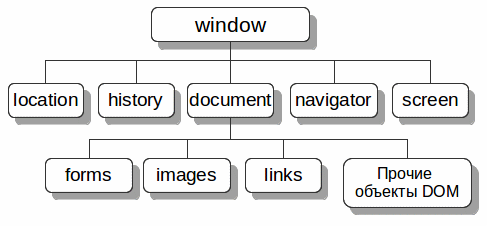
**Иерархия объектов в JavaScript**

В языке JavaScript все элементы на web-странице выстраиваются в иерархическую структуру. Каждый элемент представляется в виде объекта, каждый такой объект может иметь определенные свойства и методы. Язык JavaScript позволяет легко управлять объектами web-страницы, хотя для этого очень важно понимать иерархию объектов, на которые опирается разметка HTML.

С точки зрения языка JavaScript окно браузера – это некий объект window. Этот объект также содержит в свою очередь некоторые элементы оформления, такие как строка состояния. Внутри окна можно разместить документ HTML (в общем случае файл любого типа). Такая страница является объектом document. Это означает, что объект document представляет в языке JavaScript загруженный на настоящий момент документ HTML. Объект document является очень важным объектом в языке JavaScript. К свойствам объекта document относятся, например, цвет фона для web-страницы. Однако гораздо важнее то, что все без исключения объекты HTML являются свойствами объекта document. Примерами объекта HTML являются, к примеру, ссылка или заполняемая форма. Для получения информации о различных объектах в этой иерархии и управления ею нужно знать как в языке JavaScript организован доступ к различным объектам. Каждый объект иерархической структуры имеет свое имя.



**window** — объект, который поддерживается всеми программами просмотра, представляет окно браузера. Корень иерархии BOM, через который осуществляется доступ ко всем прочим объектам JavaScript.

**window.document** — представляет доступ ко всем элементам HTML гипертекстового документа, загруженного в заданное окно. Поддерживается всеми браузерами.

**window.history** — управляет историей просмотра веб-страниц. Для перемещения по журналу просмотра используются методы back, forward, go

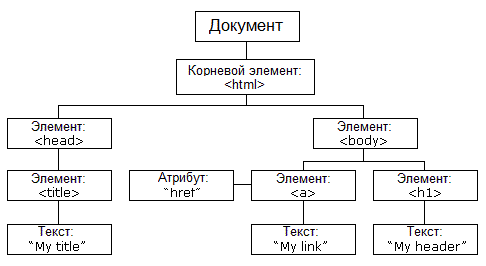
Объект window является родителем всех других объектов, ссылки на которые содержатся в соответствующих свойствах объекта window.

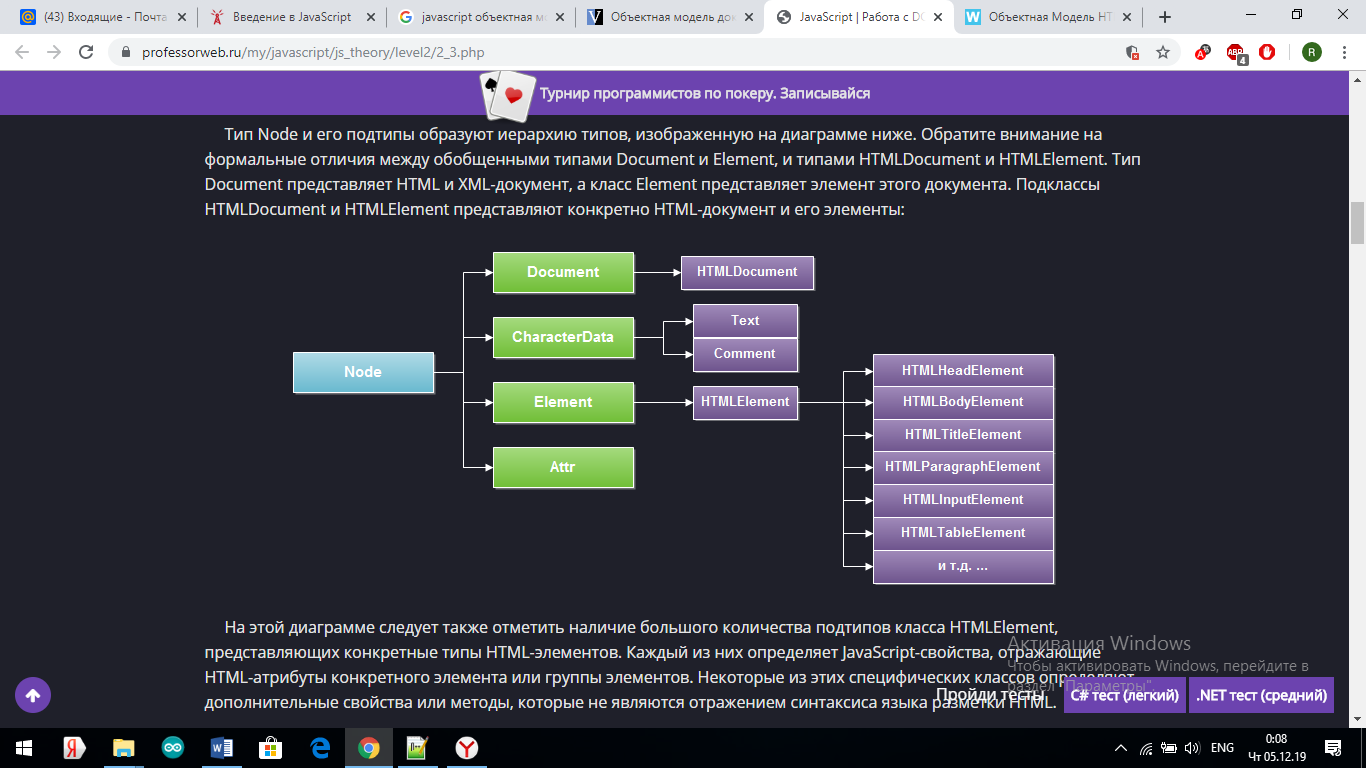
|  |  |
| --- | --- |
| **Объект** | **Краткое описание объекта** |
| window.navigator | Свойства и методы для доступа к данным, описывающих браузер. |
| window.location | Свойства и методы для работы с URL открытого окна. |
| window.history | Свойства и методы для работы со списком URL документов, которые загружались в окно браузера. |
| window.screen | Свойства и методы для доступа к параметрам и настройкам экрана монитора. |
| window.document | Объектная модель документа — содержит иерархию объектов, построенных браузером по HTML-коду, загруженному в окно документа. |

Объектная Модель Документа (DOM) – это программный интерфейс (API) для HTML и XML документов.

DOM предоставляет структурированное представление документа и определяет то, как эта структура может быть доступна из программ, которые могут изменять содержимое, стиль и структуру документа. Представление DOM состоит из структурированной группы узлов и объектов, которые имеют свойства и методы. По существу, DOM соединяет веб-страницу с языками описания сценариев либо языками программирования.

DOM спроектирован таким образом, чтобы быть независимым от любого конкретного языка программирования, обеспечивая структурное представление документа согласно единому и последовательному API





Выбор элементов документа

Работа большинства клиентских программ на языке JavaScript так или иначе связана с манипулированием элементами документа. В ходе выполнения эти программы могут использовать глобальную переменную document, ссылающуюся на объект Document. Однако, чтобы выполнить какие-либо манипуляции с элементами документа, программа должна каким-то образом получить, или выбрать, объекты Element, ссылающиеся на эти элементы документа. Модель DOM определяет несколько способов выборки элементов. Выбрать элемент или элементы документа можно:

по значению атрибута id;

по значению атрибута name;

по имени тега;

по имени класса или классов CSS;

по совпадению с определенным селектором CSS.

**Выбор элементов по значению атрибута id**

Все HTML-элементы имеют атрибуты id. Значение этого атрибута должно быть уникальным в пределах документа - никакие два элемента в одном и том же документе не должны иметь одинаковые значения атрибута id. Выбрать элемент по уникальному значению атрибута id можно с помощью метода getElementById() объекта Document:

**var section1 = document.getElementById("section1");**

Это самый простой и самый распространенный способ выборки элементов. Если сценарию необходимо иметь возможность манипулировать каким-то определенным множеством элементов документа, присвойте значения атрибутам id этих элементов и используйте возможность их поиска по этим значениям.

**Выбор элементов по значению атрибута name**

HTML-атрибут name первоначально предназначался для присваивания имен элементам форм, и значение этого атрибута использовалось, когда выполнялась отправка данных формы на сервер. Подобно атрибуту id, атрибут name присваивает имя элементу. Однако, в отличие от id, значение атрибута name не обязано быть уникальным: одно и то же имя могут иметь сразу несколько элементов, что вполне обычно при использовании в формах радиокнопок и флажков. Кроме того, в отличие от id, атрибут name допускается указывать лишь в некоторых HTML-элементах, включая формы, элементы форм и элементы <iframe> и <img>.

Выбрать HTML-элементы, опираясь на значения их атрибутов name, можно с помощью метода **getElementsByName()** объекта Document:

**var radiobuttons = document.getElementsByName("favorite\_color");**

Метод getElementsByName() определяется не классом Document, а классом HTMLDocument, поэтому он доступен только в HTML-документах и не доступен в XML-документах. Он возвращает объект NodeList, который ведет себя, как доступный только для чтения массив объектов Element.

**Выбор элементов по типу**

Метод getElementsByTagName() объекта Document позволяет выбрать все HTML или XML-элементы указанного типа (или по имени тега). Например, получить подобный массиву объект, доступный только для чтения, содержащий объекты Element всех элементов <span> в документе, можно следующим образом:

**var spans = document.getElementsByTagName("span");**

Подобно методу getElementsByName(), getElementsByTagName() возвращает объект NodeList. Элементы документа включаются в массив NodeList в том же порядке, в каком они следуют в документе, т.е. первый элемент <p> в документе можно выбрать так:

var firstParagraph = document.getElementsByTagName("p")[0];

Имена HTML-тегов не чувствительны к регистру символов, и когда getElementsByTagName() применяется к HTML-документу, он выполняет сравнение с именем тега без учета регистра символов. Переменная spans, созданная выше, например, будет включать также все элементы <span>, которые записаны как <SPAN>.

Можно получить NodeList, содержащий все элементы документа, если передать методу getElementsByTagName() шаблонный символ «\*».

Кроме того, классом Element также определяет метод getElementsByTagName(). Он действует точно так же, как и версия метода в классе Document, но выбирает только элементы, являющиеся потомками для элемента, относительно которого вызывается метод. То есть отыскать все элементы <span> внутри первого элемента <p> можно следующим образом:

var firstParagraph = document.getElementsByTagName("p")[0];

var firstParagraphSpans = firstParagraph.getElementsByTagName("span");

По историческим причинам класс HTMLDocument определяет специальные свойства для доступа к узлам определенных типов. Свойства images, forms и links, например, ссылаются на объекты, которые ведут себя как массивы, доступные только для чтения, содержащие элементы <img>, <form> и <a> (но только те теги <a>, которые имеют атрибут href). Эти свойства ссылаются на объекты HTMLCollection, которые во многом похожи на объекты NodeList, но дополнительно могут индексироваться значениями атрибутов id и name.

**Выбор элементов по классу CSS**

Значением HTML-атрибута class является список из нуля или более идентификаторов, разделенных пробелами. Он дает возможность определять множества связанных элементов документа: любые элементы, имеющие в атрибуте class один и тот же идентификатор, являются частью одного множества. Слово class зарезервировано в языке JavaScript, поэтому для хранения значения HTML-атрибута class в клиентском JavaScript используется свойство className.

Обычно атрибут class используется вместе с каскадными таблицами стилей CSS, с целью применить общий стиль отображения ко всем членам множества. Однако кроме этого, стандарт HTML5 определяет метод getElementsByClassName(), позволяющий выбирать множества элементов документа на основе идентификаторов в их атрибутах class.

Подобно методу getElementsByTagName(), метод getElementsByClassName() может вызываться и для HTML-документов, и для HTML-элементов, и возвращает «живой» объект NodeList, содержащий все потомки документа или элемента, соответствующие критерию поиска.

Метод getElementsByClassName() принимает единственный строковый аргумент, но в самой строке может быть указано несколько идентификаторов, разделенных пробелами. Соответствующими будут считаться все элементы, атрибуты class которых содержат все указанные идентификаторы. Порядок следования идентификаторов не имеет значения. Обратите внимание, что и в атрибуте class, и в аргументе метода getElementsByClassName() идентификаторы классов разделяются пробелами, а не запятыми.

Ниже приводится несколько примеров использования метода getElementsByClassName():

// Отыскать все элементы с классом "warning"

var warnings = document.getElementsByClassName("warning");

// Отыскать всех потомков элемента с идентификаторам "log"

// с классами "error" и "fatal"

var log = document.getElementById("log");

var fatal = log.getElementsByClassName("fatal error");

**Выбор элементов с использованием селекторов CSS**

Каскадные таблицы стилей CSS имеют очень мощные синтаксические конструкции, известные как селекторы, позволяющие описывать элементы или множества элементов документа. Наряду со стандартизацией [селекторов CSS3](https://professorweb.ru/my/css/css_theory/level1/1_2.php), другой стандарт консорциума W3C, известный как Selectors API, определяет методы JavaScript для получения элементов, соответствующих указанному селектору.

Ключевым в этом API является метод **querySelectorAll()** объекта Document. Он принимает единственный строковый аргумент с селектором CSS и возвращает объект NodeList, представляющий все элементы документа, соответствующие селектору.

В дополнение к методу querySelectorAll() объект документа также определяет метод querySelector(), подобный методу querySelectorAll(), - с тем отличием, что он возвращает только первый (в порядке следования в документе) соответствующий элемент или null, в случае отсутствия соответствующих элементов.

Эти два метода также определяются классом Elements. Когда они вызываются относительно элемента, поиск соответствия заданному селектору выполняется во всем документе, а затем результат фильтруется так, чтобы в нем остались только потомки использованного элемента.

Все объекты имеют методы (отделяются от объекта точкой), например:  
*document.write* позволяет писать текст в текущую страницу,  
*window.open* открывает новое окно браузера.  
Также объекты имеют свойства, например:  
*document.bgcolor* содержит значение фонового цвета текущей страницы,  
*document.title* содержит заголовок страницы.  
  
**Обработка событий**

Одним из главных (но не единственным) назначений сценариев в HTML-документе является обработка событий, таких как щелчок кнопкой мыши на элементе документа, помещение указателя мыши на элемент, перемещение указателя с элемента, нажатие клавиш и т.п. Значением таких атрибутов-событий в тегах HTML является строка, содержащая сценарий, выполняющий роль обработчика события

<div onClick="addText();"></div>

Здесь *Click* - событие (щелчок по div-у), *onClick* - обработчик события, *addText()* - имя функции, которая сработает при возникновении этого события (щелчка по div-у).  
  
События, которые поддерживаются javascript.

|  |  |  |
| --- | --- | --- |
| **событие** | **когда происходит** | **обработчик события** |
| Blur | потеря объектом фокуса | onBlur |
| Change | пользователь изменяет значение элемента | onChange |
| Click | пользователь щелкает мыщью по объекту | onClick |
| DblClick | пользователь делает двойной щелчок мышью по объекту | onDblClick |
| DragDrop | пользователь перетаскивает мышью объект | onDragDrop |
| Error | возникновение javascript-ошибки | onError |
| Focus | окно или элемент формы получает фокус | onFocus |
| KeyDown | пользователь нажимает клавишу клавиатуры | onKeyDown |
| KeyPress | пользователь удерживает нажатой клавишу клавиатуры | onKeyPress |
| KeyUp | пользователь отпускает клавишу клавиатуры | onKeyUp |
| Load | документ загружается в браузер | onLoad |
| MouseDown | пользователь нажимает кнопку мыши | onMouseDown |
| MouseOut | указатель мыши выходит за пределы элемента | onMouseOut |
| MouseOver | указатель мыши помещается над элементом | onMouseOver |
| MouseUp | пользователь отпускает кнопку мыши | onMouseUp |
| Move | пользователь перемещает окно | onMove |
| Reset | пользователь нажимает кнопку "reset" формы | onReset |
| Resize | пользователь изменяет размеры окна или элемента | onResize |
| Select | пользователь выбирает элемент формы | onSelect |
| Submit | пользователь нажимает кнопку "submit" формы | onSubmit |
| Unload | пользователь закрывает документ | onUnload |

Пример: html>

<head><title>Страница с изображением</title>

<script>

function clickimage() {

alert("Привет")

}

</script>

</head>

<body>

<p>Вариант 1

<img src="изображение0.jpg"

onclick="clickimage()"></p>

<p>Вариант 2

<img src="изображение1.jpg"

onclick='alert("Ура")'></p>

</body>

</html>

Пример

<html>

<body>

<h2>Поиск по id</h2>

<p id="demo">Искомый элемент</p>

<button type="button" onclick='document.getElementById("demo").innerHTML = "Приве JavaScript!"'>Click Me!</button>

</body>

</html>

Пример:

<html>

<head>

<title>Расчет площади прямоугольника</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma1">

Введите длину прямоугольника <input type="text" name="t1" size="10"><br><br>

Введите ширину прямоугольника <input type="text" name="t2" size="10"><br><br>

<input type="button" name="button" value="Вычислить"><br><br>

Площадь прямоугольника равна <input type="text" name="res" size="10">

</form>

</body>

</html>

function areaRectangle(){

var a=document.forma1.t1.value;

var b=document.forma1.t2.value;

var s=a\*b;

}