Основы языка JavaScript

Язык JavaScript

JavaScript - это прототипно-ориентированный язык программирования, наиболее часто используемый в составе страниц HTML для увеличения функциональности и возможностей взаимодействия с пользователями. Он был разработан фирмой **Netscape** в сотрудничестве с **Sun Microsystems** на базе языка **Sun's Java**.

JavaScript - это язык для создания **активных** клиентских страниц: с его помощью можно:

- изменять содержимое HTML-документов;
- проверять введенные пользователем в форму значения без ее пересылки на сервер;
- выполнять сложные математические вычисления;
- управлять анимацией без использования каких-либо дополнительных средств

и т.п.

Скрипты JavaScript могут выполняться в результате наступления каких-либо событий, инициированных действиями пользователя – событий.

Использование JavaScript в HTML

Чтобы выполнять скрипты, написанные на языке JavaScript необходим браузер, способный работать с JavaScript – любой современный браузер.

JavaScript программы, могут располагаться непосредственно в HTML-документах. Для этого в HTML используется пара тегов <SCRIPT> и </SCRIPT>:

<SCRIPT LANGUAGE="JavaScript"> ... программа на JavaScript ...
</SCRIPT>
Или
<SCRIPT type=«application/javascript»> ... программа на JavaScript ... </SCRIPT>

Для того чтобы браузеры, не поддерживающие скриптовые программы, могли пропустить их, обычно программы располагаются внутри блока комментариев, как показано ниже:

```
<SCRIPT LANGUAGE="JavaScript">
<!—
 ...
 программа на JavaScript
...
//-->
 </SCRIPT>
```

Скрипты JavaScript могут располагаться не только непосредственно в теле HTML-документа, но и во внешнем файле. Для определения места расположения файла, содержащего используемый код JavaScript, используется атрибут SRC тэга <SCRIPT>. Например:

<SCRIPT SRC="common.js">
</SCRIPT

Операторы JavaScript в теле тэга <SCRIPT> с атрибутом SRC игнорируются.
В атрибуте SRC можно задать любой URL, относительный или абсолютный. Например:

<SCRIPT SRC="js/func.js"></ SCRIPT >
<SCRIPT SRC="http://home.netscape.com/functions/jsfuncs.js"></ SCRIPT >

Внешние файлы с кодом JavaScript не должны содержать никаких тэгов HTML - они обязаны содержать только операторы и определения функций JavaScript. Внешние файлы JavaScript должны иметь расширение файла ".js"

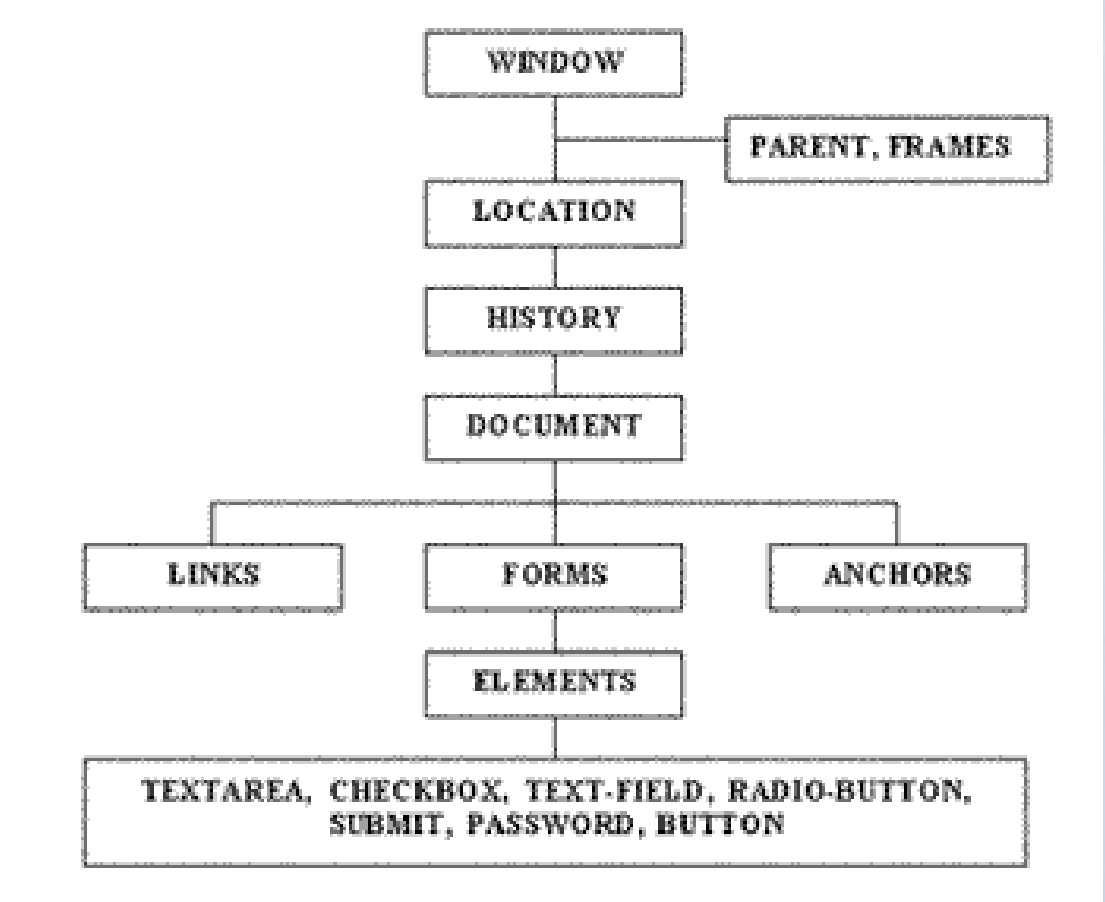
Объектная модель JavaScript

JavaScript основан на объектно-ориентированном подходе. Объект - это конструкция, обладающая *свойствами* и *методами*. Свойства являются переменными *JavaScript*. Свойства могут быть и другими объектами. Функции, связанные с объектом, называются *методами* объекта.
Объект *JavaScript* имеет свойства, ассоциированные с ним. Для обращения к свойствам объекта используется символ ".":

objectName.propertyName

Объектная модель браузера в JavaScript – это набор связанных между собой объектов, обеспечивающих доступ к содержимому страницы и ряду функций браузера. В составе объектной модели есть три основных объекта:

- **window -** представляет текущее окно браузера (находится на вершине иерархии)
- **document** – предназначен для программного представления самого документа и его содержимого.
- **frames -** предназначен для работы с фреймами в окне браузера.



window - объект, дающий доступ к окну браузера
frames - объект, дающий доступ к фреймам
- window...
  - ...

document - объект, содержащий в себе всю HTML-страницу
- all - полная коллекция всех тегов документа
- forms - коллекция форм
  - elements – коллекция элементов формы
    - button – объект, соответствующий тэгу <input type=button>
    - checkbox - объект, соответствующий тэгу <input type=checkbox>
    - radio - объект, соответствующий тэгу <input type=radio>
    - text - объект, соответствующий тэгу <input type=text>
    - textarea - объект, соответствующий тэгу <textarea>
- anchors - коллекция якорей
- applees - коллекция апплетов
- embeds - коллекция внедренных объектов
- filters - коллекция фильтров
- images - коллекция изображений
- links - коллекция ссылок
- plugins - коллекция подключаемых модулей
- scripts - коллекция блоков <script></script>
- selection - коллекция выделений
- stylesheets - коллекция объектов с индивидуально заданными стилями

history - объект, дающий доступ к истории посещенных ссылок
navigator - объект, дающий доступ к характеристикам браузера
location - объект, содержащий текущий URL
event - объект, дающий доступ к событиям
screen - объект, дающий доступ к характеристикам экрана

Из методов объекта **document** пока отметим только метод **write**, выводящий указанный текст в окно браузера, и **writeln**, выводящий текст, в конце которого устанавливается символ «возврата каретки» (для того, чтобы символ «возврата каретки» отображался браузером метод writeln должен быть заключен между тегами <PRE>. Пример.

Метод **alert** объекта **window** позволяет отображать на экране браузера различные сообщения – эти сообщения появляются в диалоговом окне, содержащем кнопку OK. Пример :

```
<SCRIPT LANGUAGE = "Java Script" >
window.alert ( "Пример вывода сообщения ")
</SCRIPT>
```

Переменные JavaScript

Переменные языка JavaScript могут хранить значения различных типов:

- **Строки** - последовательность символов;
- **Числовые значения** - целые и действительные числа;
- **Булевы значения** - только два значения **true** или **false**;
- **Массивы** - множества однотипных переменных;
- **Даты** - значения даты и времени.

Имена переменных могут начинаться с любой буквы (от a до z, или A-Z) либо с символа подчеркивания "_", оставшаяся часть может содержать цифры, символы подчеркивания и буквы. Следует отметить, что в именах переменных различаются символы верхнего и нижнего регистра

В языке JavaScript переменные можно переопределять, даже задавая другой тип данных.

```
<script language="JavaScript">
<!--
var a = 35, b = "5", c;
c = a + b;
c = c - a;
document.writeln("C="+c);
-->
</script>
```

Переменную можно определить и не используя оператор "Var" - достаточно присвоить значение - причем какой тип данных будет присвоен, такого типа и окажется переменная. Переменная может быть определена и без начальных значений

Время жизни и видимость переменной связаны с окном, в котором они созданы и зависят от того, где они определены. Переменные, определенные внутри тела JavaScript-функции видны только внутри этой функции.

Если идентификатор переменной устанавливается путём присвоения вне какой-либо функции, такая переменная называется ***глобальной***, и доступна в любом месте документа. Если переменная объявляется внутри функции, она называется ***локальной***, и доступна только внутри данной функции.

Использование var при объявлении глобальной переменной не требуется. Однако обязательно использовать var при объявлении переменой внутри функции.

JavaScript-программы содержатся в документах HTML, и при загрузке нового документа в браузер любые переменные, созданные в программе, будут удалены. Чтобы сохранить какие-либо значения при загрузке нового документа в JavaScript имеются только 2 решения:

•путем создания переменных во фреймосодержащем документе верхнего уровня;
•путем использования "cookies";

Строковые переменные

Строка представляет собой множество символов, заключенных в одинарные или двойные кавычки. **Строки** в JavaScript представляются **объектами**.

Создать объект **String** можно одним из нескольких способов:

- присваивание значения при помощи конструктора String();
- использование оператора Var и оператора присваивания для создания и инициализации строки;
- создание и инициализация строковой переменной в операторе присваивания;
- преобразование переменной числового типа путем сложения со строковым типом (10+"" ==> "10");

Примеры создания строк:

```
var myVariable = "Строка первая";
var myVariable = new String();
var myVariable = new String("Строка другая ");
```

Строковый объект может содержать специальные символы, управляющие форматированием строк:

\n - символ новой строки;

\r - символ возврата каретки;

\f - код перехода на новую страницу;

\xnn - представление символа в виде шестнадцатиричного ASCII-кода nn;

\b - код клавиши [Backspace].

Эти символы будут правильно интерпретированы только в контейнерах <textarea> и при использовании функции вывода сообщения - "Alert".

Объект String имеет только **одно свойство - length**, значением которого является количество символов в строке, содержащейся в объекте.

Методы объекта String можно разделить на две категории:

- методы форматирования HTML-документа;
- методы обработки строк.

Метод	Тип	Описание
anchor()	У	Создает именованную метку, т.е. тег <a name> из содержимого объекта(строки)
Big()	Ф	Заключает строку в контейнер <big> . . . </big>, для отображения крупным шрифтом
blink()	Ф	Заключает строку в контейнер <blink> . . . </blink>, чтобы она отображалась мигающей.
bold()	Ф	Заключает строку в контейнер . . . , чтобы она отображалась жирным шрифтом.
charAt()	У	Возвращает символ, находящийся в заданной позиции строки
fixsed()	Ф	Заключает строку в контейнер <tt> . . . </tt>, чтобы она отображалась шрифтом постоянной ширины.
fontcolor()	Ф	Заключает строку в контейнер . . . , чтобы она отображалась определенным цветом.
fontsize()	Ф	Заключает строку в контейнер . . . , чтобы она отображалась шрифтом определенного размера.
indexOf(arg1[,arg2])	У	Возвращает позицию в строке, где впервые встречается символ - arg1, необязательный числовой аргумент arg2 указывает начальную позицию для поиска.
italics()	Ф	Заключает строку в контейнер . . . , чтобы она отображалась курсивом.
lastindexOf(arg1 [,arg2])	У	Возвращает либо номер позиции в строке, где в последний раз встретился символ - arg1, либо -1, если символ не найден. Arg2 задает начальную позицию для поиска.

link()	У	Создает тег гиперсвязи <a href> . . . и помещает в него содержимое объекта
small()	Ф	Заключает строку в тег . . . , чтобы она отображалась шрифтом меньшего размера.
strike()	Ф	Заключает строку в контейнер <strike> . . . </strike>, чтобы она отображалась зачеркнутой.
Sub()	Ф	Заключает строку в контейнер _{. . .}, чтобы она отображалась как нижний индекс.
substring(arg1,arg2)	У	Возвращает подстроку длинной arg2, начиная с позиции arg1.
Sup()	Ф	Заключает строку в контейнер ^{. . .}, чтобы она отображалась как верхний индекс.
toLowerCase()	У	Преобразует все буквы строки в строчные
toUpperCase()	У	Преобразует все буквы строки в прописные

Операция конкатенации (+) может быть использована для объединения двух строковых значений, возвращая строку, которая является результатом объединения двух строк-операндов.
Например, "my " + "string" возвращает строку "my string".

Операция-аббревиатура присвоения += также может использоваться для конкатенации строк.

Например, если переменная mystring имеет значение "alpha", то выражение mystring+="bet" вычисляется в "alphabet" и это значение присваивается переменной mystring.

В примере производится подсчет количества слов в строке (предполагается, что слова разделены одним пробелом)

Числовые переменные

Числовые значения могут быть либо целыми, либо числами с плавающей точкой. Числа с плавающей точкой называют действительными или вещественными. К числовым значениям применимы операции:

- умножения (*);
- деления (/);
- сложения (+);
- вычитания (-);
- увеличения (++);
- уменьшения (--);

Кроме того, используются операции умножения, деления, сложения и вычитания в сочетании с присваиванием (*=, /=, +=, -=).

Булевы переменные

Булевы, или логические, переменные содержат только литеральные значения - true и false - и используются в логических выражениях и операторах. Вместо true и false можно использовать числовые значения 1 и 0.

Операция	Описание	Примеры, возвращающие true(var1=3, var2=4)
Равно (==)	Возвращает true, если операнды равны. Если два операнда имеют разные типы, JavaScript пытается конвертировать операнды в значения, подходящие для сравнения.	3 == var1 "3" == var1 3 == '3'
Не равно (!=)	Возвращает true, если операнды не равны. Если два операнда имеют разные типы, JavaScript пытается конвертировать операнды в значения, подходящие для сравнения.	var1 != 4 var2 != "3"
Строго равно (===)	Возвращает true, если операнды равны и одного типа.	3 === var1
Строго не равно (!==)	Возвращает true, если операнды не равны и/или не одного типа.	var1 !== "3" 3 !== '3'
Больше (>)	Возвращает true, если левый операнд больше правого операнда.	var2 > var1
Больше или равно (>=)	Возвращает true, если левый операнд больше правого операнда или равен ему.	var2 >= var1 var1 >= 3
Меньше (<)	Возвращает true, если левый операнд меньше правого операнда.	var1 < var2
Меньше или равно (<=)	Возвращает true, если левый операнд меньше правого операнда или равен ему.	var1 <= var2 var2 <= 5

Массивы в JavaScript

В языке Java Script наборы значений одного типа(массивы) чаще всего представляются объектом Array. Массивы создаются при помощи конструктора Array(). С помощью конструктора Array() не только создается сам объект array, но и могут быть присвоены начальные значения его элементам. Количество элементов массива доступно через свойство length. К элементу массива следует обращаться при помощи выражения:

arrayName[index]

```
var path = "c:/images/" ,
arrayImg = new Array();
arrayImg[0] = path+"img1.gif";
arrayImg[1] = path+"img2.gif";
```

Элементы массива также могут быть заданы как параметры конструктора:

```
var path = "c:/images/" ,
arrayImg = new Array(path+"img1.gif", path+"img2.gif");
```

Как отмечалось выше, в языке JavaScript можно создавать массив, в котором элементы имеют различный тип данных. Например:

```
var myArray = new Array(3.14, true, 85, date(), "word");
```

Размер массива и, следовательно, значение свойства length массива, создаваемого конструктором Array(), зависят от максимального значения индекса, который применялся для задания элемента массива. Например:

```
var myArray = new Array;
myArray[20] = "Это 21 элемент массива";
```

Значение свойства length объекта Array автоматически устанавливается при явном указании количества элементов в конструкторе Array():

myArray = new Array(10);

Кроме того, существует возможность задать значения элементов массива при его конструировании :

myArray = new Array(0,0,0,0,0,0);

В JavaScript существует достаточно **много встроенных методов для работы с массивами**, которые поддерживаются браузерами:

• pop - удаляет последний элемент в массиве и возвращает удалённый элемент.

• push - добавляет в конец массива произвольное количество элементов, которые он принимает в качестве параметров. Этот метод возвращает длину нового увеличенного массива.

• shift - удаляет нулевой элемент массива и возвращает удалённый элемент.

• unshift - добавляет в начало массива произвольное количество элементов, которые принимает в качестве параметров. Этот метод возвращает длину нового увеличенного массива.

• reverse - меняет порядок следования элементов в массиве на обратный и возвращает новый перевёрнутый массив.

• join - этот метод ‘склеивает’ элементы массива в одну строку, используя в качестве разделителя переданный аргумент, массив при этом не изменяется.

Аргумент у этого метода можно опускать. В этом случае элементы массива будут соединены запятыми.

• slice - возвращает часть исходного массива. В качестве параметров принимает индекс элемента, с которого будет начинаться вырезанный массив и индекс, на единицу превосходящий индекс элемента, которым вырезанный массив будет заканчиваться вырезанном массиве. Этот метод возвращает вырезанный массив, не изменяя исходный.
Второй параметр в методе slice можно опускать. В этом случае выборка элементов в новый массив продолжится до последнего элемента исходного массива.

• splice - этот метод можно использовать как для массового удаления элементов из массива, так и для массового добавления. В качестве параметров он принимает индекс элемента, с которого начинать удаление и количество удаляемых элементов. Возвращает метод массив из удалённых элементов.
Для удаления элемента из массива по индексу следует передать методу splice в качестве первого параметра индекс удаляемого элемента, а в качестве второго единицу.

• concat - склеивает массивы

• sort - используется для сортировки массива. Возвращает отсортированный массив. Стоить заметить, что массив по умолчанию сортируется в лексикографическом порядке, то есть в порядке возрастания символов в таблице ASCII. Если элементы массива числа, то этот порядок не совпадает с порядком возрастания чисел:

```
var arr = [7, 6, 5, 11, 23, 45];
arr.sort();
console.log(arr); // [11, 23, 45, 5, 6, 7]
```

Если элементы массива строки, то после сортировки они будут расположены в алфавитном порядке.

Метод sort в качестве необязательного параметра может принимать функцию. Эта функция принимает в качестве аргументов два сравниваемых по какому-то критерию элемента массива. Она должна возвращать отрицательное число, если первый аргумент меньше второго, ноль, если аргументы равны и положительное число, если второй аргумент меньше первого.

Для сортировки массива чисел по возрастанию эта функция может выглядеть так.

```
function sortFunct(a, b) {
  if (a < b) {
    return -1;
  } else if(a > b) {
    return 1;
  }
  return 0;
}
```

• filter – этот метод выбирает элементы массива по определённому критерию и возвращает их в массиве. В качестве параметра он принимает функцию. Эта функция принимает в качестве параметра элемент массива, который оценивает по какому-то признаку и должна возвращать true, если элемент удовлетворяет этому признаку, и false в противном случае. Например, отберём все чётные элементы массива:

```
var arr = [1, 2, 3, 4, 5, 6, 7, 8];
console.log(arr.filter(function (item) { // [2, 4, 6, 8]
  return item % 2 == 0;
}));
```

• forEach - эта функция принимает в качестве параметра функцию, которая будет вызвана для каждого элемента массива. Выведем с помощью это функции элементы массива в столбик на консоль.

```
var arr = ['one', 'two', 'three', 'four', 'five'];
arr.forEach(function (item) {
  console.log(item);
});
```

map - создаёт новый массив, состоящий из изменённых элементов старого. В качестве параметра этот метод принимает функцию, которая изменяет элементы массива. Увеличим все элементы массива на единицу.

```
var arr = [1, 2, 6, 8, 2, 9];

arr = arr.map(function (item) {
  return ++item;
});

console.log(arr); // [2, 3, 7, 9, 3, 10]
```

Многие из рассмотренных методов возвращают массив, что позволяет комбинировать их в цепочки вызовов, в которых каждый последующий вызов метода производится от результата вызова предыдущего. Например, выберем из массива все чётные числа разделим их на два:

```
var arr = [1, 2, 3, 4, 5, 6, 7, 8];

arr = arr
  .filter(function (item) { // отобрали чётные числа
    return item % 2 == 0;
  })
  .map(function (item) { // делим их на два
    return item / 2;
  });

console.log(arr); // [1, 2, 3, 4]
```

Преобразование типов данных

Как было отмечено выше, JavaScript это динамически типизированный язык. Это означает, что нет необходимости специфицировать тип данных переменной при её объявлении и что типы данных при необходимости автоматически конвертируются при выполнении скрипта.

При преобразовании строк и чисел используются следующие два правила:

•Преобразование **числа в строку** символов производится путем сложения числового аргумента со строковым, независимо от перестановки слагаемых. Например, если переменная varI=123, то преобразовать переменную и следовательно ее значение в строку символов можно: varI = varI + "" или наоборот: varI = "" + varI.

Если сложить не с пустой строкой: varI = varI + "456", то результатом значения переменной varI станет "123456".

Это же справедливо и в случае: varI = "456" + varI - результат: "456123";

•Преобразование **строки в число** производится путем вычитания одного операнда из другого и также независимо от их позиции. Например, если переменная varI = "123", то преобразовать ее в число можно если вычесть из нее значения 0: varI = varI - 0, и соответственно значение переменной из строкового типа преобразуется в числовой: 123. При перестановке операндов соответственно знак числового значения поменяется на противоположный.

- В отличие от преобразования числа в строку в действиях вычитания нельзя применять буквенные значения.
- Так если "JavaScript"+10 превратится в varI == "JavaScript10", то операция типа varI = "JavaScript" - 10 выдаст значение "NON" – то есть такая операция не допустима.
- Следует так же отметить, что при вычитании строкового значения из строкового же также происходит преобразование: varI = "20" - "15", значением переменной varI станет число 5.

Объекты JavaScript, соответствующие тегам HTML

Имя объекта	Краткое описание
anchor (anchors[])	Множество тегов <a name> в текущем документе
button	Кнопка, создаваемая при помощи тега <input type=button>
checkbox	Переключатель, создаваемый при помощи тега <input type=checkbox>
elements[]	Все элементы тега <form>
form (forms[])	Множество объектов тега <form> языка HTML
frame (frames[])	Фреймосодержащий документ
hidden	Скрытое текстовое поле, создаваемое при помощи тега <input type=hidden>
images (images[])	Множество изображений (тегов <img>) в текущем документе
link (links[])	Множество гиперсвязей в текущем документе
navigator	Объект, содержащий информацию о браузере, загрузившем документ
password	Поле ввода пароля, создаваемое при помощи тега <input type=password>
radio	Radio button, создаваемая при помощи тега <input type=radio>
reset	Кнопка очистки формы, создаваемая при помощи тега <input type=reset>
select (options[])	Элементы <option> объекта <select>
submit	Кнопка передачи данных, создаваемая при помощи тега <input type=submit>
text	Поле ввода, создаваемое при помощи тега <input type=text>
textarea	Поле текста, создаваемое при помощи тега <textarea>

Объект button

К свойствам и методам объекта button можно обратиться одним из способов:

buttonName.propertyName
buttonName.methodName (parameters)
formName.elements[i].propertyName
formName.elements[i].methodName (parameters)

Свойства: свойства **name** и **value** соответствует атрибутам name и value HTML-тега <input>.

Свойство **type** всегда имеет значение "button".

Методы и обработчики событий: объект button имеет метод **click**(). Обработчик событий **onClick** позволяет выполнить оператор или вызвать функцию языка JavaScript при щелчке мыши на кнопке, которой соответствует в программе определенный объект button.

Пример.

Объект checkbox

Контрольный переключатель - это флажок, который можно установить в одно из двух состояний: включено или выключено.

К объекту checkbox можно обращаться одним из способов:

checkboxName.propertyName
checkboxName.methodName (parameters)
formName.elements[i].propertyName
formName.elements[i].methodName (parameters)

Свойства:

- **checked - ** если контрольный переключатель включен, свойство **checked** имеет значение true;
- **defaultChecked** - когда в теге <input> используется атрибут checked, например <input checked type=checkbox>, свойсту **defaultChecked** также присваивается значение true.
- **name** - Свойство **name** соответствует атрибуту name тега input
- **value -** соответствует атрибуту value тега <input>.
- **type** - всегда содержит значение "checkbox".

Методы и обработчики событий: с объектом checkbox может использоваться метод **Click()**. Для объекта checkbox предусмотрен только один обработчик - **onClick()**.

Пример.

Объект radio

Для обращения к методам и свойствам селекторной кнопки используют выражения:

radioName[i].propertyName
radioName[i].methodName(parameters)
formName.elements[i].propertyName
formName.elements[i].methodName(parameters)

Следует иметь в виду, что для группы селекторных кнопок элементы массива для каждого элемента этой группы располагаются в обратном порядке.

Свойства:

checked - содержит булево значение true или false, в зависимости от того, выбрана или нет данная селекторная кнопка;

defaultChecked - соответствует атрибуту checked тега <input type="radio">, и также содержит булево значение;

length - представляет количество селекторных кнопок в объекте radio;

name - соответствует атрибуту name тега <input>

value - соответствует атрибуту value тега <input>

type - значением этого атрибута является строка "radio"

Методы и обработчики событий: Для выбора селекторной кнопки используется метод click(). Объект radio имеет обработчик событий onClick, который используется для обработки событий, связанных с активизацией селекторной кнопки.

Пример.

Объект select и массив options

Элементам списка, определенным в тегах <option>, в JavaScript соответствуют элементы массива **options**, массив является свойством объекта **select**. Если тег <select> с именем test содержит два тега <option>, в JS им соответствуют элементы test.options[0] и test.options[1].

Массив **options**, в свою очередь, имеет свойство **length**. Значение этого свойства - количество тегов <option> в заданном объекте select.

Свойства select:

- **length -** количество тегов <option>, заданных в теге <select>
- **name -** соответствует атрибуту name тега <select>
- **options -** массив значений тегов <option>
- **selectIndex -** содержит индекс выбранного элемента, а если выбрано несколько элементов, то индекс первого;
- **type -** для списков с возможностью выбора одного элемента содержит значение "select-one", а для списков с возможностью выбора нескольких элементов - значение "select-multiple"

Свойства массива options:

- **defaultSelected -** соответствует первому тегу <option>, определенному с атрибутом selected;
- **index -** номер данного элемента в массиве;
- **length -** количество элементов в списке объекта select;
- **selected -** не равно нулю, если данный элемент списка выбран
- **selectedIndex -** содержит индекс выбранного элемента
- **text -** соответствует тексту, который указан в теге <option>
- **value -** соответствует атрибуту value тега <option>

Методы и обработчики событий: объект **select** имеет методы **focus()** и **blur()**. Для объекта select можно определить обработчики событий, соответствующие атрибутам onBlur и onChange, задавая реакцию объекта на события, связанные с потерей и получением фокуса ввода.

Пример

Объект text

Для обращения к методам и свойствам объекта text используют выражения вида:

textName.propertyName
textName.methodName(parameters)
formName.elements[i].propertyName
formName.elements[i].methodName(parameters)

Свойства:

- defaultValue соответствует атрибуту value.
- value является текущее значение объекта text.
- name соответствует атрибуту name объекта text,
- type - содержит значение "text".

Методы и обработчики событий: объект text имеет три метода: **focus(), blur() и select().**
Для объектов text можно определить четыре обработчика событий: **onBlur, onChange, onFocus и onSelect**.

Объект textarea

Для обращения к методам и свойствам объекта textarea применяются типичные для элементов формы выражения:

textareaName.propertyName
textareaName.methodName(parameters)
formName.elements[i].propertyName
formName.elements[i].methodName(parameters)

Свойства: объекты textarea имеют свойства:

- **defaultValue** - значение содержит текст, помещенный в контейнер <textarea> . . . </textarea>
- **name** - соответствует атрибуту name тега <textarea>
- **value** - соответствует текущему значению объекта textarea т.е. текущему содержимому области текста;
- **type** - для объекта textarea всегда содержит значение "textarea".

Методы: focus() , blur() , select() .

Обработчики событий: onBlur, onChange, onFocus и onSelect

Пример

Массив elements

При обращении к массиву следует указывать имя формы или ее индекс в массиве **:**

formName.elements[i]
forms[i].elements[i]

Свойства:

- **length -** значение которого - количество элементов объекта form.

Массив **elements** включает данные **только для чтения,** т.е. динамически записать в этот объект какие-либо значения невозможно**.**

Пример

Объект form и массив forms

Использование массива forms: к любой форме текущего гипертекстового документа можно обращаться как к элементу массива forms.

document.forms[i]
document.forms.length
document.forms['name']

Свойства: объект **form** имеет шесть свойств:

- **action -** соответствует атрибуту **action;**
- **elements -** массив, содержащий все элементы формы**;**
- **encoding -** соответствует атрибуту **enctype;**
- **length -** количество элементов в форме**;**
- **method -** соответствует атрибуту **method;**
- **target -** соответствует атрибуту **target**

Массив **forms** имеет только одно свойство **length -** количество форм в документе**.**

Методы: метод **submit()** применяется для «оправки» формы из JavaScript-программы**.**

Обработчики событий: обработчик события onSubmit() позволяет перехватывать события, связанные с передачей данных формы. Такие события возникают либо после нажатия кнопки передачи данных, определенной тегом <input type=submit> в контейнере <form>, либо при передаче данных формы с помощью метода submit(), вызванного из JS-программы.

Операторы JavaScript

Условные Операторы

JavaScript поддерживает два условных оператора: if...else и switch.

Оператор **if...else**

Оператор if используется для выполнения определённых операторов, если логическое условие true; не обязательный блок else выполняется, если условие false:

```
if (condition) {
   statements1
}
[else {
   statements2
} ]
```

condition это может быть любое выражение JavaScript, которое вычисляется в true или false. Выполняемые операторы это любые операторы JavaScript, включая вложенные if.

Если нужно использовать более одного оператора после операторов if или else, необходимо заключать их в фигурные скобки {}.

Пример

Оператор **switch**

Оператор switch позволяет вычислять выражение и пытается найти совпадение значения выражения с оператором label. Если совпадение найдено, программа выполняет ассоциированный оператор:

```
switch (expression){
   case label :
      statement;
      break;
   case label :
      statement;
      break;
   ...
   default : statement;
}
```

Пример

Операторы циклов

JavaScript поддерживает операторы циклов: **for, do while, while**.
Кроме того, можно использовать операторы break и continue с операторами циклов.

Оператор **for**

Оператор **for** повторяется, пока специфицированное условие не станет false.

JavaScript-цикл for похож на аналогичный цикл языка C:

```
for ([initialExpression]; [condition]; [incrementExpression]) {
    statements
}
```

Пример

Оператор **do...while**

Оператор do...while повторяется, пока специфицированное выражение не станет false:

```
do {
   statement
} while (condition)
```

Пример:

```
do {
   i+=1;
   document.write(i);
} while (i<5);
```

Оператор **while**

Оператор while выполняется, пока специфицированное условие вычисляется в true:

```
while (condition) {
   statements
}
```

Пример:

```
n = 0;x = 0;
while( n < 3 ) {
   n ++;
   x += n';
}
```

Оператор **break**

Оператор break используется для прерывания выполнения цикла, либо оператора switch.

Когда break используется с операторами while, do-while, for или switch, оператор break немедленно прерывает **внутренний** цикл или **switch** и передаёт управление следующему за циклом оператору.

Пример:

```
for (i = 0; i < a.length; i++) {
   if (a[i] = theValue)
      break;
}
```

Оператор **continue**

Оператор **continue** может использоваться для рестарта оператора while, do-while, for.

Пример. Здесь цикл while с оператором continue выполняется, если i имеет значение 3. Таким образом, n получает значения 1, 3, 7 и 12.

```
i = 0;
n = 0;
while (i < 5) {
   i++;
   if (i == 3)
      continue;
   n += I;
}
```

Функции в JavaScript

Определение функции состоит из ключевого слова function, сопровождаемого:
- Именем функции;
- Списком аргументов функции, записанным в круглых скобках, и отделяемые запятыми;
- *JavaScript* операторами, заключенными в фигурных скобках, {...}.

Функция может быть рекурсивной, то есть может вызывать саму себя.
Пример

Функции с переменным числом аргументов

Возможно вызывать функцию с большим количеством аргументов, чем в ее объявлении. Для доступа к аргументам в этом случае, используется массив ***arguments*** . Это может быть полезно тогда, когда заранее неизвестно, сколько аргументов будет в функции. Чтобы определить реальное число аргументов в функции используется значение ***arguments.length***.
Пример

Базовая система событий языка JavaScript

В JavaScript поддерживается возможность обработки (задания одного или нескольких операторов или функции) различных событий, инициируемых пользователем.

События JavaScript можно разделить на несколько категорий:

- события, связанные с документами:
  - загрузка и выгрузка документов;
- события, связанные с формами:
  - щелчки мыши на кнопках
  - получение и потеря фокуса ввода и изменение содержимого полей ввода, областей текста и списков;
  - выделение текста в полях ввода и областях текста;
- события, связанные с мышью:
  - помещение указателя мыши на объект;
  - уход указателя мыши с объекта;
  - активизация объекта.

Чтобы обеспечить перехват(обработку) события, необходимо написать функцию-обработчик. В качестве обработчиков событий могут быть заданы как группы из одного или нескольких JavaScript-операторов, так и целые функции.

Имя события	**Атрибут HTML**	**Условие возникновения события**
Blur	onBlur	Потеря фокуса ввода элементом формы
Change	onChange	Изменение содержимого поля ввода иди области текста, либо выбор нового элемента списка
Click	onClick	Щелчок мыши на элементе формы или гиперсвязи
Focus	onFocus	Получение фокуса ввода элементом формы
Load	onLoad	Завершение загрузки документа
MouseOver	onMouseOver	Помещение указателя мыши на объект
MouseOut	onMouseOut	Помещение указателя мыши с объекта
Select	onSelect	Выделение текста в поле ввода или области текста
Submit	onSubmit	Передача данных формы
Unload	onUnload	Выгрузка текущего документа и начало загрузки нового

Событие onBlur

Атрибут обработчика события onBlur может использоваться со следующими тегами HTML:

<input type=". . ." onBlur="expr | function()">
<textarea onBlur="expr | function()"> . . . </textarea>
<select onBlur="expr | function()"> . . . <option> . . . </select>

С помощью атрибута onBlur задается выражение языка JavaScript(*expr*), или имя функции(*function*), которая выполняется, когда соответствующий элемент HTML-формы теряет фокус ввода. Потеря фокуса ввода происходит либо при щелчке мыши на другом элементе формы или другой формы, либо при переходе к другому элементу формы посредством клавиши [Tab]. Атрибут onBlur часто применяют для проверки данных, введенных в соответствующее поле.

Событие onChange

Атрибут обработчика события onChange можно использовать в следующих HTML-тегах:

<select onChange="expr | function()"> . . . <option> . . . </select>
<textarea onChange="expr | function()"> . . . </textarea>
<input type=text onChange="expr | function()">

Атрибут onChange задает выражение, которое должно выполняться при потере фокуса ввода элементом HTML-формы и при изменении содержимого этого элемента. Данный атрибут подобен атрибуту onBlur, однако для того чтобы возникло событие Change, содержимое поля должно быть изменено, и поле должно потерять фокус ввода.

Событие onClick

Атрибут onClick может использоваться в следующих тегах HTML:

. . .
<input.type="checkbox" onClick="expr | function()">
<input.type="radio" onClick="expr | function()">
<input.type="reset" onClick="expr | function()">
<input.type="submit" onClick="expr | function()">
<input.type="button" onClick="expr | function()">

Операторы языка JavaScript, заданные в атрибуте onClick, выполняются при щелчке мыши на таких объектах как гиперсвязь, кнопка, контрольный переключатель и т.п. Для контрольных переключателей и селекторных кнопок событие Click возникает не только при выборе элемента, но и при снятии выбора.

Событие onFocus

Атрибут обработчика события onFocus работает со следующими тегами HTML:

<input.type="text" onFocus="expr | function()">
<textarea onFocus="expr | function()"> . . . </textarea>
<select onFocus="expr | function()"> . . . <option> . . . </select>

Название говорит за себя, атрибут onFocus позволяет обрабатывать события, связанные с получением фокуса ввода. В противоположность onBlur, здесь обрабатывается событие при получении фокуса ввода.

Событие onLoad

Атрибут обработчика события onLoad работает со следующими тегами HTML:

<body onLoad="expr | function()"> . . . </body>

<frameset> . . . <frame onLoad="expr | function()"> . . . </frameset>

Атрибут onLoad, помещенный в тег <body>, активизирует заданные операторы или функцию языка JavaScript, когда загрузка текущего документа в браузер завершена.

Событие onLoad может использоваться для активизации каких-либо действий после загрузки документа, например, для активизации скрипта отображающего текущее время.

Событие unLoad

Атрибут обработчика события unLoad работает со следующими тегами HTML:

<body unLoad="expr | function()"> . . . </body>

<frameset> . . . <frame unLoad="expr | function()"> . . . </frameset>

Соответствующее событие возникает при выгрузке текущего документа, то есть вызывается функция-обработчик события перед выгрузкой документа из текущего окна или фрейма.

Событие onSelect

Атрибут onSelect может быть использован со следующими тегами:

<input type="text" onSelect="expr | function()">

<textarea onSelect="expr | function()"> . . . </textarea>

Этот атрибут запускает обработчик события, когда пользователь выделил фрагмент текста в поле ввода или области текста.

События onMouseOver и onMouseOut

OnMouseOver позволяет активизировать JavaScript-операторы, когда курсор мыши находится на объекте, а атрибут onMouseOut - когда курсор отведен от объекта. Например, для гиперсвязей:

* . . . *

* . . . *

Атрибут onMouseOver - операторы JavaScript выполняются, когда указатель мыши наведен на объект, у которого задан этот атрибут.

Атрибут onMouseOut предоставляет возможность активизировать операторы языка JavaScript в случае, когда курсор мыши выходит за пределы объекта. Обработку события MouseOut следует выполнять, когда необходимо отменить ранее заданные действия.

Пример.

Объект window. Создание и управление окнами

Метод **open**() объекта window позволяет открывать новые окна, определяя различные их характеристики (размер, координаты, наличие панели инструментов, полосы скроллинга и др.).

Метод open() объекта window вызывается следующим образом:

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

где строка "URL" - адрес ресурса, загружаемого в новое окно. Если URL не указан, окно открывается, но загрузки документа не происходит. (Если не указывается URL, - оставляются пустые кавычки и запятая).

С помощью аргумента "windowName" задают имя окна, это имя не является его заголовком.

Атрибут	Значения	Что определяет
copyhistory	[=yes|no] | [=1|0]	Сохранение истории загрузки документов в данное окно
directories	[=yes|no] | [=1|0]	Наличие в данном окне кнопок групп новостей
height	=pixelheight	Высота окна в пикселах
location	[=yes|no] | [=1|0]	Наличие поля Location
menubar	[=yes|no] | [=1|0]	Наличие меню
resizable	[=yes|no] | [=1|0]	Наличие рамки окна, позволяющего менять его размеры
scrollbars	[=yes|no] | [=1|0]	Наличие линеек прокрутки
status	[=yes|no] | [=1|0]	Наличие строки состояния
toolbar	[=yes|no] | [=1|0]	Наличие панели инструментов
width	=pixelwidth	Ширина окна в пикселах

Пример.

Динамическое создание документов

Используя JavaScript, есть возможность не только открывать новые окна браузера, но и формировать в этих окнах новые HTML-страницы.

Пример.

Программирование строки статуса(status bar)

Полем статуса (status bar) называют среднее поле нижней части окна браузера сразу под областью отображения HTML-страницы.

Программа на JavaScript имеет возможность работать с этим полем как с изменяемым свойством окна. При этом фактически с этим полем связаны два разных свойства:

window.status

window.defaultStatus

Разница между ними заключается в том, что браузер на самом деле имеет несколько состояний, которые связаны с некоторыми событиями. Состояние браузера отражается сообщением в поле статуса. Существует два состояния: нет никаких событий (defaultStatus) и происходят какие-то события (status).

Пример.

Объект location

Объект **location** отображает URL загруженного документа. Если пользователь хочет вручную перейти к какой-либо странице (набрать ее URL), то он делает это в поле location(адрес). Для перехода на произвольный адрес необходимо изменить значение **location**.

Пример.

История посещений (History)

История посещений страниц позволяет пользователю вернуться к странице, которую он просматривал некоторое время назад. История посещений в JavaScript трансформируется в объект класса **History**. Этот объект содержит массив URL тех страниц, которые пользователь посещал. Методы объекта history позволяют пользователю загружать страницы, используя URL из этого массива. При этом сам URL, как текстовая строка, программисту не доступен.

Пример.

Программирование графики

Используя объекты Image, можно вносить изменения в графические образы, присутствующие на web-странице.

Динамическая загрузка новых изображений

Для осуществления динамической смены изображений на web-странице может быть использовано свойство *src* объекта *Image* (оно соответствует атрибуту src тэга <img>). Используя JavaScript, существует возможность назначать новый адрес изображению, уже загруженному в web-страницу. И в результате, изображение будет загружено с этого нового адреса, заменив на web-странице старое.

Пример.

Упреждающая загрузка изображения

Один из недостатков динамической загрузки изображений может заключаться в том, что после записи в свойство src объекта Image нового адреса, начинается процесс загрузки соответствующего изображения. И прежде чем новое изображение будет передано от сервера к браузеру, и отобразится им, пройдет некоторое время. Иногда подобные задержки неприемлемы.

Решение этой проблемы - упреждающая загрузка изображения. Для этого необходимо создать новый объект Image и указать для него свойство src. Например:

```
hiddenImg= new Image();
hiddenImg.src= "img3.gif";
```

Чтобы вывести предзагруженное изображение на экран, можно воспользоваться строкой:

```
document.myImage.src= hiddenImg.src;
```

Изменение изображений в соответствии с событиями

Возможно создать различные эффекты, используя смену изображений в качестве реакции на определенные события. Например, Вы можете изменять изображения в тот момент, когда курсор мыши попадает на определенную часть страницы.

Исходный код такого примера выглядит следующим образом:

```
<a href="#"  onMouseOver="document.myImage2.src='img2.gif'"
onMouseOut="document.myImage2.src='img1.gif'">
<img src="img1.gif" name="myImage2" width=160 height=50 border=0></a>
```

Объект Date

Объект содержит информацию о дате и времени. Этот объект имеет множество методов, предназначенных для получения такой информации. Кроме того, объекты **Date** можно создавать и изменять, например, путем сложения или вычитания значений дат получать новую дату. Для создания объекта **Date** применяется синтаксис:

dateObj = new Date(parameters),

где dateObj - переменная, в которую будет записан новый объект Date. Аргумент parameters может принимать следующие значения:

- пустой параметр, например date() - системные дата и время;
- строку, представляющую дату и время в виде: "месяц, день, год, время", например "March 1, 2000, 17:00:00" Время представлено в 24-часовом формате;
- значения года, месяца, дня, часа, минут, секунд. Например, строка "00,4,1,12,30,0" означает 1 апреля 2000 года, 12:30;
- целочисленные значения только для года, месяца и дня, например "00,5,1" означает 1 мая 2000 года, сразу после полночи, так, как значения времени равны нулю.

Метод	Описание метода
getDate()	Возвращает день месяца из объекта в пределах от 1 до 31
getDay()	Возвращает день недели из объекта: 0 - вс, 1 - пн, 2 - вт, 3 - ср, 4 - чт, 5 - пт, 6 - сб.
getHours()	Возвращает время из объекта в пределах от 0 до 23
getMinutes()	Возвращает значение минут из объекта в пределах от 0 до 59
getMonth()	Возвращает значение месяца из объекта в пределах от 0 до 11
getSeconds()	Возвращает значение секунд из объекта в пределах от 0 до 59
getTime()	Возвращает количество миллисекунд, прошедшее с 00:00:00 1 января 1970 года.
getTimeZoneoffset()	Возвращает значение, соответствующее разности во времени (в минутах)
getYear()	Возвращает значение года из объекта
setDate(day)	С помощью данного метода устанавливается день месяца в объекте от 1 до 31
setHours(hours)	С помощью данного метода устанавливается часы в объекте от 0 до 23
setMinutes(minutes)	С помощью данного метода устанавливаются минуты в объекте от 0 до 59
setMonth(month)	С помощью данного метода устанавливается месяц в объекте от 1 до 12
setSeconds(seconds)	С помощью данного метода устанавливаются секунды в объекте от 0 до 59
setTime(timestring)	С помощью данного метода устанавливается значение времени в объекте.
setYear(year)	С помощью данного метода устанавливается год в объекте year должно быть больше 1900.
toString()	Преобразует содержимое объекта Date в строковый объект.
toLocaleString()	Преобразует содержимое объекта Date в строку в соответствии с местным временем.

Пример.

Объект Math

Объект Math является встроенным объектом языка JavaScript и содержит свойства и методы, используемые для выполнения математических операций. Объект Math включает также некоторые широко применяемые математические константы. Синтаксис:

Math.propertyName
Math.methodName(parameters)

E	Константа Эйлера. Приближенное значение 2.718 . . .
LN2	Значение натурального логарифма числа два. Приближенное значение 0.693 . . .
LN10	Значение натурального логарифма числа десять. Приближенное значение 2.302 . . .
LOG2_E	Логарифм *e* по основанию 2. Приближенное значение 1.442 . . .)
LOG10_E	Десятичный логарифм *e*. Приближенное значение 0.434 . . .
PI	Число ПИ. Приближенное значение 3.1415 . . .
SQRT2	Корень из двух

abs()	Возвращает абсолютное значение аргумента.
acos()	Возвращает арккосинус аргумента
asin()	Возвращает арксинус аргумента
atan()	Возвращает арктангенс аргумента
ceil()	Возвращает большее целое число от аргумента - округление в большую сторону. Math.ceil(3.14) вернет 4
cos()	Возвращает косинус аргумента
exp()	Возвращает экспоненту аргумента
floor()	Возвращает наибольшее целое число аргумента, отбрасывает десятичную часть
log()	Возвращает натуральный логарифм аргумента
max()	Возвращает больший из 2-х числовых аргументов. Math.max(3,5) вернет число 5
min()	Возвращает меньший из 2-х числовых аргументов.
pow()	Возвращает результат возведения в степень первого аргумента вторым. Math.pow(5,3) вернет 125
random()	Возвращает псевдослучайное число между нулем и единицей.
round()	Округление аргумента до ближайшего целого числа.
sin()	Возвращает синус аргумента
sqrt()	Возвращает квадратный корень аргумента
tan()	Возвращает тангенс аргумента