JavaScript

http://shamansir.github.io/JavaScript-Garden/

ПЛАН ЛЕКЦИИ:

- Java-апплет
- Операции JS
- Базовые операторы зыка JS

JavaScript

Разработан компаниями

Sun Microsystems и Netscape

Java Апплеты

- Java-апплет — прикладная программа на Java в форме байт-кода.
- Java-апплеты выполняются в веб-браузере с использованием виртуальной Java машины (JVM), или в Sun's AppletViewer, автономном инструменте для тестирования апплетов.
- Java-апплеты были внедрены в первой версии языка Java в 1995.
- Java-апплеты обычно пишутся на языке программирования Java, но могут быть написаны и на других языках, которые компилируются в байт-код Java, таких, как Jython.
- Апплеты используются для предоставления интерактивных возможностей веб-приложений, которые не могут быть предоставлены HTML.
- Так как байт-код Java платформо-независим, то Java-апплеты могут выполняться с помощью плагинов браузерами многих платформ, включая Microsoft Windows, UNIX, Apple Mac OS и GNU/Linux.

JavaScript- интерпретатор с элементами объектно-ориентированной модели

- JS использует методы и свойства объектов и событий
- Иерархия наследования свойств объектов
- Сложность: JS встраивается в HTML документ и взаимодействует с ним

- Скрипты могут находиться в любом месте HTML-документа
- Однако теги HTML нельзя помещать внутри JS-программы
- JS программа помещается между тегами

<script> ... </script>

- Исключение составляют обработчики событий

Главная часть → контейнер

<head>... </head>

Скрипт → HTML – документа лучше перед контейнером → <body>... </body>

Синтаксис тега:

<script language="JavaScript">
[текст программы] </script>

Выражения языка JavaScript

- Выражение - это сочетание переменных, операторов и методов, возвращающее определенное значение.
- Условные выражения используются для сравнения одних переменных с другими, а также с константами или значениями, возвращаемыми различными выражениями.

Оператор сравнения ?

(условное выражение) ? операторы_1 : операторы_2

Присваивание значений переменным:

```
type_time = (hour >= 12) ? "PM" : "AM"
```

```
if (hour >= 12)
type_time="PM";
else
type_time="AM";
```

Операции присваивания

=	Прямое присваивание значения левому операнду
+=	Складывает значения левого и правого операндов и присваивает результат левому операнду
+	Складывает значения левого и правого операндов и присваивает результат левому операнду
++	Увеличивает значение левого операнда (правый может отсутствовать)
-=	Вычитает значения левого и правого операндов и присваивает результат левому операнду
-	Вычитает значения левого и правого операндов и присваивает результат левому операнду
--	Уменьшает значение левого операнда (правый может отсутствовать)
*	Умножает значения левого и правого операндов и присваивает результат левому операнду
*=	Умножает значения левого и правого операндов и присваивает результат левому операнду
/	Делит значения левого на правого операндов и присваивает результат левому операнду
/=	Делит значения левого на правого операндов и присваивает результат левому операнду

nval *=10;
вместо:
nval = nval * 10;

Операции сравнения

==	Равенство (равно)
!=	Не равно
!	Логическое отрицание
>=	Больше или равно
<=	Меньше или равно
>	Больше
<	Меньше (по возможности желательно воздержаться от применения этого типа)

if mvar <h . . . . . .bgcolor- может интерпретироваться как начало заголовка HTML

!Теги HTML в JS программах недопустимы!

Логические операции

И **&&**

ИЛИ **||**

Эти операции применимы только к булевым значениям

Например:

bvar1 = true;

bvar2 = false;

bvar1 || bvar2

bvar1 && bvar2 false

```
if ((bvar1 && bvar2) || bvar3) {
  function1();
}
    else {
   function2();
}
```

"Активизировать функцию function1(), если обе переменные bvar1 и bvar2 содержат значения true, или хотя бы bvar3 содержит true, иначе вызвать функцию function2 "

Базовые операторы зыка JS

- Каждый оператор, если он занимает единственную строку, имеет разграничивающую точку с запятой **(;)**, обозначающую окончание оператора.
- Каждый оператор имеет собственный синтаксис.
- Синтаксис оператора - это набор правил, определяющих обязательные и допустимые для использования в данном операторе значения.
- Значения, присутствие которых является необязательным, при описании синтаксиса принято заключать в квадратные скобки, например [value].

При несоблюдении правил синтаксиса произойдет ошибка компиляции.

Операторы комментариев и примечаний

```
// Текст комментариев
/* Текст
комментариев
*/
```

Первый комментарий может иметь только одну строку, второй несколько.

Комментарии нужны для пояснений или для временного исключения некоторых фрагментов программы во время отладки.

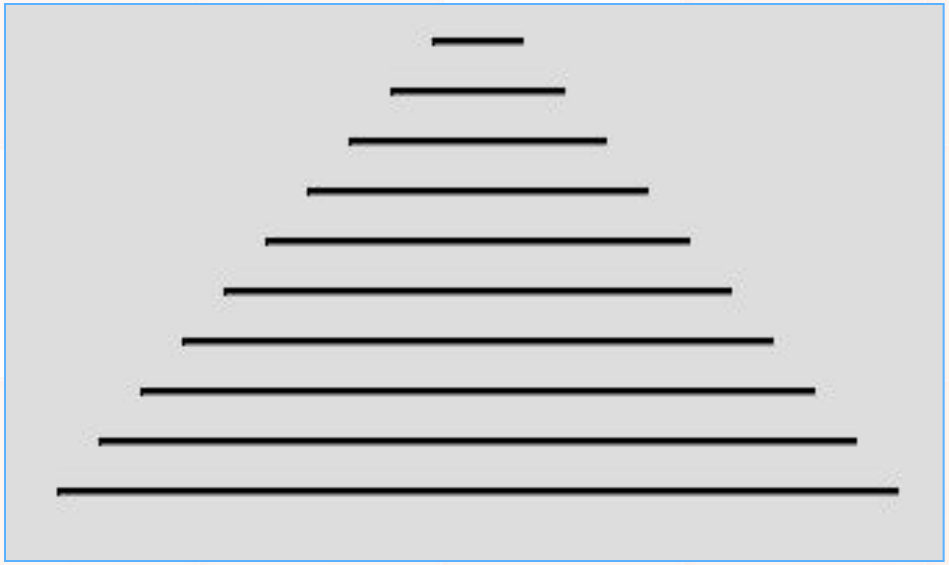
Операторы циклов

```
for ([инициализация начального значения;]
   [условие;] [механизм обновления счетчика,
   шаг]) {
программный блок
}
```

Оператор For позволяет многократно выполнять операторы в JS-программе. Оператор For может быть использован для выполнения одного или нескольких операторов. Фигурные скобки можно опустить, если тело цикла содержит только один оператор. Все параметры оператора For являются необязательными и используются для управления процессом выполнения цикла. При применении всех параметров каждую часть нужно отделять точкой с запятой (;).

Пример вывода в окне браузера горизонтальных линий

```
<html>
<head>
<script language ="JavaScript">
<!--
function testloop() {
var String1 = '<hr align="center" width="' ;
document.open();
for (var size = 5; size <= 50; size+=5)
document.writeln (String1+ size+'%">');
document.close();
}
//-->
</script>
</head>
<body>
<form>
<input type="button"
value="Test the loop"
onClick="testloop()">
</form>
</body>
</html>
```



Цикл while

```
while (условие) {
программный блок
}
```

При помощи оператора **while** можно выполнять один или несколько операторов до тех пор, пока не будет удовлетворено условие.

Если в теле цикла выполняется несколько операторов, их необходимо заключить в фигурные скобки.

Пример вывода таблицы умножения

```
<html>
<head>
<script language ="JavaScript">
function ftable(inum) {
   var iloop = 1;
   document.writeln ("ТАБЛИЦА УМНОЖЕНИЯ ДЛЯ: <b>" + inum + "</b><hr><pre>");
      /* в параметрах функции writeln применены теги HTML - это допустимо.
while (iloop <= 10) {
document.writeln(iloop + " x "+ inum + " = " + (iloop*inum));
iloop ++;
}
document.writeln("</pre>");
}
ftable(prompt ("Введите число: ", 10));
</script>
</head>
</html>
```

Теги HTML в тексте программы на JS недопустимы

Выход из цикла - оператор break

Оператор break используется для выхода из какого-либо цикла, например из цикла for или while.

Выполнение цикла прекращается в той точке, в которой размещен этот оператор, а управление передается следующему оператору, находящемуся непосредственно после цикла.

Пример применения оператора break

```
<html>
<script language ="JavaScript">
function btest() {
var index = 1;
while (index <= 10) {
if (index = = 6)
break;
index ++;
}
//После отработки оператора break управление
   переходит сюда.
}
btest();
</script>
</html>
```

Цикл while будет всегда завершаться после первых шести итераций, а значение переменной index никогда не достигнет 10-ти

Продолжение цикла - оператор continue

- Оператор continue используется для прерывания выполнения блока операторов, которые составляют тело цикла и продолжения цикла в следующей итерации. В отличие от оператора break, оператор continue не останавливает выполнение цикла, а наоборот запускает новую итерацию.
- Если в цикле while идет просто запуск новой итерации, то в циклах for запускает с обновленным шагом.

Определение функции

```
function functionname (arg, . . .) {
блок операторов
}
```

- **Функция - это блок из одного или нескольких операторов.**
- **Блок выполняет определенные действия, а затем, возможно, возвращает значение.**
- **В языке JS процедуры - подпрограммы не возвращающие значений, не различаются.**
- **Все подпрограммы описываются функциями, а если в функцию или из нее не передаются параметры - то после имени функции ставятся круглые скобки без параметров.**
- **Если функция имеет несколько аргументов, они отделяются запятой.**
- **В языке JS внутри одной функции не может существовать другой функции.**
- **Фигурные скобки определяют тело функции.**
- **Функция не может быть выполнена до тех пор, пока не будет явного обращения к ней.**
- **Если необходимо, чтобы функция возвращала определенное значение, следует использовать необязательный оператор return, при этом указав в нем выражение, значение которого требуется возвратить.**

Возврат значения функциями - оператор return

```
return (value);
return value;
```

Оператор return завершает выполнение функции и возвращает значение заданного выражения.

Скобки в этом операторе можно не использовать.

Оператор return может отсутствовать в функции, если функция не возвращает значение.

return для возврата массива

```
function retarray() {
var sarray = new Object();
sarray[1] = "Java";
sarray[2] = "Script";
return (sarray);
}
```

Обращение к аргументам функции при помощи массива argunents[ ]

```
function showargs() {
   arglist = "";
   for (var n=0; n <= arguments.length; n++)
{
        arglist += n +"." + arguments[n] + "\n";
}
alert(arglist);
}
```

Условные операторы - if . . . else

```
if (condition); {
  Программный блок1
} [ else { программный блок2 }]
```

Оператор **if . . . else** - это условный оператор, который обеспечивает выполнение одного или нескольких операторов, в зависимости от того, удовлетворяются ли условия.

Часть **condition** оператора **if** является выражением, при истинности которого выполняются операторы языка в первом программном блоке.

Программный блок должен быть заключен в **фигурные скобки**, однако если используется только один оператор, можно скобки не ставить.

Необязательная часть **else** обеспечивает выполнение операторов второго блока, в случае, если условие **condition** оператора **if** является ложным.

Пример. Смена цвета фона в зависимости от системного времени: первая половина часа пусть будет синим, вторая - черным:

```
<html>
<head>
<script language ="JavaScript">
<!--
today = new date();
minutes = today.getMinutes();
if (minutes >=0 && minutes <= 30)
   document.write("<body text=white bgcolor=blue> Это написано
    белым на синем");
     else
       document.write("<body text=red bgcolor=black> Это написано
    красным на черном");
//-->
</script>
</body>
</html>
```

Создание переменных

Переменные создаются либо при помощи **оператора var**,

либо при непосредственном присвоении значений с помощью **оператора присваивания (=).**

var variablename [= value | expression];

Оператор var создает новую переменную с именем **variablename**. Область действия этой переменной будет либо **локальной**, либо **глобальной** в зависимости от того, где создана переменная.

Переменная, созданная внутри функции будет недоступна за пределами функции, то есть переменная будет **локальной**.

Оператор this

this[.property]

- Оператор this является не столько оператором, сколько внутренним **свойством языка JavaScript**.
- Значение this представляет собой текущий объект, имеющий стандартные **свойства**, такие как **name, length и value**.
- Оператор this **нельзя использовать вне области действия** функции или вызова функции. Когда аргумент property опущен, с помощью оператора this передается текущий объект. Однако при обращении к объекту, как правило, **нужно указать его определенное свойство**.
- Оператор this применяется для "**устранения неоднозначности**" объекта с помощью привязки его в область действия текущего объекта, а также для того, чтобы сделать программу более компактной.

Оператор with

```
with (objname); {
    statements
}
```

Оператор **with** делает объект, обозначенный как **objname**, текущим объектом для операторов в программном блоке **statements**. Удобство использования этого оператора заключается в том, что такая запись позволяет **сократить объем текста программы**.

Оператор with применяется к встроенному объекту Math языка JS

```
with (Math) {
    document.writeln(PI);
}
```

Такая запись позволяет избежать использования префикса Math при обращении к константам данного объекта.

Оператор with применительно к объекту document

```
with (parent.frames [1].document) {
    writeln("Пишем сюда текст");
    write("<hr>");
}
```

В этом случае оператор **with** избавляет нас от необходимости указывать перед методами **writeln()** и **write()** документ, к которому относятся вызовы этих методов.

Вывод.

- В данной лекции были рассмотрены и использованы объекты, методы, свойства и обработчики событий

Объектная модель языка. Объекты браузера

При создании HTML-документов и JavaScript-программ необходимо учитывать структуру объектов. Все объекты можно разделить на три группы:

- Объекты браузера
- Внутренние, или встроенные, объекты языка JavaScript
- Объекты, связанные с тегами языка HTML
- Объектами браузера являются зависимые от браузера объекты: window (окно), location (местоположение) и history (история). Внутренние объекты включают простые типы данных, такие как строки (string), математические константы (math), дата (date) и другие.
- Объекты, связанные с тегами HTML, соответствуют тегам, которые формируют текущий документ. Они включают такие элементы как гиперсвязи и формы.

Методы объектов

- С объектами связаны методы, которые позволяют управлять этими объектами, а также в некоторых случаях менять их содержимое. Кроме того в языке JavaScript имеется возможность создавать свои методы объектов. При использовании метода объекта, нужно перед именем метода указать имя объекта к которому он принадлежит.
- Например, правильным обращением к методу document является выражение

document.write(),

а просто выражение write() приведет к ошибке.

Свойства объектов языка JavaScript

- **Свойство** - это именованное значение, которое принадлежит объекту. Все стандартные объекты языка JS имеют свойства. Например, в прошлой главе мы использовали в одном из примеров свойство **bgColor** объекта **document**. Данное свойство соответствует атрибуту **bgColo**r тега **<body>** - цвет фона документа

.

- Для обращения к свойству необходимо указать имена объекта и свойства, разделив их точкой.
- Каждый объект имеет собственный набор свойств. Набор свойств нового объекта можно задать при определении объекта.
- Однако, некоторые свойства объектов существуют только для чтения, и вы не можете их менять. В таких случаях можно получить только значения этих свойств. Как показывает практика, такие свойства изменять обычно без надобности и проблем в связи с этим не возникает.

Объекты браузеров

- HTML-объектами являются объекты, которые соответствуют тегам языка HTML:метки, гиперсвязи и элементы формы - **текстовые поля, кнопки, списки и др**.

- Объекты верхнего уровня, или объекты браузера, - это объекты, поддерживаемые в среде браузера: **window, location, history, document, navigator**.

Объекты, перечисленные в таблице, создаются автоматически при загрузке документа в браузер

Имя объекта	Описание
window	Объект верхнего уровня в иерархии объектов языка JavaScript. Фреймосодержащий документ также имеет объект window.
document	Содержит свойства, которые относятся к текущему HTML-документу, например имя каждой формы, цвета, используемые для отображения документа, и др. В языке JS большинству HTML-тегов соответствуют свойства объекта document.
location	Содержит свойства, описывающие местонахождение текущего документа, например адрес URL.
navigator	Содержит информацию о версии браузера. Свойства данного объекта обычно только для чтения. Например свойство: navigator.appname содержит строковое значение имени браузера.
history	Содержит информацию обо всех ресурсах, к которым пользователь обращался во время текущего сеанса работы с браузером.

Объект window

- Объект window обычно соответствует главному окну браузера и является объектом верхнего уровня в языке JavaScript, поскольку документы, собственно, и открываются в окне.
- В фреймосодержащих документах, объект window может не всегда соответствовать главному окну программы.
- Для обращения к конкретному окну следует использовать свойство frames объекта parent.
- Фреймы - это те же окна. Чтобы обратиться к ним в языке JavaScript, можно использовать массив frames.
- Например, выражение parent.frames[0] обращается к первому фрейму окна браузера. Предполагается, что такое окно существует, но при помощи метода window.open() можно открывать и другие окна и обращаться к ним посредством свойств объекта window.

Для обращения к методам и свойствам объекта window используют следующие варианты записи:

- window.propertyName
- window.methodName (parameters)
- self.propertyName
- self.methodName (parameters)
- top.propertyName
- top.methodName (parameters)
- parent.propertyName
- parent.methodName (parameters)
- windowVar.propertyName
- windowVar.methodName (parameters)
- propertyName
- methodName (parameters)

Свойства

Объект window имеет свойства:

- defaultStatus - текстовое сообщение, которое по умолчанию выводится в строке состояния (status bar) окна браузера.
- frames - массив фреймов во фреймосодержащем документе.
- length - количество фреймов во фреймосодержащем документе.
- name - заголовок окна, который задается с помощью аргумента windowName метода open().
- parent - синоним, используемый для обращения к родительскому окну.
- self - синоним, используемый для обращения к текущему окну.
- status - текст временного сообщения в строке состояния окна браузера.
- top - синоним, используемый для обращения к главному окну браузера.
- window - синоним, используемый для обращения к текущему окну.

Методы

- **Метод alert()** применяется для того, чтобы вывести на экран текстовое сообщение.
- Для открытия окна используется метод open(), а для закрытия - **метод close().**
- С помощью **метода confirm()** происходит вывод на экран окна сообщения с кнопками Yes и No, и возвращает булевое значение true или false, в зависимости от нажатой кнопки.
- Посредством **метода prompt()** на экран выводится диалоговое окно с полем ввода.
- Метод **setTimeout()** устанавливает в текущем окне обработку событий, связанных с таймером.
- Метод **clearTimeout()** отменяет обработку таких событий.

Обработчики событий

- Объект **window** не обрабатывает события до тех пор, пока в окно не загружен документ.
- Однако можно обрабатывать события, связанные с загрузкой и выгрузкой документов.
- Обработчики таких событий задаются как значения атрибутов **onLoad и onUnload**, определяемых в теге **<body>.**
- Эти же атрибуты могут быть определены в тегах **<frameset>** фреймосодержащих документов.

пример:

- Загрузка страницы

http://my.site.ru в окно размером в 640x480 пикселов:

```
myWin = open ("http://my.site.ru",
"myWin",
"width=640, height=480");
```

Закрыть это окно можно из любого другого окна используя:

```
myWin.close();
```

Объект document

- Объект document соответствует всему гипертекстовому документ, вернее, той его части, которая заключена в контейнер <body> . . . </body>. Документы отображаются в окнах браузера, поэтому каждый из них связан с определенным окном. Все HTML-объекты являются свойствами объекта document, поэтому они находятся в самом документе. Например, в языке JS к первой форме документа можно обратиться, используя выражение:

```
document.forms[0]
```

в то время как к первой форме во втором фрейме следует обращаться выражением:

```
parent.frames[1].document.forms[0]
```

- Объект document удобно использовать для динамического создания HTML-документов.
- Для этого применяется HTML-контейнер

  <body> . . . </body>.
- Хотя в этом контейнере можно установить множество различных свойств документа, все же имеются такие свойства, значения которых нельзя установить с помощью этих тегов. Синтаксис тега я не буду приводить, - его можно найти в спецификации HTML. Мы же, будем считать, что синтаксис HTML знаем.

Свойства объекта document

- document.propertyName
- Объект document имеет достаточно много свойств, каждое из которых соответствует определенному HTML-тегу в текущем документе:
- alinkColor- соответствует атрибуту alink тега <body>;
- anchors- массив, который соответствует всем меткам в документе;
- bgColor- соответствует атрибуту bgColor (цвет фона) тега <body>;
- cookie- представляет собой фрагмент информации, записанный на локальный диск ("ключик");
- fgColor- соответствует атрибуту fgColor (цвет текста) тега <body>;

- fgColor- соответствует атрибуту fgColor (цвет текста) тега <body>;
- forms- массив, содержащий все теги <form> в текущем документе;
- images- масив изображений, ссылки на которые заданы в текущем документе;
- lastModified- дата последнего изменения текущего документа;
- linkColor- соответствует атрибуту linkColor (цвет гиперсвязи по умолчанию);
- links- масив, содержащий все гипервязи в текущем документе;
- location- соответствует адресу URL текущего документа;
- referrer- соответсствует адреу URL документа, из которого пользователь перешел к текущему документу;
- title- соответствует содержимому контейнера <title> . . . </title>;
- vlinkColor- соответствует атрибуту vlinkColor (цвет <FONT COLOR="#800080">посещенной связи) тега <body>.

Методы объекта document

document.methodName (parameters)

Метод **clear()** предназначен для очистки текущего документа.

Лучше использовать для очистки методы **open() и close()**.

Для записи информации в браузер применяют методы **write() и writeln()**. Поскольку эти методы записывают текст в браузер в HTML-формате, вы можете создавать любой HTML-документ динамически, включая готовые приложения на языке JavaScript.

Если в окно загружен документ, то запись данных поверх него может привести к сбою. Поэтому в окно следует записывать поток данных, для чего с помощью метода **document.open()** нужно открыть документ, а затем, вызвав необходимое количество раз метод **document.wtite()**, записать данные в документ.

В заключение, чтобы послать данные в браузер, следует вызвать метод **document.close()**.

Обработчики событий

В тегах <body> и <frame> можно использовать обработчики событий, связанных загрузкой и выгрузкой документа, onLoad и onUnload. Примеры использования событий будем разбирать позже.

Для записи текста в HTML-формате в браузер иногда применяют функцию document.writeln().

Например, можно динамически создавать теги изображений, выводя изображения на экран посредством следующего:

```
document.open();
document.writeln("<img
  sr='myimage.gif'>");
document.close();
```

- С помощью JavaScript программ, а в частности при помощи объекта document, можно создавать законченные HTML-документы и другие JavaScript программы. Например:
-

```
document.open();
document.writeln("<script language='JavaSript'>"+
"alert('Hello World!')"+
"</script>");
document.close();
```

- Заметьте, что в приведенных примерах неколько строк объединяются при помощи операции сложения +. Этот способ удобно применять, когда строки текста программы слишком длинны, чтобы поместиться в редактируемом окне, или когда сложные строки необходимо разбить на несколько простых.

Объект location

- Данный объект сохраняет местоположение текущего документа в виде адреса URL этого документа.
- При управлении объектом location существует возможность изменять адрес URL документа.
- Объект location связан с текущим объектом window - окном, в которое загружен документ.
- Документы не содержат информации об адресах URL.
- Эти адреса являются свойством объектов window.

Объект location

[windowVar.]location.prepertyName

где windowVar - необязательная переменная, задающая конкретное окно, к которому хотите обратиться. Эта переменная также позволяет обращаться к фрейму во фреймосодержащем документе при помощи свойства parent - синонима, используемого при обращении к объекту window верхнего уровня, если окон несколько. Объект location является свойством объекта window. Если вы обращаетесь к объекту location без указания имени окна, то подразумевается свойство текущего окна.

- Свойство location объекта window легко перепутать со свойством location объекта document. Значение свойства document.location изменить нельзя, а значение свойства location окна - можно, например при помощи выражения window.location.property. Значение document.location присваивается объекту window.location при первоначальной загрузке документа, потому, что документы всегда загружаются в окна.

Свойства

Объект location имеет следующие свойства:

- hash - имя метки в адресе URL (если задано);
- host - часть hostname:port адреса URL текущего документа;
- hostname - имя хоста и домена (или цифровой IP-адрес) в адресе URL текущего документа;
- href - полный адрес URL текущего документа;
- pathname - часть адреса URL, описывающая каталог, в котором находится документ;
- port - номер порта, который использует сервер;
- protocol - префикс адреса URL, описывающий протокол обмена, (например, http:);
- target - соответствует атрибуту target в теге <href>.

Методы и обработчики событий

- Для объекта location методы, не определены, также не связан с какими-либо обработчиками событий.

Примеры

- Чтобы присвоить свойству location текущего окна в качестве значения новый адрес URL, используйте такой вид:

```
self.location="http://wdstudio.al.ru";
```

- который в данном случае загружает в текущее окно Web-страницу. Вы можете опустить объект self, поскольку он является ссылкой на текущее окно.
- Чтобы загрузить ресурс в фреймосодержащий документ, можно записать так:

```
parent.frames[0].location = "http://my.site.ru";
```

- где parent.frames[0] соответствует первому фрейму в текущем документе.

Объект history

- Объект history содержит список адресов URL, посещенных в этом сеансе. Объект history связан с текущим документом. Несколько методов этого объекта позволяют загружать в браузер различные ресурсы и обеспечивают навигацию по посещенным ресурсам.
-
- Синтаксис:

history.propertyName

history.methodName (parameters)

Свойства: Значением свойства length является количество элементов в списке объекта history

Объект history

- Методы
- Метод **back()** позволяет загружать в браузер предыдущий ресурс, в то время как метод **forward()** обеспечивает обращение к следующему ресурсу в списке.
- С помощью метода **go()** можно обратиться к ресурсу с определенным номером в списке объекта **history**.
- Обработчики событий для объектов history не определены.

Примеры использования объекта history:

- Чтобы посмотреть предыдущий загруженный документ, воспользуйтесь оператором:

history.go(-1);

или

history.back();

- Для обращения к истории конкретного окна или фрейма применяют объект parent:

parent.frames[0].history.forward();

загружает в первый фрейм предыдущий документ.

- А если открыто несколько окон браузера можно использовать вид:
  - window1.frames[0].history.forward();

здесь в первый фрейм окна window1 будет загружен следующий документ из списка объекта history

Объект navigator

- Объект navigator содержит информацию об используемой в настоящее время версии браузера. Этот объект применяется для получения информации о версиях.
- Синтаксис:

navigator.propertyName

- Методы и события, как и не трудно догадаться не определены для этого объекта. Да и свойства только для чтения, так как ресурс с информацией о версии недоступен для редактирования.

Свойства

- appCodeName - кодовое имя браузера;
- appName - название браузера;
- appVersion - информация о версии браузера;
- userAgent - кодовое имя и версия браузера;
- plugins - массив подключаемых модулей (похоже только для Netscape);
- mimeTypes - поддерживаемый массив типов MIME.

Выводы

- Здесь я попыталась ввести понятия объектов и связанных с ними методов, свойств и обработчиков событий.
- Также описала объекты браузера. В следующих лекциях будут описаны остальные объекты языка JavaScript.

Внутренние объекты

В этой лекции мы рассмотрим внутренние объекты языка JavaScript. В предыдущей части рассматривались объекты браузера.

Внутренние объекты не относятся к браузеру или загруженному в настоящее время HTML-документу. Эти объекты могут создаваться и обрабатываться в любой JavaScript-программе.

- Они включают в себя простые типы, такие как строки, а также более сложные объекты, в частности даты.
- Имя объекта Описание
- Array Массив. Не поддерживается в браузерах старых версий
- Date Дата и время
- Math Поддержка математических функций
- Object Обобщенный объект. Не поддерживается в старых версиях IE - до 4, NN - до 3.
- String Текстовая строка. Не поддерживается в старых версиях

Объект array

- Array - это многомерное упорядоченное множество объектов, обращение к объектам ведется при помощи целочисленного индекса. Примерами объектов-массивов в браузере служат гиперсвязи, метки, формы, фреймы.

Массив можно создать одним из следующих способов:

- используя определенную пользователем функцию для присвоения объекту многих значений;
- используя конструктор Array();
- используя конструктор Object().
- Объекты-массивы не имеют ни методов, ни свойств.

Объект Date

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

  Для создания объекта Date применяется синтаксис:

dateObj = new Date(parameters)

где dateObj - переменная, в которую будет записан новый объект Date.

Аргумент parameters может принимать следующие значения:

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

Как уже говорилось ранее данный объект имеет множество методов, свойств объект Date не имеет.

Методы.

Метод **Описание метода**

- getDate() Возвращает день месяца из объекта в пределах от 1 до 31
- getDay() Возвращает день недели из объекта: 0 - вс, 1 - пн, 2 - вт, 3 - ср, 4 - чт, 5 - пт, 6 - сб.
- getHours() Возвращает время из объекта в пределах от 0 до 23
- getMinutes() Возвращает значение минут из объекта в пределах от 0 до 59
- getMonth() Возвращает значение месяца из объекта в пределах от 0 до 11
- getSeconds() Возвращает значение секунд из объекта в пределах от 0 до 59
- getTime() Возвращает количество миллисекунд, прошедшее с 00:00:00 1 января 1970 года.
- getTimeZoneoffset() Возвращает значение, соответствующее разности во времени (в минутах)
- getYear() Возвращает значение года из объекта

Методы.

Метод Описание метода

- Date.parse(arg) Возвращает количество миллисекунд, прошедшее с 00:00:00 1 января 1970 года. Arg - строковый аргумент.
- setDate(day) С помощью данного метода устанавливается день месяца в объекте от 1 до 31
- setHours(hours) С помощью данного метода устанавливается часы в объекте от 0 до 23
- setMinutes(minutes) С помощью данного метода устанавливаются минуты в объекте от 0 до 59
- setMonth(month) С помощью данного метода устанавливается месяц в объекте от 1 до 12
- setSeconds(seconds) С помощью данного метода устанавливаются секунды в объекте от 0 до 59

Методы.

Метод **Описание метода**

- setTime(timestring)С помощью данного метода устанавливается значение времени в объекте.
- setYear(year) С помощью данного метода устанавливается год в объекте year должно быть больше 1900.
- toGMTString() Преобразует дату в строковый объект в формате GMT.
- toString() Преобразует содержимое объекта Date в строковый объект.
- toLocaleString() Преобразует содержимое объекта Date в строку в соответствии с местным временем.
- Date.UTC(year, month, day [,hours][,mins][,secs]) Возвращает количество миллисекунд в объекте Date, прошедших с с 00:00:00 1 января 1970 года по среднему гринвичскому времени.

Разберем пару примеров:

- В данном примере приведен HTML-документ, в заголовке которого выводится текущие дата и время.

```
<html>
<head>
<script language "JavaScript">
<--
function showh() {
  var theDate = new Date();
  document.writeln("<table cellpadding=5 width=100%
   border=0>" +
      "<tr><td width=95% bgcolor=gray align=left>" +
      "<font color=white>Date: " + theDate +
      "</font></td></tr></table><p>");}
showh();
//-->
</script>
</head>
</html>
```

Разберем еще один пример. Подобный мы уже разбирали, когда рассматривали условные операторы, просто вспомним его и немного изменим: пусть меняются графические бэкграунды в зависимости от времени суток.

```
<html>
<script language "JavaScript">
<--
theTime = new Date();
theHour = theTime.getHours();
if (18 > theHour)
  document.writeln("<body background='day.jpg'
   text='Black'>");
else
  document.writeln("<body background='night.jpg'
   text='White'>");
//-->
</script>
</body>
</html>
```

Вероятно, вы успели заметить, что тег <body> создается в JavaScript-программе, а закрывается уже в статическом тексте HTML. Это вполне допустимо, так, как все теги расположены в правильном порядке. В данном примере предполагается, что файлы рисунков находятся в том же каталоге. Вы можете здесь задать полный адрес URL.

Объект Math

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

Синтаксис:

```
Math.propertyName
Math.methodName(parameters)
```

Свойства

Свойствами объекта Math являются математические константы:

- E Константа Эйлера. Приближенное значение 2.718 . . .
- LN2Значение натурального логарифма числа два. Приближенное значение 0.693 . . .
- LN10 Значение натурального логарифма числа десять. Приближенное значение 2.302 . . .
- LOG2_E Логарифм e по основанию 2 (не вижу смысла в этой константе - это же корень из двух.) Приближенное значение 1.442 . . .)
- LOG10_E Десятичный логарифм e. Приближенное значение 0.434 . . .
- PI Число ПИ. Приближенное значение 3.1415 . . .
- SQRT2Корень из двух, (ыгы, это все равно, как еще и натуральный логарифм 2*e в степени 1/2)

Методы

Методы объекта Math представляют собой математические функции.

- abs() Возвращает абсолютное значение аргумента.
- acos() Возвращает арккосинус аргумента
- asin() Возвращает арксинус аргумента
- atan() Возвращает арктангенс аргумента
- ceil() Возвращает большее целое число аргумента, округление в большую сторону. Math.ceil(3.14) вернет 4
- cos() Возвращает косинус аргумента
- exp() Возвращает экспоненту аргумента
- floor() Возвращает наибольшее целое число аргумента, отбрасывает десятичную часть

Методы

- max() Возвращает больший из 2-х числовых аргументов. Math.max(3,5) вернет число 5
- min() Возвращает меньший из 2-х числовых аргументов.
- pow() Возвращает результат возведения в степень перврго аргумента вторым. Math.pow(5,3) вернет 125
- random() Возвращает псевдослучайное число между нулем и единицей.
- round() Округление аргумента до ближайшего целого числа.
- sin() Возвращает синус аргумента
- sqrt() Возвращает квадратный корень аргумента
- tan() Возвращает тангенс аргумента

Обработчиков событий нет для внутренних объектов.
Синтаксис очень прост, вызывается метод как любая функция, но это все же метод и не забывайте указывать префикс Math перед методом:

```
var mpi = Math.Pi
```

В данном случае переменной mpi присвоится значение Пи.
Или, например,

```
var myvar = Math.sin(Math.Pi/4)
```

Строковые объекты

- Строка (string) в языке JavaScript представляется в виде последовательности символов, заключенных в двойные или одинарные кавычки. Для управления строковыми объектами используется синтаксис:

stringName.propertyName

stringName.methodName(parameters)

Здесь stringName - имя объекта String. Строки можно создавать тремя способами:

- 1. Создать строковую переменную при помощи оператора var и присвоить ей строковое значение;
- 2. Присвоить значение строковой переменной только посредством оператора присваивания (=);
- 3. Использовать конструктор String().

Свойства

- Значением свойства length является длина строки.
- Например, выражение

  "Script".length

- вернет значение 6, поскольку строка "Script" содержит 6 символов.

Методы

- Вызов метода осуществляется обычно: **"Строка или строковая переменная".метод()**,
- в данном случае метод без параметров, имеются методы и с параметрами. Заметьте, строка или строковая переменная, к которой применяется метод - объект, и никак не аргумент!

Метод Описание метода

- **big()** Аналогично тегам HTML <big> . . .</big>. позволяет отобразить более крупным шрифтом.
- **blink()** Заставляет строку мигать. (Этим почти никто не пользуется).
- **bold()** Название говорит за себя - делает символы жирными.
- **charAt(arg)** Возвращает символ, находящийся в заданной позиции строки. Пример: vpos = "Sc<FONT COLOR="#FF0000">ript".charAt(3); переменной vpos будет присвоено значение "r".
- **fixed()** Анологично <tt> . . .</tt> вывод строки фиксированного размера.
- **fontcolor(arg)** Анологично . . .. Аргумент метода может быть как триплетом RGB, так и зарезервированным словом.
- **fontsize(arg)** Позволяет изменять размер шрифта. Аргумент в условных единицах. Анологично . . ..
- Также можно использовать вид **+size** или **-size** для увеличения или уменьшения шрифта на size единиц, например: **"строка".fontsize(+1)**.

- **indexOf(arg1[,arg2])** Возвращает позицию в строке, где впервые встречается символ - arg1, неябязательный числовой аргумент arg2 указывает начальную позицию для поиска.
- **italics()** Аналогично тегам HTML . . .. позволяет отобразить италиком.
- **lastIndexOf(arg1[,arg2])** Возвращает либо номер позиции в строке, где в последний раз встретился символ - arg1, либо -1, если символ не найден. Arg2 задает начальную позицию для поиска.
- **link()** Аналогично тегам HTML <a href> . . .. позволяет преобразовать строку в гиперсвязь.
- **small()** Аналогично тегам HTML <small> . . .</small>. позволяет отображать строку мелким шрифтом.
- **strike()** Аналогично тегам HTML <strike> . . .</strike>. позволяет отображать строку зачеркнутой.

- **sub()** Аналогично тегам HTML _{. . .}. позволяет отображать строку нижним индексом.
- **substring(arg1,arg2)** Позволяет извлеч подстроку длинной arg2, начиная с позиции arg1
- **sup()** Аналогично тегам HTML ^{. . .}. позволяет отображать строку верхним индексом.
- **toLowerCase()** Преобразует символы строкового объекта в строчные
- **toUpperCase()** Преобразует символы строкового объекта в прописные

Вот, пожалуй, и весь список методов объекта String. Примеры их использования будут приводиться по ходу рассмотрения других объектов. К строковым методам, как видно из таблицы относятся методы-функции операций над строками и в то же время как методы форматирования.

Объекты, соответствующие тегам HTML - 1

- Многие объекты языка JavaScript соответствуют тегам, формирующим HTML-документы. Каждый такой объект состоит во внутренней иерархии, поскольку все они имеют общий родительский объект - браузер, который представлен объектом window.
- Некоторые объекты языка JavaScript имеют потомков. В частности, гиперсвязь является объектом, наследованным из объекта document. В языке JS наследованные объекты называются также свойствами. Например, множество гиперсвязей является свойством объекта document, а links - именем этого свойства. Таким образом, трудно провести четкую границу между объектами и свойствами. **Гиперсвязь, являясь объектом, в то же время представляет собой свойство links объекта document.**

Рассмотрим пример. Напишем простенькую программку и посмотрим, как будут создаваться объекты HTML. Тоесть, при загрузке HTML-документа в браузер соответственно будут созданы HTML-объекты на JavaScript. Теги HTML собственно служат исключительно для удобства написания документа:

```
<html>
<head>
<title>Пример программы</title>
</head>
<body bgcolor="White">
<form>
<input type="checkbox" checked
  name="chck1">Item 1
</form>
</body>
</html>
```

Посмотрим эквивалентную запись на JavaScript

```
document.title="Пример программы"
document.bgColor="White"
document.forms[0].chck1.defaultChecked=true
```

Как видно из примера, тегу <title> соответствует свойство document.title, а цвету фона документа, установленному в теге <body>, - свойство document.bgColor.
Переключателю checkbox с именем chck1, определенному в форме, соответствует выражение document.forms[0].chck1.
Свойство defaultChecked принадлежит объекту check-box и может принимать значения true или false в зависимости от того, указан ли в теге <input> атрибут checked. Когда этот атрибут задан, переключатель отображается на экране как включенный по умолчанию.

- Поскольку документ может включать несколько таких объектоов, как гиперсвязи, формы и другие объекты, многие из них являются массивами. Для **обращения к определенному элементу массива** нужно указать его индекс. Например, **forms[0]** - первая форма текущего документа. Ко второй форме можно обратиться соответственно **forms[1]**. Заметьте, что индексы массивов в JS программах всегда начинаются с **нуля**.
- В нашем примере объект верхнего уровня - window, потому, что любой документ загружается в окно. Например, выражения document.forms[0] и window.document.forms[0] обращаются к одному и тому же объекту, а именно к первой форме текущего документа. Однако если необходимо обратиться к форме в другом окне (фрейме), следует использовать выражение вида

parent.frames[n].document.forms[n]

где n является индексом нужного элемента массива.

Имя объектаКраткое описание

- **anchor (anchors[])** Множество тегов <a name> в текущем документе
- **button** Кнопка, создаваемая при помощи тега <input type=button>
- **checkbox** Контрольный переключатель, создаваемый при помощи тега <input type=checkbox>
- **elements[]** Все элементы тега <form>
- **form (forms[])** Множество объектов тега <form> языка HTML
- **frame (frames[])** Фреймосодержащий документ
- **hidden** Скрытое текстовое поле, создаваемое при помощи тега <input type=hidden>
- **images (images[])** Множество изображений (тегов <img>) в текущем документе
- **link (links[])** Множество гиперсвязей в текущем документе

Имя объектаКраткое описание

- **navigator** Объект, содержащий информацию о браузере, загрузившем документ
- **password** Поле ввода пароля, создаваемое при помощи тега <input type=password>
- **radio** Селекторная кнопка (radio button), создаваемая при помощи тега <input type=radio>
- **reset** Кнопка перегрузки, создаваемая при помощи тега <input type=reset>
- **select (options[])** Элементы <option> объекта <select>
- **submit** Кнопка передачи данных, создаваемая при помощи тега <input type=submit>
- **text** Поле ввода, сооздаваемое при помощи тега <input type=text>
- **textarea** Поле текста, создаваемое при помощи тега <textarea>

Объекты, которым соответствуют массивы, являются многомерными объектами.

В некоторых HTML-тегах можно определить несколько элементов, например множество элементов списка в теге <select>.

Рассмотрим тег <select>, содержащий два элемента:

```
<form>
<select name="primer">
<option>Опция1
<option>Опция2
</select>
</form>
```

- Тег <select> сам по себе является объектом, однако для обращения к отдельным элементам этого объекта (тегам <option>) используется массив option. Данный массив представляет собой множество значений, которые соответствует тегам <option>, содержащимся в теге <select>. В нашем примере создается два объекта: первый - объект select в целом (к нему обращаются, чтобы выяснить, сколько элементов он фактически соодержит), второй - массив options (он позволяет обращаться к отдельным элементам, содержащимся в первом объекте). Таким образом, некоторые объекты могут использовать объекты-массивы для обращения к содержащимся в них элементам. Однако это не является правилом, все зависит от структуры рассматриваемых объектов и тех объектов, из которых они унаследованы. Например, HTML-тегам <a name> . . . соответствует объект anchor, являющийся элементом массива anchors, и в то же время эти теги встречаются сами по себе, а не в других тегах. Просто родительским объектом (parents) для объекта anchors является объект document, а в документе может быть определено множество меток. Окна тоже могут содержать множество документов, связанных с ними через фреймы.

Объект anchor и массив anchors

- Anchor - это элемент текста, который является объектом назначения для тега гиперсвязи <a href>, а также свойством объекта document. Тегу <a name> в языке JavaScript соответствует объект anchor, а всем тегам <a name>, имеющимся в документе, - массив anchors.
- Являясь объектами назначения для гиперсвязей <a name>, метки в основном используются для индексирования содержимого гипертекстовых документов, обеспечивая быстрое перемещение к определенной части документа при щелчке мыши на гиперсвязи, которая обращается к данной метке. Тег <a>, задающий метки и гиперсвязи, имеет синтаксис:

```
<a [href=location]
    [name="anchorName"]
    [target="windowName"] >
    anchorText
    </a>
```

- Как вы успели заметить, обычная схема построения гиперсвязей. Значение location задает имя метки.
- Когда значение определено, данный тег задает гиперсвязь.
- Location может также включать и URL, например: **href="http://wdstudio.al.ru/index.htm#netscape".**
- Заметьте, что перед и после знака # пробелы не допустимы.
- Атрибут **name="anchorName"** определяет имя метки, которая будет объектом назначения для гипертекстовой связи в текущем HTML-документе: .
- В данном случае netscape - имя метки, которая будет объектом назначения для гипертекстовой связи.
- Атрибут **target="windowName"** - имя объекта-окна или его синонима: self, top и др., в котрое загружается документ, запрашиваемый при активизации гиперсвязи.

- Значение **anchorText з**адает текст, который будет отображаться на экране в том месте, где находится метка, и является необязательным.
- Например: ** **.
- А вот с атрибутом **href -** текст в большинстве случаев должен быть виден на экране, иначе как активизировать гиперсвязь.
- Атрибут **target** также может существовать только с атрибутом **href.**

Массив anchors

- Посредством массива anchors программа на языке JavaScript может обращаться к метке текущего гипертекстового документа. Каждому тегу <a name> текущего документа соответствует элемент массива anchors. Для того чтобы программа выполнялась правильно, в соответствующих атрибутах name должны быть заданы имена всех меток. Если документ содержит именованную метку, определенную HTML-тегом

  < a name="s1">Selection1

- то этой метке в JS-программе соответствует объект **document.anchors[0]**.

Массив anchors

- Чтобы перейти к этой метке посредством гиперсвязи, пользователь должен щелкнуть мышью на тексте, определенном в контейнере . . . . К массиву anchors можно обращаться при помощи следующих операторов:

document.anchors[i]

document.anchors.length

- где i - индекс запрашиваемой метки. Свойство length позволяет определить количество меток в документе, хотя элементы, соответствующие отдельным меткам, будут содержать значение null. Это значит, что нельзя обращаться к именам отдельных меток через элементы массива.

Свойства

- Массив anchors имеет только одно свойство **length**, которое возвращает значение, соответствующее количеству меток в документе.
- Массив anchors является структурой только для чтения.
- Методов и обработчиков событий объекты anchors не имеют.

Объект button

- Кнопка - это область окна, которая реагирует на щелчки мыши и может активизировать оператор или функцию языка JavaScript при помощи атрибута события onClick.
- Кнопки являются свойствами объекта form и должны быть заключены в теги *<form> . . . </form> языка HTML*.

Синтаксис:

```
<input type="button"
name="buttonName"
value="buttonText"
[onClick="handlerText"]>
```

Атрибут name задает имя кнопки и в языке JS ему соответствует свойство name нового объекта buttun. Атрибут value определяет надпись на кнопке, которой соответствует свойство value. К свойствам и методам объекта button можно обратиться одним из способов:

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

- Здесь **buttonName** - значение атрибута name, а formName - либо значение атрибута name объекта form, либо элемент массива forms. Переменная i является индексом, используемым для обращения к отдельному элементу массива, в данном случае к элементу button.

Свойства

- Свойства **name** и **value** объекта button соответствует атрибутам name и value HTML-тега <input>.
- Обратившись к значениям этих свойств, можно вывести полный список кнопок, имеющихся в текущем документе.
- Свойство **type** объекта button всегда имеет значение "button".

Методы и обработчики событий

- Объект button имеет метод **click()** - о нем будем говорить позже.
- Обработчик событий **onClick** позволяет выполнить оператор или вызвать функцию языка JavaScript при щелчке мыши на кнопке, которой соответствует в программе определенный объект button.

Приведем простой пример, например, выведем текущую дату и время посредством нажатия кнопки. Будем использовать событие onClick для вызова метода alert() и конструктора Date()
Пример схематичный, объект должен быть определен

```
<form>
<input type="button"
value="Date and Time"
onClick='alert(Date())'>
</form>
```

Объект checkbox

- Контрольный переключатель - это кнопка(флажок), которую можно установить в одно из двух состояний: включено или выключено.
- Объекты checkbox являются свойствами объекта form и должны быть помещены в теги **<form> . . . </form>.**

Простой контрольный переключатель:

- Checkbox1
- Checkbox2
- Checkbox3

Синтаксис:

```
<input name="checkboxName"
type="checkbox"
value="checkboxValue"
[checked]
[onClick="handlerText"]>textToDisplay
```

- где атрибут name является именем объекта checkbox.
- Ему соответствует свойство *name* объекта языка JavaScript. Атрибут *value* определяет значение, которое передается серверу при пересылки значений элементов формы, если контрольный переключатель включен.
- Необязательный атрибут *checked* указывает, что контрольный переключатель должен быть включен по умолчанию.
- Если этот атрибут задан, свойство *defaultChecked* имеет значение true.
- При помощи свойства *checked* можно определить, включен ли контрольный переключатель.
- Текст, отображаемый рядом с контрольным переключателем, задается строкой *textToDisplay*.

К объекту checkbox можно обращаться одним из способов:

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

- где checkboxName - значение атрибута name объекта checkbox, а formName - имя объекта form или формы, которой принадлежит данный контрольный переключатель. Другими словами, к форме можно обращаться как к элементу массива forms, например forms[0] - для обращения к первой форме документа, либо по имени объекта form, если оно определено в атрибуте name HTML-тега <form>.

- К любому элементу формы можно обратиться так же, как к элементу массива elements, который является свойством объекта form.
- В этом случае для обращения к определенному контрольному переключателю следует использовать его порядковый номер (i) в массиве всех элементов формы.

Свойства

- Если контрольный переключатель включен, свойство checked имеет значение true.
- Когда в теге <input> используется атрибут checked, например <input checked type=checkbox>, свойсту defaultChecked также присваивается значение true.
- Свойство name соответствует атрибуту name тега <input name= . . . type=checkbox>,
- а свойство value - атрибуту value тега <input>. (оно и правильно: ключевые слова и должны соответствовать чтобы путаницы не было). Свойство type объекта checkbox всегда содержит значение "checkbox".

Методы и обработчики событий

- Метод ***Click()*** может использоваться с объектом ***checkbox***, мне не приходилось его использовать, но есть много замечаний в адрес этого метода, - в некоторых браузерах он должным образом не работает. Но тем не менее он имеется.
- Для объекта ***checkbox*** предусмотрен только один обработчик - ***onClick()***.

Массив elements

- Массив elements содержит все элементы HTML-формы - контрольные переключатели (checkbox), селекторные кнопки (radio-button), текстовые объекты (text) и другие, - в том порядке, в котором они определены в форме. Этот массив можно использовать для доступа к элементам формы в JS-программе по их порядковому номеру, не используя свойства name этих элементов. Массив elements, в свою очередь, является свойством объекта forms, поэтому при обращении к нему следует указывать имя формы, к элементу которой вы хотите обратиться:

  formName.elements[i]

  formName.elements[i].length

- Здесь formName может быть либо именем объекта form, определенным при помощи атрибута name в теге <form>, либо элементом массива forms, например forms[i], где i - переменная, которая индексирует элементы массива. Значением свойства length является количество элементов, содержащихся в форме. Массив elemments включает данные только для чтения, т.е. динамически записать в этот объект какие-либо значения невозможно.

Свойства

- Объект **elements** имеет только одно свойство, **length**, значением которого является количество элементов объекта form.

  document.forms[0].elements.length

- возвратит значение, соответствующее количеству элементов в первой форме текущего документа.

Пример

Создадим пару элементов, например поля ввода для имени и адреса:

Имя: Адрес:

Нажав на эту кнопку, можно увидеть элементы формы, назовем ее "Форма для примера". Третьим элементом будет кнопка, вызывающая функцию на JavaScript. Она также находится в данной форме.

Внимание: не корректно работает в Internet Explorer-е. Дело в том, что в этом браузере элементы формы хранятся не в виде строки. В NN должно быть нормально. IE 3.01 может даже вызвать ошибку. IE 4 и выше ошибки не выдает.

Теперь рассмотрим текст этой программы:

```
<html>
<head>
<script language="JavaScript">
<!--
function showElem(f) {
var formEl = " ";
for (var n=0; n < f.elements.length; n++) {
formEl += n + ":" + f.elements [n] +"\n";
}
alert("Элементы в форме '" + f.name + "' :\n\n" + formEl );
}
//-->
</script>
</head>
<body>
<form name="Форма для примера">
Имя:
<input type="text" size=10 name="fullname">
Адрес:
<textarea name="adr"></textarea>
<BR>
<input type="button" value="Смотрим элементы" onClick="showElem(this.form)">
</form>
</body>
</html>
```

- Здесь функция перебирает все элементы массива elements заданной формы, в данном примере их три, формирует строку formEl, содержащую информацию об элементах данного массива. IE покажет здесь в виде "n:[object]" то есть этот браузер не содержит в массиве elements строки с информацией об объекте формы. Созданная строка (для удобства читаемости разделена "переводом строки \n") выводится в окне предупреждения с помощью метода alert().
- Функция showEl() вызывается с аргументом this.form, который обращается к текущей форме. Если оператор this опустить, то из функции showEl() к форме придется обращаться с помощью выражения document.forms[n], - это не очень удобно, так как мы обращаемся из текущей формы.

Объект form и массив forms

- **Форма** - это область гипертекстового документа, которая создается при помощи контейнера **<form> . . . </form>** и содержит элементы, позволяющие пользователю вводить информацию.
- Многие HTML-теги, например теги, определяющие поля ввода (text field), области текста (textarea), контрольные переключатели (checkbox), селекторные кнопки (radio button) и списки (selection list), располагаются только в контейнере <form> . . . </form>.
- Всем перечисленным элементам в языке JavaScript соответствуют отдельные объекты.
- Программы на языке JS могут обрабатывать формы непосредственно, получая значения, содержащиеся в необходимых элементах (например для проверки ввода обязательных данных). Кроме того, данные из формы обычно передаются для обработки на удаленный Web-сервер.

Синтаксис:

```
<form name="formName"
target="windowname"
action="serverURL"
method="get" | "post"
enctype="encodingType"
[onSubmit="handlerText"]>
</form>
```

- Атрибут **name** - строка, определяющая имя формы. Атрибут **target** задает имя окна, в котором должны обрабатываться события, связанные с изменением элементов формы. Для этого требуется наличие окна или фрейма с заданным именем. В качестве значений данного атрибута могут использоваться и зарезервированные имена **_blank, _parent, _self и _top.**
- Атрибут **action** задает адрес URL сервера, который будет получать данные из формы и запускать соответствующий CGI-скрипт. Также можно послать данные из формы по электронной почте, указав при этом значения этого атрибута адрес URL типа mailto: . . .

Формы, передаваемые на сервер, требуют задания метода передачи (submission), который указывается при помощи атрибута method. Метод GET присоединяет данные формы к строке адреса URL, заданного в атрибуте action. При использовании метода POST информация из формы посылается как отдельный поток данных. В последнем случае CGI-скрипт на сервере считывает эти данные из стандартного входного потока (standard input stream). Кроме того, на сервере устанавливается переменная среды с именем QUERY_STRING, что обеспечивает еще один способ получения этих данных.

- **Атрибут enctype задает тип кодировки MIME (Multimedia Internet Mail Extensions) для посылаемых данных. Типом MIME по умолчанию является тип application/x-www-form-urlencoded.**
- К свойствам и методам формы в JavaScript-прграмме можно обратиться одним из способов:

  formName.propertyName

  formName.methodName (parameters)

  forms[i].propertyName

  forms[i].methodName (parameters)

- Здесь formName соответствует атрибуту name объекта form, а i является целочисленной переменной, используемой для обращения к отдельному элементу массива forms, который соответствует определенному тегу <form> текущего документа.

Использование массива forms

- К любой форме текущего гипертекстового документа можно обращаться как к элементу массива forms. Для этого необходимо указать индекс запрашиваемой формы. Например, forms[0] - первый тег <form> в текущем документе.

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

- Переменная i - это индекс, соответствующий запрашиваемой форме.
- Выражение вида

```
document.forms[i]
```

- можно также присвоить переменной

```
var myForm = document.forms[i];
```

Теперь, если в форме имеется, к примеру, поле ввода, определенное в HTML-теге

```
<form>
<input type=text name=myField size=40>
. . .
</form>
```

- то в JS-программе к этому полю позволяет обращаться переменная myForm. В частности, при помощи следующего оператора содержимое данного поля ввода присваивается новой переменной с именем result:

```
var result = myForm.myField.value;
```

- Значение свойства length соответствует количеству форм в документе:

  var numForms = document.forms.length
- Массив forms содержит данные, которые используют только для чтения.

Свойства

Объект form имеет шесть свойств, большинство из них соответствуют атрибутам тега <form>:

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

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

Методы

- Метод submit() применяется для передачи формы из JavaScript-программы.
- Его можно использовать вместо тега **<input type=submit>**, имеющегося в большинстве форм, информация которых должна передаваться на сервер.

Обработчики событий

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

Пример. При нажатии кнопки передачи данных содержимое текстового поля посылается адресату по электронной почте:

- Отсюда вы можете послать почту. Перед отправкой последует запрос на отправку почты, - это срабатывает защита на вашем компьютере. Ничего, кроме содержимого формы не отправит!

```
<form nethod="post"
action="mailto:my@mail.ru"
enctype="text/plain">
<input type="submit" value="Отправить почту">
<input type="reset" value="Очистить форму">
<textarea name="email" rows=5 cols=60>
</textarea>
</form>
```

Объекты, соответствующие тегам HTML - 2

Массив frames

- К отдельным фреймам можно обращаться при помощи массива frames и свойства parent.
- Например, если имеется два фрейма, определенных в HTML-тегах:

```
<frameset rows="50%,50%">
<frame name="top" src="file1.htm">
<frame name="bot" src="file2.htm">
</frameset>
```

- Для обращения к первому фрейму вы можете использовать выражение parent.frames[0], и соответственно ко второму - parent.frames[1]. Таким образом, для обращения к отдельным фреймам и к свойству length массива frames используются выражения вида:

  frameRef.frames[i]

  frameRef.frames.length

  windowRef.frames[i]

  windowRef.frames.length

- Для определения количества фреймов во фреймосодержащем документе применяется свойство length. Все данные массива frames предназначены только для чтения.

Свойства

Объект frame имеет следующие свойства:

- frames - массив, содержащий все фреймы в окне;
- name - соответствует атрибуту name тега <frame>;
- length - количество дочерних фреймов в родительском окне (фрейме).

Кроме того, можно использовать такие синонимы:

- parent - синоним для окна или фрейма с текущим фреймосодержащим документом;
- self - синоним для текущего фрейма;
- window - синоним для текущего фрейма.

Массив frames имеет всего одно свойство length, значением которого является количество дочерних фреймов в родительском фрейме.

Методы и обработчики событий

- Во фреймосодержащих документах могут быть использованы методы **clearTimeout()** и **setTimeout()**.
- В теге **<frameset>** определяют обработчики событий, связанные с загрузкой и выгрузкой документов **onLoad** и **onUnLoad**.
- Об этих методах и событиях будем говорить позже. Пока мы ими пользоваться не будем. Забегать вперед тоже не очень хорошо.

Скрытый объект

- Что это такое.
- Это поле, которое может передаваться из формы например на сервер, находиться в тегах <form> . . . </form>, при этом не отображаться на экране. Для чего оно нужно? Ну например, что-то формируется JS программой и это нужно передать, при этом выводить эту информацию нет смысла. Это текстовые поля позволяют сохранять определенные значения в структурах, отличных от переменных языка JS, хотя данные значения существуют до тех пор, пока загружен текущий документ. Скрытое поле, как уже говорилось является свойством объекта form и должно помещаться в тегах <form> . . . </form>.

HTML-тег имеет синтаксис:

<input type="hidden"
[name="hiddenName"]
[value="textValue"]>

- Атрибут name задает имя поля и является необязательным. Значение текстового поля указывают при помощи атрибута value, который позволяет задавать и значение поля по умолчанию. К свойствам скрытых объектов можно обращаться посредством одного из следующих выражений:

  fieldName.propertyName

  formName.elements[i].propertyName

- где fieldName - имя скрытого поля, заданное в атрибуте name тега <input>, а formName - имя формы, в которой определено скрытое поле.

Атрибут name задает имя поля и является необязательным. Значение текстового поля указывают при помощи атрибута value, который позволяет задавать и значение поля по умолчанию. К свойствам скрытых объектов можно обращаться посредством одного из следующих выражений:

fieldName.propertyName

formName.elements[i].propertyName

где fieldName - имя скрытого поля, заданное в атрибуте name тега <input>, а formName - имя формы, в которой определено скрытое поле.

Свойства

Скрытый объект имеет свойства:

- name - соответствует атрибуту name тега <input>;
- value - соответствует атрибуту value тега <input>;
- type - соответствует атрибуту type и содержит значение "hidden".

Скрытые объекты не имеют методов и обработчиков событий.

Пример

- В следующей форме определено скрытое поле hfield шириной 20 символов, по умолчанию имеет значение "page 1":

```
<form name="hiddenField">
<input name="hfield" type="hidden"
  size=20 value="page 1">
</form>
```

Значение этого поля можно изменить с помощью оператора следующего вида:

```
document.hiddenField.hfield.value = "page 2";
```

Объект image и массив images

Браузер Microsoft Internet Explorer версии ниже 4, не поддерживает массив images.

В браузере рисунки рассматриваются как объекты image, а все рисунки, содержащиеся в текущем документе, помещаются в массив images, который можно использовать для обращения к любому рисунку, определяемому тегом <img>.

В частности, можно динамически обновлять изображения, изменяя их свойство src.

Для начала приведем тег <img>, распишем полностью:

```
<img src="fileOrUrl"
    [alt="altText"]
    [lowsrc="fileOrUrl"]
    [width=w]
    [heigth=h]
    [border=b]
    [vspace=v]
    [hspace=h]>
```

В атрибуте src содержится имя или адрес URL файла, который нужно вывести в документе. Рисунок должен храниться в формате GIF, JPEG, или PNG. С помощью атрибута **alt** задается альтернативный текст, появляющийся на экране: в момент загрузки текста, если пользователь заблокировал вывод изображений и поясняющая надпись под курсором мыши. Атрибут **lowsrc**, NN его поддерживает, IE не имеет смысла его использовать. Он позволяет предварительно выводить на экран изображение с низким разрешением. При этом рисунок загружается в два этапа. Атрибуты **width** (ширина) и **height** (высота) позволяют задать размеры рисунка в пикселах, атрибут **border** - ширину рамки в пикселах, а атрибуты vspace и **hspace** - размеры вертикального и горизонтального зазоров между границами изображения и другими элементами документа.

Для обращения к свойствам объекта image используется следующий синтаксис:

document.images[i].propertyName

где i - индекс элемента массива, который соответствует нужному рисунку.

Первым рисунком в документе будет **document.images[0]**.

Массив images является свойством объекта document, поэтому при обращении к рисунку необходим префикс **document** к имени массива.

Тег **<img>** не имеет атрибута name, поэтому выражение вида "**document.imgName**" приведет к ошибке.

Свойства. Все свойства объектов image соответствуют атрибутам тега <img>, за исключением свойства complete.

Эти свойства, кроме свойств src и lowsrc, значения которых могут быть изменены динамически, имеют значения только для чтения:

- src - соответствует атрибуту src тега <img>;
- lowsrc - соответствует атрибуту lowsrc тега <img>;
- height - соответствует атрибуту height тега <img>;
- width - соответствует атрибуту width тега <img>;
- border - соответствует атрибуту border тега <img>;
- vspace - соответствует атрибуту vspace тега <img>;
- hspace - соответствует атрибуту hspace тега <img>;
- complete - содержит булево значение, которое указывает, загружен рисунок в браузер или нет (true - загружен, false - нет);
- type - для объектов image содержит значение "image".

Перед загрузкой рисунка появляется его рамка, внутри которой отображается строка, заданная в атрибуте **alt** (в версии 5 и выше IE, пользователь при желании может отключить рамки с alt-текстом отображаемые в момент загрузки рисунка).

Рисунок можно изменять динамически, присваивая атрибуту src или lowsrc в качестве значения новый адрес URL

(локально проверить это не удается, так как lowsrc загрузится мгновенно. Ошибок по крайней мере при применении этого атрибута не выдает.)

Методы и обработчики событий.

Объект image не имеет методов.

Обработчики событий:

- **onAbort** - обработка события, возникающего при прерывании загрузки рисунка, т.е. при нажатии клавиши [Esc] или активизации новой гиперсвязи, в то время, когда рисунок загружается;
- **onError -** обработка события, связанного с ошибкой загрузки рисунка, т.е. когда невозможно найти рисунок по указанному адресу URL ;
- **onLoad** - соответствующее событие, инициализируется в начале загрузки рисунка. При загрузке анимированного GIF-а это событие возникает несколько раз и зависит от числа кадров анимационной последовательности.

Объект link и массив links

- Объект **link (гиперсвязь)** отображается как участок текста или графического объекта, щелчок мыши на котором позволяет перейти к другому Web-ресурсу. Тег языка HTML, а мы помним, что рассматриваем объекты соответствующие тегам HTML, имеет следующий вид:

```
<a href=locationOrURL
[name="anchorName"]
[target="windowOrFrameName"]
[onClick="handlerText"]
[onMouseOver="handlerText"]>
linkText
</a>
```

- Атрибут **href** определяет имя файла, или адрес URL для объекта, который загружается при активизации гиперсвязи.
- Атрибут **name** задает имя гиперсвязи, превращая ее в объект **anchor** (метку).
- С помощью атрибута **target** в определенный фрейм текущего фреймосодержащего документа можно загрузить документ, URL которого указан в значении атрибута href.
- Атрибут **linkText** представляет собой текст, отображаемый в HTML-документе как гиперсвязь, которая активизируется щелчком мыши.
- Для обращения к свойству объекта link используются выражения типа:

document.links[i].propertyName

где i - индекс данной связи в массиве гиперсвязей links текущего документа.

Масив links

- В программе на языке JavaScript к гиперсвязям можно обращаться как к элементам массива links. Например, если в документе определены два тега < a href>, то в JS-программе к этим гиперсвязям можно обращаться с помощью выражений **document.links[0]** и **document.links[1]**.
- Синтаксис выражений для обращений к массиву links следующий:

document.links[i]

document.links.length

где переменная i - индекс гиперсвязи. Значением свойства length является количество гиперсвязей в текущем документе.

Объекты link представляют собой объекты только для чтения, поэтому динамически изменять гиперсвязи в документе нельзя.

Свойства

Для объекта link определены следующие свойства:

- hash - задает имя метки в адресе URL, если она существует ;
- host - задает часть hostname:port адреса URL, определенного в гиперсвязи;
- hostname - задает имя хоста и домена (или IP-адрес) в адресе URL, определенном в гиперсвязи;
- href - задает полный адрес URL, определенный в гиперсвязи;
- pathname - задает часть адреса URL, которая описывает путь к документу и находится после части hostname:port;
- port - задает коммуникационный порт, который использует сервер;
- protocol - задает начало адреса URL, включая двоеточие, например http:;
- target - соответствует атрибуту target тега <a href>.

Массив links имеет всего одно свойство, length, значением которого является количество гиперсвязей в текущем документе.

Методы и обработчики событий

- Для **объекта link** методы не определены.
- В **тегах <a href>** могут использоваться обработчики событий щелчка мыши и ее перемещения - onClick и onMouseOver.

Пример

- При подведении указателя мыши на гиперсвязь, в строке состояния браузера появится текст "Текст в строке состояния при подведении мыши на гиперсвязь".

```
<a href="#" onMouseOver="window.status='Текст в строке состояния при подведении мыши на гиперсязь';
  return true">
Подведите сюда курсор мыши
</a>
```

- В данном случае гиперсвязь указывает на пустой документ - "#". Это выбрано для примера в случае щелчка на гиперсвязи ничего не грузилось.

Модифицирование веб-страниц

Для генерирования нового и модификации уже имеющегося HTML-кода на странице первым делом вы должны идентифицировать элемент (тег) на странице, а далее выполнить над ним какие-либо действия.

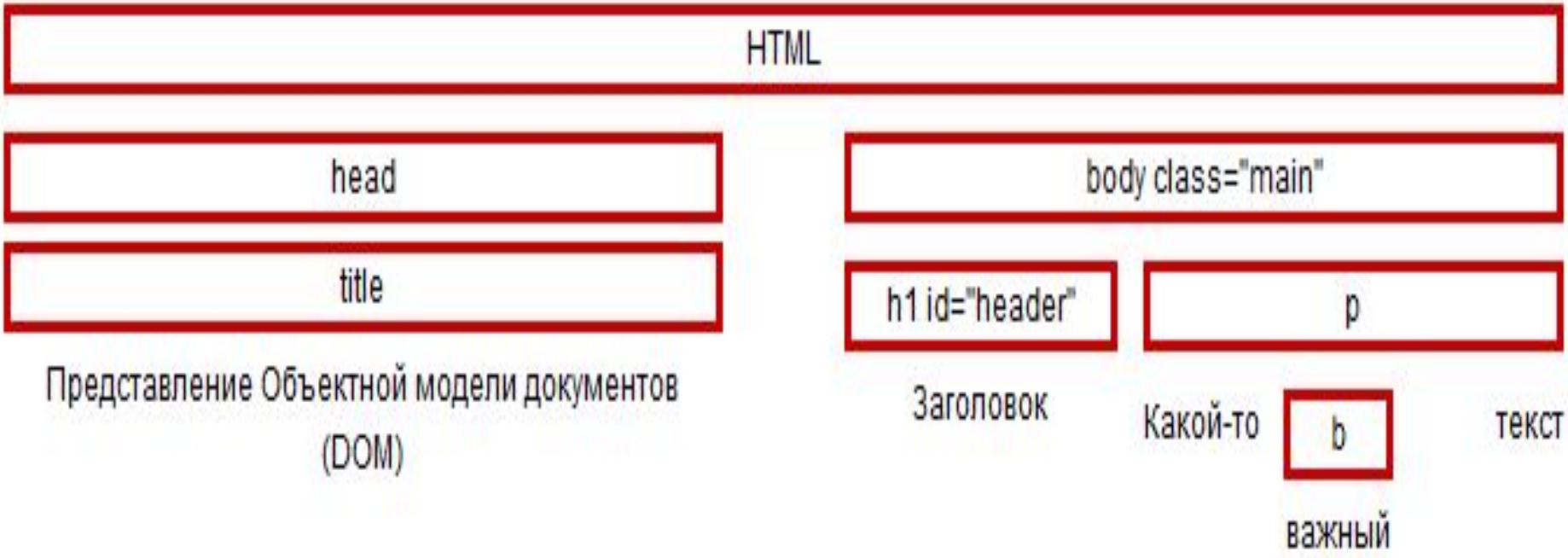
При загрузке HTML-документа на страницу выводится его содержимое, при этом браузер запоминает модель HTML, т.е. теги, их атрибуты и последовательность их появления на странице.

Во время загрузки страницы Web-обозреватель создает экземпляр объекта **HTMLDocument** и сохраняет его в переменной **document**, дополнительно считывает код HTML элементов и создает их внутреннее представление в виде экземпляров соответствующих объектов. Так, для абзаца создается экземпляр объекта **HTMLParagraphElement**, для гиперсылки - **HTMLLinksElement**, для картинки - **HTMLImageElement**, для таблицы - **HTMLTableElement** и т.д.

Текстовое содержимое (узлы) представлено как экземпляр объекта Text. Все объекты, представляющие элементы страницы являются потомками объекта HTMLElement.

Такое представление содержимого называется **объектной моделью документа (Document Object Model), сокращенно DOM.**

```
<html>
<head>
<title>Представление Объектной модели документа
  (DOM)</title>
</head>
<body class="main">
<h1 id="header">Заголовок</h1>
<p>Какой-то <b>важный</b> текст</p>
</body>
</html>
```



- Структура HTML-страницы часто изображается в виде генеалогического древа, где одни теги включают в себя другие и называются родительскими (предками), а вложенные теги - дочерними (потомками). Теги h1 и p называются сестринскими (братскими) и они также являются дочерними элементами по отношению к body.

Война браузеров

- Объектная модель документов сама по себе не является частью диалекта JavaScript, это стандарт консорциума W3C, к которому производители большинства браузеров привели свои программы.
- Объектная модель документов позволяет JavaScript обмениваться информацией с веб-страницей и изменять на ней HTML-код. Но проблема в том, что Web-обозреватели по-разному поддерживают объектную модель документа.
- К примеру, Internet Explorer (IE) никак не обрабатывает промежутки между тегами, незаполненные текстом, а Opera и Firefox подсчитают их за пустой текст. Web-сценарий тоже представлены по-разному.

Получение доступа к узлам

- Для того, чтобы получить доступ к отдельному элементу на странице нужно его как-нибудь обозвать.
- В примере выше мы присвоили тегу <h1> необязательный атрибут id со значением header и теперь можем получить к нему доступ.
- Существует три способа: прямой доступ; через коллекции и с помощью свойств и методов объектной модели документа (DOM).
- *Работа с последним - является хорошим тоном программирования.*

Прямой доступ

- Используя такой способ, мы обращаемся к элементу прямо по имени, затем пишем свойство или метод:

```
header.style.color = '#cc0000';
// устанавливаем бордовый цвет заголовка
```

Доступ через коллекции

Коллекция - представлена в виде ассоциативного массива. Объект HTMLdocument поддерживает большое количество коллекций:

// Экземпляры объекта HTMLCollection, кроме последнего

- all Все элементы страницы
- anchors Все якоря страницы
- applets Все элементы ActiveX
- embeds Все модули расширения
- forms Все Web-формы
- images Все графические изображения
- links Все гиперссылки
- scripts Все Web-сценарии (только IE & Opera)
- styleSheets Все таблицы стилей

- Доступ к нашему элементу мы можем получить по строковому индексу, который совпадает с именем элемента страницы:

document.all['header'];

// получение доступа через коллекции

- Также доступ можно получить подставив числовой индекс элемента страницы. Например, код доступа к первой картинке на странице следующий:

document.images[0];

// доступ к самому первому изображению, если оно есть

Доступ с помощью свойств и методов объектной модели документа (DOM)

Существует два основных метода доступа к узлам:

getElementById()

getElementsByTagName()

Метод getElementById()

Находит нужный элемент с определенным идентификатором. В нашем случае заголовок h1 имеет уникальный id со значением header:

```
// объектная модель документа (DOM)
var ourHeader = document.getElementById('header');
ourHeader.innerHTML = 'Объектная модель документа';
```

Команда getElementById() - это метод объекта document, а 'header' - простой литерал переданный как параметр, обозначающий уникальность имени идентификатора. Причем в качестве параметра может быть и переменная.

В примере выше мы получили доступ к нашему заголовку и произвели его замену, используя свойство innerHTML.

Атрибут name

Аналогичный подход можно применить и с помощью атрибута name:

```
<p name="newAtr">Новый параграф</p>
```

В этом случае для получения доступа к узлу применяется метод **getElemenstByName(),** который возвращает массив экземпляров объекта HTMLElement с данным именем:

```
var newPar = document.getElementsByName('newAtr');
var result = newPar[0];
```

Свойства и методы объекта HTMLElement

Объектная модель документа предлагает несколько способов доступа к соседним узлам.

Рассмотрим их.

Свойство childNodes

Содержит все дочерние элементы по отношению к текущему и при этом вложены непосредственно в него.

Похож на массив, возвращенный методом **getElementsByTagName**:

```
// Объектная модель документа
  var head = document.getElementById('header');
// получаем доступ к тегу h1
var kinder = head.childNodes;
// находим дочерний узел (сам вложенный текст)
var textKinder = kinder[0].nodeValue;
// вытаскиваем текст с помощью свойства nodeValue
alert(textKinder);
// выводим результат в модальное окно
```

В нашем примере первым делом получаем доступ к заголовку h1 с уникальным идентификатором header. Первый и единственный дочерний элемент - сам текст.

Определяем его с помощью свойства **.childNodes**.

Стоит отметить, что мы получили только доступ к тексту, чтобы его вывести мы используем свойство

.nodeValue.

Свойство firstChild

- Возвращает первый дочерний элемент по отношению к текущему.
- Если дочерних элементов нет, возвращается значение null:

```
// Объектная модель документа
var head = document.getElementById('header');
var firstKind = head.firstChild;
var val = firstKind.nodeValue;
alert(val);
```

Свойство lastChild

- Возвращает последний дочерний элемент по отношению к текущему, т.е. **антипод** свойства **firstChild**.
- Если текущий элемент не содержит дочерних элементов, возвращается значение null:

```
// Объектная модель документа
    var lastKind = document.body.lastChild;
// ссылка на последний элемент тела страницы
```

Свойство parentNode

- Возвращает родительский элемент по отношению к текущему:

// Объектная модель документа

```
var head = document.getElementById('header');
var predok = head.parentNode;
```

// в нашем примере ссылаемся на тег body

Свойство nextSibling

- Указывает на узел, следующий за текущим. Если элемент последний, то возвращает значение null:

```
// Объектная модель документа
  var head = document.getElementById('header');
var nextel = head.nextSibling;
// в нашем примере ссылаемся на след. тег p
if(! nextel) {                    // если элемент последний, то...
          alert('Элемент является последним!');
          }
else {
     var val = nextel.lastChild.nodeValue;  // вытаскиваем
   последний дочерний элемент
     alert(val); // выводим результат в модальное окно
};
```

Свойство previousSibling

- Указывает на узел, предыдущий по отношению к текущему.
- Если элемент первый, то возвращает значение null:

```
// Объектная модель документа
var x = document.body.lastChild;
var prev = x.previousSibling;
```

Метод hasChildNodes

- Не принимает параметров и возвращает значение true, если находит дочерние элементы, в противном случае возвращает значение false:

```
// Объектная модель документа
var head = document.getElementById('header');
var nextel = head.nextSibling;
// в нашем примере ссылаемся на след. тег p
var result = nextel.hasChildNodes();
alert(result);
```

Определение событий

- Все действия пользователя (нажатие на кнопки клавиатуры, клики мыши или ее перемещение, загрузка страницы, наведение фокуса и т.п.), на которые реагирует веб-обозреватель, именуются **событиями**.
- Язык JavaScript - клиентский язык, т.е. событийно-управляемый. Без него страницы были бы не в состоянии отвечать на действия посетителя или предлагать что-либо интерактивное, динамичное и впечатляющее.
- Подготовка Web-страницы к ответу на события проходит в два этапа:

  - идентифицируем элемент страницы, реагирующий на событие;

  - присваиваем событие обработчику

Сразу отметим, что существуют разные модели обработки событий.

Одна из которых стандартизирована объектной моделью документа, ее еще называют модель Firefox, она более прогрессивная и в ней больше возможностей, но зато не поддерживается Internet Explorer (IE < 8). Другие более простые, но зато поддерживается всеми современными Web-обозревателями. Рассмотрим их.

Встроенные javascript события

- Один из самых простых и непрофессиональных способов исполнения функции в момент запуска события называют регистрацией встроенных событий, когда обработчик события присваивается прямо в HTML-код:

```
// javascript события
<p> Нажмите на ссылку и получите результат! А вот и сама
<a href="#" ONCLICK="alert('событие onclick javascript');">Ссылка на javascript события мыши</a>
</p>
```

Привязка через свойства объектов

Обработчики событий оформляются в виде функции в случае их привязки к событиям через соответствующие свойства объектов, представляющих элементы страницы:

```
<p id="par">Наведите курсор мышки на текст</p>
<script type="text/javascript">
var text = document.getElementById('par');
text.onmouseover = function() {
                        this.style.color = '#ff0000';
                      };
text.onmouseout = function() {
                        this.style.color = '#000000';
                      };
</script>
```

Привязка через свойства объектов

```
// javascript события  (пример №2)

   <html>
   <head>
   <meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
   <title>Введение в JS</title>
<script type="text/javascript">
function parClick() {
        alert('Вы кликнули по абзацу и получили ответ от javascript события мыши');
                };
</script>
  </head>
  <body>
<p id="par2">Кликни меня</p>
<script type="text/javascript">
var text = document.getElementById('par2');
text.onclick = parClick;
</script>
  </body>
  </html>
```

Привязка обработчика к событию с помощью функции-слушителя

- Стандартами DOM рекомендуется использование именно этого способа, но, к большому сожалению, не все Web-обозреватели его поддерживают.
- В этой модели привязка к заданному элементу страницы и событию осуществляется с использованием метода **addEventListener()** объекта HTMLElement:

```
// javascript события
    <html>
  <head>
  <meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
  <title>javascript события</title>
<script type="text/javascript">
                var n = 0;
function parClick() {
                var val = document.getElementById('par3');
                if(n == 1) {
                        val.style.color = '#000000';
                        n--;
                    }
                else {val.style.color = '#ff0000';
                        n++;
                    };
            };
</script>
 </head>
 <body>
<p id="par3">Кликни меня два раза или более.</p>
<script type="text/javascript">
var text = document.getElementById('par3');
text.addEventListener('click', parClick, false);
</script>
 </body>
 </html>
```

В качестве первого параметра передается имя события в виде строки формата DOM (строка кода №25). Вторым параметром передается сама функция-слушитель (parClick). Третий параметр указывает Web-обозревателю, следует ли перехватывать события, возникающие в дочерних по отношению к текущему элементах страницы (булевое значение false - отключает перехват, true - включает).

Метод removeEventListener объекта HTMLElement позволяет удалить подключенную ранее функцию-слушателя:

```
// javascript события
  text.removeEventListener('click', parClick, false);
```

Internet Explorer использует метод **attachEvent()** для выполнения той же задачи:

```
// javascript события
  text.attachEvent('onclick', parClick);
```

сделаем так, чтобы работало и в Explorer:

```
// javascript события
    <html>
  <head>
  <meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
  <title>javascript события</title>
<script type="text/javascript">
var textColor = ['#330000','#006600','#cc0000',‘
#3366ff','#660033','#cc6600'];
var textBgColor = ['#99fff','#cccc66','#33ffff','#cc9999',
'#ff9999'];
function parClick() {
            var val = document.getElementById('par3');
            val.style.color = textColor[Math.round(Math.random() * 6)];
            val.style.backgroundColor = textBgColor[Math.round(Math.random() * 5)];
          };



</script>
  </head>
  <body>
```

Объект Event

- Web-обозреватели позволяют нам получить дополнительную информацию о событиях, например, о том, была ли нажата клавиша или координаты курсора мыши.
- Для этих целей объектная модель документа предусматривает особый объект **Event**, который поддерживает весьма большой набор свойств, дозволяющих нам отслеживать каждое наступившее событие. Следует также отметить, что получение такой информации в разных браузерах выполняется по разному и модель обработки события в этом случае не играет никакой роли!

Предположим, что у нас есть элемент DIV, в котором мы поместили абзац со ссылкой внутри:

```
<div id="main">
<p id="par1">Это абзац со ссылкой внутри
<a id="link1"
href="10-2.html">ССЫЛКА</a></p>
</div>
```

Теперь присвоим обработчик событию щелчка по ссылке, где пропишем, что никуда переходить нам не нужно и выведем сообщение в модальное окно:

```
// Всплытие и перехват событий   *** JS-код ***
  window.onload = scriptAfterLoad; // выполнит ф-цию scriptAfterLoad после загрузки стр.

function scriptAfterLoad() {
        var link1 = document.getElementById('link1');
        if(link1.addEventListener) {
        link1.addEventListener('click', clickLink, false);
        }
        else {
        link1.attachEvent('onclick', clickLink)
        };
}; // end scriptAfterLoad Func
  function clickLink(event) {
        alert('Кликнули по ссылке!');
        (window.event) ? event.returnValue = false : event.preventDefault();
}; // end Func
```

- Если вы сохраняете сценарий в отдельный файл, который прикрепляете к странице между парным тегом **head** или просто прописываете вначале, то нужно исполнить его только после загрузки страницы.
- В этом случае нам поможет событие **load** объекта **window**. После этого первым делом идентифицируем наш элемент - ссылку.
- После определяем метод с которым будем работать: **attachEvent()** для Internet Explorer или **addEventListener()** для остальных, поддерживающих стандарты консорциума W3C, внутри присваиваем событие, на которое нужно реагировать и функцию обработчик **clickLink**.
- В обработчике с помощью команды **alert()** выводим в модальное окно сообщение, что кликнули именно по ссылке и далее пишем условие, в зависимости от типа браузера, т.к. в Web-обозревателе Firefox свои свойства и методы для событий, которые отличаются от остальных, но есть и схожие, правда их не так много.
- Чтобы отменить поведение для данного события в Firefox используют вызов метода **preventDefault** объекта **Event**, который не принимает никаких параметров и ничего не возвращает.
- Для этих же целей в остальных браузерах поддерживается свойство **returnValue**. Значение false отменяет поведение.

Теперь представим, что к тегу <p> вам нужно привязать еще одно событие click. Допишем наш скрипт и выведем на экран сообщение о щелчке по абзацу:

```
// Всплытие и перехват событий   *** JS-код ***
  window.onload = scriptAfterLoad;
  function scriptAfterLoad() {
     var mainDiv = document.getElementById('main');
     mainDiv.style.border = '1px solid #cc0000'; // сделали рамку для div
          var link1 = document.getElementById('link1');
     if(link1.addEventListener) {
     link1.addEventListener('click', clickLink, false);
     }
     else {
     link1.attachEvent('onclick', clickLink)
     };
```

```
//*************************************************************//
        var par1 = document.getElementById('par1');
        par1.style.background = '#f9f9f9'; // background для абзаца
        if(par1.addEventListener) {
        par1.addEventListener('click', clickPar, false);
        }
        else{
              par1.attachEvent('onclick', clickPar)
              };
}; // end scriptAfterLoad Func


function clickPar(event) {                  // для ссылки
        alert('Кликнули по абзацу!');
}; // end Func

  function clickLink(event) {                  // для абзаца
        alert('Кликнули по ссылке!');
        (window.event) ? event.returnValue = false : event.preventDefault();
}; // end Func
```

Все необходимые сведения о Web-обозревателе и системе у пользователя можно узнать при помощи объекта **Navigator**.

Данный объект поддерживаем множество свойств и один бесполезный метод javaEnabled, который не принимает параметров и возвращает true, если выполняются сценарии JavaScript и false - в противном случае:

- appCodeName Возвращает имя исходного кода программного ядра обозревателя.
- appName Возвращает имя программы обозревателя.
- appVersion Возвращает версию программы обозревателя
- browserLanguage Возвращает код языка программы обозревателя*
- cookiEnabled Возвращает true, если разрешен прием куки, false - нет.
- cpuClass Возвращает наименование процессора клиентского компьютера*
- lanquage Возвращает код языка программы обозревателя*
- onLine Возвращает true, если клиент подключен к инету, false - нет*
- platform Возвращает обозначение операционной системы клиентского компьютера.
- systemLanquageВозвращает код языка операционной системы клиента.
- userAgent Возвращает строку индексирующую обозреватель
- userLanquage Аналог browserLanquage*

Примечание * - свойство поддерживается отдельными браузерами.