Язык Javascript

Краткое введение в *Javascript*

Основная идея *Javascript:*
Возможность изменения значений атрибутов HTML-контейнеров и свойств среды отображения в процессе просмотра HTML-страницы пользователем.

Javascript это:

1. Объектно-ориентированный язык разработки встраиваемых приложений, выполняемых как на стороне клиента, так и на стороне браузера.
2. Интерпретируемый язык. Его интерпретатор обычно встроен в браузер.
3. Основное назначение – определять «динамическое» поведение страниц при загрузке (формирование страницы перед ее открытием) и при работе пользователя со страницей.
4. Текст на *Javascript* может быть вложен в HTML-страницу непосредственно или находиться в отдельном файле (как CSS).
5. Похож на языки *Java* и *C#* синтаксически, но сильно отличается от них по внутреннему содержанию.

Области использования *Javascript:*

1. Динамическое создание документа с помощью сценария.
2. Проверка заполнения полей форм HTML до передачи на сервер.
3. Создание динамических HTML страниц совместно с CSS и DOM.

Характеристика *Javascript*

Некоторые важнейшие характеристики *Javascript* :

1. Язык объектно-ориентированного программирования. Объекты в языке имеют «тип», «атрибуты» и «методы»

   ```
   "John,Jane,Paul,Michael".split(",").length
   ```

2. Переменные не имеют заранее заданного типа, то есть в разные моменты времени могут содержать значения разных типов

   ```
   var number = 25;  number = (number < 0);  number = "25";
   ```

3. Типы объектов могут быть: `number`, `string`, `function`, `object`, `undefined`. Оператор `typeof` позволяет «вычислить» тип объекта.

   ```
   typeof 25 == "number"   typeof null == "object"
   ```

Объекты, встроенные в браузеры

При программировании можно использовать ряд встроенных объектов. Основные из них это:

- `window` : представляет «глобальный контекст» и позволяет работать с атрибутами и методами окна.
- `document` : загруженная страница со своей структурой элементов.
- `navigator` : объект, представляющий браузер и его свойства.
- `location` : характеристики текущего URL (порт, хост и т.п.).
- объекты, представляющие элементы различных типов в HTML-странице, такие как `<body>`, `<link>`, `<img>` и т.п.
- события (events), возникающие от действий пользователя, например, нажатие кнопки мыши (`click`), загрузка новой страницы (`load`) и т.д.

Размещение кода JavaScript на HTML-странице

JavaScript-код исполняет браузер . В него встроен *интерпретатор JavaScript*. Выполнение программы зависит от того, когда и как этот интерпретатор получает управление.

Четыре основных способа функционального применения *JavaScript*:

- гипертекстовая ссылка (схема *URL*);

http://sut.ru/directory/page.html

<A HREF="JavaScript:код_программы">...</A>

- обработчик события (в атрибутах, отвечающих событиям);

<FORM><INPUT TYPE=button VALUE="Кнопка"
onClick="alert('Вы нажали кнопку');"></FORM>

- подстановка (entity);
- вставка (контейнер <*SCRIPT*>).

Два способа вставки JavaScript-кода на HTML-странице:

- написать текст кода непосредственно внутри контейнера *SCRIPT*:

<SCRIPT> a = 5;</SCRIPT>

- вынести код JavaScript в отдельный файл и затем включить его в HTML-страницу :

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

Размещение кода JavaScript внутри HTML-документа

Фрагменты кода можно включать в заголовок или тело HTML-документа. Кроме того, можно разместить код в отдельном файле, а в HTML-странице разместить ссылку на этот файл.

```
<html>
  <head>
    <script type="text/javascript"> ... </script>
    <script type="text/javascript" src="scripts/myscript1.js/>
  <head>
  <body>
    <script type="text/javascript"> ... </script>
    <script type="text/javascript" src="scripts/myscript2.js/>
  </body>
</html>
```

Код, ссылки на который размещены в заголовке, просто подсоединяется к странице и может быть использован, например, для определения реакций на пользовательские события.
Код, ссылки на который размещены в теле, исполняется при загрузке страницы и может непосредственно использоваться для формирования содержания страницы во время загрузки.

Два простых примера

Метод `document.write` используется для непосредственного включения HTML-текста в содержимое страницы, например, можно сгенерировать длинный текст в параграфе:

```
<body>
  <p>
    <script type="text/javascript">
      for (var i = 0; i < 100; ++i) {
        document.write("Hello, world! ");
      }
    </script>
  </p>
</body>
```

Два простых примера (продолжение)

Во втором примере датчик случайных чисел используется для генерации случайной ссылки (из заданного набора):

```
<body>
  <p>
    <script type="text/javascript">
      var rand = Math.random();          // в диапазоне: [0, 1)
      var numb = Math.floor(rand * 10);
      var image = "images/image" + numb + ".jpg";
      var insert = "<img class=\"floatRight\" src=\"" +
                   image + "\" alt=\"Фотография цветочка\"/>";
      document.write(insert);
    </script>
  </p>
</body>
```

Некоторые сведения о синтаксисе

Описание переменных:

```
var count = 25,
    msg = 'Сообщение об ошибке';
var nullVar;   // получает начальное значение null
```

Операции такие же, как в Java и C#, но более широко используется преобразование типов

```
+   -   *   /   %   ++  --   =   +=   -=   *=
/=  %=  ==  !=  >   <   >=   <=  &&   ||   !

2 + '3' == '23', но     2 + 3 == 5
```

Многие операторы очень похожи на соответствующие операторы Java и C#, но могут иметь некоторые отличия в семантике.

```
for (var i = 0; i < 100; ++i) { ... }
if (x * y < 100) { ... } else { ... }
try { ... } catch (e) { ... } finally { ... }
```

Язык ядра JavaScript

Типы данных

JavaScript поддерживает следующие встроенные структуры и типы данных:

- литералы;
- переменные;
- массивы;
- функции;
- объекты.

Литералы

Литералом называют данные, которые используются в программе непосредственно. При этом под данными понимаются числа или строки текста. Все они рассматриваются в JavaScript как элементарные типы данных.

Примеры литералов:

числовой литерал: 10 (целый литерал)
числовой литерал: 2.310 (вещественный литерал)
числовой литерал: 2.3e+2 (иная форма записи)
строковый литерал: 'Это строковый литерал‘
строковый литерал: "Это строковый литерал"
булевый литерал: true, false

Переменные

Переменная — это область памяти, имеющая свое имя и хранящая некоторые данные. Имя переменной начинается с буквы латинского алфавита или знака _.

Переменные в *JavaScript* объявляются с помощью оператора *var*, при этом можно давать или не давать им начальные значения:

```
var k; var h='Привет!';
```

Можно объявлять сразу несколько переменных в одном операторе *var* (тем самым уменьшая размер кода), но тогда их надо писать через запятую. При этом тоже можно давать или не давать начальные значения:

```
var k, h='Привет!';
var t=37, e=2.71828;
```

Тип переменной определяется по присвоенному ей значению. Язык *JavaScript* — слабо типизирован: в разных частях программы можно присваивать одной и той же переменной значения различных типов, и интерпретатор будет "на лету" менять тип переменной. Узнать тип переменной можно с помощью оператора *typeof():*

```
var i=5;          alert(typeof(i));
i= 'Привет!';       alert(typeof(i));
```

Массивы

Массив создается оператором *new* и специальным конструктором *Array*.

Если ему передается единственный аргумент, причем целое неотрицательное число, то создается незаполненный массив соответствующей длины.

Если же передается один аргумент, не являющийся числом, либо более одного аргумента, то создается массив, заполненный этими элементами:

```
a = new Array();                // пустой массив (длины 0)
b = new Array(10);              // массив длины 10
c = new Array(10,'Привет');     // массив из двух элементов: числа и строки
d = [5, 'Тест', 2.71828, 'Число e'];  // краткий способ создать массив из 4 элементов
```

Элементы массива нумеруются с нуля.

Поэтому в последнем примере значение d[0] равно 5, а значение d[1] равно 'Тест'.

Массив может состоять из разнородных элементов. Массивы не могут быть многомерными, однако ничто не мешает завести массив, элементами которого будут тоже массивы.

Для массивов определено несколько методов:

- join(), позволяет объединить элементы массива в одну строку;
- reverse(), применяется для изменения порядка элементов массива на противоположный;
- sort(), интерпретирует элементы массива как строковые литералы и сортирует массив в алфавитном (т.н. лексикографическом) порядке.

Синтаксис: *имя_массива.метод(необязательные аргументы)*

a = new Array('мать','видит','дочь'); a.reverse().

а также свойство length, которое позволяет получить число элементов массива.

Тип Array

Существует несколько способов создания массива:

```
var holidays = ["1 января", "7 января", "23 февраля"];
var holidays = new Array("1 января", "7 января", "23 февраля");
var holidays = new Array(3);
holidays[0] = "1 января";
holidays[1] = "7 января";
holidays[2] = "23 февраля";
```

Атрибут массива: `length` – длина массива.

```
var myArray = new Array();
myArray[2] = new Date(2008,2,23);
myArray[5] = new Date(2008,5,9);
myArray.length == 6
```

Тип Array (продолжение)

Методы, определенные для работы с массивом:
concat, join, pop, push, shift , unshift, slice

```
var names = ["Петя", "Вася"];
names = names.concat(["Сережа", "Наташа"], ["Оля", "Люба"]);
        names == ["Петя", "Вася", "Сережа", "Наташа", "Оля", "Люба"]
var s = names.join(';');
        s == "Петя;Вася;Сережа;Наташа;Оля;Люба"
var e = names.pop();
        e == "Люба"
        names == ["Петя", "Вася", "Сережа", "Наташа", "Оля"]
var l = names.push("Саша");
        l == 6
        names == ["Петя", "Вася", "Сережа", "Наташа", "Оля", "Саша"]
```

shift и unshift – точно так же, как pop и push, но с началом массива.

```
names = names.slice(1, 4);
        names == ["Вася", "Сережа", "Наташа", "Оля"]
```

Тип Array (продолжение)

Еще методы, определенные для работы с массивом:
reverse, sort, splice, toString

```
var names = ["Вася", "Сережа", "Наташа", "Оля"];
names.reverse();
        names == ["Оля", "Наташа", "Сережа", "Вася"]
names.sort();
        names == ["Вася", "Наташа", "Оля", "Сережа"]
var a = [5, 3, 40, 1, 10, 100].sort();
        a == [1, 10, 100, 3, 40, 5]
var a = [5, 3, 40, 1, 10, 100].sort(function(a,b){return a-b;});
        a == [1, 3, 5, 10, 40, 100]
names.splice(1, 2, "Саша", "Таня", "Нина");
        names == ["Вася", "Саша", "Таня", "Нина", "Сережа"]
```

toString – точно так же, как join(',').

```
        names.toString() == "Вася,Саша,Таня,Нина,Сережа"
```

Выражения и операторы

Выражение – комбинация переменных, литералов и операторов.

Результат – одно значение определенного выше типа.

В JavaScript определены три типа сложных выражений:

- арифметическое (вычисляется число);
- строковое (вычисляется строка);
- логическое (имеет значение *true* или *false*).

Арифметические операторы: + - * / % ++ --

Сокращенные формы операторов присваивания: *x+=y* *соотв.* *x=x+y*

Операторы сравнения: > < >= <= == !=

Логические операторы: И ИЛИ НЕ

Строковые операторы: конкатенация (все операнды приводятся к типу string, если хотя бы один из операндов содержит строковый литерал)

Основные встроенные типы

Есть набор встроенных «классов», порождающих «объекты», различающиеся набором атрибутов и методов. Программисты могут динамически изменять поведение этих «классов» и создавать свои собственные. Каждый «класс» является объектом, у которого есть «прототип», определяющий набор атрибутов и методов у всех вновь создаваемых объектов этого класса.

Типы, встроенные в язык, это:

- `Number` : 64-х-разрядные числа с плавающей точкой.
- `String` : строки с символами в формате Unicode.
- `Array` : массивы с переменными границами.
- `Function` : Функции. Каждая функция, кроме того, может служить конструктором объекта.
- `Boolean, Date, Math, RegExp` : логические значения, даты,...

Тип String

Строки заключаются либо в апострофы, либо в двойные кавычки

```
var slogan = "Don't be evil!";
var image ='<img class="to-right" src="myphoto.jpg"/>';
```

escape-последовательности: `\\` `\'` `\"` `\t` `\n`

Операции над строками: `+` `<` `>` `==` `!=`

```
"2" + "3"       "23"            "a" == "A"     false
"10" < "5"      true            5 == "5"       true
10 < "5"        false           5 === "5"      false
5 + "5"         "55"
```

Атрибут строки: `length` – длина строки.

```
"abc".length == 3
```

Преобразования типов: `String(n)` `Number(s)`

```
String(10) < "5" == true     Number('3.' + '14') == 3.14
```

Стандартные методы объектов типа `String`

```
charAt, indexOf, lastIndexOf, replace, split,
substr, substring, toLowerCase, toUpperCase
```

Примеры:

`"Google".charAt(3)`	`"g"`
`"Google".indexOf("o")`	`1`
`"Google".lastIndexOf("o")`	`2`
`"Google".replace("o", "oo")`	`"Gooogle"`
`"Google".replace(/o/g, "oo")`	`"Goooogle"`
`"Google".split("o")`	`["G","","gle"]`
`"Google".substr(1,3)`	`"oog"`
`"Google".substring(1,3)`	`"oo"`
`"Google".toLowerCase()`	`"google"`
`"Google".toUpperCase()`	`"GOOGLE"`

Тип Number

Числа – это 64-х-разрядные двоичные числа с плавающей точкой.

```
Number.MIN_VALUE        5e-324
Number.MAX_VALUE        1.79769313486231570e+308
Number.NaN              NaN
```

Операции над числами: `+ - * / % < > == !=`

```
3.14 % 2                1.14
```

Функции преобразования: `parseInt, parseFloat, Number, toString`

```
parseInt("3.14")        3
parseFloat("*3.14")     NaN
Number("3.xaxa")        NaN
3.14.toString()         "3.14"
isNaN(3.14 / 0)         true
```

Тип Boolean

Стандартные логические значения – `true` и `false`. Однако в качестве условий можно использовать любое значение.

"Истинные" условия:

```
if (2 < 5)
if (25)
if ('Google могуч и ужасен')
```

"Ложные" условия:

```
if ("")
if (0)
if (null)
```

Логические условия используются в условных операторах и операторах циклов.

```
if (x < y) { z = x; } else { z = y; }
while (x < 100) { x = x * 2; n++; }
do { x = Math.floor(x / 2); n++; } while (x > 0);
for (var y = 0, x = 0; x < 100; ++x) { y += x; }
```

Тип Date

Объекты типа Date содержат дату в виде числа миллисекунд, прошедших с 1 января 1970 г. Диапазон от -10^8 до 10^8 дней от 1 января 1970 г.

Конструкторы:

```
var now = new Date();                  // сейчас
var january1st1970 = new Date(0);  // дата в миллисекундах
var gagarin = new Date(1961, 4, 12);
var newYear = new Date("January 1, 2009");
```

Методы, применимые для работы с датами: getDate, getMonth, getFullYear, getTime, getTimezoneOffset, setDate, setFullYear,...

```
function DaysToDate(day, month) {
  var now = new Date(), year = now.getFullYear();
  var bd = new Date(year, month-1, day);
  var fullDay = 24 * 60 * 60 * 1000;
  var diff = Math.ceil((bd - now) / fullDay);
  return diff < 0 ? diff + 365 : diff;
}
```

todate.html

Операторы языка

Общий перечень этих операторов выглядит следующим образом (сразу оговоримся, что этот список неполный):

- {...}
- if ... else ...
- ()?
- while
- for
- break
- continue
- return

{...}

Фигурные скобки определяют составной оператор JavaScript — ***блок***. Основное назначение блока — определение тела цикла, тела условного оператора или функции.

if ... else ...

Условный оператор применяется для ветвления программы по некоторому логическому условию. Есть два варианта синтаксиса:

if (логическое_выражение) оператор_1;

if (логическое_выражение) оператор_1; else оператор_2;

()?

Этот оператор, называемый *условным выражением*, выдает одно из двух значений в зависимости от выполнения некоторого условия.

Синтаксис его таков:

(логическое_выражение)? значение_1 : значение_2

Следующие два фрагмента равносильны:

```
TheFinalMessage = (k>5)? 'Готово!' : 'Подождите...';
if(k>5) TheFinalMessage = 'Готово!';
                    else    TheFinalMessage = 'Подождите...';
```

while

Оператор *while* задает цикл.

Определяется он в общем случае следующим образом:

while (условие_продолжения_цикла) тело_цикла;

Такой цикл используется, когда заранее неизвестно количество итераций, например, в ожидании некоторого события.

Пример:

```
var s='';
while (s.length<6)
{ s=prompt('Введите строку длины не менее 6:','');
}
alert('Ваша строка: ' + s + '. Спасибо!');
```

for

Оператор *for* — это еще один оператор цикла.

В общем случае он имеет вид:

for (инициализация_переменных_цикла; условие_продолжения_цикла; модификация_переменных_цикла) тело_цикла;

Пример использования оператора:

```
document.write('Кубы чисел от 1 до 100:');
for (n=1; n<=100; n++)
document.write('<BR>'+n+'<sup>3</sup> = '+ Math.pow(n,3));
```

break

Оператор break позволяет досрочно покинуть тело цикла.

Распечатаем только кубы, не превышающие 5000.

```
document.write('Кубы чисел, меньшие 5000:');
for (n=1; n<=100; n++)
{
s=Math.pow(n,3);
if(s>5000) break;
document.write('<BR>'+n+'<sup>3</sup> = '+s);
}
```

continue

Оператор *continue* позволяет перейти к следующей итерации цикла, пропустив выполнение всех нижестоящих операторов в теле цикла.

Пример: вывести кубы чисел от 1 до 100, превышающие 10 000, то:

```
document.write('Кубы чисел от 1 до 100, большие 10 000:');
for (n=1; n<=100; n++)
{
s=Math.pow(n,3);
if(s <= 10000) continue;
document.write('<BR>'+n+'<sup>3</sup> = '+s);
}
```

return

Оператор *return* используют для возврата значения из *функции* или обработчика события.

Пример с функцией:

```
function sign(n)
{  if (n>0) return 1;
   if (n<0) return -1;
   return 0;
}
```

При использовании в обработчиках событий оператор *return* позволяет отменить или не отменять действие по умолчанию, которое совершает браузер при возникновении данного события.

Пример:

```
<FORM ACTION="newpage.html" METHOD=post>
<INPUT TYPE=submit VALUE="Отправить?"
onClick="alert('Не отправим!');
return false;">
</FORM>
```

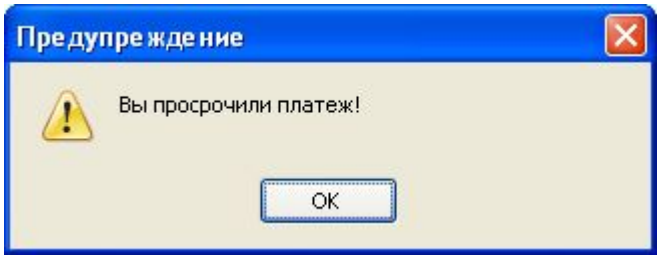
В этом примере без оператора *return false* пользователь увидел бы окно предупреждения *"Не отправим!"* и далее был бы перенаправлен на страницу *newpage.html.*

Оператор же *return false* позволяет отменить отправку формы, и пользователь лишь увидит окно предупреждения

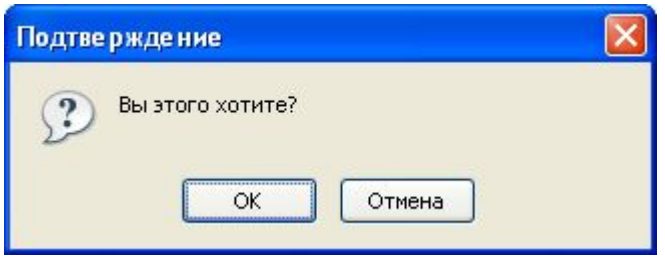
Сообщения, выдаваемые в popup-окнах

Три стандартные функции используются для генерации сообщений в popup-окнах: `alert`, `confirm`, `prompt`.

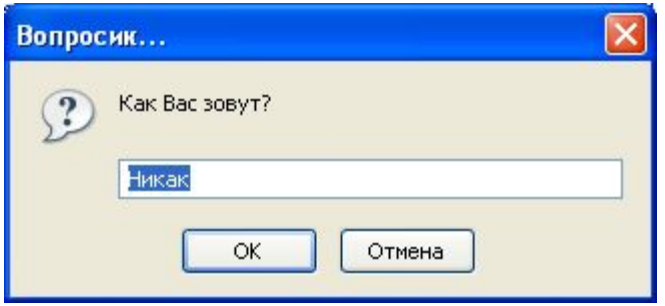
```
alert('Вы просрочили платеж!');
```




```
confirm('Вы этого хотите?');
```




```
var name = prompt('Как Вас зовут?',
    'Никак', 'Вопросик...');
```



События и реакции на них

Имеется большое количество событий, которые можно разделить на следующие классы:

- события от мыши (`click`, `dblclick`, `mousedown`,...);
- события от клавиатуры (`keypress`, `keydown`,...);
- события от элементов ввода (`focus`, `submit`, `select`,...);
- события страницы (`load`, `unload`, `error`,...);

Один из способов программирования состоит в определении реакции на события непосредственно в описании элемента, например:

```
<p>День независимости России
   <span style="color: blue; text-decoration: underline;"
         onclick=
           "alert('Осталось ' + DaysToDate(12, 6) + 'дней');">
   12 июня</span>.
</p>
```

holidays.html

Недостаток этого способа: `javascript`-текст опять смешивается с содержанием страницы.

Работа с таймером

Можно создать таймер и определить реакцию на событие от таймера.

```
var timer = setTimeout(func, timeinterval);
```

`func` – это функция или строка с кодом; `timeinterval` – время в миллисекундах. Таймер срабатывает один раз и запускает функцию.

```
function launchTimer() {
  setTimeout("alert('Зенит – чемпион!');", 2000);
}
```

Теперь можно запустить этот таймер, например, по событию `click`:

```
<body>
  <p>Нажми <span onclick="launchTimer();">сюда!</span></p>
</body>
```

Пока событие еще не случилось, таймер можно остановить:

```
var timer = setTimeout(func, timeinterval);
clearTimeout(timer);
```

settimer.html

Работа с интервальным таймером

Таймер может срабатывать многократно через равные промежутки времени. Такой таймер создается с помощью функции `setInterval` и останавливается с помощью функции `clearInterval`.

```
var timer = setInterval(func, timeinterval);

function launchInterval() {
  timer = setInterval("alert('Зенит – чемпион!');", 2000);
}

function stopTimer() {
  if (timer) clearInterval(timer);
  timer = null;
}

<body>
  <p>Нажми <span onclick="launchInterval();">сюда,</span>
     чтобы запустить.</p>
  <p>Нажми <span onclick="stopTimer();">сюда,</span>
     чтобы остановить.</p>
</body>
```

setinterval.html