Основы Java Script

Web-конструирование:

- **Структура** веб-документа определяется средствами **HTML**.
- **Отображение** документа задается при помощи **CSS.**
- **Функциональность на стороне клиента** обеспечивается **JavaScript.**
- Обработка данных и динамическое формирование документа (ответа) по запросу клиента осуществляется серверными средствами (например, PHP и mySQL, Perl, Java….).

Итак, **JavaScript – язык программирования клиентских сценариев**. Сегодня он является самым массовым языком программирования.

Программа (сценарий) на языке JavaScript **встраивается** в текст веб-страницы. Миллионы веб-страниц, размещенных в Интернете, содержат сценарии на JavaScript. Такие сценарии добавляют в интерактивные веб-страницы динамичные эффекты.

С помощью JavaScript можно создавать разнообразные варианты меню и навигационных панелей, открывать всевозможные окна с заданными параметрами, реализовывать интерактивные тесты для самопроверки, и т.д. Интересной сферой применения является и программирование различных декоративных эффектов.

Будучи вполне полноценным алгоритмическим языком, JavaScript *лишен возможностей работы с файловой системой*. Это ограничение продиктовано соображениями безопасности клиентского компьютера.

JavaScript является *интерпретируемым* языком. Записанные в текстовом формате операторы JavaScript анализируются встроенным в браузер интерпретатором и выполняются либо *по мере интерпретации*, либо *в ответ на наступление некоторых событий*.

Сценарии JavaScript могут быть сохранены во внешних файлах, присоединяя которые к документам веб-ресурса можно использовать общие библиотеки, описывающие глобальные переменные, объекты и функции.

JavaScript является объектно-ориентированным языком, построенным на **прототипах** (а не **классах**!). Основную мощь этому языку придает **объектная модель документа** (**DOM**), описывающая множество объектов, сопоставленных элементам HTML-документа.

Встраивание кода Java Script в документ HTML

Содержащийся в html-документе программный код называют **сценарием (*script*)**.

Код JavaScript может быть использован в HTML-документе несколькими способами.

1) Программный код внутри элемента (контейнера) script

```
<script type="text/JavaScript">
... инструкции ...
</script>
```

На практике наряду с атрибутом **type** используют и атрибут **language**, несмотря на то, что в спецификации HTML он объявлен нежелательным (***depricated***). Атрибут **language** указывает на язык (напр, **VBscript**) и версию.

Например, при задании **language="JavaScript1.5"**

сценарий будет выполняться лишь в браузерах с поддержкой JavaScript1.5.

Элементы **script** могут содержаться как в разделе **head**, так и в **body**. При интерпретации документа они выполняются *последовательно*.

- В контейнере **head**, как правило, объявляются и инициализируются глобальные переменные и размещаются описания функций.
- В контейнере **body** фрагменты сценария обычно реализуют вставку динамически формируемого содержимого, *например,* обработку форм, вывод изображения или даты изменения документа …

2) Программный код во внешнем файле

Подключаемые внешние файлы с кодом JavaScript, определяются в разделе **head** с помощью элемента **script**. Путь к файлу задается значением атрибута **src**:

```
<script type="text/JavaScript" src="myF.js"> </script>
```

Подключение файлов предпочтительно, если одни и те же функции, объекты, глобальные переменные используются не в одном, а в нескольких документах.

Эти файлы (обычно имеющие расширение **.js**) перечисляют в разделе **head** соответствующих документов. Это упрощает поддержку сценариев и позволяет ускорить загрузку за счет кэширования браузером файла с кодом JavaScript.

3) Программный код в обработчиках событий

Обработчики событий указываются как **атрибуты html-элементов**.

Значениями этих атрибутов могут быть инструкции JavaScript.

Примеры:

- По щелчку мыши текст " Нажми!" станет красным

```
<b onClick="this.style.color='red'"> Нажми!</b>
```

- По наведению мыши меняется изображение:

```
<img src="lisa.gif" name="ris"
     onMouseOver="document.ris.src='volk.gif'"
     onMouseOut= "document.ris.src='zaj.gif'"
     onMouseDown="document.ris.src='lisa.gif'">
```

- Использование функции, которая задается в блоке **head** или внешнем файле:

```
<img src="lisa.jpg" id="ris"
     onMouseOver = "changeImg('volk.jpg')"
     onMouseOut = "changeImg('lisa.jpg')"
```

4) Программный код в гиперссылках (URL типа JavaScript)

В гиперсылке можно указать значение атрибута `href` как сценарий:

```
<a href = “javascript: список инструкций”>
```

В этом случае при выборе гиперссылки браузер исполняет код JavaScript.

Примеры:

- Возврат на предыдущую страницу

```
<a href="javascript:history.back()">назад</a>
```

- Открытие нового окна 300x200px и загрузке в него документа next.htm.

```
<a href="javascript:window.open('next.htm','newWin',
        width=300,height=200); void(0)">новое окно</a>
```

При использовании кода в гиперссылках следует помнить:
если **последняя** инструкция возвращает какое-то значение,
то его строковый эквивалент будет выведен в документ, заменив его содержимое.

Если в приведенном коде убрать `void(0)` , то новое окно, конечно, откроется, но в документе появится строка `[object]` или `[objectWindow]`(в зависимости от браузера), поскольку метод `window.open()` возвращает значение объектного типа. Во избежание этого применяют инструкцию `void(0)`.

Определенным недостатком последнего способа встраивания кода можно считать отображение в строке статуса не адреса целевого документа, а последовательности инструкций JavaScript, непонятной для большинства пользователей.

Схема изучения:

- **Базовые определения, семантика, синтаксис: переменные, операторы, выражения, типы данных, функции …**
- **Алгоритмические конструкции (ветвление, повторения…)**
- **Объекты, свойства и методы, события**
- **Объектные модели языка и браузера**

Основные понятия

Java Script – скриптовый объектно-ориентированный язык программирования, основанный на прототипах. Стандарт **ECMA-262**. Интерпретируется браузером.

- Java Script - **регистрозависимый** язык. ("верблюжья" нотация lastModified) Чувствительны к регистру имена (идентификаторы) переменных, функций, ключевые слова. Все ключевые слова используют **нижний** регистр.
- Имена могут содержать символы ASCII (латинские буквы, цифры), символы подчеркивания "_". Первый символ **не** должен быть цифрой. В качестве имен нельзя использовать ключевые и зарезервированные слова.
- Выражения разделяются точкой с запятой, которую можно опустить, если следующее выражение записывается на новой строке *(лучше не опускать).*
- Комментарии: // однострочный комментарий /* многострочный…….. . …….комментарий */

Типы данных

- Переменные **не имеют строгой типизации** (динамический контроль типов)
- Поддерживаются следующие типы данных:
  базовые: **number, boolean, string; undefined, null**
  ссылочные: **Array, Object.**
- Объявляются с помощью оператора **`var`**, который можно опускать. Возможно объявление с одновременной инициализацией, напр: **`var s = 123`**
- Для объявления локальных переменных в теле функции **`var`** обязателен!

Числа

- представляются в формате с плавающей точкой (и целые тоже!) - 64 бит).

Напр., **`123`** **`3.14169256`** **`.35`** **`1.2e-12`**

`015` `восьмеричное число начинается с` **`0`** 8+5=13

`0x5C` `шестнадцатеричное число начинается с` **`0x`** 16*5+12=92

Логические (булевы) величины

- представляются ключевыми словами **`true`** и **`false`**.

Строки

- заключаются в апострофы либо в двойные кавычки.

Для вывода кавычек и некоторых (управляющих) символов внутри строки перед ними должен стоять символ "\" (косая черта, слеш) - **экранирование**:

`\'` **`\"`** **`\\`** **`'`** **`"`** **`\`** или **`\n`** - перенос строки

Массивы

Массив представляет собой поименованный объект, доступ к значениям которого осуществляется с помощью индексов (целых чисел, **начиная с нуля!**).

Объявляется с помощью **new**, создающего новый *экземпляр* объекта **Array**.

```
teachers = new Array();                          //объявление без инициализации
teachers[0] ='Иванов'; teachers[1] = 'Петров';   //инициализация
days = new Array('Пн','Вт','Ср','Чт'); //объявление с инициализацией
otm = ['балл', 1, 2, 3, 'уд', 5, 6, 7, 8, 'отл', 10];  //присвоение с инициализацией
```

Объекты

В языке JavaScript определен ряд **встроенных** объектов для работы со строками, числами, датами…

Объектная модель документа DOM (***Document Object Model***), описывает множество объектов, соответствующих элементам **HTML-документа** и **браузера.**

Объект имеет **свойства, методы** (функции, определенные для объекта), **события**.

К экземплярам объектов в сценариях обращаются

по именам (name), идентификаторам (id),

по индексам (ключам), как к элементам массива**.**

При обращении к свойствам и методам объектов используется **точечная нотация:** имя переменной объектного типа отделяется от имени свойства или метода **точкой**.

При обращении к элементу массива индекс (ключ) в квадратных скобках ["ris2"].

Главным в иерархии объектов DOM является окно браузера – объект **window**.

С этим объектом связано большое количество свойств и методов.

Например, у объекта **window** есть метод **open()**, открывающий новое окно с указанными параметрами. Вызвать его можно следующим образом:

```
window.open('qq.htm','new','width=300,height=200,toolbar=1')
  // откроется окно 300x200,  в котором будет только панель инструментов.
```

Объект **screen** дает доступ к параметрам экрана монитора клиентского компьютера (конечно, эти параметры можно лишь узнать, но не изменить).

Методы для этого объекта не определены, но определен ряд свойств.

Наиболее полезные **width**, **height** (ширина и высота в пикселах), **availWidth** и **availHeight** (доступная ширина и высота в пикселах).

Обращение к свойствам: `w = screen.width;  h = screen.height;`

Операторы

Основные операторы (***operator***) в порядке убывания приоритета.

Оператор	выполняемая операция
. (точка)	доступ к свойству или методу объекта
[ ]	доступ к элементу массива
( )	вызов функции
++ , --	приращение (**инкремент**), уменьшение (**декремент**)
-	унарный минус
!	отрицание (логическое NOT)
* , / , %	умножение, деление, взятие по модулю
+ , -	сложение и вычитание
+	сцепление (конкатенация) строк
< , <= , > , >=	меньше, меньше или равно, больше, больше или равно
== , !=	проверка равенства и неравенства
=== , !==	проверка идентичности (т.е. равенства и неравенства без преобразования типа операндов)
&&	логическое **И** (второй операнд вычисляется, если первый **true**)
||	логическое **ИЛИ** (второй операнд вычисляется, если первый равен **false**)
=	присваивание
+= , -= , *= , /=	присваивание с операцией

new	создание экземпляра объекта
delete	удаление свойств объекта
typeOf	возвращает тип операнда
void	возвращает неопределенное значение

Примеры: ***a++; a=a+1; b--; b=b-1;***
a+=7; a=a+7; b-=3; b=b-3;
c*=4; c = c*4; d/=3; d=d/3;
7 % 2 1 8 % 2 0

Преобразование типов

a=7; b = 2; *// числа* ***c =’3’ ;*** *// строка*

результаты выполнения операций: ***a+b*** *сумма чисел* ***9***

a-c *разность* ***4*** ***a*c*** *произведение* ***21*** // преобразование в числа

но ***a+c*** *строка* ***‘73’*** // конкатенация с преобразованием в строки

Инструкции

Простыми **инструкциями** (англ. ***Statement***) в JavaScript являются операторы присваивания, вызовы методов объектов, операторы инкремента и декремента и т.д. Иногда *Statement* переводят как *оператор, выражение…*

Последовательности инструкций можно объединять в блоки, получая **составную инструкцию** (или *составной оператор*).

Для этого последовательность инструкций заключается в **фигурные скобки {...}**.

Инструкции могут быть вложены одна в другую с любой степенью вложенности.

Алгоритмические конструкции

Цикл while с предусловием

```
while (условие) инструкция_выполняемая_в_случае_истинности ;
```

Условие - выражение, имеющее значение логического типа,

например: `x > 2.5` или `(a != 0) && (b > 1)`

Цикл while с постусловием

```
Do инструкция_выполняемая_в_случае_истинности_условия
while (условие) ;
```

Цикл с параметром for

```
for (нач_знач_счетчика; проверка_условия; изменение_счетчика)
    инструкция ;
```

Примеры: вычисление суммы и произведения чисел от 1 до 10.

```
x = 0; y= 1;
 for (i=1; i<=10; i++ )
 { x+=i; y*=i; };
```

```
x = 0; y= 1;  i=1;
 while (i<=10)
 { x+=i; y*=i;  i++  };
```

Цикл for…in (перебор всех элементов массива, коллекции)

```
for (var prop in document)
    document.write(prop);
```

Ветвление if

```
if (условие)
    инструкция_выполняемая_в_случае_истинности
else  инструкция_выполняемая_в_случае_ложности;
```

Например, в результате выполнения следующего кода переменная `mes` получит строковое значение 'отлично'.

```
ball =5;
 if (ball == 5) mes ='отлично'
     else mes ='маловато';
```

Если необходимо последовательно рассмотреть выполнение нескольких условий, используются вложенные условные конструкции.

В простейших случаях вместо условной инструкции можно использовать **оператор условного присваивания** **= ? :**

```
mes = (ball == 5) ? 'отлично' : 'маловато' ;
```

Инструкция switch

Если вариантов значения условия много, и конструкция **if** получается слишком громоздкой, удобна инструкция **switch** следующего формата:

```
switch (выражение )
     {   case значение1 : инструкция;  break;
         case значение2 : инструкция;  break;
          ...
         case значениеN : инструкция;  break;
         default : инструкция по умолчанию;  }
```

Оператор **break** прекращает выполнение инструкции **switch** в случае найденного соответствия. Если его не поставить, будут выполняться все последующие проверки **case**. Вариант по умолчанию (**default**) указывать необязательно.

Например, следующий цикл прервется на 2-й проверке и установит значение 'хорошо':

```
ball = 4;
switch (ball)
{    case 5: mes ='отлично';     break;
     case 4: mes ='хорошо';      break;
     case 3: mes ='удовл';       break;
     case 2: mes ='плохо';       break;          };
```

Приведенный пример иллюстрирует смысл инструкции **switch**. Конечно, в данном случае рациональнее было бы использовать массивы.

Инструкции break и continue

Инструкция **break** приводит к выходу из цикла или инструкции **switch**. В случае вложенных циклов выход происходит из самого внутреннего.

Инструкция **continue** прерывает выполнение текущей итерации цикла и запускает следующую.

Инструкция var

Инструкция **var** служит для **явного объявления** глобальных и локальных переменных. При объявлении глобальной переменной с одновременной инициализацией эту инструкцию можно опустить.

При объявлении **локальной** переменной инструкция **var** обязательна.

Функции

Функция объявляется с помощью ключевого слова **function**,
за которым (не менее чем через один пробел) следуют:

- **имя** функции.
- заключенный в круглые скобки **список аргументов** через запятую.
  Круглые скобки **обязательны** даже, если функция не имеет аргументов.
- заключенный в фигурные скобки **набор инструкций - тело функции**.
  Фигурные скобки **обязательны**, даже если тело функции – одна инструкциея.
  Инструкции выполняются при вызове функции.

```
function имя_функции(арг1, арг2, арг3,… )
         {    инструкции_тела_функции     };
```

Замечание! Это **не единственный** способ определения функции.

В теле функции может присутствовать инструкция **return**,
которая прекращает выполнение функции и возвращает указанное выражение:

```
return выражение;
```

Пример **анонимной** (безымянной) функции:

```
S = function(a,b)
    {  return a*b  };
```

Функции могут быть **вложенными**, т.е. одна функция определяется в теле другой.

Функции могут быть заданы **рекурсивно**, т.е. функция может вызывать саму себя.

Пример:

```
function fact(n)                  //вычисляет и возвращает факториал
   {  if (n =< 1) return 1;
       return n*fact(n-1);  };
```

Области видимости переменных

Английский термин ***scope*** переводят как «область действия» или «область определения» переменной.

Область видимости переменной – это блок программы, в котором переменная определена (действует).

Переменная, объявленная вне какой бы то ни было функции (***глобальная***), определена для всего сценария (уровень объекта **window**). При объявлении глобальной переменной с одновременной инициализацией инструкцию **var** можно опустить.

Переменная, объявленная при помощи инструкции **var** в **теле функции**, является **локальной**, т.е. действует только в теле функции. Ее значение доступно лишь при вызове этой функции. Примеры:

```
var x;                    //явное объявление без инициализации
a = 123;                  //неявное объявление с инициализацией
var q ='привет';          //явное объявление с инициализацией
```

Объектные модели JavaScript.

В классовой модели каждый объект относится к некоторому **классу** (*типу данных*). Класс описывает общие свойства и методы группы объектов. Отдельные представители класса называются ***экземплярами*** класса (**instance of class**) или просто ***объектами*.**

Напр, Бобик является экземпляром класса *собака*, т.е. имеет все свойства и методы, присущие собакам.

Объекты Java Script строятся на основе **прототипов**. Классы **не** объявляются.

Базовым объектом является «родовой» объект **Object**. Это «объект вообще», для которого еще не определены ни свойства, ни методы. Отдельные представители называются ***экземплярами*** прототипа (**instance of object**) или просто ***объектами*.**

Встроенные объекты языка JavaScript – стандарт ECMA 262

Примеры встроенных объектов: **Array** – массив, **Date** - дата и время, **Math** – математика….

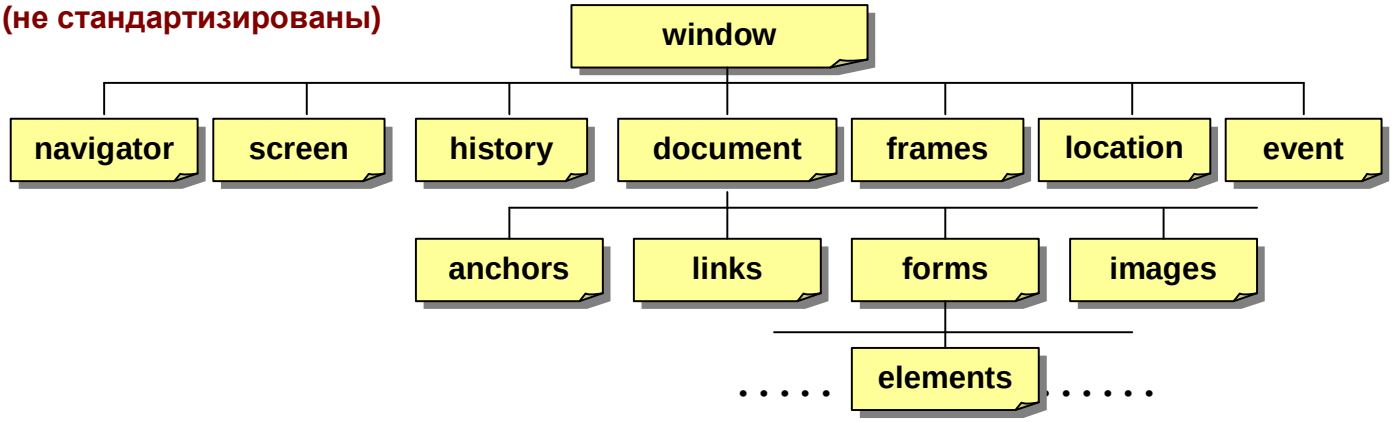
Объектная модель документа (DOM)

Объектная модель документа **не является частью языка JavaScript**.

Это **отдельные стандарты**, в настоящее время развиваемые под эгидой **W3C**.

DOM (*Document Object Model*) – это интерфейс прикладного программирования для представления документа, обеспечения доступа к его элементам и интерактивного изменения свойств. DOM обеспечивает механизмы для изменения самой структуры документа (добавление и удаление элементов, изменение их содержимого).

Объектная модель браузера и документа (не стандартизированы)



Экземпляры объектов **создаются с помощью конструктора (**инструкция **new)**:

Например: **stud= new Array();** массив; **d = new Date();** дата и время

Некоторые объекты существуют только в ***единственном*** экземпляре – это т.н. **глобальные объекты.**

Например: **Math** — матем. функции; **navigator** — браузер; **screen** — монитор…..

Используются **без конструктора:** **Math.sin(Math.PI*ugol/180);**
navigator.appName; **screen.width;**

События Java Script

В языке ActionScript реализована **событийная** модель программирования.

В JavaScript определен ряд событий, связанных с элементами документа.
Обработчики дают возможность организовать реакцию на события.
Обработчик может указываться как **атрибут элемента**;
значением этого атрибута является **выражение** JavaScript.

Примеры:

- Наведение курсора мыши на элемент сопровождаются изменением цвета фона.

```
<p onMouseOver="this.style.bgColor='yellow'"
   onMouseOut="this.style.bgColor='green'">Внимание!</p>
```

- При попытке пользователя закрыть окно и выгрузить документ выводится сообщение

```
<body onUnLoad="alert('окно закрывается!')">
```

- При щелчке мышью по изображению выполняется функция пользователя

```
<img src="cat.jpg" onClick="showPict()">
```

Часть обработчиков поддерживаются практически всеми HTML-элементами. Некоторые события можно имитировать с помощью соответствующих методов.

Список событий по спецификации **HTML 4.0** (+ некоторые события MS IE)

Обработчик события	**Поддерживающие HTML-элементы и объекты**	**Описание**	**Метод имитации**
onclick	Практически все*	Одинарный щелчок (нажата и отпущена кнопка мыши)	
ondblclick	Практически все*	Двойной щелчок	
onfocus	**a, area, button, input, label, select, textarea**	Получение элементом фокуса (щелчок мышью на элементе или очередное нажатие клавиши табуляции)	**focus()**
onblur	**a, area, button, input, label, select, textarea**	Потеря текущим элементом фокуса, т.е. переход к другому элементу. Возникает при щелчке мышью вне элемента либо нажатии клавиши табуляции	**blur()**
onchange	**input, select, textarea**	Изменение значений элементов формы. Возникает после потерей элементом фокуса, т.е. после события **blur**	
onselect	**input, textarea**	Выделение текста в текущем элементе	**select()**
onreset	**form**	Сброс данных формы (щелчок по кнопке **<input type="reset">**)	**reset()**
onsubmit	**form**	Отправка данных формы (щелчок по кнопке **<input type="submit">**)	**submit()**

onload	**body, frameset**	Закончена загрузка документа	
onunload	**body, frameset**	Попытка закрытия окна браузера и выгрузки документа	
onkeydown	Практически все *	Нажата клавиша на клавиатуре	
onkeyup	Практически все *	Отпущена клавиша на клавиатуре	
onkeypress	Практически все *	Нажата и отпущена клавиша на клавиатуре	
onmousedown	Практически все *	Нажата кнопка мыши впределах текущего элемента	
onmouseup	Практически все *	Отпущена кнопка мыши в пределах текущего элемента	
onmousemove	Практически все *	Перемещение курсора мыши в пределах текущего элемента	
onmouseover	Практически все *	Курсор мыши наведен на текущий элемент	
onmouseout	Практически все *	Курсор мыши выведен запределы текущего элемента	
onmove	**window**	Перемещение окна	
onresize	**window**	Изменение размеров окна	
onabort	**img**	Прерывание загрузки изображения	
onerror	**img, window**	Возникновение ошибки выполнения сценария	

***** Практически все – *все HTML-элементы, за исключением:*
html, head, title, meta, base, frame, frameset, iframe, style, font, basefont, br, script, applet, param

Заметим, что специфические события, применимые лишь к отдельным элементам, в основном относятся к элементам форм.

Элементы форм представляют особый класс HTML-элементов, так как их функциональность напрямую связана с работой сценариев (клиентских или серверных). Для работы с ними приходится обрабатывать особые события.

Встроенные (глобальные) функции языка

eval(строка)

Функция выполняет переданную ей строку так, как если бы это была JavaScript-інструкция. Если результат выполнения имеет значение, то оно будет возвращено функции **eval().**

Пример 1

```
var x = eval("100/5-15");
alert("x=" + x);                    // будет выведено  x = 5
```

Пример 2

```
eval("var d = 10; alert('d='+d)");      // вывод  d = 10
```

Пример 3 (калькулятор)

```
var str = "Math.sqrt(100)+5";
alert(eval(str));                       // вывод  15
```

parseInt(строка, осн) или parseInt(строка)

Преобразование строки в целое число.
В качестве параметра **осн** указывается основание системы счисления. Если параметр **осн** отсутствует или система счисления конфликтует с первыми символами, JavaScript использует систему счисления, анализируя первые символы строки:

символы	система
0	8 (восьмеричная)
0x	16 (шестнадцатеричная)
другие	10 (десятичная)

Если **parseInt** сталкивается с недопустимым символом, то возвращает значение, основанное на подстроке, следующей до этого символа, игнорируя все последующие. Если первый же символ не допустим, **parseInt** возвращает значение **NaN**.

Пример 1

```
var num = parseInt("-46.35");
alert(num);                             // вывод  -46
```

Пример 2

```
var num = parseInt("111", 2);
alert(num);                             // вывод  7
```

parseFloat(строка)

Преобразование строки в число с плавающей точкой.
Если **parseFloat** сталкивается с недопустимым символом, то возвращает значение, основанное на подстроке, следующей до этого символа, игнорируя все последующие. Если первый же символ не допустим, **parseFloat** возвращает **NaN.**

Пример

```
var num = parseFloat("3.47");
alert(num);                             // вывод  3.47
```

isNaN(строка)

Если строка не является числом, возвращает **true**, иначе -- **false**.

Пример

```
alert(isNaN("3.47"))                    // будет выведено  false
```

Встроенные объекты Java Script

объект Math Использование математических функций и констант.

Объект существует в **единственном** экземпляре.
Нельзя создать свой собственный экземпляр этого объекта.

методы объекта Math -функции

abs(arg) Возвращает абсолютное значение аргумента.

sin(arg) **cos(arg)** **tan(arg)** тригонометрич функции, *арг* в радианах.

asin(arg) **acos(arg)** **atan(arg)** обратные тригонометрические функции, возвращают значения в радианах.

exp(arg) **log(arg)** Экспонента и натуральный логарифм (основание e)

pow(arg1,arg2) Возвращает ***arg1*** в степени ***arg2***.

sqrt(arg) Корень квадратный из аргумента.

ceil(arg) **floor(arg)** **round(arg)** Округление

random() генерирует случайное число из диапазона **[0,1]**.

min(arg1,arg2) **max(arg1,arg2)** **min/max** из двух числовых аргументов

свойства объекта Math - константы

E Основание натурального логарифма (константа Эйлера). **PI**

LN10 **LN2** Натуральный логарифм числа 10. или 2

LOG10E **LOG2E** Логарифм числа E по основанию 10 или 2

SQRT2 **SQRT1_2** Квадратный корень из 2. или из ½

объект Array Представление массивов данных и операций над ними.

создание экземпляра

```
new Array();
new Array(e1,e2,...,eN);
```

где **e1, e2, ..., eN** - элементы массива.

Конструктор без параметров создает пустой массив.

Экземпляр также создается по умолчанию при таком описании переменной:

```
var wd = ["Пн","Вт","Ср"];
```

Эта запись эквивалентна следующей:

```
var wd = new Array("Пн","Вт","Ср");
```

Нумерация элементов массива начинается с нуля!

Для доступа к элементам массива используют конструкцию: **имя_массива[индекс]**

Длина массива (число элементов в нем) **может меняться** во время работы программы.

методы объекта Array

concat(array) Возвращает массив, полученный добавлением массива array.
Исходный массив не меняется.

Пример:

```
var ar1 = new Array (1,2);  var ar2 = new Array (3,4);
var ar  = ar1.concat(ar2);
      alert(ar1+"\n"+ar2+"\n"+ar);   // будет выведено  1, 2, 3, 4
```

reverse() Изменяет порядок следования элементов массива на обратный (первый элемент станет последним)

Пример:
```
var ar = new Array (1,2,3);
ar.reverse();     alert(ar);          // будет выведено  3, 2, 1
```

slice(ind1,ind2) или **slice(ind1)**

"Вырезает" из исходного массива элементы с индекса **ind1** по **ind2-1**. Исходный массив не меняется.

- Если **ind2 < 0** (меньше нуля), то отсчет последнего выделенного элемента ведется с конца массива. Вырезаются элементы с **ind1** по **ar.length-ind2-1**.
- Если **ind2** опущен, то вырезаются элементы с **ind1** до конца массива, т.е по **ar.length-1**.

Пример:
```
var ar  = new Array (0,1,2,3);  var ar1 = ar.slice(1,3);
var ar2 = ar.slice(1,-1);       var ar3 = ar.slice(1);
                  // будет выведено  ar = 0,1,2,3   ar1 = 1,2   ar2 = 1,2   ar3 = 1,2,3
```

sort(func) или **sort()** Сортировка массива.

Параметром является имя функции, которая задает правила сравнения двух элементов. Если параметр опущен, элементы сортируются по возрастинию или по алфавиту.

Пример:
```
var ar = new Array("zebra","ant","dog","cat");
ar.sort();                          // будет выведено  ant, cat, dog, zebra
```

Функция **func** содержит два аргумента и возвращает:

- отрицательное число, когда первый аргумент считается расположенным левее второго в смысле определяемого порядка;
- 0, когда аргументы считаются равнозначными в смысле определяемого порядка;
- положительное число, когда первый аргумент считается правее второго в смысле определяемого порядка.

Пример:
```
var ar = new Array (26,71,9,1);
    func Compare(a,b)
        {  return a-b ; }
ar.sort(Compare);  alert(ar);  // будет выведено  1, 9, 26, 71
```

s.split('d') преобразование строки в массив (расщепление по разделителю **d**)

Пример:
```
var ar = new Array();     str = "Пн,Вт,Ср,Чт,Пт,Сб,Вс";
ar = str.split(',');                  // получим массив дней недели
```

свойство объекта Array

length Длина массива (число элементов).

Пример:
```
var ar = new Array(0,1,2,3,4,5,6,7,8,9);
alert(ar.length);                     // будет выведено  10
```

объект String Представление строк и работа с ними.

создание экземпляра **new String(str)** где **str** строка.

По умолчанию экземпляр создается и при таком описании переменной:

```
var str="Скоро сессия";
```

Эта запись эквивалентна следующей:

```
var str=new String("Скоро сессия");
```

методы объекта String

charAt(ind) Возвращает символ с указанной позиции (нумерация с нуля).

Пример:
```
var str= new String("абвгд");   var ind = 2;
alert(str.charAt(ind));          // будет выведено в
```

indexOf(subStr, startInd) или **indexOf(subStr)**

Метод возвращает позицию, с которой начинается подстрока **subStr** в строке.

Поиск начинается с позиции **startInd** до конца строки.

Если **startInd** не задан, поиск начинается с нулевой позиции.

Если подстрока не найдена, метод возвращает -1.

Пример:
```
var str = new String("чрезвычайный");
var ind = str.indexOf("чай",3) ; alert(ind);   // вывод 6
```

lastIndexOf(subStr,startInd) или **lastindexOf(subStr)**

Метод возвращает позицию, с которой начинается подстрока **subStr** в строке.

Поиск начинается с позиции **startInd** до начала строки.

Если **startInd** не задан, поиск начинается с конца строки к ее началу.

Пример:
```
var str = new String("чрезвычайный");
var ind = str.lastIndexOf("чай",9); alert(ind);   //вывод 6
```

substring(ind1,ind2) или **substring(ind1)**

Возвращает подстроку от позиции **ind1** до позиции **ind2-1**.

- Если **ind1 < 0**, он полагается равным нулю.
- Если **ind2** больше длины строки, он полагается равным длине строки.
- Если **ind1** равен **ind2**, возвращается пустая строка.
- Если **ind2** опущен, возвращается конец строки, начиная с позиции **ind1**.

Пример:
```
var str = new String("чрезвычайный");
var s = str.substring(6,9) ; alert(s);   // вывод чай
```

toLowerCase() **toUpperCase()**

Возвращает строку, преобразованную к нижнему(верхнему) регистру (исходная строка никак не меняется).

свойство объекта String

length Длина строки.

Пример:
```
var str = "Интернет"; alert(str.length);   // вывод 8
```

объект Date работа с датой и временем.

создание экземпляра

```
new Date()
new Date(year, month, dat)
new Date(year, month, dat, hours, minutes, seconds)
```

Экземпляр, созданный без использования параметров, содержит текущую дату и время.
Параметры **month, day, year, hours, minutes, seconds** являются целыми числами
и задают соответствующие поля объекта части даты и времени.
Если часы, минуты и секунды не заданы, они устанавливаются в ноль.

параметр	*значение*	*диапазон*
year	год	напр, 2012
month	месяц	0..11 (январь..декабрь)
dat	день месяца (от 1 до 31).	
day	день недели (от 0 (воскресенье) до 6 (суббота)).	
hours	часы	0..23
minutes	минуты	0..59
seconds	секунды	0..59

методы объекта Date

getYear() setYear(year)

Возвращает (устанавливает) год. До 2000 года возвращает только две последние цифры.

getMonth() setMonth(month)

Возвращает (устанавливает) месяц.

getDate() setDate(dat)

Возвращает (устанавливает) день месяца (дату).

getDay()

Возвращает день недели (**0-воскресенье, 1-понедельник, ... 6-суббота**).

getHours() setHours(hours)

Возвращает (устанавливает) час.

getMinutes() setMinutes(minutes)

Возвращает (устанавливает) минуты.

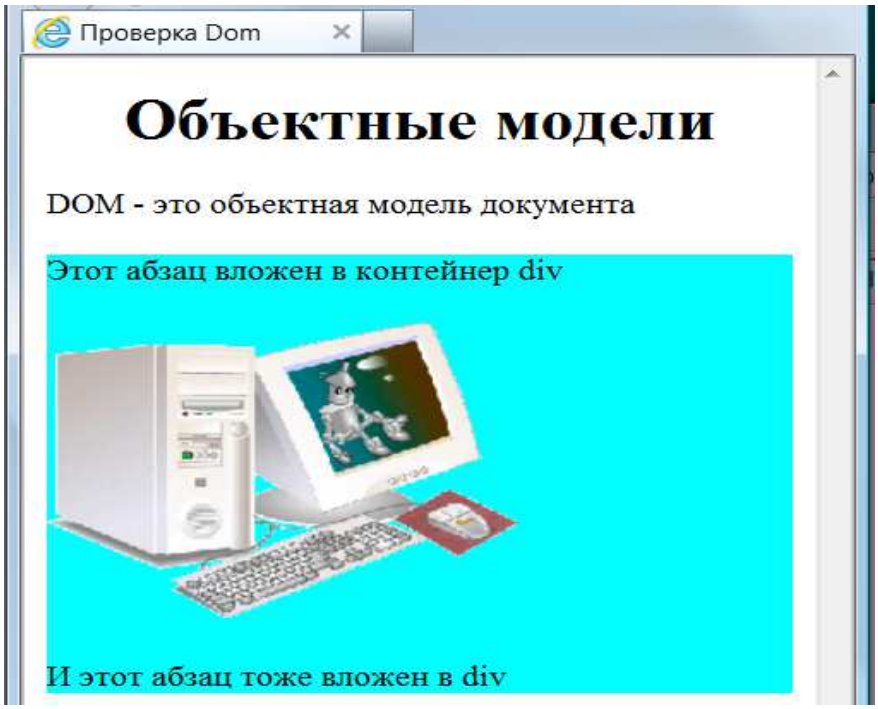
getSeconds() setSeconds(minutes)

Возвращает (устанавливает) секунды.

getTime() setTime(time)

Возвращает число миллисекунд прошедших с 0 часов 1 января 1970.
Устанавливает число миллисекунд прошедших с 0 часов 1 января 1970

Объектные модели документа и браузера



File Edit Format

```
1  <html>
2  <head>
3  <title>Проверка Dom</title>
4  </head>
5  <body>
6  <h1 align=center>Объектные модели</h1>
7  <p id=p1>DOM - это объектная модель документа</p>
8  <div style="background-color: cyan;">
9    <p>Этот абзац вложен в контейнер div</p>
10   <img src=comp.gif alt="компьютер">
11    <p>И этот абзац тоже вложен в div</p>
12 </div>
13 </body>
14 </html>
```

Структура гипертектового документа представляется браузером в виде иерархического **дерева**.

Каждый элемент, точнее **узел** (*англ*. **Node**) дерева соответствует определенному объекту в документе.

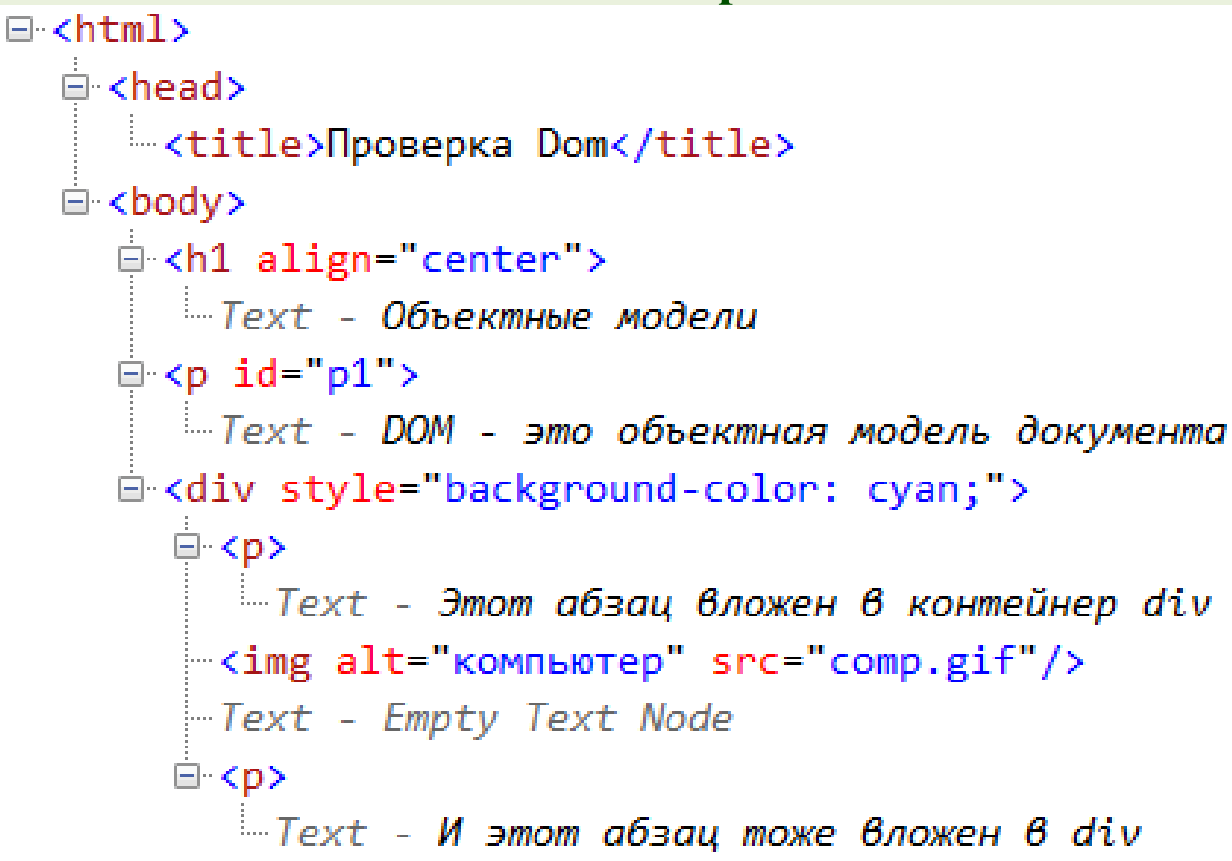
Модель **DOM** определяет 12 типов узлов.

Большинство из них - для XML.

В HTML в основном используются 6 типов:

Номер	Тип	Описание	примеры
1	**элемент**	элемент HTML или XML документа	<body>…<body> , … , <p>…<p> <img…>, <a...>….
2	**атрибут**	атрибут элемента	src, href,..
3	**текст**	фрагмент текста	
8	**комментарий**	комментарий	
9	**документ**	корневой объект	<html>
10	**тип документа**	определение типа	<!DOCTYPE HTML….>

DOM-дерево



Обращение к элементам - узлам DOM-дерева

document.getElementById('мой_id') - по идентификатору **id**

document.getElementsByTagName('тег') - по тегу

document.getElementById('d1').parentNode родитель

document.getElementById('d1').childNodes дети

document.getElementById('d1').previousSibling пред брат

document.getElementById('d1').nextSibling след брат

для сокращения записи – оператор **with**

например:
```
with (document.getElementById('d1')) {
        parentNode.tagName; childNodes.length; ….
```

document.images[i] по индексу **i** элемента в коллекции **images**

document.all[i].tagName по индексу **i** элемента в коллекции **all**

Примеры обработки событий

(функции-обработчики вызываются из элементов форм)

```
function showNodeP()  {              // обращение к узлу по id  (p1)
   with (document.getElementById('p1'))   {
         var msg = nodeName +"\n" + nodeType +"\n"+ nodeValue;  }
  alert("Узел\n" + msg);        }

function showNodeD()  {            // обращение к узлу по id  (d1)
   with (document.getElementById('d1'))  {
        var childs = childNodes;      var msg="";                 // + дети
        for (var i=0;  i < childs.length;  i++)
              msg += " --- " + childs[i].tagName +"\n" ;
    alert(nodeName +"\n" + msg);     }   }

function showTags()  {                 элементам коллекции all
   msg="";      var N = document.all.length;
   for (i=0;  i<N;  i++)
       msg += i + "  " + document.all[i].tagName +"\n";
  alert("Количество объектов = " + N + "\n" + msg);    }
```

```
function showRis()  {      // обращение к узлу по имени тега  (img)
   ris = document.getElementsByTagName('img');    msg = ' ';
   for (i=0;  i<ris.length;  i++)
       msg+= '\n' + ris[i].width + ' x '+ ris[i].height + ' - ' + ris[i].alt;
  alert('К-во: ' + ris.length + '\nширина, высота, подсказка: ' + msg); }

function showAbz()  {      // обращение к узлу по имени тега  (p)
    abz = document.getElementsByTagName('p');     msg="";
       for (i=0;  i<abz.length;  i++)
           with (abz[i])   msg += "\n"+ nodeName + "\n";
    alert("Всего абзацев: " + abz.length + msg);          }

function showAbz()  {         // обращение к узлу по имени тега  (p)
    abz = document.getElementsByTagName('p');     msg="";
    for (i=0;  i<abz.length;  i++)
      msg += "\n"+ abz[i].nodeName + "\n" +
          abz[i].firstChild.nodeName + ":   " + abz[i].firstChild.nodeValue;
    alert("Всего абзацев: " + abz.length + msg);      }
```

"Родственники"

```
function showParent()  {                    // обращение к родителю
     var parent = document.getElementById('d1').parentNode;
     alert(parent.tagName +"\n");    }

function showChilds()  {                        // обращение к детям
    var childs = document.getElementById('d1').childNodes;
    msg="";     for (var i=0; i<childs.length; i++)
                      msg +=  childs[i].tagName +"\n" ;
     alert(msg);    }

function showPrevSib()  {         // к предыдущему брату
    var prevSib = document.getElementById('d1').previousSibling;
    alert(prevSib.tagName +"\n");    }

function showNextSib()  {         // обращение к следующему брату
    var nextSib = document.getElementById('d1').nextSibling;
    alert(nextSib.tagName +"\n");     }
```

Реализация DOM

Все интерпретаторы языка JavaScript в той или иной степени поддерживают объектную модель документа (**DOM**).

Стандарт **DOM0** поддерживается всеми браузерами, но предоставляет лишь базовые возможности. Современные браузеры ориентированы на стандарт **DOM2** (к сожалению в несколько различающейся интерпретации).

Стандарт **DOM2 (*Object Document Model level2* - DOM2)** определен организацией WorldWideWeb Consorcium (**W3C**). Он поддерживает объекты модели DOM0 и предлагает концепцию представления элементов HTML-документа. Акцент модели DOM2 - иерархическая структура документа, в которой каждый элемент является объектом (с точки зрения алгоритмического языка) со своими свойствами, методами и событиями.

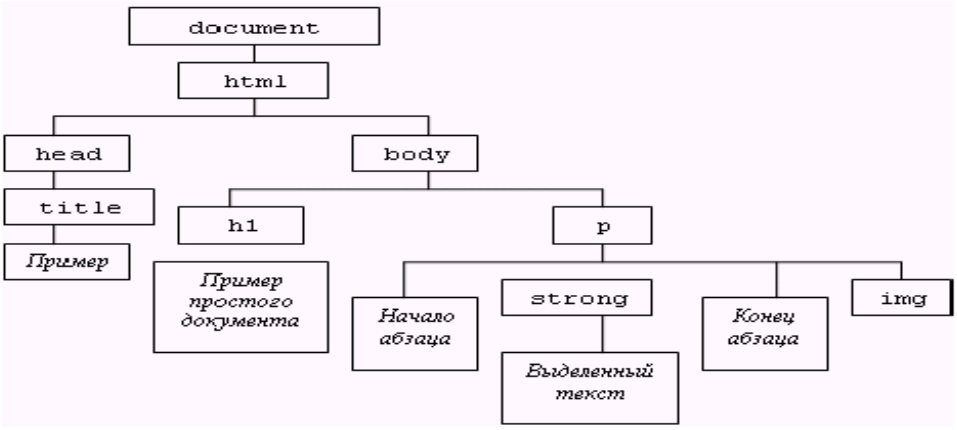
Модель DOM2 отображает реальную иерархическую структуру документа. Например, в любом HTML- документе есть разделы **head** и **body**, в каждом из которых своя иерархия элементов.

В разделе **head** указываются элементы, которые непосредственно не отображаются в окне браузера, но могут влиять на общее отображение и функциональность документа (определение внешних таблиц CSS, коды JavaScript), на идентификацию документа (многие meta-элементы). Раздел **body** имеет явную иерархическую структуру.

Рассмотрим, например, очень простое описание HTML-документа

```
<html>
    <head>
      <title>Пример</title>
    </head>
  <body>
    <h1>Пример простого документа</h1>
    <p>Начало абзаца<strong>выделенный текст</strong>продолжение
абзаца <img src="test.jpg"></p>
  </body>
</html>
```

Этот документ может быть представлен деревом, состоящим из следующих узлов:



Такую иерархическую модель документа и предлагает стандарт DOM2. Отдельного рассмотрения интерфейса прикладного программирования (API) в нем нет. Однако, именно обращение к ряду свойств и методов, представленных в DOM API, позволяет сценариям JavaScript обеспечивать необходимую функциональность в большинстве задач.

Свойства HTML-элементов

DOM2 определяет ряд общих свойств элементов, а также наборы свойств, специфичных для каждого элемента.

При этом имена атрибутов HTML транслируются в имена свойств следующим образом:

- Имя атрибута HTML, представляющего одно слово, транслируется в имя свойства **без изменений**. Все буквы в имени свойства **строчные**.
  Исключение: атрибут `class` транслируется в свойство `className`.
- Имя атрибута HTML, представляющего несколько слов, транслируется в имя свойства, в котором все слова идут слитно (без дефисов и прочих разделителей). Первые буквы второго и последующих свойств являются прописными.
  Например: `maxLength`, `borderColor` и т.д.

Все элементы имеют свойства `id, style, title, lang, dir` и `className`.
В дополнение к этим общим свойствам, каждый элемент обладает специфическими.
Например: для элемента `img` это свойства `src, width` и `height`.

Доступ к свойствам элементов документа позволяет организовать их интерактивное изменение.

Идентификация элементов документа

Модели документа DOM0 и DOM2 предоставляют свои возможности идентификации элементов документа. Для чего нужна эта идентификация? Для того, чтобы при реализации определенного события (`click, mouseOver, mouseOut`, ...)

- указанному элементу присвоить определенные свойства стиля: цвет, фон, видимость и т.д (скажем, при реализации меню);
- организовать смену изображений (то, что называется ***rollover image***);
- обработать данные форм на стороне клиента; и т.д.

Итак, чтобы изменить какие-то свойства элемента HTML-документа или применить соответствующие методы, надо иметь механизм, позволяющий идентифицировать необходимый элемент. Возможности модели DOM0 беднее, но начнем с них. Разумеется, средства идентификации DOM0 распространяются и на модель DOM2, имеющую дополнительные возможности.

- **Ссылка на текущий элемент**

Во всех моделях документа при вызове обработчика для **текущего** элемента используется ключевое слово `this`. Например:

```
<span onMouseOver="this.style.color='red'"
      onMouseOut="this.style.color='blue'"
onClick="this.style.color='springgreen'"> тест  </span>
```

Здесь заданы обработчики трех событий для текущего элемента.

- **Ссылки на объекты, вложенные в объект document**

Эти объекты представляют собой коллекции (массивы): массив гиперссылок **links[]**, массив изображений **images[]**, массив форм **forms[]** и т.д. Индексация массивов начинается с 0, поэтому, например, к i-му изображению документа можно обратиться **images[i-1]**. Очевидно, что такая возможность актуальна лишь в случаях, когда требуется перебрать **все** элементы данной группы. Например, перебрать все элементы формы с целью выяснить, какие из них являются **checkbox**-ами и включены ли они.

- **Ссылки на имя (name) HTML-элемента**

Некоторые HTML-элементы имеют атрибут **name**. Это изображения **img**, а также формы и их элементы. К ним можно обращаться, используя значение атрибута **name**, причем по крайней мере в двух вариантах.

Пример. Пусть изображение в HTML описано так:

```
<img src="view.jpg" name="ris">
```

Тогда в сценарии JavaScript это изображение идентифицируется как **document.images['ris']** или **document.images.ris**

Первый вариант напоминает обращение к элементу ассоциативного массива. Действительно, все объекты JavaScript по сути являются ассоциативными массивами, ключи которых совпадают с именами свойств. Это и объясняет возможность применения двух приведенных вариантов.

Помимо перечисленных, модель DOM2 поддерживает и другие варианты идентификации элементов документа, основным из которых является следующий:

- **Идентификация по ID**

Если у HTML-элемента указан атрибут **id**, то к нему можно обращаться, используя значение этого атрибута. При этом применяется метод **getElementById()**. Обратите внимание на то, что это - метод объекта **document**. Ниже приведен пример.

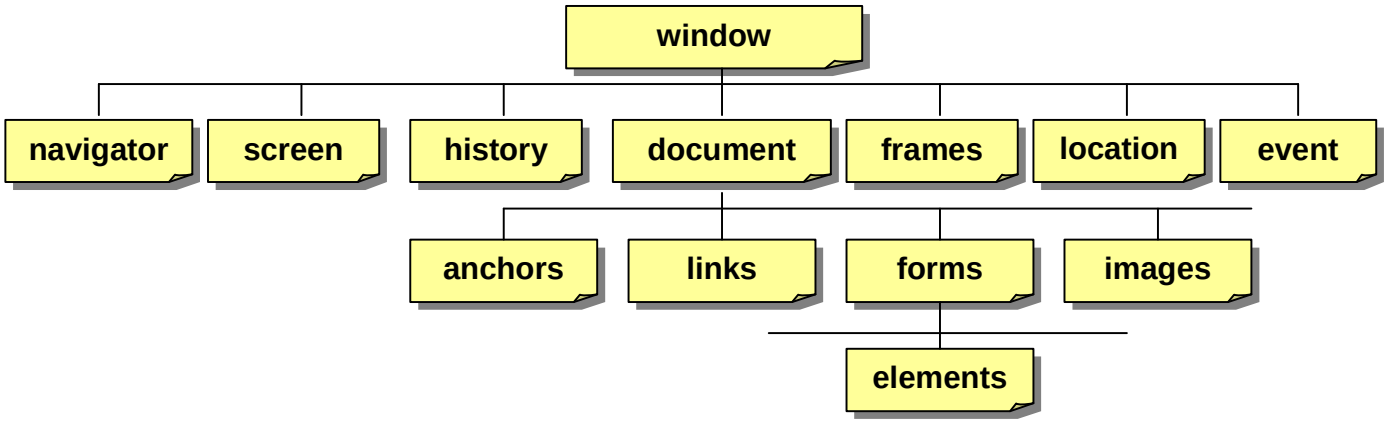
```
<p id="q">этот текст меняет цвет</p>
<p onClick="document.getElementById('q').style.color ='red'">
   щелкни мышкой, и предыдущий текст станет красным </p>
```

Щелчок мышью по элементу, в котором задан обработчик **onClick**, приведет к изменению цвета элемента с **id="q"**.

Обращение ко всем элементам указанного типа

Метод **getElementsByTagName()** объекта **document** возвращает массив элементов с указанным именем тега. Например, **getElementsByTagName('h2')** возвратит массив всех заголовков 2- го уровня.

Основные объекты браузера и HTML-документа



Основным объектом является **window**, ссылающийся на текущее окно браузера. Остальные объекты являются свойствами корневого объекта **window**. Почти все эти объекты имеют много свойств, с ними связаны события и методы, использование которых позволяет создавать сценарии, обеспечивающие необходимую функциональность.

Объект Window. Окна и фреймы.

Работа с окнами (текущими, открываемыми из сценария или фреймами в JavaScript организуется по одной схеме.

К объектам **текущего окна** обращаются, используя **window** или **self**, причем в этом случае указание окна можно опускать, т.е., равносильны три варианта обращения к свойству **bgcolor** (фоновый цвет) окна **Window:**

```
window.bgcolor   self.bgcolor   bgcolor
```

Замечание. Это свойство приводится исключительно для примера. Использование его в реальных сценариях неактуально.

К фреймам обращаются по их именам, заданным в HTML-описании значениям атрибута **name**. Если определен набор вложенных фреймов, то могут быть использованы обращения к родительскому фрейму как к объекту **parent**. Объект **top** ссылается на верхний уровень набора фреймов –текущее окно.

Свойства объекта Window

Для объекта **Window** определен ряд свойств, наиболее полезные и употребительные из которых сами являются объектами. На схеме показаны лишь высшие уровни иерархии свойств объектного типа. Заметим, что объект **Document**, являясь свойством объекта **Window**, сам обладает свойствами объектного типа. В свою очередь, ряд его свойств также представляют собой объекты и т.д.

Есть у объекта **Window** и скалярные свойства, наиболее полезные из которых приведем:

- **name** – строка, содержащая имя окна.
- **status** – текст строки состояния; **defaultStatus** - текст строки состояния по умолчанию.
- **opener** – ссылка на объект window, из сценария которого было открыто текущее окно.
- **parent** – ссылка на объект window, содержащий текущее окно или фрейм.
- **closed** – равно true, если открытое сценарием окно было закрыто, и false в противном случае.

Основные методы объекта Window

- **open()** – открывает новое окно (см. в отдельном разделе).
- **close()** – закрывает окно.
- **resizeTo(w,h)** –изменяет размеры окна до ширины **w** и высоты **h**.
- **resizeBy(x,y)** – изменяет размеры окна на **x** пикселов по ширине и на **y** пикселов по высоте.
- **moveTo(x0,y0)** – перемещает окно так, чтобы координаты его левого верхнего угла приняли значения **x0** и **y0**.
- **moveBy(x,y)** – перемещает окно на x пикселов по горизонтали и y пикселов по вертикали.
- **focus()** – передает окну фокус; при этом окно перемещается на передний план поверх других

Метод open()

Метод **`window.open()`** вызывается с такими аргументами:

`window.open( адрес файла, имя окна, параметры )` где

• *адрес файла* - адрес файла, первоначально загружаемого в новое окно;

• *имя окна* – условное имя, смысл которого будет пояснен ниже.

• *параметры* - набор элементов нового окна браузера - строка, в которой через запятую в любом порядке перечисляются пары параметр=значение.

Если какой-то параметр не указан, будет применено значение по умолчанию.

Параметр	значение	описание
`width`	размер в пикселах	ширина нового окна
`height`	размер в пикселах	высота нового окна
`left`	размер в пикселах	абсцисса левого верхнего угла нового окна
`top`	размер в пикселах	ордината левого верхнего угла нового окна
`toolbar`	1/ 0 / yes / no	вывод панели инструментов
`location`	1/ 0 / yes / no	вывод адресной строки
`directories`	1/0/yes /no	вывод панели ссылок
`menubar`	1/0/yes /	no вывод строки меню
`scrollbars`	1/0/yes /no	вывод полос прокрутки
`resizable`	1/0/yes /no	возможность изменения размеров окна
`status`	1/0/yes /no	вывод строки статуса
`fullscreen`	1/0/yes /no	вывод на полный экран

Например, при выполнении инструкции

```
window.open('test.htm','new', 'width=300, height=200, toolbar=1')
```

откроется окно 300x200, в котором из стандартных интерфейсных элементов будет только панель инструментов. Поскольку не указаны координаты верхнего левого угла, расположение окна зависит от умолчаний браузера.

Поясним смысл 2-го аргумента (условное имя окна) метода **`open()`**. При последовательном применении метода **`open()`** с одинаковым значением 2-го аргумента новые окна не открываются, а очередной целевой документ загружается в ранее открытое окно с тем же именем. Если же указывается условное имя окна, ранее не задававшееся, открывается новое окно с указанными параметрами.

Замечание. Если вызов метода **`open()`** загружает новый документ в уже открытое окно, это окно **не** становится активным. То есть, если оно закрыто окнами других приложений, то **не** «всплывет» поверх других. Пользователь не заметит, что в это окно загрузился новый документ. Для того, чтобы активизировать окно с вновь загруженным документом, следует применить метод **`focus()`**.

Прочие методы

В предыдущем разделе мы применили вызов метода **open()** в процедурном формате:

```
window.open(список аргументов)
```

С точки зрения пользователя при этом открывается новое окно с указанными параметрами, и в него загружается целевой документ. Для программиста же важно, что вызов этого метода **создает новый объект Window**. Однако, пока непонятно, каким образом можно реализовать доступ к этому новому объекту из сценария.

Проблема решается следующим образом. Метод **open()** **возвращает значение** объектного типа. Таким образом, для того, чтобы иметь возможность дальнейшей работы с окном, открытым сценарием, следует сохранить возвращаемое методом **window.open()** значение в некоторой переменной. Обращение к этой переменной и реализует доступ к методам и свойствам нового окна. Именно к ней можно применять перечисленные выше методы: **close()** **resizeTo(w,h)** **resizeTo(x,y)** **moveTo(x0,y0)** **moveBy(x,y)** **focus()**

Итак, присвоим возвращаемое методом **window.open()** значение некоторой переменной **newWin:**

```
newWin = window.open('test.htm','new','width=300,height=200');
```

Теперь это окно можно закрыть, переместить, изменить его размеры из сценария, используя соответствующие методы.

Приведем примеры обращения к переменной **newWin**

Пример 1. **Закрытие окна.**

Открытое из сценария окно можно закрыть, применив к объекту **newWin** метод **close()**. Перед выполнением этого метода надо убедиться в том, что окно открыто, например так:

```
if (!newWin)
{ alert('Окно newWin еще не открывалось') }
      else if ( newWin.closed)
{ alert('Окно newWin уже закрыто') }
      else {
  newWin.close() };
```

Пример 2. **Изменение параметров окна.**

Обращаясь к переменной **newWin**, можно получать и изменять свойства нового окна. При этом надо убедиться в том, что окно открыто. Например, изменим текст строки статуса и цвет фона документа с помощью операторов:

```
if (!newWin)
 { alert('Окно newWin еще не открывалось') }
else if ( newWin.closed)
      { alert('Окно newWin уже закрыто')}
else {newWin.status = 'Свойства окна изменены';
       newWin.document.bgColor ='#CCCCCC';      };
```

Пример 3. **Открытие нового окна в центре экрана.**

Необходимость открыть окно заданных размеров – очень распространенная ситуация. Например, надо показать иллюстрацию в увеличенном масштабе. Актуальный пример – фотоархив. Одно из распространенных (и вполне технологичных) решений состоит в выводе заранее заготовленных миниатюр, щелчком по которым в отдельном окне показывается полномасштабное изображение. При этом, возможны разные варианты:

- Все изображения одинакового размера (точнее, при предварительной обработке приведены к одинаковому размеру). В этом случае миниатюры также одинакового размера.
- Изображения разного размера и разных пропорций. Но миниатюры представляют собой вырезанные и масштабированные фрагменты одинакового размера.
- Изображения разного размера и разных пропорций. Миниатюры также имеют разные пропорции, но масштабированы таким образом, чтобы вписаться в квадрат заданных размеров. Именно такой вариант реализован в большинстве программ просмотра графических объектов. Так организован и вывод миниатюр в MS Windows в режиме «эскизы страниц».

Не рассматривая здесь вопросы подготовки миниатюр и задания параметров их отображения с помощью CSS, приведем лишь один из вариантов функции, открывающей окно заданных размеров по центру экрана монитора и загружающей в это окно документ с указанного адреса. При этом предположим, что в новом окне будут открываться изображения разного размера.

Планируя последовательные открытия изображений, согласимся, что новые окна не должны множиться. Значит, все они (см. раздел «Метод open()») должны иметь одно условное имя, задаваемое в качестве 2-го аргумента метода `open()`. Как отмечалось выше, в таком случае необходимо после загрузки нового содержимого активизировать окно, чтобы оно переместилось на передний план поверх других окон приложений. Здесь можно применить по крайней мере два подхода:

- Передать фокус ранее открытому окну и изменить его размеры и положение, чтобы при новых размерах окно снова расположилось по центру экрана.
- Закрыть окно с изображением, если оно уже было ранее открыто, а потом открыть новое окно нужного размера.

Более рациональным представляется второй вариант.

```
// Объявляем глобальную переменную newWin без инициализации. При вызове функции openWin() ей будет присвоено значение объектного типа, ссылающееся на новое окно.
var newWin;
function openWin(addr,w,h)
// Открывает окно с условным именем ' new' шириной w  высотой h и загружает в него документ с адресом addr.  Окно располагается по центру свободной части экрана.
// Закрываем окно, если оно ранее было открыто:
if (newWin) newWin.close();
// Определяем координаты левого верхнего угла окна, используя свойства объекта Screen:
var x = (screen.availWidth-w)*0.5;
var y = (screen.availHeight-h)*0.5;
// Открываем новое окно с указанным документом:
newWin = window.open(addr,'new','width='+w+',height='+h+',left='
+x+',top='+y);     };
```

Объекты браузера

- Объект **Screen** позволяет узнать (*но не изменить!*) разрешение клиентского экрана и глубину цвета. Свойства доступны только для чтения (**ReadOnly**).
  Определив разрешение, можно предусмотреть разные варианты компоновки страницы, устанавливать размеры и положение окон, открывающихся из сценария.
  Методы для этого объекта не определены, но определен ряд свойств. Наиболее полезные:
  width и **height** - ширина и высота экрана в пикселах;
  availWidth и **availHeight** - доступная ширина и высота экрана.
- Объект **Navigator** дает информацию о версии браузера.
  Ее можно использовать при создании «браузеронезависимого» сценария.
  Однако, зачастую более удобен другой подход, описанный далее
- Объект **Location** дает доступ к URL документа, отображаемого в окне браузера, позволяет определить полный URL, а также его части: протокол, доменное имя…
  В отличие от двух предыдущих объектов, его свойства доступны не только для чтения (**ReadOnly**), но и для изменения. В зависимости от выполнения условий, определенных в сценарии, можно загрузить нужный документ как в текущее окно или его фрейм, так и в любое из окон, открытых из сценария. Этот объект имеет два метода:
  reload() перезагружает указанный в качестве аргумента документ;
  replace() загружает указанный документ, который замещает текущий в списке истории просмотра.
- Объект **History** имеет одно свойство **length** (количество просмотренных в данном сеансе документов), и три метода, позволяющих перемещаться по истории:
  back() на один шаг назад по истории просмотра;
  forward() на один шаг вперед по истории просмотра;
  go(n) на n шагов по истории просмотра (если n>0, то вперед, при n<0 назад).

Пример: Возврат на предыдущую страницу

```
<a href="javascript:history.back()">назад</a>
```

Определение возможностей браузера

Разные браузеры отличаются степенью поддержки JavaScript и возможностей DOM.

Пример

Для идентификации элементов применяется метод **getElementById():**

```
if (document.getElementById)
      { описание инструкций, использующих метод getElementById() }
else  {альтернативный вариант };
```

Именно факт поддержки тех или иных свойств и методов, связанных с моделями DOM0 и DOM2, может служить удобным методом определения типа браузера и его версии. Например, глобальным переменным **isMozilla, isIE6** и **isOpera** присваиваются булевы значения, которые потом удобно будет использовать в сценарии в контексте условных инструкций.

Пример 2

```
var isMozilla =  (document.getElementById && !document.all);
var isIE6 =(document.getElementById && document.all);
var w= navigator.userAgent.toLowerCase();
var opera=( w.indexOf('opera')!=-1 );
```

Объект Document

Объект **Document** особенно важен при разработке **динамичеких страниц**.

Приведенная здесь схема иерархии объектов включает только основные свойства этого объекта, определенные в базовой объектной модели документа (*DOM0*).

Более поздние стандарты DOM1 и DOM2 представляют документ HTML в виде дерева и дают доступ ко всем элементам, существенно расширяя возможности разработчика.

Свойства объекта Document

Начнем со *скалярных* свойств, общих для всех браузеров.

Большинство их доступны как для чтения, так и для изменения. Отметим, что значения свойств, связанных с цветами текста, фона и гиперссылок, можно изменять динамически лишь в тех случаях, когда не заданы соответствующие описания CSS, которые имеют больший приоритет. Все значения свойств -строковые.

- **title -** текст заголовка документа (содержимое элемента title);
- **fgColor** и **bgColor** - цвет текста и цвет фона документа;
- **linkColor, vLinkColor, aLinkColor** - цвета непосещенных, посещенных и активных гиперссылок;
- l**astModified** (только для чтения) - дата изменения документа;
- **referrer** (только для чтения) - URL документа, ссылка в котором привела к загрузке текущего;
- **URL** (и устаревшее **location**) -URL документа.

Более интересны и полезны для разработчика *свойства-объекты* (*свойства-массивы*) объекта **Document**. Все они, естественно, имеют свойство **length** (количество элементов в массиве). Большинство свойств, специфичных для объектов, хранящихся в этих массивах, ассоциируются с атрибутами соответствующих элементов HTML. Вот лишь некоторые из них:

- объект **Form** (форма) имеет свойства **name, action, method**;
- объект **Anchor** (закладка) имеет единственное свойство **name**;
- объект **Link** (ссылка) имеет свойства **href, target**;
- объект **Image** (изображение) имеет свойства **src, width, height**.

Коллекция (совокупность элементов) **-- это структура**, похожая на массив.

Элементы коллекции являются объектами и доступны как элементы массива по индексу. Индексация начинается с нуля. В качестве индекса можно использовать **имя объекта**, если оно задано в соответствующем теге при помощи атрибута **name** (или **id**).

К объектам документа, хранящимся в **коллекциях** (**all, images, forms и applets**, а также к любым элементам форм можно обращаться разными способами.

Пусть, например, в документе описано изображение

```
<img src="images/cat.jpg" id="cat_id" name="cat_name">
```

и оно является **k**-ым изображением, встречающимся в документе. К этому элементу **img** можно обратиться несколькими способами (см. раздел «Идентификация элементов документа»):

- Как к элементу массива **images**, используя его индекс(индексация начинается с **0**):
  document.images[k-1]
- Как к элементу массива **images**, используя значение атрибута **name** как ключ массива:
  document.images['cat_name']
- Используя значение атрибута **name** как свойство объекта :
  document.cat_name
- Используя значение атрибута **id** и свойство **getElementById**:
  document.getElementById('cat_id')

Методы объекта Document

- **open()** - открывает новый документ; при этом все его содержимое удаляется.
- **close()** - закрывает ранее открытый документ.
- **write()** - записывает в документ заданную в качестве аргумента строку.
- **writeln()** - аналогичен предыдущему, но выведенная в документ строка заканчивается символом перевода строки.

Методы **write()** и **writeln()** весьма полезны и используются для динамического формирования содержимого документа.

Пример: включение в документ даты его последнего изменения:

```
<script type="text/JavaScript"> document.write(document.lastModified); </script>
```

Коллекция Images

Свойства объекта Images

Для работы с изображениями используется объект **Images**, вложенный в объект **Document**. Каждому HTML-элементу **img**, встречающемуся в документе, сопоставляется элемент массива **Images** со свойствами **src, width, height** и так далее (названия свойств соответствуют атрибутам элемента **img**). Например, решение задачи смены изображения сводится к изменению значения свойства **src** элемента массива **Images**. Получить доступ к свойствам изображений можно несколькими способами.

Пусть, например, в документе описано изображение

```
<img src="images/cat.jpg" id="cat_id" name="cat_name">
```

и оно является k-ым изображением, встречающимся в документе. Тогда следующие конструкции, ссылающиеся на свойство **src** изображения равносильны:

- Обращаемся к элементу массива **images**, используя его индекс (индексация начинается с0): **window.document.images[k-1].src** Этот вариант может быть удобен, когда надо работать с несколькими последовательно расположенными в документе изображениями.
- Используем значение атрибута **id**: **window.document.cat_id.src**
- Используем значение атрибута **name**: **window.document.cat_name.src**

Смена изображений

Итак, смена изображения сводится к изменению значения свойства **src** определенного элемента массива **images**. Обращаться к этому свойству можно любым из описанных ранее способов. Например, используем атрибут name элемента **img**.

Сценарий может работать в связи с любым из доступных в языке событий мыши. Атрибуты **onMouseOver, onMouseOut, onClick** и т.д. в HTML 4.0 и выше могут применяться с большинством элементов. При желании выполнить скрипт, например, при наведении курсора мыши на изображение, применима конструкция вида

```
<img src="cat.gif" name="pic" onMouseOver = "…инструкции…">
```

Чаще всего встречается два варианта смены изображений.

1. Изображение меняется при наведении на него курсора мыши.

Пусть в документе есть изображение **ris1.jpg** и при наведении курсора мыши требуется его сменять изображением **ris2.jpg**. Опишем функцию **changeImg()**:

```
function changeImg(ris)
          {   document.pict.src = ris + '.jpg';  };
```

Применим ее при наступлении событий **onMouseOver** и **onMouseOut**:

```
<img src="ris1.jpg" name="pict" border=0
 onMouseOver="changeImg('ris2')" onMouseOut="changeImg('ris1')">
```

2. Изображение меняется при наведении курсора на другие элементы документа

Пусть в документе есть изображение:
`<img src="cat.jpg" name="pict1" border=0>` которое надо сменять в зависимости от того, над какой гиперссылкой находится курсор мыши.

Как и раньше, опишем функцию `changeImg1()`

```
function changeImg1(source)
          {   document.pict1.src = source + '.jpg';  };
```

В элементе `a` каждой гиперссылки наряду с адресом целевого документа указываем атрибуты `onMouseOver` и `onMouseOut`.

Получится описание примерно следующего вида:

```
<a href="адрес " onMouseOver="changeImg1('dog')"
                 onMouseOut="changeImg1('cat')" >
```

3. Предварительная подгрузка изображений

Для того, чтобы эффект смены изображений проявился сразу после загрузки страницы, надо подгрузить изображения, которые будут сменять исходные в ответ на события мыши, написав в начале документа соответствующий инструкции. Для предыдущего примера можно сделать так:

```
pic1 = new Image(); pic1.src = 'dog1.jpg';
pic2 = new Image(); pic2.src = 'dog2.jpg';
pic3 = new Image(); pic3.src = 'dog3.jpg';
```

Изображения будут подгружены заранее и в нужный момент взяты из кэша.

4. Еще о смене изображений

В любом сценарии, в частности и при организации смены изображений, надо стараться сделать код рациональным. Вот очень простой, но вполне эффектный пример: при наведении курсора мыши на миниатюры меняется увеличенное изображение. Файлы изображений находятся в папке `Images` и называются `1.jpg, 2.jpg` и `3.jpg`. Файлы миниатюр называются так же и находятся во вложенной папке (подкаталоге) `Images/Thumbs`.

Понятно, что надо выполнить предварительную подгрузку и написать функцию, реализующую смену изображений. В начале сценария определим переменную: `imageDir = Images/'` и рассмотрим три варианта.

1 вариант

Функция:

```
function changeImg(imgAddr)
          { document.bigImg.src = imgAddr }
```

Вызов функции (например, для 1-ой миниатюры):

```
<img src="Images/Thumbs/1.jpg" width="50" height="50"
          onMouseOver="changeImg('Images/1.jpg')">
```

2 вариант

Пользуемся тем, что имена файлов изображений представляют последовательность `1.jpg`, `2.jpg` . . . и находятся в каталоге `imageDir`. Функция слегка изменилась:

```
function changeImg(imgAddr)
     {document.bigImg.src = imageDir + imgAddr + '.jpg' }
```

Вызов функции получился короче и прозрачнее:

```
<img src="Images/Thumbs/1.jpg" width="50" height="50"
          onMouseOver="changeImg(1)">
```

3 вариант

А если к тому же динамически сформировать последовательности миниатюр (ведь их могло бы быть много), получится совсем компактно. Пользуясь тем, что имена файлов изображений представляют последовательность **1.jpg , 2.jpg**, выведем изображения в цикле, используя метод **write()** объекта **document**.

```
for (i=1; i<=3; i++)
    document.write('<img src="Images/Thumbs/' + i + '.jpg"
    width="50" height="50" '+ ' onMouseOver="changeImg(' + i + ')">');
```

Формы и элементы форм

Задание форм и их элементов в HTML

Элементы форм знакомы всем пользователям глобальной сети. Это поля ввода текста и пароля, стандартные кнопки, переключатели, «флажки», выпадающие списки и т.д. Наиболее очевидные варианты применения: ввод ключевых слов в поисковых системах, работа с электронной почтой через веб-интерфейс, регистрация на сайте, веб-анкеты, online-тесты.

Использование форм предполагает **интерактивность** и, следовательно, обеспечение функциональности требует программирования (клиентского или серверного). В большинстве случаев формы используются для передачи данных на сервер, однако и на стороне клиента есть задачи, в которых удобно применение форм (например, календарь или калькулятор).

Роль HTML состоит в описании необходимых элементов и компоновке их на странице. Все **атрибуты** форм и их элементов, описанных с помощью HTML, **транслируются** в соответствующие **свойства объектов** DOM, которые используются в сценариях JavaScript.

Формы

Описания управляющих элементов форм, призванных обеспечивать необходимую функциональность, должны быть расположены в контейнере **form**.

Контейнер form

- **name** имя формы; используется в клиентских и серверных сценариях для идентификации вложенных управляющих элементов
- **action** адрес файла серверного сценария, который будет обрабатывать заполненную и переданную форму
- **method** метод передачи данных серверу (по умолчанию **get**)
- **enctype** тип содержимого, используемый для отправки формы на сервер (по умолчанию **application/x-www-form-urlencoded**)
- **acceptcharset** список кодировок символов ввода данных, которые будут обрабатываться сервером
- **target** имя окна или фрейма для загрузки документа, сгенерированного сценарием на основании принятых из формы данных (по умолчанию **_self**, т.е. результат обработки формы загружается в то же окно или фрейм)
- **onSubmit** подтверждение отправки данных формы на сервер
- **onReset** сброс значений элементов формы в значения по умолчанию

Заметим, что большинство перечисленных атрибутов актуальны при обработке данных формы серверным сценарием. Три атрибута имеют значения по умолчанию.

Все перечисленные атрибуты формально необязательны, однако

- • при работе на стороне клиента необходимо задавать имя формы **name**, чтобы иметь возможность обращаться к элементам формы из сценария JavaScript;
- • отправка данных формы на сервер требует, как минимум, указания атрибута **action**, определяющего серверный сценарий обработки.

Атрибут method

Атрибут **method** имеет два основных возможных значения: **get** (по умолчанию) и **post**.

При передаче данных методом **get** данные формы отправляются на сервер в заголовке запроса, а при использовании метода **post** – в теле запроса. Передача текстовых данных может осуществляться любым из этих методов. А вот бинарные данные могут быть отправлены только методом **post**. Это происходит в случае загрузки файла не сервер (всем понятный пример – приложение к электронному письму). В этом случае необходимо указать **enctype="multipart/form-data"** (см. далее примечения к атрибуту **enctype**).

Итак, за исключением случая передачи бинарных данных (только методом **post**) равным образом можно применять оба возможных метода. На сложность разработки серверного сценария, принимающего данные из формы, это никоим образом не влияет. Какой же метод предпочтителен?

В подавляющем большинстве случаев применяется метод **get**. При передаче методом **get** пользователь видит данные формы в адресной строке. Символы, отличные от стандартной латиницы кодируются. Например, пробелу соответствует шестнадцатиричный код **%20** . Каждый наблюдает такую ситуацию, работая с поисковыми системами. Наберем, например, в Яndex словосочетание «***язык HTML***» и увидим в адресной строке: слово «***язык***» закодировано последовательностью **% FF%E7%FB%EA**, а «HTML» передается как есть. Получив такой запрос, соответствующая серверная программа обратится к базе данных, сгенерирует и отправит браузеру список результатов поиска.

При передаче методом **post** данные формы отправляются на сервер "как есть". При этом пользователь не видит в адресной строке ничего «лишнего». Зато появляются другие проблемы: попытка обновить страницу вызывает маловразумительное сообщение: «… обновление страницы невозможно без повторной отправки данных…».

Резюме. По-видимому, следует выбирать заданный по умолчанию метод `get` во всех случаях, кроме передачи на сервер бинарных данных.

Атрибут enctype

Этот атрибут имеет два возможных значения:

- **application/x-www-form-urlencoded (по умолчанию)**
- **multipart/form-data**

Первое значение используется в абсолютном большинстве случаев. Нет смысла указывать его явно – оно предполагается по умолчанию.

Второй тип кодировки (**multipart/form-data**) указывается в единственном случае - при загрузке на сервер бинарного файла. При этом обязательно задание атрибута **method="post"**.

Элементы форм в HTML

Элементы форм можно условно разбить на несколько групп. Все элементы имеют различную функциональность и разный внешний вид. У каждого из них есть определенное количество атрибутов (которые как раз и обеспечивают функциональность и, в меньшей степени, параметры внешнего отображения). Отметим лишь самые необходимые атрибуты, чтобы хотя бы увидеть, как выглядят те или иные элементы форм. При работе с формами, как на стороне клиента, так и с помощью серверных сценариев, очевидна необходимость указания и прочих атрибутов.

Особенности синтаксиса:

- Элементы **`input`** не имеют содержимого и, соответственно, при их задании не указывается закрывающий тег.
- Остальные элементы форм (**`textarea`**, **`select`**, **`option`**) являются контейнерами и имеют конечный тег. Для элемента **`option`** конечный тег может быть опущен.

Элементы input

Элементы **`input`**, в зависимости от значения атрибута **`type`**, не только выглядят, но и функционируют принципиально по-разному. Большинство из них могут быть равным образом использованы как в клиентских, так и в серверных сценариях. Представим их в виде следующих групп.

Кнопки

`<input type="submit">` Отправляет данные из формы на сервер (предполагается, что серверная программа их сможет принять и правильно обработать).

`<input type="reset">` Очищает форму, восстанавливая значения по умолчанию.

`<input type="button">` Выглядит как системная кнопка, но нажатие на нее всего лишь запускает клиентский сценарий JavaScript (этого же эффекта можно достичь использованием обработчика **`onClick`** для любого элемента HTML-документа).

`<input type="image" src="адрес графического файла">` Как и кнопка типа **`submit`**, отправляет данные из формы на сервер, но выглядит нестандартно: вид задается изображением, адрес которого должен быть указан как значение атрибута **`src`**.

Текстовые поля

`<input type="text">` Однострочное поле ввода текста.

`<input type="password">` Поле ввода пароля.

Переключатели

`<input type="checkbox">` Флажок (включатель-выключатель).

Флажки могут быть отмечены независимо друг от друга.

`<input type="radio">` Радиокнопка (переключатель с зависимой фиксацией).

Имеет смысл говорить не об одной радиокнопке, а о группе радиокнопок.

Групп в форме может быть несколько, но в каждой группе в текущий момент только одна из кнопок может быть выбрана.

Поле выбора файла

`<input type="file">` Позволяет выбрать файл для загрузки на сервер.

Скрытое поле

`<input type="hidden">` Позволяет передавать данные, не введенные пользователем, а полученные в результате работы сценария.

Многострочное текстовое поле textarea

```
<textarea>   ...   текст по умолчанию ...   </textarea>
```

Выпадающий список select и его элементы option

```
<select>
      <option>Москва</option>     <option>Петербург</option>
      <option>Минск</option>      <option>Киев</option>
<select>
```

Атрибуты элементов форм

Как и подавляющее большинство элементов HTML, все элементы форм имеют общие атрибуты **id, style, class, title.**

Атрибут **type** элемента **input** обсуждался в предыдущем разделе.

Перечислим прочие атрибуты, начиная с тех, которые применимы со всеми элементами форм, и заканчивая специфическими для отдельных элементов.

Со всеми элементами форм можно указывать атрибуты

name, disabled, tabindex, accesskey.

- **name** – условное имя элемента формы. Используется клиентскими и серверными сценариями для обращения к элементу.
- **disabled** – задание этого атрибута делает элемент временно недоступным. При необходимости это значение можно изменить из сценария.
- **tabindex** – порядок в последовательности перехода по элементам с помощью клавиши табуляции.
- **accesskey** – клавиша доступа к элементу.

Примечание Атрибут **name** служит для идентификации элементов форм при обращении к ним из сценария. Следовательно, значение этого атрибута должно быть уникальным для каждого элемента. Для элемента **input** типа **radio** уникальным должно быть **имя группы** зависимых радиокнопок. Именно единое имя группы и позволяет браузеру трактовать радиокнопки как зависимые.

Атрибуты элементов input разных типов

value – значение. Атрибут обязателен для радиокнопок и флажков!.

Смысл его для разных элементов форм различен.

- Для текстовых элементов это текст по умолчанию (если не указан, текстовое поле пусто).
- Для кнопок значение **value** задает надпись на кнопке.
- Заданное для поля выбора файла значение **value** игнорируется современными браузерами из соображений безопасности. Так что, фактически, можно считать, что элемент **<input type="file">** не имеет атрибута **value**.

size – для однострочного текстового поля и поля ввода пароля задает ширину в символах.

maxLength – для однострочного текстового поля и поля ввода пароля задает максимальное количество вводимых символов.

readonly – указание этого атрибута для однострочного текстового поля и поля ввода пароля делает текст временно недоступным для изменения. При необходимости это значение можно изменить из сценария.

checked – указание этого атрибута для флажка или радиокнопки делает соответствующий элемент выбранным по умолчанию.

Атрибуты элемента textarea

Помимо перечисленных общих атрибутов, элемент **textarea** имеет еще два, задающих его размеры: **rows** и **cols** – ширина и высота текстового поля в символах.

Атрибуты элемента select

Элемент **select** служит контейнером для элементов списка **option**. Помимо перечисленных общих атрибутов, у этого элемента есть следующие атрибуты:

size – количество одновременно видимых элементов при нераскрытом состоянии списка (по умолчанию 1).

multiple – указание этого атрибута разрешает множественный выбор (по умолчанию выбирается единственный элемент).

Атрибуты элемента option

Элемент **option** определяет элемент списка. Наличие атрибута **selected** делает соответствующий элемент списка выбранным по умолчанию.

Работа с формами в JavaScript

В JavaScript определен ряд свойств и методов для работы с формами и ее элементами. Кроме того, предусмотрены специфические обработчики событий, обеспечивающие функциональность сценария. Все значения свойств имеют **строковый** тип.

Как и в общем случае трансляции атрибутов элемента HTML в свойства объекта модели DOM, HTML-**атрибутам** форм и их элементов сопоставлены соответствующие **свойства объектов** JavaScript. Кроме того, предусмотрены и другие свойства, полезные для применения в сценариях.

Свойства и методы формы в JavaScript

В JavaScript определены следующие **свойства**, часть из которых представляет собой **трансляцию атрибутов** HTML-элемента **form**.

Свой+ство	атрибут HTML	Описание
Name	**name**	Имя формы
action	**action**	Адрес файла серверного сценария, который будет обрабатывать заполненную и переданную форму
method	**method**	Метод передачи данных серверу
encoding	**enctype**	Тип содержимого, используемый для отправки формы
target	**target**	Имя окна или фрейма для загрузки документа, сгенерированного сценарием на основании принятых из формы данных
length	**-**	Количество элементов формы
elements[]	**-**	Массив элементов формы

Определены **два метода**, эмулирующие нажатие на кнопки типа **submit** и **reset**: **submit()** **reset()**

Свойства элементов форм в JavaScript

Для элементов форм в JavaScript определен ряд свойств, часть из которых представляет собой трансляцию атрибутов соответствующих HTML элементов.

Общие свойства

Для всех типов элементов форм определены следующие свойства.:

Свойство	атрибут HTML	Описание
`name`	`name`	Имя элемента формы
`value`	`value`	Строка, определяющее значение, отображаемое элементом и/или используемая серверным сценарием, обрабатывающим полученные их формы данные. Это свойство задано для всех элементов форм, кроме `select`
`disabled`	`disabled`	Булево значение, определяющее доступность элемента в текущем контексте
`form`	`-`	Имя формы

Для отдельных элементов форм в дополнение к общим свойствам определены **частные**:

Элементы input

Свойство	атрибут HTML	тип элемента **input**	Описание
`Type`	`type`	все	Тип элемента `input`
`DefaultValue`	`-`	text, password	Текст, первоначально отображающийся в элементе формы
`Checked`	`-`	radio, checkbox	Булево значение, определяющее отмечен ли в данный момент элемент формы
`DefaultChecked`	`checked`	radio, checkbox	Булево значение, определяющее отмечен ли элемент формы по умолчанию

Элемент textarea

`defaultValue` – текст, первоначально отображающийся в многострочном текстовом поле.

Элемент select

`length` - количество элементов списка

`options[]` - Массив элементов списка

`selectedIndex` - Номер выбранного элемента списка

Элемент option

`text` - Строка, задающая текст элемента списка

`selected` -Булево значение, определяющее выбран ли в данный момент элемент списка

`defaultSelected` - Булево значение, определяющее выбраный элемент списка по умолчанию

Идентификации элементов форм

Существенным моментом программирования форм является идентификация их элементов.

Массив (коллекция) форм **forms[]** вложен в объект **document**; элементы форм являются вложенными объектами самой формы; элементы списка являются вложенными объектами объекта **select**.

Напомним, что объекты JavaScript по сути представляют собой ассоциативные массивы со строковыми ключами – именами свойств.

Рассмотрим пример обращения к элементам формы из сценария. Пример условный, в нем представлены три разных элемента, обращение к которым принципиально различно.

По типу идентификации элементы можно разделить на три группы:

• радиокнопки; • элементы списка; • прочие элементы форм.

Нетрудно догадаться о причинах именно такого разделения. В отличие от прочих элементов форм, группы радиокнопок и элементы списка представляют собой массивы. А массив элементов списка к тому же является вложенным по отношению к объекту **select**.

Пример. Пусть в документе задана следующая форма, например, фрагмент online-анкеты).

```
<form name="f">
Ваше имя:  <input type="text" name="yourName">
Ваш пол: <input type="radio" name="sex"value="male">мужской<br>
         <input type="radio" name="sex" value="female">женский<p>
Город:<br>  <select name="town">
     <option value="msk">Москва</option>
     <option value="mn">Минск</option>
     <option value="other">другой</option>
     </select>
</form>
```

Во всех случаях при идентификации элементов форм используются имена (значения атрибута **name**) заданных элементов HTML. По имени идентифицируется и сама форма; для нашей формы это обращение **document.f**

Рассмотрим варианты идентификации элементов.

Текстовое поле yourName.

Варианты идентификации: **document.f.yourName** или **document.f['yourName']**

Группа из двух **радиокнопок** с именем **sex** представляет собой массив, в котором к отдельной кнопке с индексом **i** (индексация начинается с нуля) можно обращаться следующим образом: **document.f.sex[i]** или **document.f['sex'][i]**

Список с именем **town**. Его элементы представляет собой массив **options[]**. К элементу этого массива с индексом **i** можно обращаться следующим образом:

document.f.options[i] или **document.f['options'][i]**

Альтернативная возможность идентификации - **универсальная идентификация**.

Все элементы формы представлены массивом **elements[],** в котором содержатся в порядке их объявления в документе HTML.