

Основы JavaScript



JavaScript

Функции в JavaScript. Объявление и ВЫЗОВ

В своей простейшей форме **функция** представляет собой часть программного кода, который в любое время может быть вызван по его имени. Зачастую нам надо повторять одно и то же действие во многих частях программы. Например, красиво вывести сообщение необходимо при приветствии посетителя, при выходе посетителя с сайта, еще где-нибудь.

Чтобы не повторять один и тот же код во многих местах, придуманы функции. **Функции** являются основными «строительными блоками» программы.

Примеры встроенных функций вы уже видели - это **alert (message)**, **prompt (message, default)** и **confirm (question)**. Но можно создавать и свои.

Функции в JavaScript. Объявление и ВЫЗОВ

Пример объявления функции:

```
function showMessage() {  
    alert('Привет всем присутствующим!');  
}
```

Вначале идет ключевое слово **function**, после него *имя функции*, затем *список параметров* в скобках (в примере выше он пустой) и *тело функции* — код, который выполняется при её вызове. Объявленная функция доступна по имени, например:

```
function showMessage() {  
    alert('Привет всем присутствующим!');  
}  
showMessage();  
showMessage();
```

Функции в JavaScript. Объявление и ВЫЗОВ

Этот код выведет сообщение два раза. Уже здесь видна **главная цель создания функций: избавление от дублирования кода.**

Если понадобится поменять сообщение или способ его вывода — достаточно изменить его в одном месте: в функции, которая его выводит.

Функция может содержать **локальные переменные**, объявленные через `var`. Такие переменные видны **только внутри (!!!)** функции.

Неважно, где именно в функции и сколько раз объявляется переменная. Любое объявление срабатывает один раз и распространяется на всю функцию.

Переменные, объявленные на уровне всего скрипта, называют **«глобальными переменными»**.

Делайте глобальными только те переменные, которые действительно имеют общее значение для вашего проекта. Пусть каждая функция работает «в своей песочнице».

Функции в JavaScript. Объявление и ВЫЗОВ

При вызове функции ей можно передать **данные**, которые та использует по своему усмотрению. Параметры копируются в локальные переменные функции.

Функцию можно вызвать с любым количеством аргументов.

Например, функцию показа сообщения `showMessage(from, text)` можно вызвать с одним аргументом:

```
showMessage("Маша");
```

Если параметр не передан при вызове — он считается равным **undefined**.

Стиль объявления функций в JavaScript

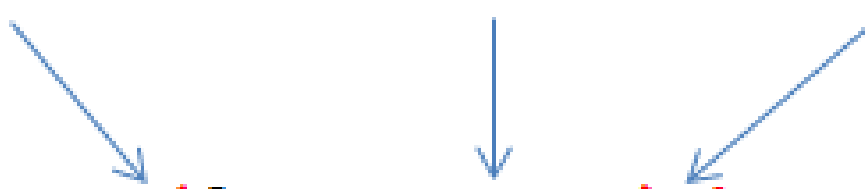
В объявлении функции есть **правила** для расстановки пробелов. Они отмечены стрелочками:

после названия
скобка и параметры
без пробела

параметры
через пробел

пробел перед {

```
function showMessage(fromName, text) {  
    var phrase = 'Сообщение от ' + fromName;  
  
    alert(phrase);  
    alert(text);  
}
```



Инструкция return для функции в JavaScript

Функция может вернуть некоторое значение (результат работы функции) программе, которая её вызвала. Возвращаемое значение передаётся в точку вызова функции.

Инструкция **return** внутри функции служит для определения значения, возвращаемого функцией. В качестве возвращаемого значения может быть значение любого типа. Если в качестве возвращаемого значения используется выражение, в программу возвращается не само выражение, а результат его вычисления.

Инструкция **return** имеет следующий синтаксис:

```
return выражение;
```

Для дальнейшего использования возвращаемого значения, результат выполнения функции можно присвоить например переменной:

```
function calc(a) {  
  return a * a;  
}  
  
var x = calc(5);  
document.write(x);
```

Инструкция `return` для функции в JavaScript

Инструкция `return` может быть расположена в любом месте функции. Обратите внимание, при выполнении кода внутри функции, как только будет достигнута инструкция `return`, функция возвращает значение и немедленно завершает своё выполнение. Код, расположенный в теле функции после инструкции `return`, будет проигнорирован.

```
var a = 1;
function foo() {
  ++a;
  return;
  ++a;
}
foo();
document.write(a); // => 2
```


Инструкция return для функции в JavaScript

Внутри функции может быть использовано несколько инструкций **return**, например если возвращаемое значение зависит от некоторых условий выполнения:

```
function check(a, b) {  
  if(a > b) return a;  
  else return b;  
}  
  
document.write(check(3, 5));
```

Инструкция return для функции в JavaScript

Если инструкция **return** не указана или не указано возвращаемое значение, то функция вернёт значение **undefined**.

```
// Инструкция return не указана
function bar() { document.write("функция bar выполнена"); }

// Возвращаемое значение не указано
function foo(a) {
  if(!a) return;
  document.write(a);
}

var a = bar(); // undefined
var b = foo(); // undefined

document.write("a: " + a + "<br> b: " + b);
```

Выбор имени функции в JavaScript

Имя функции следует тем же правилам, что и имя переменной. Основное отличие — оно должно быть **глаголом**, т.к. функция — это действие.

Как правило, используются глагольные префиксы, обозначающие общий характер действия, после которых следует уточнение.

Функции, которые начинаются с **"show"** — что-то показывают:

showMessage(..) // префикс show, "показать" сообщение

Функции, начинающиеся с **"get"** — получают, и т.п.:

getAge(..) // get, "получает" возраст

calcD(..) // calc, "вычисляет" дискриминант

createForm(..) // create, "создает" форму

checkPermission(..) // check, "проверяет" разрешение, возвращает true/false

Это очень удобно, поскольку взглянув на функцию — мы уже примерно представляем, что она делает, даже если функцию написал совсем другой человек, а в отдельных случаях — и какого вида значение она возвращает.

Выбор имени функции в JavaScript

Одна функция — одно действие!!!

Функция должна делать только то, что явно подразумевается её названием. И это должно быть одно действие.

Если оно сложное и подразумевает поддействия — может быть имеет смысл выделить их в отдельные функции? Зачастую это имеет смысл, чтобы лучше структурировать код.

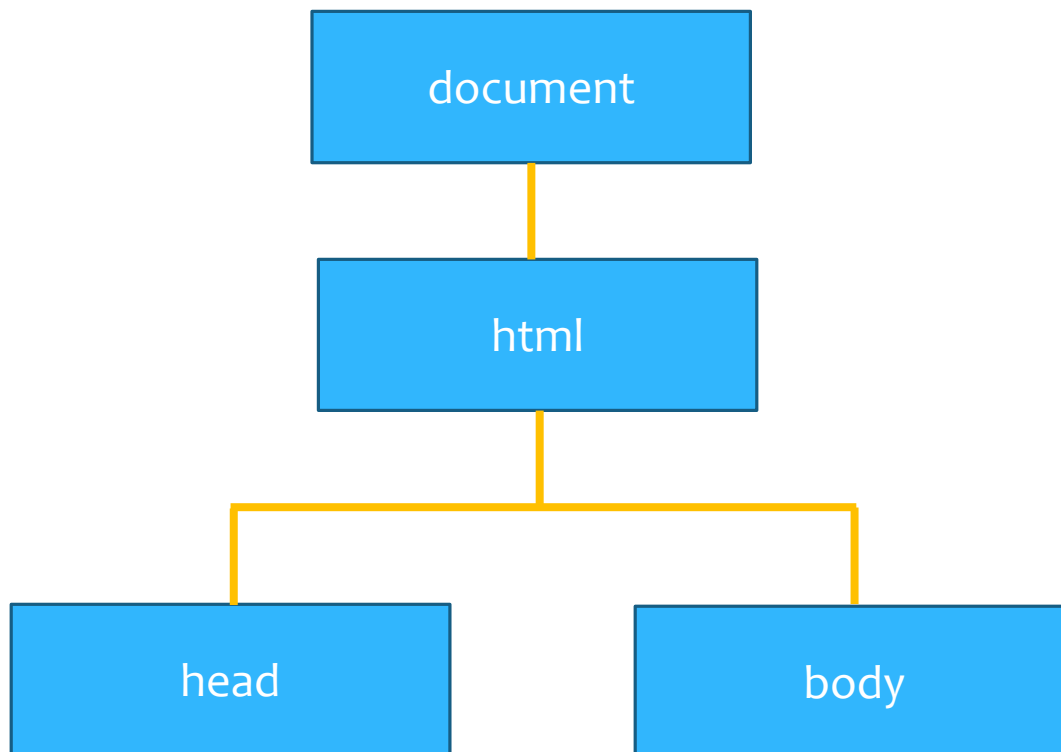
... Но самое главное — в функции не должно быть ничего, кроме самого действия и поддействий, неразрывно связанных с ним.

Например, функция проверки данных (скажем, **"validate"**) не должна показывать сообщение об ошибке. Её действие — проверить.

Объекты в JavaScript. Структура HTML и DOM

Главное предназначение языка HTML — это определение форматирования и структуры документа на уровне составляющих его элементов. Например, пустой HTML-документ имеет следующую иерархическую структуру:

```
<html>  
  <head></head>  
  <body></body>  
</html>
```



Объектная модель окна документа

Преимущество JavaScript, даже по сравнению со сложными языками программирования, состоит в том, что он позволяет управлять браузером.

Сценарий позволяет загрузить в окне браузера новую Web-страницу, управлять внешним ее видом и преобразовывать документ, а также запускать новые окна браузера. Для того чтобы справиться со всеми этими операциями, в JavaScript включена специальная иерархическая структура — **объектная модель документа (DOM)**.

В соответствии с ней все объекты сценария организованы в древовидную структуру, отвечающую за все без исключения элементы Web-страницы: окно, документ, рисунок и т.п. Объекты **DOM**, как и другие объекты, имеют свойства, описывающие Web-страницу, и методы, позволяющие управлять частью документа.

Объекты браузера упорядочены в иерархическом порядке, создавая структуру у родительских и дочерних объектов. При определении объекта вы сначала вводите имя родительского объекта, после него ставите разделитель (точку), а затем имя дочернего объекта.

Например, объекты, соответствующие рисункам Web-страницы, соответствуют дочерним объектам родительского объекта `document`. В следующем примере определен объект `image9`, который является дочерним по отношению к объекту `document`, который, в свою очередь, тоже дочерний — только уже по отношению к объекту `window`:

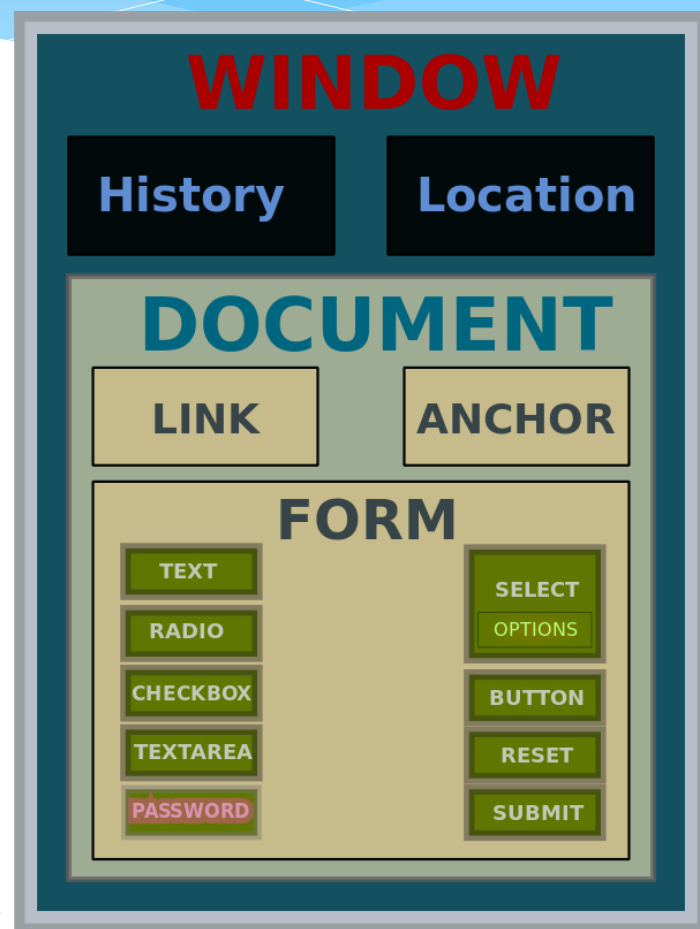
```
window.document.image9
```

Объектная модель окна документа

Модель DOM не накладывает ограничений на структуру документа. Любой документ известной структуры с помощью DOM может быть представлен в виде дерева узлов, каждый узел которого представляет собой элемент, атрибут, текстовый, графический или любой другой объект. Узлы связаны между собой отношениями "родительский-дочерний".

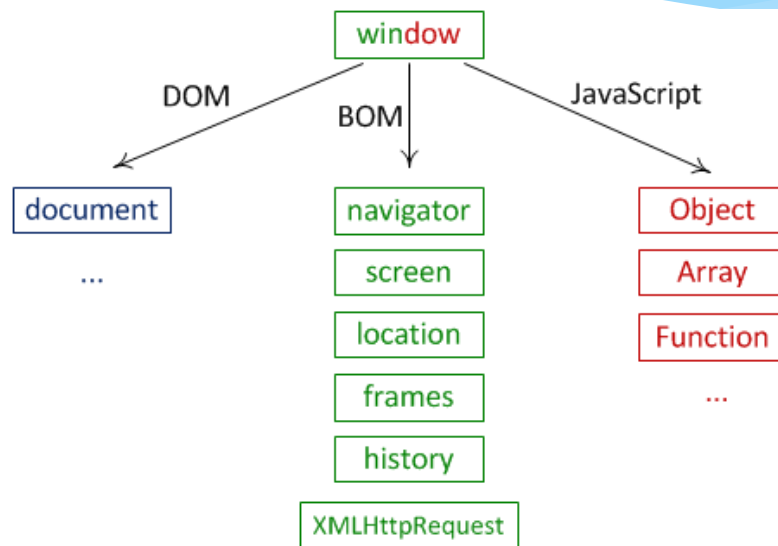
Изначально различные браузеры имели собственные модели документов (DOM), несовместимые с остальными. Для обеспечения взаимной и обратной совместимости, специалисты международного консорциума W3C классифицировали эту модель по уровням, для каждого из которых была создана своя спецификация. Все эти спецификации объединены в общую группу, носящую название W3C DOM (принята в апреле 2004 года).

Веб-браузеры не обязаны использовать **DOM**, чтобы исполнять HTML-документ. Однако **DOM** требуется для скриптов JavaScript, которые желают наблюдать или изменять веб-страницу динамически. Другими словами, **Document Object Model** — это инструмент, с помощью которого JavaScript видит содержимое HTML-страницы и состояние браузера.



Объектная модель окна документа

Сам по себе язык **JavaScript** не предусматривает работы с браузером. Он вообще не знает про HTML. Но позволяет легко расширять себя новыми функциями и объектами. На рисунке ниже схематически отображена структура, которая получается если посмотреть на совокупность браузерных объектов с «высоты птичьего полёта».



Как видно из рисунка, на вершине стоит **window**. У этого объекта двойная позиция — он с одной стороны является глобальным объектом в **JavaScript**, с другой — содержит свойства и методы для управления окном браузера, открытия новых окон, например:

```
// открыть новое окно/вкладку с URL http://www.exam.by  
window.open('http://www.exam.by');
```


Объектная модель браузера BOM

BOM — это объекты для работы с чем угодно, кроме документа.

Например:

- Объект **navigator** содержит общую информацию о браузере и операционной системе. Особенно примечательны два свойства: **navigator.userAgent** — содержит информацию о браузере и **navigator.platform** — содержит информацию о платформе, позволяет различать Windows/Linux/Mac и т.п.
- Объект **location** содержит информацию о текущем URL страницы и позволяет перенаправить посетителя на новый URL.
- Функции **alert/confirm/prompt** — тоже входят в **BOM**.

Пример использования:

```
alert( location.href ); // выведет текущий адрес
```

Большинство возможностей BOM стандартизированы в **HTML 5**, хотя различные браузеры и предоставляют зачастую что-то своё, в дополнение к стандарту.

Объектная модель окна документа

Глобальный объект **document** даёт возможность взаимодействовать с содержимым страницы.

Пример использования:

```
document.body.style.background = 'blue';  
alert( 'Элемент BODY стал синим, а сейчас обратно вернётся' );  
document.body.style.background = '';
```

Он и громадное количество его свойств и методов описаны в [стандарте W3C DOM](#).

По историческим причинам когда-то появилась первая версия стандарта **DOM Level 1**, затем придумали ещё свойства и методы, и появился **DOM Level 2**, на текущий момент поверх них добавили ещё **DOM Level 3** и готовится **DOM 4**.

Современные браузеры также поддерживают некоторые возможности, которые не вошли в стандарты, но де-факто существуют давным-давно и отказываться от них никто не хочет. Их условно называют «**DOM Level 0**».

Также информацию по работе с элементами страницы можно найти в стандарте [HTML 5](#).

Уровни W3C DOM

Текущим уровнем спецификаций DOM является Уровень 2, но тем не менее некоторые части спецификаций Уровня 3 являются рекомендуемыми W3C.

Уровень 0

Включает в себя все специфические модели DOM, которые существовали до появления Уровня 1, например, `document.images`, `document.forms`, `document.layers` и `document.all`. Необходимо обратить внимание, что эти модели формально не являются спецификациями DOM, опубликованными W3C, а скорее являются информацией о том, что существовало до начала процесса стандартизации.

Уровень 1

Базовые функциональные возможности DOM (HTML и XML) в документах, такие как получение дерева узлов документа, возможность изменять и добавлять данные.

Уровень 2

Поддержка так называемого пространства имён XML <--filtered views--> и событий.

Уровень 3

Состоит из шести различных спецификаций:

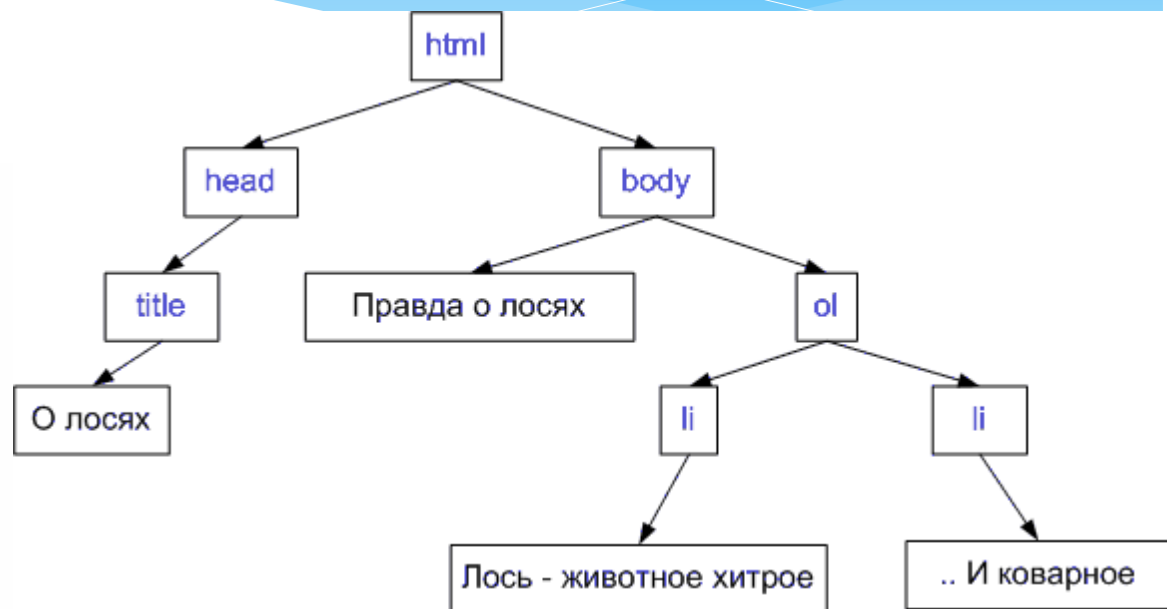
1. DOM Level 3 Core;
2. DOM Level 3 Load and Save;
3. DOM Level 3 XPath;
4. DOM Level 3 Views and Formatting;
5. DOM Level 3 Requirements;
6. DOM Level 3 Validation.

Эти спецификации являются дополнительными расширениями DOM.

Объекты в JavaScript. Структура HTML и DOM

Рассмотрим пример посложнее:

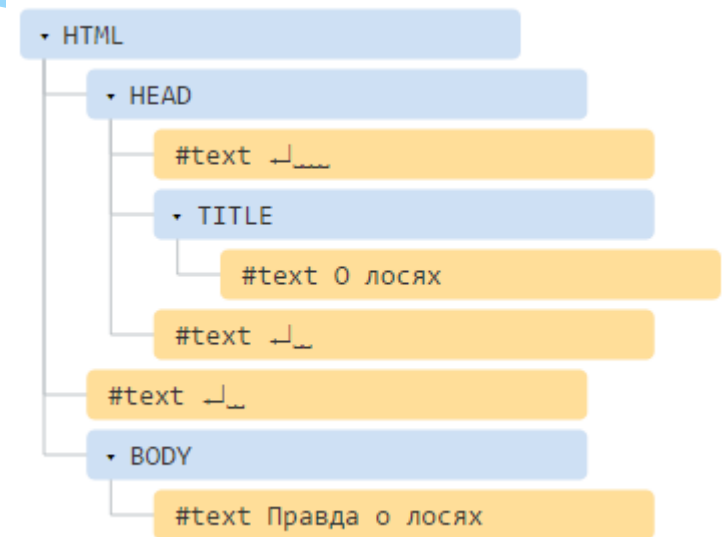
```
<html>  
  <head>  
    <title>  
      О лосях  
    </title>  
  </head>  
  <body>  
    Правда о лосях.  
    <ol>  
      <li>  
        Лось - животное хитрое  
      </li>  
      <li>  
        .. И коварное  
      </li>  
    </ol>  
  </body>  
</html>
```



Пример DOM

Построим, для начала, дерево **DOM** для следующего документа. Его вид:

```
<!DOCTYPE HTML>
<html>
<head>
  <title>0 лосях</title>
</head>
<body>
  Правда о лосях
</body>
</html>
```



В этом дереве выделено два типа узлов.

Теги образуют **узлы-элементы (element node)**. Естественным образом одни узлы вложены в другие. Структура дерева образована исключительно за счет них.

Текст внутри элементов образует **текстовые узлы (text node)**, обозначенные как **#text**. Текстовый узел содержит исключительно строку текста и не может иметь потомков, то есть он всегда на самом нижнем уровне.

Пример DOM

Всё, что есть в HTML, находится и в DOM!!!

Даже директива `<!DOCTYPE...>`, которую мы ставим в начале HTML, тоже является DOM-узлом, и находится в дереве DOM непосредственно перед `<html>`. На предыдущем слайде этот факт скрыт, поскольку мы с этим узлом работать не будем, он никогда не нужен.

Даже сам объект **document**, формально, является DOM-узлом, самым-самым корневым.

Всего различают 12 типов узлов, но на практике мы работаем с четырьмя из них:

Документ — точка входа в DOM.

Элементы — основные строительные блоки.

Текстовые узлы — содержат, собственно, текст.

Комментарии — иногда в них можно включить информацию, которая не будет показана, но доступна из JS.

Возможности, которые дает DOM

Зачем, кроме красивых рисунков, нужна иерархическая модель DOM?

DOM нужен для того, чтобы манипулировать страницей — читать информацию из HTML, создавать и изменять элементы.

Узел HTML можно получить как **document.documentElement**, а BODY — как **document.body**.

Получив узел, мы можем что-то сделать с ним. Например, можно поменять цвет **BODY** и вернуть обратно:

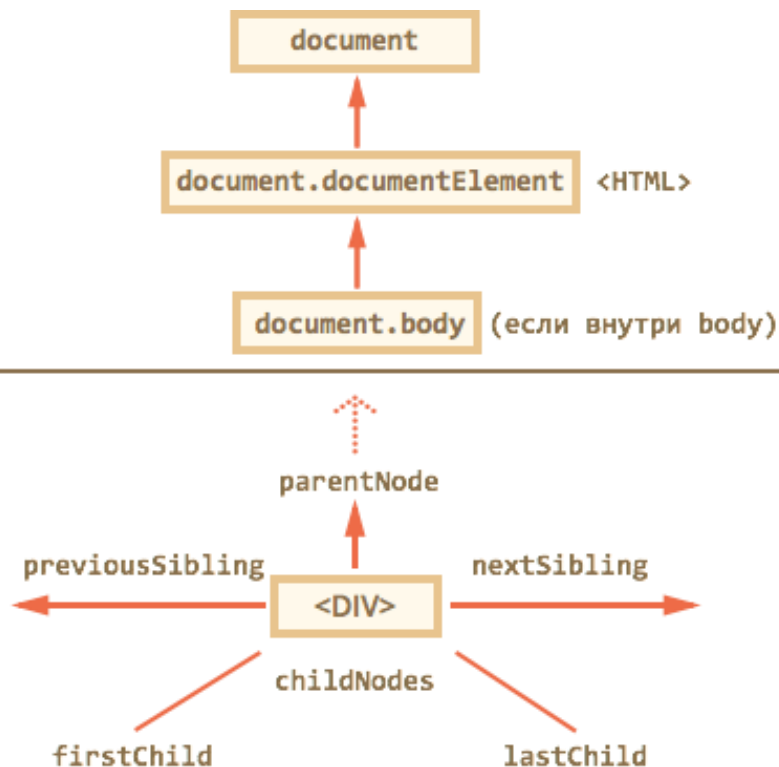
```
document.body.style.backgroundColor = 'red';  
alert( 'Поменяли цвет BODY' );
```

```
document.body.style.backgroundColor = '';  
alert( 'Сбросили цвет BODY' );
```

DOM предоставляет возможность делать со страницей всё, что угодно.

Навигация по DOM-элементам

DOM позволяет делать что угодно с HTML-элементом и его содержимым, но для этого нужно сначала нужный элемент получить. Доступ к DOM начинается с объекта **document**. Из него можно добраться до любых узлов. Так выглядят основные ссылки, по которым можно переходить между узлами DOM:



<HTML> = document.documentElement

Первая точка входа — **document.documentElement**. Это свойство ссылается на DOM-объект для тега **<html>**.

<BODY> = document.body

Вторая точка входа — **document.body**, который соответствует тегу **<body>**.

В современных браузерах (кроме IE8-) также есть **document.head** — прямая ссылка на **<head>**. В мире DOM в качестве значения, обозначающего «нет такого элемента» или «узел не найден», используется не **undefined**, а **null**.

Дети: `childNodes`, `firstChild`, `lastChild`

Здесь и далее мы будем использовать два принципиально разных термина.

Дочерние элементы (или дети) — элементы, которые лежат непосредственно внутри данного. Например, внутри `<HTML>` обычно лежат `<HEAD>` и `<BODY>`.

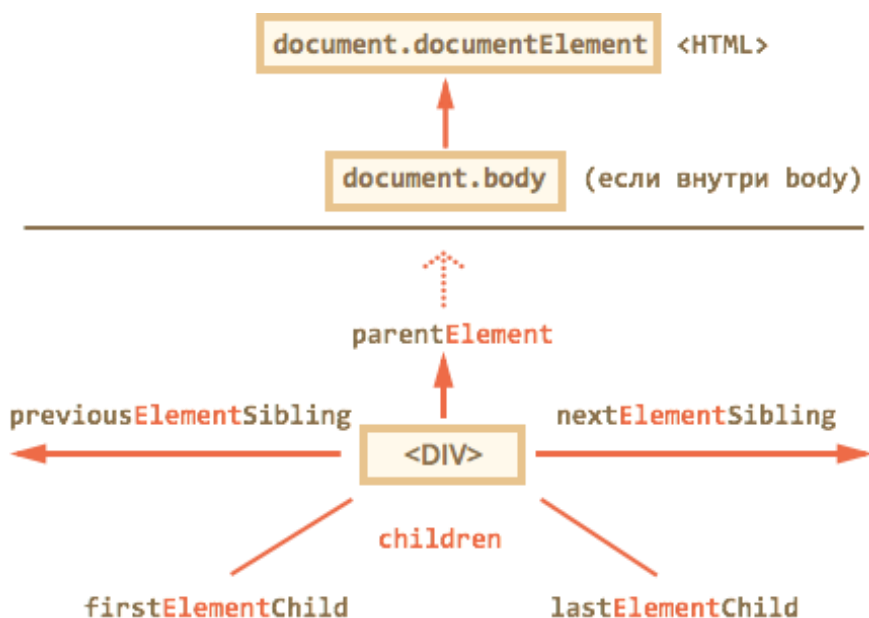
Потомки — все элементы, которые лежат внутри данного, вместе с их детьми, детьми их детей и так далее. То есть, всё поддерево DOM.

Доступ к элементам слева и справа данного можно получить по ссылкам **`previousSibling`** / **`nextSibling`**.

Родитель доступен через **`parentNode`**. Если долго идти от одного элемента к другому, то рано или поздно дойдёшь до корня DOM, то есть до **`document.documentElement`**, а затем и **`document`**.

Навигация только по элементам

Навигационные ссылки, описанные выше, равно касаются всех узлов в документе. В частности, в **childNodes** сосуществуют и текстовые узлы и узлы-элементы и узлы-комментарии, если есть. Но для большинства задач текстовые узлы нам не интересны. Поэтому посмотрим на дополнительный набор ссылок, которые их не учитывают:

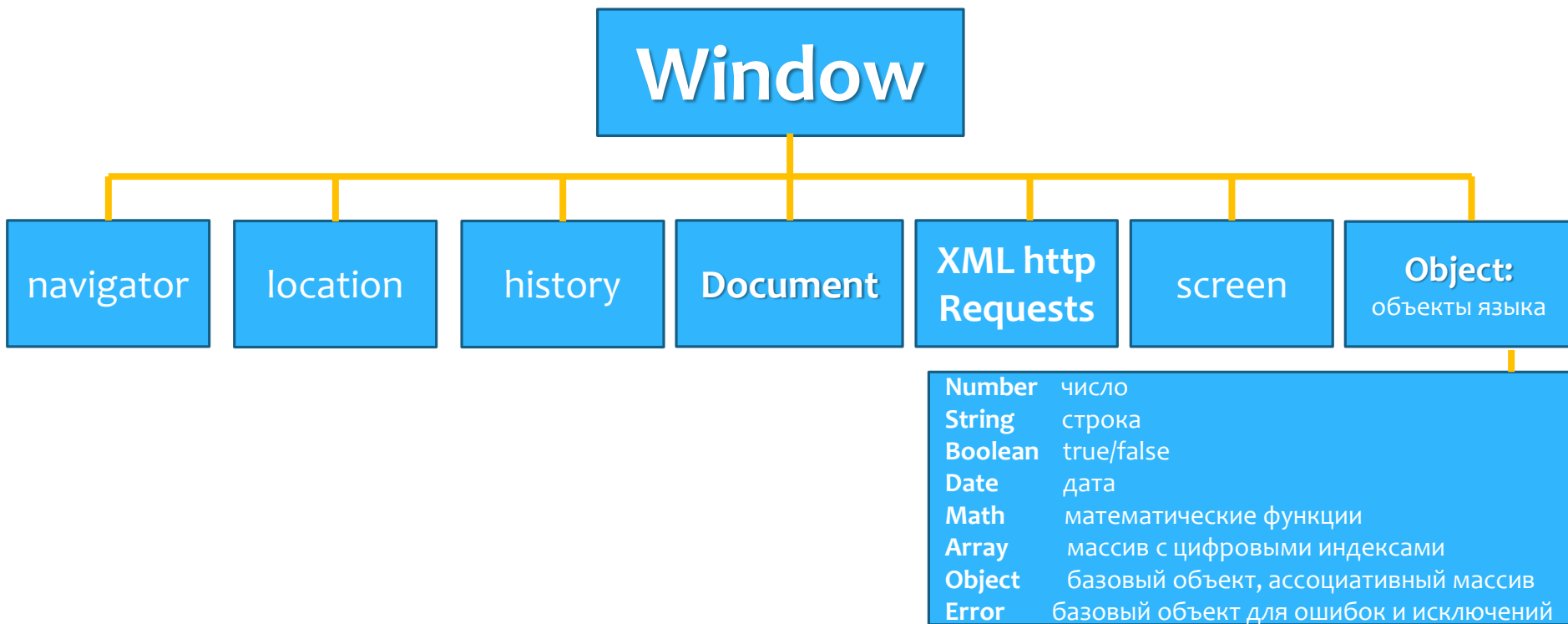


Эти ссылки похожи на те, что раньше, только в ряде мест стоит слово **Element**:

- **children** — только дочерние узлы-элементы, то есть соответствующие тегам.
- **firstElementChild**, **lastElementChild** — соответственно, первый и последний дети-элементы.
- **previousElementSibling**, **nextElementSibling** — соседи-элементы.
- **parentElement** — родитель-элемент.

Объектная модель окна документа

Как и предполагало название, формальная **DOM** описывает, в основном, HTML-документ и его содержимое. Но уже вскоре Вам придется в своих приложениях обращаться не только к элементам документа, но и к элементам среды – окнам браузера. **Общая иерархическая структура объектной модели** современных браузеров выглядит примерно так:



Объектная модель окна документа

- **Объект окна (window).** Вверху иерархической структуры находится окно (`window`). Этот объект представляет ту часть окна браузера, в которой отображается содержимое HTML-документа. В многофреймовой среде каждый фрейм (или кадр) также является окном (детально это пока рассматривать не будем). Поскольку все события, относящиеся к документу, происходят именно в окне, то оно и является самым общим элементом в иерархической структуре объектов. В нем, в буквальном смысле, размещен документ.
- **Объект navigator.** С помощью этого объекта определяется браузер, в котором выполняется сценарий. Он содержит сведения о производителе и версии браузера. Этот объект имеет атрибут только чтения (чтобы “зловредные” сценарии не могли получить доступ к свойствам браузера).
- **Объект экрана (screen).** Это еще один объект с атрибутом только чтения. Объект `screen` позволяет определить физическую среду, в которой запускается браузер. Например, с его помощью определяется разрешение экрана.
- **Объект history.** Каждый браузер сохраняет сведения об уже посещенных страницах (кнопка **Back (Назад)** на панели инструментов) в специальном стеке, представленном объектом `history`. Как и следовало ожидать, этот объект функционально дублирует кнопки **Back (Назад)** и **Forward (Вперед)**.
- **Объект location.** С помощью этого объекта проще всего загрузить в текущее окно целевую страницу или фрейм. Сведения об URL-адресе документа представлены таким образом, чтобы в сценарии нельзя было отследить защищенные Web-узлы.
- **Объект документа (Document).** Каждый HTML-документ, загружаемый в окно браузера, становится объектом `document` (документ). В иерархической структуре положение объекта документа является весьма важным. В объекте `document` содержится большинство остальных типов объектов модели. Со стороны это выглядит просто здорово: в документе находится то, что используется в сценарии.

Объекты в JavaScript.

Объект **window**

Объект. Метод. Свойства.

методы

**open()
write()
close()**

```
window.open("URL", "windowName", ["windowFeatures, . . ."]);
```

Свойства
атрибуты

height=100
Высоту окна в пикселах

scrollbars[=yes|no] | [=1|0]
Наличие линеек прокрутки

```
myWin=window.open("http://al.ru/exewin.htm",  
"wind1", "width=200,height=100,scrollbars=no,menubar=no");
```

Объекты в JavaScript.

Объект **document**

Содержит информацию относительно текущего документа, и с помощью методов позволяет также выводит для пользователя на экран текст HTML.

Методы

- close
- open
- write
- writeln

Свойства

AlinkColor отражает атрибут ALINK

anchor - массив, содержащий все якоря в документе

BgColor отражает атрибут BGCOLOR

cookie определяет cookie

FgColor отражает атрибут TEXT

form - массив, содержащий все формы в документе

LastModified отражает дату, когда последний раз документ был изменен

LinkColor отражает атрибут LINKS

link - массив, содержащий все links в документе

Referrer отражает URL вызывающего документа

title отражает содержание тега <TITLE >

URL отражает полный URL документа

VlinkColor отражает атрибут VLINK

Объекты в JavaScript.

Отображение и получение информации. Диалоги.

Window

```
alert("message")
```

`message` - любая строка или свойство существующего объекта.

Используйте метод `alert`, чтобы показать сообщение, которое не требует решения пользователя. Аргумент `message` определяет сообщение в диалоговом окне.

```
prompt(message, [inputDefault])
```

`message` - любая строка; строка вводится как сообщение.

`inputDefault` - строка, целое число, или свойство существующего объекта, который представляет значение по умолчанию области ввода.

Объекты в JavaScript.

```
confirm ("message")
```

`message` - любая строка или свойство существующего объекта.

Описание

Используйте метод `confirm`, чтобы пользователя подтвердил принятое решение. Аргумент `message` определяет сообщение, которое побуждает пользователя для решения. Подтверждение возвращает методу, `true`, если пользователь выбирает "ОК" или `false` если пользователь выбирает "Cancel".

```
document.write (expression1 [, expression2], ... [, expressionN])
```