Пустой набор символов, заключенный в одинарные или двойные кавычки, является пустой строкой. Число, заключенное в кавычки, также является строкой.

var money = ""; // пустая строка, ноль символов

var work = 'test';

var day = "Sunday";

var x = "150";

JavaScript

В строку в двойных кавычках можно включить одиночную кавычку и наоборот. Кавычка того же типа отключается с помощью символа обратного слэша \ (так называемая **escape-последовательность**):

document.writeln("\"Доброе утро, Иван Иваныч!\"\n"); // выведет на экран "Доброе утро, Иван Иваныч!"

JavaScript

Строки можно сравнивать, а также объединять с помощью операции конкатенации +. Благодаря автоматическому приведению типов можно объединять числа и строки. Строки являются постоянными, после того, как строка создана, она не может быть изменена, но может быть создана новая строка путем объединения других строк.

##### 2.2.2. Числовой тип (number)

Используется для числовых значений. Числа в языке JavaScript бывают двух типов: целые числа (integer) и числа с плавающей точкой (floating-point number). Целочисленные величины могут быть положительными, например 1, 2, и отрицательными, например –1, –2, или равными нулю. 1 и 1.0 — одно и то же значение. Большинство чисел в JavaScript записываются в десятичной системе счисления, также может использоваться восьмеричная и шестнадцатеричная системы.

В **десятичной** системе значения числовых переменных задаются с использованием арабских цифр 1, 2, 3, 4, 5, 6, 7, 8, 9, 0.

В **восьмеричном** формате числа представляет собой последовательность, содержащая цифры от 0 до 7 и начинающаяся с префикса 0.

Для **шестнадцатеричного** формата добавляется префикс 0x (0X), за которым следует последовательность из цифр от 0 до 9 или букв от a (A) до f (F), соответствующие значениям от 10 до 15.

var a = 120; // целое десятичное числовое значение

var b = 012; // восьмеричный формат

var c = 0xfff; // шестнадцатеричный формат

var d = 0xACFE12; // шестнадцатеричный формат

JavaScript

Числа с плавающей точкой представляют собой числа с дробной десятичной частью, либо это числа, выраженные в экспоненциальном виде. Экспоненциальная запись чисел предполагает следующий вид: число с дробной десятичной частью, за ним следует буква e, которая может быть указана как в верхнем, так и в нижнем регистре, далее — необязательный знак + или - и целая экспонента.

var a = 6.24; // вещественное число

var b = 1.234E+2; // вещественное число, эквивалентно 1.234 Х 10²

var c = 6.1e-2; // вещественное число, эквивалентно 6.1 Х 10‾²

JavaScript

##### 2.2.3. Логический тип (boolean)

Данный тип имеет два значения, true (истина), false (ложь). Используется для сравнения и проверки условий.

var answer = confirm("Вам понравилась эта статья?\n Нажмите ОК. Если нет, то нажмите Cancel.");

if (answer == true)

{

alert("Спасибо!");

}

JavaScript

Также существуют специальные типы простых значений:

**нулевой тип** — данный тип имеет одно значение null, которое используется для представления несуществующих объектов.

**неопределенный тип** — тип переменной underfined означает отсутствие первоначального значения переменной, а также несуществующее свойство объекта.

**Составные типы данных** состоят из более чем одного значения. К ним относятся объекты и особые типы объектов — массивы и функции. Объекты содержат свойства и методы, массивы представляют собой индексированный набор элементов, а функции состоят из коллекции инструкций.

#### 2.3. Глобальные и локальные переменные

Переменные по области видимости делятся на **глобальные** и **локальные**. **Область видимости** представляет собой часть сценария, в пределах которой имя переменной связано с этой переменной и возвращает ее значение. Переменные, объявленные внутри тела функции, называются **локальными**, их можно использовать только в этой функции. Локальные переменные создаются и уничтожаются вместе с соответствующей функцией.

Переменные, объявленные внутри элемента <script>, или внутри функции, но без использования ключевого слова var, называются **глобальными**. Доступ к ним может осуществляться на протяжении всего времени, пока страница загружена в браузере. Такие переменные могут использоваться всеми функциями, позволяя им обмениваться данными.

Глобальные переменные попадают в **глобальное пространство имен**, которое является местом взаимодействия отдельных компонентов программы. Не рекомендуется объявлять переменные таким способом, так как аналогичные имена переменных уже могут использоваться любым другим кодом, вызывая сбой в работе скрипта.

Глобальное пространство в JavaScript представляется глобальным объектом window. Добавление или изменение глобальных переменных автоматически обновляет глобальный объект. В свою очередь, обновление глобального объекта автоматически приводит к обновлению глобального пространства имен.

Если глобальная и локальная переменная имеют одинаковые имена, то локальная переменная будет иметь преимущество перед глобальной.

Локальные переменные, объявленные внутри функции в разных блоках кода, имеют одинаковые области видимости. Тем не менее, рекомендуется помещать объявления всех переменных в начале функции.

### 3. Ключевые слова JavaScript

**Ключевое слово** представляет собой зарезервированное слово языка JavaScript. Зарезервированные слова обозначают операторы, типы данных, функции, методы и т.д.

|  |  |  |
| --- | --- | --- |
| ТАБЛИЦА 1. КЛЮЧЕВЫЕ СЛОВА JAVASCRIPT | | |
| break | finally | return |
| catch | for | static |
| class | function | super |
| case | if | switch |
| const | implements | this |
| continue | import | throw |
| debugger | in | true |
| default | instanceof | try |
| delete | interface | typeof |
| do | let | var |
| else | new | void |
| enum | package | while |
| export | private | with |
| extends | protected | yield |
| false | public |  |

### 4. Оформление имен переменных и функций в коде JavaScript

Функции-конструкторы (которые вызываются вместе с оператором new), начинаются с заглавной буквы:  
var boy = new Person();

Если имя переменной или функции-конструктора состоит из нескольких слов, первые буквы каждого слова вводятся в верхнем регистре:  
MyConstructor().

Имена обычных функций и методов — первый символ вводится в нижнем регистре:  
getLastName().

Переменные, которые не являются функциями, вводятся строчными буквами, при этом слова разделяются символом подчеркивания:  
first\_name/

Имена переменных, значения которых неизменны на протяжении всего времени выполнения сценария, а также имена глобальных переменных, записываются заглавными буквами:  
var PI = 3.14;  
MAX\_COST = 800;

# 

# Выражения в JavaScript

**Выражения в JavaScript** представляют собой комбинации операндов и операторов.

Операции в выражениях выполняются последовательно в соответствии со значением приоритета (чем больше значение приоритета, тем он выше). Возвращаемый результат не всегда имеет значение того же типа, что и тип обрабатываемых данных. Например, в операциях сравнения участвуют операнды различных типов, но возвращаемый результат всегда будет логического типа.

РИС. 1. СТРУКТУРА ВЫРАЖЕНИЯ В JAVASCRIPT

**Операнды** — это данные, обрабатываемые сценарием JavaScript. В качестве операндов могут быть как простые типы данных, так и сложные, а также другие выражения.

**Операторы** — это символы языка, выполняющие различные операции с данными. Операторы могут записываться с помощью символов пунктуации или ключевых слов.

В зависимости от количества операндов различают следующие типы операторов:  
**унарный** — в операции участвует один операнд;  
**бинарный** — в операции участвуют два операнда;  
**тернарный** — комбинирует три операнда.

Простейшая форма выражения — **литерал** — нечто, вычисляемое само в себя, например, число 100, строка "Hellow world". Переменная тоже может быть выражением, так как она вычисляется в присвоенное ей значение.

## Выражения и операторы в JavaScript

* **СОДЕРЖАНИЕ:**
* [1. Арифметические операторы](https://html5book.ru/vyrazheniya-v-javascript/#part1)
* [2. Операторы присваивания](https://html5book.ru/vyrazheniya-v-javascript/#part2)
* [3. Операторы инкремента и декремента](https://html5book.ru/vyrazheniya-v-javascript/#part3)
* [4. Операторы сравнения](https://html5book.ru/vyrazheniya-v-javascript/#part4)
* [5. Логические операторы](https://html5book.ru/vyrazheniya-v-javascript/#part5)
* [6. Побитовые операторы](https://html5book.ru/vyrazheniya-v-javascript/#part6)
* [7. Строковые операторы](https://html5book.ru/vyrazheniya-v-javascript/#part7)
* [8. Специальные операторы](https://html5book.ru/vyrazheniya-v-javascript/#part8)
* [9. Комментарии в JavaScript](https://html5book.ru/vyrazheniya-v-javascript/#part9)

### 1. Арифметические операторы

**Арифметические операторы** предназначены для выполнения математических операций, они работают с числовыми операндами (или переменными, хранящими числовые значения), возвращая в качестве результата числовое значение.

Если один из операндов является строкой, интерпретатор JavaScript попытается преобразовать его в числовой тип, а после выполнить соответствующую операцию. Если преобразование типов окажется невозможным, будет получен результат NaN (не число).

|  |  |  |
| --- | --- | --- |
| ТАБЛИЦА 1. АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ | | |
| **Оператор/Операция** | **Описание** | **Приоритет** |
| + Сложение | Складывает числовые операнды. Если один из операндов — строка, то результатом выражения будет строка. | 12 |
| - Вычитание | Выполняет вычитание второго операнда из первого. | 12 |
| - Унарный минус | Преобразует положительное число в отрицательное, и наоборот. | 14 |
| \* Умножение | Умножает два операнда. | 13 |
| / Деление | Делит первый операнд на второй. Результатом деления может являться как целое, так и число с плавающей точкой. | 13 |
| % Деление по модулю (остаток от деления) | Вычисляет остаток, получаемый при целочисленном делении первого операнда на второй. Применяется как к целым числам, так и числам с плавающей точкой. | 13 |

var x = 5, y = 8, z;

z = x + y; // вернет 13

z = x - y; // вернет -3

z = - y; // вернет -8

z = x \* y; // вернет 40

z = x / y; // вернет 0.625

z = y % x; // вернет 3

JavaScript

### 2. Операторы присваивания

**Операторы присваивания** используются для присваивания значений переменным. Комбинированные операторы позволяют сохранить первоначальное и последующее значение в одной переменной.

|  |  |  |
| --- | --- | --- |
| ТАБЛИЦА 2. ОПЕРАТОРЫ ПРИСВАИВАНИЯ | | |
| **Оператор/Операция** | **Описание** | **Приоритет** |
| = Присваивание | Используется для присваивания значения переменной. | 2 |
| +=, -=, \*=, /=, %= Комбинированный оператор | Выполняет присваивание с операцией. Между первым и вторым операндом выполняется соответствующая операция, затем результат присваивается первому операнду. | 2 |

var a = 5; // присваиваем переменной a числовое значение 5

var b = "hellow"; // сохраняем в переменной b строку hellow

var m = n = z = 10; // присваиваем переменным m, n, z числовое значение 10

x += 10; // равнозначно x = x + 10;

x -= 10; // равнозначно x = x - 10;

x \*= 10; // равнозначно x = x \* 10;

x /= 10; // равнозначно x = x / 10;

x %= 10; // равнозначно x = x % 10;

JavaScript

### 3. Операторы инкремента и декремента

Операции **инкремента и декремента** являются унарными и производят увеличение и уменьшение значения операнда на единицу. В качестве операнда может быть переменная, элемент массива, свойство объекта. Чаще всего такие операции используются для увеличения счетчика в цикле.

|  |  |  |
| --- | --- | --- |
| ТАБЛИЦА 3. ОПЕРАТОРЫ ИНКРЕМЕНТА И ДЕКРЕМЕНТА | | |
| **Оператор/Операция** | **Описание** | **Приоритет** |
| ++x Префиксный инкремент | Увеличивает операнд на единицу. | 14 |
| x++ Постфиксный инкремент | Прибавляет к операнду единицу, но результатом выражения будет являться первоначальное значение операнда. | 14 |
| --x Префиксный декремент | Уменьшает на единицу операнд, возвращая уменьшенное значение. | 14 |
| x-- Постфиксный декремент | Уменьшает на единицу операнд, возвращая первоначальное значение. | 14 |

var x = y = m = n = 5, z, s, k, l;

z = ++x \* 2; /\* в результате вычислений вернет значение z = 12, x = 6, т.е. значение x сначала увеличивается на 1, а после выполняется операция умножения \*/

s = y++ \* 2; /\* в результате вычислений вернет значение s = 10, y = 6, т.е. сначала выполняется операция умножения, а после в переменной y сохраняется увеличенное на 1 значение \*/

k = --m \* 2; // вернет значение k = 8, m = 4

l = n-- \* 2; // вернет значение l = 10, n = 4

JavaScript

### 4. Операторы сравнения

**Операторы сравнения** используются для сопоставления операндов, результатом выражения может быть одно из двух значений — true или false. Операндами могут быть не только числа, но и строки, логические значения и объекты. Однако сравнение может выполняться только для чисел и строк, поэтому операнды, не являющиеся числами или строками, преобразуются.

Если оба операнда не могут быть успешно преобразованы в числа или строки, операторы всегда возвращают false.

Если оба операнда являются строками/числами или могут быть преобразованы в строки/числа, они будут сравниваться как строки/числа.

Если один операнд является строкой/преобразуется в строку, а другой является числом/преобразуется в число, то оператор попытается преобразовать строку в число и выполнить сравнение чисел. Если строка не является числом, она преобразуется в значение NaN и результатом сравнения будет false.

Чаще всего операции сравнения используются при организации ветвлений в программах.

|  |  |  |
| --- | --- | --- |
| ТАБЛИЦА 4. ОПЕРАТОРЫ СРАВНЕНИЯ | | |
| **Оператор/Операция** | **Описание** | **Приоритет** |
| == Равенство | Проверяет две величины на совпадение, допуская преобразование типов. Возвращает true, если операнды совпадают, и false, если они различны. | 9 |
| != Неравенство | Возвращает true, если операнды не равны | 9 |
| === Идентичность | Проверяет два операнда на «идентичность», руководствуясь строгим определением совпадения. Возвращает true, если операнды равны без преобразования типов. | 9 |
| !== Неидентичность | Выполняет проверку идентичности. Возвращает true, если операнды не равны без преобразования типов. | 9 |
| > Больше | Возвращает true, если первый операнд больше второго, в противном случае возвращает false. | 10 |
| >= Больше или равно | Возвращает true, если первый операнд не меньше второго, в противном случае возвращает false. | 10 |
| < Меньше | Возвращает true, если первый операнд меньше второго, в противном случае возвращает false. | 10 |
| <= Меньше или равно | Возвращает true, если первый операнд не больше второго, в противном случае возвращает false. | 10 |

5 == "5"; // вернет true

5 != -5.0; // вернет true

5 === "5"; // вернет false

false === false; // вернет true

1 !== true; // вернет true

1 != true; // вернет false, так как true преобразуется в 1

3 > -3; // вернет true

3 >= "4"; // вернет false

JavaScript

### 5. Логические операторы

**Логические операторы** позволяют комбинировать условия, возвращающие логические величины. Чаще всего используются в условном выражении if.

|  |  |  |
| --- | --- | --- |
| ТАБЛИЦА 5. ЛОГИЧЕСКИЕ ОПЕРАТОРЫ | | |
| **Оператор/Операция** | **Описание** | **Приоритет** |
| && Логическое И | Возвращает true, только если оба операнда истинны. При выполнении операции сначала проверяется значение первого операнда. Если оно имеет значение false, то значение второго оператора не учитывается и результату выражения присваивается false. | 5 |
| || Логическое ИЛИ | Возвращает true, если хотя бы один операнд истинен, т.е. проверяет истинность как минимум одного условия. | 4 |
| ! Логическое НЕ | Изменяет значение оператора на обратное - с true на false и наоборот. | 14 |

(2 < 3) && (3===3); // вернет true, так как выражения в обеих скобках дают true

(x < 10 && x > 0); // вернет true, если значение x принадлежит промежутку от 0 до 10

!false; // вернет true

JavaScript

### 6. Побитовые операторы

**Побитовые операторы** работают с операндами как с 32-битной последовательностью нулей и единиц и возвращают числовое значение, означающее результат операции, записанное в десятичной системе счисления. В качестве операндов рассматриваются целые числа, дробная часть операнда отбрасывается. Побитовые операции могут использоваться, например, при шифровании данных, для работы с флагами, разграничения прав доступа.

|  |  |  |
| --- | --- | --- |
| ТАБЛИЦА 6. ПОБИТОВЫЕ ОПЕРАТОРЫ | | |
| **Оператор/Операция** | **Описание** | **Приоритет** |
| & Побитовый И | Если оба бита равны 1, то результирующий бит будет равен 1. В противном случае результат равен 0. | 8 |
| | Побитовый ИЛИ | Если один из операндов содержит в позиции 1, результат тоже будет содержать 1 в этой позиции, в противном случае результат в этой позиции будет равен 0. | 6 |
| ^ Исключающее ИЛИ | Если одно, и только одно значение содержит 1 в какой-либо позиции, то и результат будет содержать 1 в этой позиции, в противном случае результат в этой позиции будет равен 0. | 7 |
| ~ Отрицание | Выполняется операция побитового отрицания над двоичным представлением значения выражения. Любая позиция, содержащая 1 в исходном выражении, заменяется на 0. Любая позиция, содержащая 0 в исходном выражении, становится равной 0. Положительные числа начинаются с 0, отрицательные - с -1, поэтому ~ n == -(n+1). | 14 |
| << Побитовый сдвиг влево | Оператор сдвигает биты первого операнда влево на число битовых позиций, установленных вторым операндом. Для заполнения позиций справа используются нули. Возвращают результат того же типа, что левый операнд. | 11 |
| >> Побитовый сдвиг вправо | Оператор сдвигает биты первого операнда вправо на число битовых позиций, установленных вторым операндом. Цифры, сдвинутые за пределы диапазона, удаляются. Самый старший бит (32й) не меняется, чтобы сохранить знак результата. Если первый операнд положителен, старшие биты результата заполняются нулями; если первый операнд отрицателен, старшие биты результата заполняются единицами. Сдвиг значения вправо на одну позицию эквивалентен делению на 2 (с отбрасыванием остатка), а сдвиг вправо на две позиции эквивалентен делению на 4 и т. д. | 11 |
| >>> Побитовый сдвиг вправо без учета знака | Оператор сдвигает биты первого операнда вправо на число битовых позиций, установленных вторым операндом. Слева добавляются нули независимо от знака первого операнда. Цифры, сдвинутые за пределы диапазона, удаляются. | 11 |

var x = 9, y = 5, z = 2, s = -5, result; // 9 эквивалентно 1001, 5 эквивалентно 0101

result = x & y; // вернет 1 (эквивалентно 0001)

result = x | y; // вернет 13 (эквивалентно 1101)

result = x ^ y; // вернет 12 (эквивалентно 1100)

result = ~ y; // вернет -6 (эквивалентно 1100)

result = x << y; // вернет 288 (эквивалентно 100100000)

result = x >> z; // вернет 2 (эквивалентно 10)

result = s >>> z; // вернет 1073741822 (эквивалентно 111111111111111111111111111110)

JavaScript

### 7. Строковые операторы

Существует несколько операторов, которые работают со строками особым образом.

|  |  |  |
| --- | --- | --- |
| ТАБЛИЦА 7. СТРОКОВЫЕ ОПЕРАТОРЫ | | |
| **Оператор/Операция** | **Описание** | **Приоритет** |
| + Конкатенация | Оператор работает слева направо, выполняя объединение строк. Если первый операнд является строкой, последующие операнды будут преобразованы в строки и далее выполнится их объединение. | 12 |
| += Конкатенация с присваиванием | Выполняется объединение двух строк и результат присваивается переменной. | 12 |
| >, <, >=, <=, == Сравнение | Строки сравниваются по алфавиту, буквы в верхнем регистре всегда меньше букв в нижнем регистре. Сравнение строк основывается на номерах символов, указанных в стандарте Unicode, где прописные буквы идут раньше, чем строчные. | 10 |

"1" + "10"; // вернет "110"

"1" + 10; // вернет "110"

2 + 5 + " цветных карандашей"; // вернет "7 цветных карандашей"

"Цветных карандашей " + 2 + 5; // вернет "Цветных карандашей 25"

"1" > "10"; // вернет false

"10" <= 10; // вернет true

"СССР" == "ссср"; // вернет false

x = "micro"; x+= "soft"; // вернет "microsoft"

JavaScript

### 8. Специальные операторы

|  |  |  |
| --- | --- | --- |
| ТАБЛИЦА 8. СПЕЦИАЛЬНЫЕ ОПЕРАТОРЫ | | |
| **Оператор/Операция** | **Описание** | **Приоритет** |
| . Обращение к свойству | Осуществляет доступ к свойству объекта. | 15 |
| , Множественное вычисление | Вычисляет несколько независимых выражений, записанных в одну строку. | 1 |
| [] Индексация массива | Осуществляет доступ к элементам массива или свойствам объекта. | 15 |
| () Вызов функции, группировка | Группирует операции или вызывает функцию. | 15 |
| typeof Определение типа данных | Унарный оператор, возвращает тип данных операнда. | 14 |
| instanceof Проверка типа объекта | Оператор проверяет, является ли объект экземпляром определенного класса. Левый операнд должен быть объектом, правый - должен содержать имя класса объектов. Результат будет true, если объект, указанный слева, представляет собой экземпляр класса, указанного справа, в противном случае - false. | 10 |
| in Проверка наличия свойства | В качестве левого операнда должна быть строка, а правым - массив или объект. Если левое значение является свойством объекта, вернется результат true. | 10 |
| new Создание объекта | Оператор создает новый объект с неопределенными свойствами, затем вызывает функцию-конструктор для его инициализации (передачи параметров). Также может применяться для создания массива. | 1 |
| delete Удаление | Оператор позволяет удалять свойство из объекта или элемент из массива. Возвращает true, если удаление прошло успешно, в противном случае false. При удалении элемента массива его длина не меняется. | 14 |
| void Определение выражения без возвращаемого значения | Унарный оператор, отбрасывает значение операнда и возвращает underfined. | 14 |
| ?: Операция условного выражения | Тернарный оператор, позволяет организовать простое ветвление. В выражении участвуют три операнда, первый должен быть логическим значением или преобразовываться в него, а второй и третий - любыми значениями. Если первый операнд равен true, то условное выражение примет значение второго операнда; если false - то третьего. | 3 |

document.write("hello world"); // выводит на экран строку hello world

i = 0, j = 1; // сохраняет значения в переменных

function1(10, 5); // вызов функции function1 с параметрами 10 и 5

var year = [2014, 2015]; // создает массив с элементами

typeof {a:1}; // вернет "object"

var d = new Date(); // создаем новый объект с помощью конструктора Date()

d instanceof Date; // вернет true

var mycar = {make: "Honda", model: "Accord", year: 2005};

"make" in mycar; // вернет true

var obj = new Object(); // создает пустой объект

var food = ["milk", "bread", "meat", "olive oil", "cheese"];

delete food[3]; // удаляет четвертый элемент из массива food

Нажмите здесь, ничего не произойдет

x > 10 ? x \* 2 : x / 2; // возвращает значение x \* 2, если x > 10, в противном случае x / 2

JavaScript

### 9. Комментарии в JavaScript

Однострочный комментарий: перед текстом комментария нужно поставить символы //.

// Это однострочный комментарий

JavaScript

Многострочный комментарий помещается между символами /\* и \*/.

/\* Это многострочный

комментарий \*/

JavaScript

# Циклы JavaScript

**Циклы JavaScript** обеспечивают многократное выполнение повторяющихся вычислений. Они оптимизируют процесс написания кода, выполняя одну и ту же инструкцию или блок инструкций, образующих тело цикла, заданное число раз (используя переменную-счётчик) или пока заданное условие истинно. Циклы выполняют обход последовательности значений. Однократное выполнение цикла называется **итерацией**.

На производительность цикла влияют количество итераций и количество операций, выполняемых в теле цикла каждой итерации.

В JavaScript существуют следующие операторы цикла:

1) for используется когда вы заранее знаете, сколько раз вам нужно что-то сделать;  
2) for...in используется для обхода свойств объектов;  
3) while используется когда вы не знаете, сколько раз нужно что-то сделать;  
4) do...while работает аналогично с оператором while. Отличается тем, что do...while всегда выполняет выражение в фигурных скобках, по крайней мере один раз, даже если проверка условия возвращает false.

## Типы циклов в JavaScript, управление циклом

* **СОДЕРЖАНИЕ:**
* [1. Цикл for](https://html5book.ru/cikly-javascript/#part1)
* [2. Цикл for…in](https://html5book.ru/cikly-javascript/#part2)
* [3. Цикл while](https://html5book.ru/cikly-javascript/#part3)
* [4. Цикл do…while](https://html5book.ru/cikly-javascript/#part4)
* [5. Бесконечный цикл](https://html5book.ru/cikly-javascript/#part5)
* [6. Вложенный цикл](https://html5book.ru/cikly-javascript/#part6)
* [7. Управление циклом](https://html5book.ru/cikly-javascript/#part7)

### 1. Цикл for

Цикл for используется для выполнения итераций по элементам массивов или объектов, напоминающих массивы, таких как arguments и HTMLCollection. Условие проверяется перед каждой итерацией цикла. В случае успешной проверки выполняется код внутри цикла, в противном случае код внутри цикла не выполняется и программа продолжает работу с первой строки, следующей непосредственно после цикла.

Следующий цикл выведет на консоль строчку Hello, JavaScript! пять раз.

for (var i = 0; i < 5; i++) {

console.log(i + ": Hello, JavaScript!");

}

JavaScript

#### 1.1. Как работает цикл for

Цикл for состоит из трёх разных операций:

Шаг 1. **инициализация** var i = 0; — объявление переменной-счётчика, которая будет проверяться во время выполнения цикла. Эта переменная инициализируется со значением 0. Чаще всего в качестве счётчиков цикла выступают переменные с именами i, j и k.

Шаг 2. **проверка условия** i < 5; — условное выражение, если оно возвращает true, тело цикла (инструкция в фигурных скобках) будет выполнено. В данном примере проверка условия идёт до тех пор, пока значение счётчика меньше 5.

Шаг 3. **завершающая операция** i++ — операция приращения счётчика, увеличивает значение переменной var i на единицу. Вместо операции инкремента также может использоваться операция декремента.

По завершении цикла в переменной var i сохраняется значение 1. Следующий виток цикла выполняется для for (var i = 1; i < 5; i++) { }. Условное выражение вычисляется снова, чтобы проверить, является ли значение счётчика i всё ещё меньше 5. Если это так, операторы в теле цикла выполняются ещё раз. Завершающая операция снова увеличивает значение переменной на единицу. Шаги 2 и 3 повторяются до тех пор, пока условие i < 5; возвращает true.

#### 1.2. Вывод значений массива

Чтобы вывести значения массива с помощью цикла for, нужно задействовать свойство массива length. Это поможет определить количество элементов в массиве и выполнить цикл такое же количество раз.

Приведённый ниже скрипт выведет на экран пять сообщений с названиями цветов:

var flowers = ["Rose", "Lily", "Tulip", "Jasmine", "Orchid"];

for (var i = 0; i < flowers.length; i++){

alert(flowers[i] + " - это цветок.");

}

JavaScript

Если значение свойства length не изменяется в ходе выполнения цикла, можно сохранить его в локальной переменной, а затем использовать эту переменную в условном выражении. Таким образом можно повысить скорость выполнения цикла, так как значение свойства length будет извлекаться всего один раз за всё время работы цикла.

var flowers = ["Rose", "Lily", "Tulip", "Jasmine", "Orchid"], len = flowers.length;

for (var i = 0; i < len; i++){

alert(flowers[i] + " - это цветок.");

}

JavaScript

### 2. Цикл for...in

Циклы for...in используются для обхода свойств объектов, не являющихся массивами. Такой обход также называется **перечислением**. При обходе рекомендуется использовать метод hasOwnProperty(), чтобы отфильтровать свойства, которые были унаследованы от прототипа.

Для примера создадим объект с помощью литерала объекта.

var user = {

name: 'Alice',

age: 25,

country: 'Russia'

};

for (var prop in user) {

console.log(prop + ": " + user[prop]);

}

JavaScript

Предположим, что в сценарии до или после создания объекта user прототип объекта Object был расширен дополнительным методом clone().

if (typeof Object.prototype.clone === 'undefined') {

Object.prototype.clone = function () {};

}

JavaScript

Так как цепочка наследования прототипа постоянно проверяется интерпретатором, то все объекты автоматически получают доступ к новому методу.

Чтобы избежать обнаружения этого метода в процессе перечисления свойств объекта user, используется метод hasOwnProperty(), который отфильтрует свойства прототипа.

var user = {

name: 'Alice',

age: 25,

country: 'Russia'

};

if (typeof Object.prototype.clone === 'undefined') {

Object.prototype.clone = function () {};

}

for (var prop in user) {

if (user.hasOwnProperty(prop)) {

console.log(prop + ": " + user[prop]);

}

}

JavaScript

3. Цикл while

Цикл while - цикл с предварительной проверкой условного выражения. Инструкция внутри цикла (блок кода в фигурных скобках) будет выполняться в случае, если условное выражение вычисляется в true. Если первая проверка даст результат false, блок инструкций не выполнится ни разу.

После завершения итерации цикла условное выражение опять проверяется на истинность и процесс будет повторяться до тех пор, пока выражение не будет вычислено как false. В этом случае программа продолжит работу с первой строки, следующей непосредственно после цикла (если таковая имеется).

Данный цикл выведет на экран таблицу умножения для числа 3:

var i = 1;

var msg = '';

while (i < 10) {

msg+= i + ' x 3 = ' + (i \* 3) + '<br>';

i++;

}

document.write(msg);

JavaScript

### 4. Цикл do...while

Цикл do...while; проверяет условие продолжения после выполнения цикла. В отличие от цикла while, в do...while; тело цикла выполняется как минимум один раз, так как условие проверяется в конце цикла, а не в начале. Данный цикл используется реже, чем while, так как на практике ситуация, когда требуется хотя бы однократное исполнение цикла, встречается редко.

var result = '';

var i = 0;

do {

i += 1;

result += i + ' ';

} while (i < 5);

document.write(result);

JavaScript

В следующем примере операторы внутри цикла выполняются один раз, даже если условие не выполняется.

var i = 10;

do {

document.write(i + ' ');

i++;

} while (i < 10);

JavaScript

### 5. Бесконечные циклы

При создании любого цикла можно создать бесконечный цикл, который никогда не завершится. Такой цикл может потенциально продолжать работать до тех пор, пока работает компьютер пользователя. Большинство современных браузеров могут обнаружить это и предложат пользователю остановить выполнение скрипта. Чтобы избежать создания бесконечного цикла, вы должны быть уверены, что заданное условие в какой-то момент вернёт false. Например, следующий цикл задаёт условие, которое никогда не возвращает ложь, так как переменная i никогда не будет меньше 10:

for (var i = 25; i > 10; i++) {

document.write("Это предложение будет выводиться бесконечно...<br>");

}

JavaScript

### 6. Вложенные циклы

Цикл внутри другого цикла называется **вложенным**. При каждой итерации цикла вложенный цикл выполняется полностью. Вложенные циклы можно создавать с помощью цикла for и цикла while.

for (var count = 1; count < 3; count++) {

document.write(count + ". Строка цикла<br>");

for (var nestcount = 1; nestcount< 3; nestcount++) {

document.write("Строка вложенного цикла<br>");

}

}

JavaScript

7. Управление циклом

Циклом можно управлять с помощью операторов break; и continue;.

#### 7.1. Оператор break;

Оператор break; завершает выполнение текущего цикла. Он используется в исключительных случаях, когда цикл не может выполняться по какой-то причине, например, если приложение обнаруживает ошибку. Чаще всего оператор break; является частью конструкции if.

Когда оператор break; используется без метки, он позволяет выйти из цикла или из инструкции switch. В следующем примере создаётся счётчик, значения которого должны изменяться от 1 до 99, однако оператор break прерывает цикл после 14 итераций.

for (var i = 1; i < 100; i++) {

if (i == 15) {

break;

}

document.write(i);

document.write(' <br>');

}

JavaScript

Для вложенных циклов оператор break; используется с меткой, с помощью которой завершается работа именованной инструкции. Метка позволяет выйти из любого блока кода. Именованной инструкцией может быть любая инструкция, внешняя по отношению к оператору break;. В качестве метки может быть имя инструкции if или имя блока инструкций, заключенных в фигурные скобки только для присвоения метки этому блоку. Между ключевым словом break; и именем метки не допускается перевод строки.

outerloop:

for(var i = 0; i < 10; i++) {

innerloop:

for(var j = 0; j < 10; j++) {

if (j > 3) break; // Выход из самого внутреннего цикла

if (i == 2) break innerloop; // То же самое

if (i == 4) break outerloop; // Выход из внешнего цикла

document.write("i = " + i + " j = " + j + "<br>");

}

}

document.write("FINAL i = " + i + " j = " + j + "<br>");

JavaScript

#### 7.2. Оператор continue;

Оператор continue; останавливает текущую итерацию цикла и запускает новую итерацию. При этом, цикл while возвращается непосредственно к своему условию, а цикл for сначала вычисляет выражение инкремента, а затем возвращается к условию.

В этом примере на экран будут выведены все чётные числа:

var i;

for(i = 1; i <= 10; i++) {

if (i % 2 !== 0) {

continue;

}

document.write("<br><b>чётное число</b> = " + i);

}

JavaScript

Оператор continue; также может применяться во вложенных циклах с меткой.

outerloop:

for (var i = 0; i < 3; i++)

{

document.write("внешний цикл: "+i+"");

for (var j = 0; j < 5; j++)

{

if (i == 1)

break;

if (j == 3)

continue outerloop;

document.write("вложенный цикл: "+j+"");

}

}

document.write("Все циклы выполнены"+"");

# DOM — объектная модель документа

DOM была придумана создателями браузеров. Консорциум World Wide Web Consortium (W3C) стандартизировал DOM как единую спецификацию.

W3C определяет DOM как «интерфейс, не зависящий от платформы и языка, который позволяет программам и сценариям динамически получать доступ к содержимому и структуре документов и обновлять их». То есть, данная модель не предоставляет конкретную реализацию, а лишь описывает объекты, свойства, методы.

Концепция DOM реализована с разной степенью поддержки во всех основных браузерах. С помощью объектной модели документа можно создавать документы, перемещаться по их структуре, а также добавлять, изменять или удалять элементы и содержимое.

Когда пользователь запрашивает определенную веб-страницу, механизм рендеринга начинает получать HTML, CSS и JS-файлы запрошенной веб-страницы через сетевой уровень. После получения содержимого страницы, HTML и CSS-файлы анализируются по частям. Узлы DOM создаются для формирования дерева DOM и дерева CSSOM.

Дерево визуализации создается путем объединения деревьев DOM и CSSOM. Дерево визуализации содержит как информацию о порядке элементов, так и информацию об их стиле. На этом этапе каждому узлу присваиваются специальные координаты, чтобы гарантировать, что каждый элемент отображается в одном и том же положении на экране. После чего выполняется обход дерева визуализации, и каждый узел, упомянутый в дереве, отображается на экране.

* **СОДЕРЖАНИЕ:**
* [1. Введение в DOM](https://html5book.ru/dom-obektnaya-model-dokumenta/#intro)
* [2. Интерфейсы и узлы DOM](https://html5book.ru/dom-obektnaya-model-dokumenta/#dom-structure)
* [3. Свойства узлов](https://html5book.ru/dom-obektnaya-model-dokumenta/#node-props)
* [3.1. Определение типа и имени узла: свойства nodeName и nodeType](https://html5book.ru/dom-obektnaya-model-dokumenta/#node-type-name)
* [3.2. Получение значения узла: свойство nodeValue](https://html5book.ru/dom-obektnaya-model-dokumenta/#node-value)
* [3.3. Отношения между узлами](https://html5book.ru/dom-obektnaya-model-dokumenta/#node-relations)
* [3.3.1. Свойство childNodes](https://html5book.ru/dom-obektnaya-model-dokumenta/#childNodes)
* [3.3.2. Свойство parentNode](https://html5book.ru/dom-obektnaya-model-dokumenta/#parentNode)
* [3.3.3. Свойства previousSibling и nextSibling](https://html5book.ru/dom-obektnaya-model-dokumenta/#previousSibling)
* [3.3.4. Свойства firstChild и lastChild](https://html5book.ru/dom-obektnaya-model-dokumenta/#firstChild)
* [3.3.5. Свойство parentElement](https://html5book.ru/dom-obektnaya-model-dokumenta/#parentElement)
* [3.3.6. Свойство ownerDocument](https://html5book.ru/dom-obektnaya-model-dokumenta/#ownerDocument)
* [3.4. Другие свойства узлов: baseURI, isConnected и textContent](https://html5book.ru/dom-obektnaya-model-dokumenta/#other-props)
* [4. Манипуляции узлами](https://html5book.ru/dom-obektnaya-model-dokumenta/#node-methods)
* [4.1. Создание новых узлов](https://html5book.ru/dom-obektnaya-model-dokumenta/#create-nodes)
* [4.1.1. Метод appendChild()](https://html5book.ru/dom-obektnaya-model-dokumenta/#appendChild)
* [4.1.2. Метод cloneNode()](https://html5book.ru/dom-obektnaya-model-dokumenta/#cloneNode)
* [4.1.3. Метод insertBefore()](https://html5book.ru/dom-obektnaya-model-dokumenta/#insertBefore)
* [4.1.4. Метод replaceChild()](https://html5book.ru/dom-obektnaya-model-dokumenta/#replaceChild)
* [4.1.5. Метод removeChild()](https://html5book.ru/dom-obektnaya-model-dokumenta/#removeChild)
* [4.2. Проверка позиции узла в дереве DOM: методы compareDocumentPosition() и contains()](https://html5book.ru/dom-obektnaya-model-dokumenta/#node-position)
* [4.3. Определение идентичности узлов с помощью метода isEqualNode()](https://html5book.ru/dom-obektnaya-model-dokumenta/#isEqualNode)
* [4.4. Определение отношений между узлами](https://html5book.ru/dom-obektnaya-model-dokumenta/#node-relations)
* [5. Пространства имен XML: методы ookupNamespaceURI(), isDefaultNamespace() и lookupPrefix()](https://html5book.ru/dom-obektnaya-model-dokumenta/#namespaces)
* [6. Объединение родственных текстовых узлов в один с помощью метода normalize()](https://html5book.ru/dom-obektnaya-model-dokumenta/#normalize)

## 1. Введение в DOM

DOM — это API для HTML и XML-документов. **API** расшифровывается как интерфейс прикладного программирования, который можно определить как совокупность способов связи между различными компонентами программного обеспечения. Другими словами, API позволяет одной программе взаимодействовать с другой.

Спецификация DOM определяет набор интерфейсов для доступа и управления объектами HTML и XML-документов. **Интерфейсы** являются абстракцией, средствами определения способа доступа и управления внутренними представлениями документа в приложении.

Согласно терминологии W3C DOM, каждый документ представлен в виде **дерева**, а каждый автономный элемент или текстовый блок известен как **узел** — фундаментальный строительный блок. Термин «узел» происходит из сети, где он используется для обозначения точки подключения, и сама сеть представляет собой набор узлов. Узлы имеют отношения родитель-потомок, когда один контейнер содержит другой. Узел — это общее имя для объекта любого типа в иерархии DOM.

Когда происходит загрузка соответствующей страницы, в памяти браузера реализуется поддержка структуры объектов, сгенерированных согласно использованным в документе HTML-тегам. У каждого узла есть родительский узел (за исключением корневого узла document) и любое количество дочерних узлов. Одни типы узлов могут иметь дочерние узлы различных типов, другие являются так называемыми **листовыми** узлами, у которых не может дочерних узлов. Дерево состоит из узлов, но только некоторые из них являются HTML-элементами.

Узлы не представляют структуру данных, они представляют объекты, которые имеют функции и идентичность. В объектно-ориентированных языках программирования данные инкапсулируются в объекты, которые скрывают данные, защищая их от прямых внешних манипуляций. Функции, связанные с этими объектами, определяют, как ими можно манипулировать.

## 2. Интерфейсы и узлы DOM

Ядро DOM API состоит из Node, Element, Document и других относительно универсальных интерфейсов. Стандарт DOM также включает интерфейсы, специфичные как для HTML-документов, так и тегов многих HTML-элементов. Эти интерфейсы, например, HTMLBodyElement, HTMLTitleElement, обычно определяют набор свойств, отражающих атрибуты тега, которые обеспечивают удобный доступ к значениям атрибутов.

Базовый интерфейс, от которого наследуются все интерфейсы HTML-элементов и который должен использоваться элементами, не имеющими дополнительных атрибутов, — это интерфейс HTMLElement.

|  |  |
| --- | --- |
| ТАБЛИЦА 1. СПИСОК ИНТЕРФЕЙСОВ ДЛЯ HTML-ЭЛЕМЕНТОВ | |
| **Элемент(ы)** | **Интерфейс(ы)** |
| <a> | HTMLAnchorElement : HTMLElement |
| <abbr> | HTMLElement |
| <address> | HTMLElement |
| <area> | HTMLAreaElement : HTMLElement |
| <article> | HTMLElement |
| <aside> | HTMLElement |
| <audio> | HTMLAudioElement : HTMLMediaElement : HTMLElement |
| <b> | HTMLElement |
| <base> | HTMLBaseElement : HTMLElement |
| <bdi> | HTMLElement |
| <bdo> | HTMLElement |
| <blockquote> | HTMLQuoteElement : HTMLElement |
| <body> | HTMLBodyElement : HTMLElement |
| <br> | HTMLBRElement : HTMLElement |
| <button> | HTMLButtonElement : HTMLElement |
| <canvas> | HTMLCanvasElement : HTMLElement |
| <caption> | HTMLTableCaptionElement : HTMLElement |
| <cite> | HTMLElement |
| <code> | HTMLElement |
| <col> | HTMLTableColElement : HTMLElement |
| <colgroup> | HTMLTableColElement : HTMLElement |
| <data> | HTMLDataElement : HTMLElement |
| <datalist> | HTMLDataListElement : HTMLElement |
| <dd> | HTMLElement |
| <del> | HTMLModElement : HTMLElement |
| <details> | HTMLDetailsElement : HTMLElement |
| <dfn> | HTMLElement |
| <dialog> | HTMLDialogElement : HTMLElement |
| <div> | HTMLDivElement : HTMLElement |
| <dl> | HTMLDListElement : HTMLElement |
| <dt> | HTMLElement |
| <em> | HTMLElement |
| <embed> | HTMLEmbedElement : HTMLElement |
| <fieldset> | HTMLFieldSetElement : HTMLElement |
| <figcaption> | HTMLElement |
| <figure> | HTMLElement |
| <footer> | HTMLElement |
| <form> | HTMLFormElement : HTMLElement |
| <h1> — <h6> | HTMLHeadingElement : HTMLElement |
| <head> | HTMLHeadElement : HTMLElement |
| <header> | HTMLElement |
| <hgroup> | HTMLElement |
| <hr> | HTMLHRElement : HTMLElement |
| <html> | HTMLHtmlElement : HTMLElement |
| <i> | HTMLElement |
| <iframe> | HTMLIFrameElemen : HTMLElement |
| <img> | HTMLImageElement : HTMLElement |
| <input> | HTMLInputElement : HTMLElement |
| <ins> | HTMLModElement : HTMLElement |
| <kbd> | HTMLElement |
| <label> | HTMLLabelElement : HTMLElement |
| <legend> | HTMLLegendElement : HTMLElement |
| <li> | HTMLLIElement : HTMLElement |
| <link> | HTMLLinkElement : HTMLElement |
| <main> | HTMLElement |
| <map> | HTMLMapElement : HTMLElement |
| <mark> | HTMLElement |
| <menu> | HTMLMenuElement : HTMLElement |
| <meta> | HTMLMetaElement : HTMLElement |
| <meter> | HTMLMeterElement : HTMLElement |
| <nav> | HTMLElement |
| <noscript> | HTMLElement |
| <object> | HTMLObjectElement : HTMLElement |
| <ol> | HTMLOListElement : HTMLElement |
| <optgroup> | HTMLOptGroupElement : HTMLElement |
| <option> | HTMLOptionElement : HTMLElement |
| <output> | HTMLOutputElement : HTMLElement |
| <p> | HTMLParagraphElement : HTMLElement |
| <param> | HTMLParamElement : HTMLElement |
| <picture> | HTMLPictureElement : HTMLElement |
| <pre> | HTMLPreElement : HTMLElement |
| <progress> | HTMLProgressElement : HTMLElement |
| <q> | HTMLQuoteElement : HTMLElement |
| <rp> | HTMLElement |
| <rt> | HTMLElement |
| <ruby> | HTMLElement |
| <s> | HTMLElement |
| <samp> | HTMLElement |
| <script> | HTMLScriptElement : HTMLElement |
| <section> | HTMLElement |
| <select> | HTMLSelectElement : HTMLElement |
| <slot> | HTMLSlotElement : HTMLElement |
| <small> | HTMLElement |
| <source> | HTMLSourceElement : HTMLElement |
| <span> | HTMLSpanElement : HTMLElement |
| <strong> | HTMLElement |
| <style> | HTMLStyleElement : HTMLElement |
| <sub> | HTMLElement |
| <summary> | HTMLElement |
| <sup> | HTMLElement |
| <table> | HTMLTableElement : HTMLElement |
| <tbody> | HTMLTableSectionElement : HTMLElement |
| <td> | HTMLTableCellElement : HTMLElement |
| <template> | HTMLTemplateElement : HTMLElement |
| <textarea> | HTMLTextAreaElement : HTMLElement |
| <tfoot> | HTMLTableSectionElement : HTMLElement |
| <th> | HTMLTableCellElement : HTMLElement |
| <thead> | HTMLTableSectionElement : HTMLElement |
| <time> | HTMLTimeElement : HTMLElement |
| <title> | HTMLTitleElement : HTMLElement |
| <tr> | HTMLTableRowElement : HTMLElement |
| <track> | HTMLTrackElement : HTMLElement |
| <u> | HTMLElement |
| <ul> | HTMLUListElement : HTMLElement |
| <var> | HTMLElement |
| <video> | HTMLVideoElement : HTMLMediaElement : HTMLElement |
| <wbr> | HTMLElement |

Интерфейс Node (который в свою очередь реализует интерфейс EventTarget) является основным типом данных для всей объектной модели документа. Node представляет собой абстрактный интерфейс и не существует как узел. Он используется всеми узлами — Document, DocumentType, DocumentFragment, Element, Text, Attr, CDATASection, ProcessingInstruction и Comment.

Хотя все объекты, реализующие интерфейс Node, предоставляют методы для работы с потомками, не все объекты, реализующие интерфейс Node, могут иметь потомков (например, текстовые узлы).

## 3. Свойства узлов

Все объекты узлов, например, Element, Text и т.д., наследуют свойства и методы от интерфейса Node. Эти свойства и методы являются базовыми значениями и функциями для управления, проверки и обхода DOM.

Многие свойства DOM доступны только для чтения. Это означает, что их можно использовать для получения информации, но не для ее установки или обновления.

### 3.1. Определение типа и имени узла: свойства nodeName и nodeType

У объектов Node есть свойства идентификации, такие как nodeName и nodeType.

|  |  |  |
| --- | --- | --- |
| ТАБЛИЦА 2. ТИПЫ И ИМЕНА УЗЛОВ | | |
| **Интерфейс** | **nodeType** | **Значение nodeName** |
| Element | Node.ELEMENT\_NODE (1) | полное имя HTML-тега в верхнем регистре |
| Attr | Node.ATTRIBUTE\_NODE (2) | полное имя атрибута |
| Text | Node.TEXT\_NODE (3) | #text |
| CDATASection | Node.CDATA\_SECTION\_NODE (4) | #cdata-section |
| ProcessingInstruction | Node.PROCESSING\_INSTRUCTION\_NODE (7) | имя, определяющее приложение, для которого предназначена инструкция |
| Comment | Node.COMMENT\_NODE (8) | #comment |
| Document | Node.DOCUMENT\_NODE (9) | #document |
| DocumentType | Node.DOCUMENT\_TYPE\_NODE (10) | название типа документа |
| DocumentFragment | Node.DOCUMENT\_FRAGMENT\_NODE (11) | #document-fragment |

<!DOCTYPE html>

<html lang="ru">

<body>

<p class="example">Какой-то текст</a>

</body>

</html>

HTML

Свойство nodeName возвращает имя/название в виде строки, либо постоянное значение (которому предшествует символ #). Используется только для чтения.

console.log(document.doctype.nodeName)); // html

console.log(document.nodeName); // #document

console.log(document.body.nodeName); // BODY

console.log(document.createDocumentFragment().nodeName); // #document-fragment

console.log(document.querySelector('p').nodeName); // P

console.log(document.querySelector('p').firstChild.nodeName); // #text

console.log(document.querySelector('p').nodeType === 1); // true

JavaScript

Свойство nodeType возвращает целочисленный идентификатор типа узла. Интерфейс Node содержит константы, которые соответствуют этим числам. Определение типа узла полезно в сценариях, когда перебирается неизвестный набор узлов. Определение типа узла, для которого вы можете написать сценарий, может быть очень удобным, если вы хотите знать, какие свойства и методы доступны для сценария узла.

console.log(document.doctype.nodeType); // 10

console.log(document.nodeType); // 9

console.log(document.createDocumentFragment().nodeType); // 11

console.log(document.querySelector('p').nodeType); // 1

console.log(document.querySelector('p').firstChild.nodeType);// 3

JavaScript

### 3.2. Получение значения узла: свойство nodeValue

Свойство nodeValue возвращает или устанавливает значение узла.

Для большинства типов узлов возвращает null и любой набор операций игнорируется. Для узлов типа TEXT\_NODE, COMMENT\_NODE, CDATA\_SECTION\_NODE и PROCESSING\_INSTRUCTION\_NODE, значение соответствует текстовым данным, содержащимся в объекте. Для узла ATTRIBUTE\_NODE вернет значение атрибута.

Использование данного свойства полезно для извлечения текстовых строк из узлов Text и Comment.

console.log(document.doctype.nodeValue); // null

console.log(document.nodeValue); // null

console.log(document.createDocumentFragment().nodeValue); // null

console.log(document.querySelector('p').nodeValue); // null

console.log(document.querySelector('p').firstChild.nodeValue); // Какой-то текст

JavaScript

### 3.3. Отношения между узлами

Все узлы в документе имеют связи с другими узлами. Эти отношения описываются в терминах традиционных семейных отношений, как если бы дерево документа было генеалогическим древом.

<ul id="list">

<li>Пункт 1</li>

<li>Пункт 2</li>

<li><a href="https://google.com">Пункт 3</a></li>

<li>Пункт 4</li>

</ul>

HTML

#### 3.3.1. Свойство childNodes

Каждый узел имеет свойство childNodes, которое извлекает все дочерние узлы объекта и сохраняет их возвращает живой NodeList, содержащий всех потомков данного узла. Свойство используется только для чтения. Живой NodeList означает то, что если потомки узла изменяются, объект NodeList автоматически обновляется. NodeList — это объект типа массива, используемый для хранения упорядоченного списка узлов, доступных по позиции. NodeList не является экземпляром Array. Вы можете получить доступ к дочернему элементу текущего элемента через счетчик массива или метод item().

const elem = document.getElementById('list');

const count = elem.childNodes.length;

const firstChild = elem.childNodes.item(0);

console.log(count); // 9

console.log(firstChild); // [object Text]

JavaScript

Когда DOM конструируется браузером, текстовые узлы создаются как из пробелов, так и из текстовых символов, поэтому в данном примере каждый символ разрыва строки считается как текстовый узел.

#### 3.3.2. Свойство parentNode

Каждый узел имеет свойство parentNode, указывающее на его родителя в дереве документа. Свойство возвращает ссылку на внешний узел, который отражается как объект, принадлежащий документу. Все узлы, содержащиеся в списке дочерних узлов, имеют одного и того же родителя, поэтому каждое из их свойств родительского узла указывает на один и тот же узел. Если нет такого узла, по причине того, что узел находится вверху дерева или не относится к дереву, данное свойство вернет null. Свойство parentNode текстового фрагмента — это внешний узел или элемент, охватывающий этот фрагмент.

const elem = document.getElementById('list');

const parentElem = elem.parentNode;

document.write(parentElem); // [object HTMLBodyElement]

JavaScript

#### 3.3.3. Свойства previousSibling и nextSibling

Можно переходить от одного узла в списке к другому, используя свойства previousSibling и nextSibling. Свойства используются только для чтения.

Свойство previousSibling возвращает узел, представляющий предыдущий одноуровневый узел в дереве или null, если не такого узла.

Свойство nextSibling возвращает узел, представляющий следующий одноуровневый узел в дереве или null, если не такого узла.

Первый узел в списке имеет значение null для его свойства previousSibling, а последний узел в списке имеет значение null для его свойства nextSibling.

const list = document.querySelector('ul');

console.log(list.nextSibling.nodeName); // #text

console.log(list.previousSibling.nodeName); // #text

JavaScript

#### 3.3.4. Свойства firstChild и lastChild

Свойства firstChild и lastChild возвращают узел, представляющий первый или последний потомок узла в дереве или null, если узел не имеет потомков. Свойства используются только для чтения. Они полезны, когда вы хотите вставить новый дочерний элемент до или после всех остальных, и вам нужна точка отсчета для методов, которые добавляют элементы в список узлов документа.

<p id="p1"></p>

<p id="p2"> </p>

HTML

console.log(document.getElementById('p1').firstChild); // null

console.log(document.getElementById('p2').lastChild.nodeName); // #text

JavaScript

#### 3.3.5. Свойство parentElement

Свойство parentElement возвращает элемент, который является родителем данного узла. Если узел не имеет родителя или если родитель не элемент, это свойство вернет null. Свойство работает только с элементами HTML, которые отражаются как объекты документа, тогда как узел не обязательно является элементом HTML (например, атрибутом или текстовым фрагментом). Используется только для чтения.

<ul id="list">

<li>Пункт 1</li>

<li>Пункт 2</li>

<li><a href="https://google.com">Пункт 3</a></li>

<li>Пункт 4</li>

</ul>

<p id="result"></p>

HTML

const parentEl = document.getElementById('list').parentElement;

document.getElementById('result').innerHTML = 'Объект: ' + parentEl + '' + ' Имя узла:' + ' ' + parentEl.nodeName;

JavaScript

#### 3.3.6. Свойство ownerDocument

Свойство ownerDocument возвращает документ, в контексте которого узел был создан. Если нет связанного с ним документа, возвращает null. Свойство ownerDocument объекта предоставляет способ создания ссылок на другие объекты в том же документе или для доступа к свойствам и методам объекты документа, обеспечивая быстрый доступ к узлу документа без необходимости обхода иерархии узлов до самого верха. Свойство полезно при определении, к какому документу принадлежит элемент <iframe>, новое окно, вкладка или XML-документ. Используется только для чтения.

<iframe src="https://player.vimeo.com/video/159120552?byline=0&portrait=0&badge=0" width="640" height="360" frameborder="0" webkitallowfullscreen mozallowfullscreen allowfullscreen></iframe>

HTML

const frame = document.querySelector('iframe');

console.log(frame.ownerDocument.location.href);

JavaScript

### 3.4. Другие свойства узлов: baseURI, isConnected и textContent

#### Свойство baseURI

Свойство baseURI возвращает базовый URL-адрес документа. В HTML это соответствует протоколу, доменному имени и структуре каталогов, все до последнего /. Это свойство удобно в приложениях, которые импортируют данные XML, и в этом случае источник элемента XML, вероятно, отличается от страницы HTML, на которой он обрабатывается. Используется только для чтения.

<p id="result"></p>

HTML

const pageURI = document.baseURI;

document.getElementById('result').innerHTML = 'Базовый URL страницы: ' + pageURI;

JavaScript

Узнать базовый URL

ФИГУРА 2. РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ КОДА

#### Свойство isConnected

Свойство isConnected возвращает логическое значение, указывающее, подключен ли узел (напрямую или косвенно) к объекту контекста, например объекту Document в случае обычного DOM или ShadowRoot в случае теневого DOM. Используется только для чтения.

const test = document.createElement('p');

console.log(test.isConnected); // false

document.body.appendChild(test);

console.log(test.isConnected); // true

JavaScript

#### Свойство textContent

Свойство textContent можно использовать для получения всех дочерних текстовых узлов, а также для установки содержимого узла в определенный текстовый узел. Когда он используется для получения текстового содержимого узла, он возвращает объединенную строку всех текстовых узлов, содержащихся в узле. В отличие от innerText, возвращает любой встроенный стиль или код сценария вместе с другим текстом. Когда свойство textContent используется для установки текста, содержащегося в узле, оно удалит все дочерние узлы, заменив их на новое текстовое содержимое.

<p id="result"></p>

<p id="text">Текст</p>

HTML

document.getElementById('result').textContent = 'Какой-то текст';

const text = document.getElementById('text').textContent;

console.log(text); // Текст

JavaScript

## 4. Манипуляции узлами

### 4.1. Создание новых узлов

#### 4.1.1. Метод appendChild()

Метод appendChild() добавляет узел (или несколько узлов) в конец списка дочерних узлов родительского узла, на котором вызывается метод. Если дочерних узлов нет, узел добавится в качестве первого дочернего узла.

<p>Привет</p>

HTML

const elementNode = document.createElement('b');

const textNode = document.createTextNode(' мир!');

//добавим эти узлы к DOM

document.querySelector('p').appendChild(elementNode);

document.querySelector('b').appendChild(textNode);

JavaScript

ФИГУРА 3. РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ КОДА

#### 4.1.2. Метод cloneNode()

Метод cloneNode() позволяет дублировать один узел или узел и все его дочерние узлы. Принимает один логический аргумент, указывающий, следует ли выполнять глубокое копирование. Когда аргумент равен true, клонируется узел и все его поддерево; когда false, клонируется только начальный узел. Возвращаемый клонированный узел принадлежит документу, но ему не назначен родительский узел. Таким образом, клонированный узел не существует в документе до тех пор, пока не будет добавлен с помощью appendChild(), insertBefore() или replaceChild().

При клонировании узла элемента также клонируются все его атрибуты и их значения (включая встроенные события). Все, что добавлено с помощью addEventListener() или node.onclick, не клонируется.

<ul id="menu">

<li>Главная</li>

<li>Услуги</li>

<li>О нас</li>

<li>Контакты</li>

</ul>

HTML

const menu = document.querySelector('#menu').cloneNode(true);

menu.setAttribute('id', 'menu-mobile');

document.body.appendChild(menu);

JavaScript

ФИГУРА 4. РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ КОДА

#### 4.1.3. Метод insertBefore()

Метод insertBefore() вставляет узел перед другим узлом в качестве дочернего узла родительского узла. Принимает два параметра, указанные через запятую, — узел для вставки и узел, перед которым будет вставлен новый узел. Если у родительского узла нет дочерних узлов, то после вставки узел будет единственным дочерним узлом.

<ul id="menu">

<li>Услуги</li>

<li>О нас</li>

<li>Контакты</li>

</ul>

HTML

const menu = document.getElementById('menu');

// создаем новый узел li

const li = document.createElement('li');

li.textContent = 'Главная';

// вставляем новый узел перед первым элементом списка

menu.insertBefore(li, menu.firstElementChild);

JavaScript

#### 4.1.4. Метод replaceChild()

Метод replaceChild() заменяет один узел другим. Метод принимает два аргумента: узел для вставки и узел для замены. Когда узел вставляется с помощью replaceChild(), все его указатели отношений дублируются из узла, который он заменяет. Несмотря на то, что замененный узел технически по-прежнему принадлежит тому же документу, он больше не имеет определенного местоположения в документе.

<div id="div-1">Элемент 1</div>

<div id="div-2">Элемент 2</div>

HTML

//замена узла элемента

const divFirst = document.getElementById('div-1');

var newP = document.createElement('p');

newP.textContent = 'Новый элемент 1';

divFirst.parentNode.replaceChild(newP, divFirst);

//замена текстового узла

const divSecond = document.getElementById('div-2').firstChild;

const divText = document.createTextNode('Элемент 2 новый текст');

divSecond.parentNode.replaceChild(divText, divSecond);

JavaScript

ФИГУРА 5. РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ КОДА

#### 4.1.5. Метод removeChild()

Метод removeChild() удаляет дочерний узел из родительского узла. Удаление происходит в несколько этапов: сначала нужно выбрать узел, который хотите удалить; затем получить доступ к его родительскому элементу, как правило, с помощью свойства parentNode. После чего вызвать метод removeChild(), передавая ему ссылку на удаляемый узел.  
Удалённый дочерний узел остаётся в памяти, но больше не является частью DOM. Повторно использовать удалённый узел можно с помощью ссылки на объект — oldChild.

//удаление узла элемента без указания его родительского узла

const divFirst = document.getElementById('div-1');

divFirst.parentNode.removeChild(divFirst);

//удаление текстового узла

const divSecond = document.getElementById('div-2').firstChild;

divSecond.parentNode.removeChild(divSecond);

JavaScript

### 4.2. Проверка позиции узла в дереве DOM: методы compareDocumentPosition() и contains()

#### Метод compareDocumentPosition()

Метод compareDocumentPosition() определяет связь между двумя узлами и возвращает битовую маску, указывающую на связь. Значения битовой маски показаны в следующей таблице. Узел может иметь более одного типа отношений с другим узлом. Например, когда узел одновременно содержит (16) и предшествует (4), возвращаемое значение из compareDocumentPosition() будет равно 20.

|  |  |
| --- | --- |
| **Числовой код** | **Описание** |
| 1 | DOCUMENT\_POSITION\_DISCONNECTED Означает, что два узла не принадлежат одному и тому же документу. |
| 2 | DOCUMENT\_POSITION\_PRECEDING Означает, что переданный узел предшествует выбранному узлу. |
| 4 | DOCUMENT\_POSITION\_FOLLOWING Означает, что переданный узел следует за выбранным узлом. |
| 8 | DOCUMENT\_POSITION\_CONTAINS Означает, что переданный узел является предком выбранного узла. |
| 16 | DOCUMENT\_POSITION\_CONTAINED\_BY Означает, что переданный узел является потомком выбранного узла. |
| 32 | DOCUMENT\_POSITION\_IMPLEMENTATION\_SPECIFIC Означает, что два узла не имеют отношения друг к другу или являются двумя атрибутами одного и того же элемента. |

<p id="p1">Первый параграф</p>

<p id="p2">Второй параграф</p>

HTML

const p2Position = p1.compareDocumentPosition(p2); // p1 - узел, который сравнивается, p2 - узел, с которым идёт сравнение

console.log(p2Position); // 4, то есть p2 идет за p1

JavaScript

#### Метод contains()

Метод contains() позволяет узнать, является ли данный узел потомком другого. Метод вызывается для узла-предка, с которого должен начинаться поиск, и принимает единственный аргумент, который является предполагаемым узлом-потомком. Метод возвращает значение true, если узел является потомком данного узла, в противном случае возвращает false.

<body>

<header></header>

<main></main>

<footer></footer>

<script></script>

</body>

HTML

const footerNode = document.querySelector('footer');

const scriptNode = document.querySelector('script');

console.log(footerNode.contains(scriptNode)); // false

JavaScript

Метод также можно использовать, например, для проверки, содержит ли данный элемент указанный класс:

<body class="post-template-default single single-post single-format-standard"></body>

HTML

const bodyClass = document.body;

console.log(bodyClass.classList.contains('post-template-default')); // true

JavaScript

### 4.3. Определение идентичности узлов с помощью метода isEqualNode()

Метод isEqualNode() проверяет равенство двух узлов. Два узла равны, если выполняются следующие условия:

* Узлы одного типа;
* Атрибуты nodeName, localName, namespaceURI, prefix и nodeValue нулевые или имеют одинаковую длину и идентичны посимвольно;
* Атрибуты NamedNodeMaps равны нулю или имеют одинаковую длину, и для каждого узла, существующего в одной карте, есть узел, который существует в другой карте и является равным, хотя и не обязательно с тем же индексом.
* Списки узлов дочерних узлов равны, то есть они оба нулевые или имеют одинаковую длину и содержат одинаковые узлы с одним и тем же индексом. Рекомендуется нормализовать узлы перед сравнением.

<ul>

<li><a href="">Ссылка</a></li>

<li><a href="">Ссылка</a></li>

</ul>

<p>Параграф 1</p>

<p>Параграф 2</p>

HTML

const listItem = document.querySelectorAll('li');

console.log(listItem[0].isEqualNode(listItem[1])); // true

const p = document.querySelectorAll('p');

console.log(p[0].isEqualNode(p[1])); // false

JavaScript

### 4.4. Определение отношений между узлами: методы hasChildNodes() и getRootNode()

#### Метод hasChildNodes()

Метод hasChildNodes() возвращает true, если узел имеет один или несколько дочерних узлов, и false в противном случае. Этот способ более эффективный, чем запрос childNodes.length.

<ul id="list">

<li>Пункт 1</li>

<li>Пункт 2</li>

<li><a href="https://google.com">Пункт 3</a></li>

<li>Пункт 4</li>

</ul>

HTML

const listItems = document.getElementById("list");

if ( listItems.hasChildNodes() ) {

listItems.removeChild( listItems.firstElementChild );

}

JavaScript

#### Метод getRootNode()

Метод getRootNode() возвращает корневой узел. Вызов для элемента внутри стандартной веб-страницы вернет объект HTMLDocument, представляющий всю страницу.  
Вызов для элемента внутри теневого DOM вернет связанный с ним ShadowRoot.

<div class="js-parent">

<div class="js-child"></div>

</div>

<div class="js-shadowHost"></div>

HTML

const parent = document.querySelector('.js-parent'),

child = document.querySelector('.js-child'),

shadowHost = document.querySelector('.js-shadowHost');

console.log(parent.getRootNode().nodeName); // #document

console.log(child.getRootNode().nodeName); // #document

// создадим теневое дерево

const shadowRoot = shadowHost.attachShadow({ mode: 'open' });

shadowRoot.innerHTML = 'Содержимое теневого дерева';

const shadowChild = shadowRoot.querySelector('.js-shadowChild');

console.log(shadowChild.getRootNode() === shadowRoot); // true

console.log(shadowChild.getRootNode({ composed: false }) === shadowRoot); // true

console.log(shadowChild.getRootNode({ composed: true }).nodeName); // #document

JavaScript

### 5. Пространства имен XML: методы lookupNamespaceURI(), isDefaultNamespace() и lookupPrefix()

Пространства имен XML позволяют размещать в одном документе элементы из разных языков, основанных на XML, не опасаясь конфликтов имен элементов. Пространства имен объявляются с использованием семейства зарезервированных атрибутов. Имя такого атрибута должно быть либо xmlns, либо иметь xmlns: в качестве префикса.

Описанные ниже методы в первую очередь полезны, когда у вас есть ссылка на узел, но вы не знаете его связи с остальной частью документа.

<svg xmlns:svg="http://www.w3.org/2000/svg" height="1"></svg>

HTML

#### Метод lookupNamespaceURI()

Метод lookupNamespaceURI() принимает префикс в качестве параметра и возвращает пространство имен, представленное префиксом (prefix), или пространство имен по умолчанию, если аргумент prefix является пустой строкой.

Когда метод вызывается с префиксом в качестве аргумента, он должен возвращать URI пространства имен для данного префикса. Когда метод lookupNamespaceURI() вызывается с пустой строкой в качестве аргумента (представляющего пространство имен по умолчанию), он должен выполнить одно из следующих действий:

Если существует пространство имен по умолчанию, возвращает URI пространства имен по умолчанию. Если пространства имен по умолчанию нет, возвращает либо пустую строку, либо null, либо, если это разрешено привязкой языка, — undefined.

const svgElem = document.getElementsByTagName('svg')[0];

console.log(svgElem.lookupNamespaceURI("")); // http://www.w3.org/2000/svg

JavaScript

#### Метод isDefaultNamespace()

Метод isDefaultNamespace() принимает URI пространства имен в качестве аргумента. Он возвращает логическое значение, которое равно true, если пространство имен является пространством имен по умолчанию на данном узле, и false в противном случае. Пространство имен HTML-элемента по умолчанию всегда равно "". Для элемента SVG он задается атрибутом xmlns.

const svgElem = document.getElementsByTagName('svg')[0];

console.log(svgElem.isDefaultNamespace("")); // false

console.log(svgElem.isDefaultNamespace("http://www.w3.org/2000/svg")); // true

JavaScript

#### Метод lookupPrefix()

Метод lookupPrefix() возвращает строку, содержащую префикс для данного URI пространства имен, если он присутствует, и null в противном случае. Если с префиксом пространства имен связано более одного префикса, возвращаемый префикс пространства имен зависит от реализации.

const svgElem = document.getElementsByTagName('svg')[0];

console.log(svgElem.lookupPrefix("http://www.w3.org/2000/svg")); // null

JavaScript

### 6. Объединение родственных текстовых узлов в один с помощью метода normalize()

Метод normalize() используется для объединения родственных текстовых узлов в один текстовый узел. Родственные текстовые узлы обычно встречаются тогда, когда текст программно добавляется в DOM.

<div></div>

HTML

const pElem = document.createElement('p');

const text1 = document.createTextNode('Привет');

const text2 = document.createTextNode(' мир!');

pElem.appendChild(text1);

pElem.appendChild(text2);

document.querySelector('div').appendChild(pElem);

console.log(document.querySelector('p').childNodes.length); // 2

document.querySelector('div').normalize();

console.log(document.querySelector('p').childNodes.length); // 1

JavaScript