Оглавление

Глава 1. Что такое язык JavaScript

JavaScript - это язык программирования, используемый в составе страниц HTML для увеличения функциональности и возможностей взаимодействия с пользователями. Он был разработан фирмой Netscape в сотруднечестве с Sun Microsystems на базе языка Sun's Java .С помощью JavaScript на Web-странице можно сделать то, что невозможно сделать стандартными тегами HTML. Скрипты выполняются в результате наступления каких-либо событий, инициированных действиями пользователя.

Несмотря на отсутствие прямой связи с языком Java, JavaScript может обращаться к внешним свойствам и методам Java- апплетов, встроенных в страницу HTML. Разница сводится к тому, что апплеты существуют вне браузера, в то время как программы JavaScript могут работать только внутри браузера. На первой взгляд кажется, что найти информацию по JavaScript несложно. Сначала создается впечатление, что ее можно увидеть везде: на сервере Natscape, в виде электронных руководств и примеров, во многих других местах. Тем не менее разыскать информацию об объектах, операторах, цветах и всем прочем в одном источнике, чтобы она была всегда под рукой, трудно.

JavaScript - это язык для создания активных клиентских страниц: с его помощью можно изменять содержимое HTML-документов, управлять анимацией без использования каких-либо дополнительных средств, проверять введенные пользователем в форму значения без ее пересылки на сервер, выполнять сложные математические вычисления, поиск по Web-узлу и т.п.

Так как программы на JavaScript выполняются на клиентском компьютере, вопросы защищенности информации выступают на первый план. С помощью JavaScript нельзя читать клиентские файлы и записывать что-либо на диск, за некоторыми исключениями.

Тем не менее язык JavaScript полностью отвечает потребностям большинства Web-мастеров - это простой и мощный язык, позволяющий превратить статические HTML-документы в интерактивные.

Глава 2. Использование JavaScript в HTML

Чтобы запускать скрипты, написанные на языке JavaScript вам понадобится браузер, способный работать с JavaScript - например Netscape Navigator или Microsoft Internet Explorer (MSIE). С тех пор, как оба этих браузера стали широко распространеными, множество людей получили возможность работать со скриптами, написанными на языке JavaScript. Несомненно, это важный аргумент в пользу выбора языка JavaScript, как средства улучшения ваших Web-страниц. Перед изучением языка JavaScript вы должны познакомиться с основами другого языка - HTML. При этом, возможно, Вы обнаружите, что много хороших средств диалога можно создать, пользуясь лишь командами HTML.

В отличие от Java-аплетов и элементов ActiveX, загружаемых отдельно от документа, в котором они используются, программы, написанные на языке JavaScript, располагаются непосредственно в HTML-документах. Для этого используется специальный тэг <SCRIPT> и парный ему </SCRIPT>:

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

Атрибут LANGUAGE указывает, на каком языке написана данная программа, - в нашем случае это JavaScript. Для того что-бы браузеры, не поддерживающие скриптовые программы, могли пропустить их, программы располагаются внутри блока комментариев:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
...
```

программа на JavaScript
...
//-->
</SCRIPT>
Обычно функции, составляющие программу, располагаются в секции <HEAD> HTML-документа. Так как эта секция загружается первой, гарантируется, что такие функции будут загружены раньше, чем пользователь сможет их вызвать с помощью тех или иных интерфейсных средств, располагаемых в секции <BODY>. Посмотрим, как это выглядит внутри HTML-документа:
<HTML>
<HEAD>
<TITLE>Пример программы на JavaScript<TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
...
программа на JavaScript
...
//-->
</SCRIPT>
</HEAD>
<BODY>
...
Текст HTML-документа и вызов функций на JavaScript
...
</BODY>
</HTML>
В языке JavaScript существует два типа комментариев. К первому относятся однострочные комментарии, выделяемые в тексте символами "//":
// Эта строка - комментарий;
или
askUser(); //запросить данные от пользователя
Ко второму типу относятся многострочные комментарии:
/*
Это - многострочный комментарий, который полностью
игнорируется интерпретатором JavaScript
*/

Задание 1

1. Напишите тэги, внутри которых располагаются команды JavaScript

Глава 3. Основные элементы языка - переменные, функции, объекты, методы

Переменные

JavaScript распознает следующие типы величин:
§ Числа, типа 42 или 3.14159
§ Логические (Булевы), значения true или false
§ Строки, типа "Howdy!"
§ Пустой указатель, специальное ключевое слово, обозначающее нулевое значение
Это относительно малый набор типов значений, или *типов данных*, которые позволяют вам выполнять функции в ваших приложениях. Не существует никакого явного различия между целыми числами и реально-оцененными числа.

Как и в других языках программирования, в *JavaScript* термин "переменная" означает *нечто, значение чего может быть изменено*. Переменной может быть число, слово, последовательность символов или любая их комбинация. Для определения переменной используется ключевое слово **var**:

```
var x=3.14159;
var y=2;
var z=3;
```

Здесь вы присвоили *числовым* переменным x значение 3.14159, y - 2 и z - 3. Поскольку x, y и z являются переменными, их значения могут быть в какой-то момент изменены:

```
x=y+z; //x=5 (2+3)
```

В *JavaScript* существуют и *строковые* переменные. Они могут содержать набор символов или символы и цифры. Строковые переменные задаются с помощью кавычек:

```
var str1="String variable";
var str2="Another string variable";
```

Для задания кавычек внутри строковых переменных следует использовать символ "\":

```
var str3="That\'s Ok!":
```

С помощью этого же символа "\" в строковую переменную можно включать специальные и управляющие символы, например:

```
\r - возврат каретки
\n - переход на новую строку
\t - табуляция
```

Правила именования

Имена переменных и функций в *JavaScript* должны начинаться с буквы ("A"-"Z", "a"-"z") или подчеркивания ("_"). Последующие символы могут быть цифрами (0-9) и буквами.

Преобразование типов данных

Тип переменной зависит от того, какой тип информации в ней хранится. *JavaScript* не является жестко типизированным языком. Это означает, что вы не должны точно определять тип данных переменной, в момент ее создания. Тип переменной присваивается переменной автоматически в течение выполнения скрипта. Так, например, вы можете определить переменную следующим образом:

```
var answer = 42
```

А позже, вы можете присвоить той же переменной, например следующее значение:

```
answer = "Thanks for all the fish..."
```

Вообще, в выражениях, включающих числовые и строковые значения, *JavaScript* преобразовывает числовые значения в строковые. Например, рассмотрим следующие утверждение:

```
x = "The answer is - " + 42
y = 42 + " - is the answer."
```

Первое утверждение будет строка "The answer is - 42 ". Второе утверждение возвращает строку " 42 - is the answer".

JavaScript предоставляет несколько специальных функций для управления строковыми и числовыми значениями:

§ eval вычисляет строку, представляющую любые *JavaScript* литералы или переменные, преобразовывая ее в число.

§ parseInt преобразовывает строку в целое число в указанном основании системы счисления, если возможно.

§ parseFloat преобразовывает строку в число с плавающей точкой, если возможно.

Область действия переменных

Область действия переменных - то, где вы можете использовать их в скрипте. В *JavaScript* , существует две области действия, которые переменные могут иметь:

Глобальная: Вы можете использовать переменную где-нибудь в приложении.

Локальная: Вы можете использовать переменную внутри текущей функции.
Чтобы объявить локальную переменную внутри функция, используйте ключевое слово **var**, например:

```
var total = 0
```

Чтобы объявить глобальную переменную, объявите переменную назначения, которая просто присваивает значение переменной (или в функции или вне функции), например:

```
total = 0
```

Лучше всего объявлять глобальные переменные в начале вашего скрипта, так, чтобы функции наследовали переменную и ее значение.

Целые числа (Integers)

Целыми называют числа вида 1, 164, 102390. Они могут быть выражены в десятичном (по основанию 10), шестнадцатеричном (по основанию 16), или восьмеричном (по основанию 8) представлении. Десятичный литерал целого числа состоит из последовательности цифр без ввода 0 (ноля).
Целое число может быть выражено в восьмеричном или шестнадцатеричном скорее чем в десятиричное. Шестнадцатеричные числа включают цифры 0-9 и буквы a-f и A-F, в *JavaScript* они записываются с комбинацией символов 0x или 0X (ноль-x) перед числом. Восьмиричные числа включают только цифры 0-7 и в *JavaScript* записываются с ведущего нуля.
Например, десятичное число 23 представляется в шестнадцатеричном виде как 0x17 и в восьмеричном как 027

Литералы с плавающей точкой

Литералы с плавающей точкой представляют собой дробные части целых чисел и должны включать в себя по крайней мере одну цифру и десятичную точку либо символ экспоненты ("e" или "E"). В следующих примерах приведены различные варианты представления одного и того же числа:
§ 3.1415927
§ 31415927e-7
§ .31415927E1

Логические Литералы

Логические значения имеют только два значения, **истина (true)** или **ложь (false)**. В некоторых реализациях языка *JavaScript* 0 (false) и 1 (true) не могут быть использованы в качестве логических значений.

Строки

Строковые литералы - ноль или большее количество знаков, расположенные в двойных (") или одинарных (') кавычках. Строки должен быть разделены кавычками того же самого типа; то есть или обе одинарные кавычки или двойные кавычки. Использование обратной двойной черты "\" позволяет вставлять в строку специальные символы. Приведем примеры строковых литералов:
§ "Blah"
§ 'Blah'
§ "1234"
§ "one line \n another line"

Зарезервированные слова

В языке *JavaScript* имеется ряд зарезервированных ключевых слов. Они подразделяются на три катэгории: слова, зарезервированные в *JavaScript* (табл. 1), слова, зарезервированные в языке Java (табл. 2), и слова, использования которых следует избегать.

Таблица 1. Слова, зарезервированные в JavaScript				
break	for	new	true	with
continue	function	null	typeof	

else	if	return	var	
false	in	this	while	

Таблица 2. Слова, зарезервированные в языке Java				
abstract	default	implements	static	void
boolean	do	import	super	
byte	double	instanceof	switch	
case	extends	int	synchronized	
catch	final	interface	throw	
char	finally	long	throws	
class	float	native	transient	
const	goto	package	try	

Следует избегать использования названий встроенных объектов и статических функций, таких, например, как String или parseInt.

Объектная модель JavaScript

JavaScript основан на простом объектно-ориентированном примере. Объект - это конструкция со свойствами, которые являются переменными *JavaScript*. Свойства могут быть другими объектами. Функции, связанные с объектом известны как *методы* объекта.

В дополнение к объектам, которые сформированы в Navigator client и LiveWire server, вы можете определять ваши собственные объекты.

- Объекты и Свойства
- Функции и Методы
- Создание Новых Объектов

Объекты и Свойства

Объект *JavaScript* имеет свойства, ассоциированные с ним. Вы обращаетесь к свойствам объекта следующей простой системой обозначений:

```
objectName.propertyName
```

И имя объекта и имя свойства чувствительны к регистру. Вы определяете свойства, приписывая значение. Например, пусть существует объект, с именем *myCar* (мы обсудим, как создавать объекты позже - теперь, только принимаем, что объект уже существует). Вы можете дать свойства, именованные **make** , **model** , и **year** следующим образом:

```
myCar.make = "Ford"
myCar.model = "Mustang"
myCar.year = 69;
```

Вы можете также обратиться к этим свойствам, используя систему обозначений таблицы следующим образом:

```
myCar["make"] = "Ford
myCar["model"] = "Mustang"
myCar["year"] = 69;
```

Этот тип таблицы известен как ассоциативная таблица, потому что каждый элемент индекса также связан со значением строки. Чтобы пояснять, как это делается, следующая функция

показывает свойство объекта, когда вы проходите объект и имя объекта как аргументы функции:
Так, обращение к функции `show_props(myCar, "myCar")` возвращает следующее:
myCar.make = Ford
myCar.model = Mustang
myCar.year = 67
Вы можете также определять свойства, используя порядковые числа, например:

temp[0] = 34
temp[1] = 42
temp[2] = 56

Эти утверждения создают три свойства объекта *temp*, и вы должны обращаться к этим свойствам как temp[*i*], где *i* - целое число между 0 и 2.

Функции и Методы

Функции - один из фундаментальных встроенных блоков в *JavaScript*. Функция - *JavaScript* процедура - набор утверждений, которые выполняют определенную задачу.
Определение функции состоит из ключевого слова **function** , сопровождаемого

- Именем функции
- Списком аргументов функции, приложенной в круглых скобках, и отделяемые запятыми
- *JavaScript* утверждениями, которые определяют функцию, приложенные в фигурных скобках, {...}

Вы можете использовать любые функции, определенные в текущей странице. Лучше всего определять все ваши функции в `HEAD` страницы. Когда пользователь загружает страницу, сначала загружаются функции.
Утверждения в функциях могут включать другие обращения к функции.
Например, есть функция с именем pretty_print: function pretty_print(string) { document.write(" " + string) }
Эта функция принимает строку как аргумент, прибавляет некоторые тэги HTML, используя оператор суммы (+), затем показывает результат в текущем документе.
Определение функции не выполняет ее. Для этого вы должны *вызвать* функцию, чтобы выполнить ее. Например, вы можете вызывать функцию pretty_print следующим образом: pretty_print("This is some text to display")
Аргументы функции не ограничены только строками и числами.
Аргументы функции сохраняются в таблице. Внутри функции, вы можете адресовать параметры следующим образом:

```
functionName.arguments [i]
```

Где *functionName* - имя функции, и *i* - порядковое число аргумента, начинающегося с нуля. Так, первый аргумент в функции, с именем myfunc, будет myfunc.arguments [0]. Общее число аргументов обозначено переменным arguments.length.
Функция может даже быть рекурсивной, то есть она может вызывать себя. Например, существует функция, которая вычисляет факториалы:

```
function factorial(n)
 {
 if ((n == 0) || (n == 1))
   return 1
 else {
```

```
    result = (n * factorial(n-1))
    return result
  }
}
```

Вы можете показывать факториалы от одного до пяти следующим образом:

```
for (x = 0; x < 5; x++) {
  document.write(x, " factorial is ", factorial(x))
  document.write("")
}
```

Результаты будут следующие:
факториал нуля - 1
факториал единицы- 1
факториал двойки - 2
факториал тройки - 6
факториал четверки - 24
факториала пятерки - 120

Функции с Переменными Числовыми Аргументами

Вы можете вызывать функцию с большим количеством аргументов, чем она формально объявлена, используя массив *arguments* . Это часто полезно тогда, когда вы не знаете заранее, сколько аргументов будут в функции. Вы можете использовать arguments.length, чтобы определить число аргументов в функции, и затем обращаться к каждому аргументу, используя массив *arguments* .

Например, рассмотрим функцию, определенную, чтобы создать списки HTML. Единственый формальный аргумент функции - строка, которая является "U", если список неупорядочен или "O", если список упорядочен (пронумерован). Функция определена следующим образом:

```
function list(type) { document.write("<" + type + "L") //начинается список for (var i = 1; i < list.arguments.length; i++) // Повторить через аргументы document.write("" + list.arguments[i]) document.write("</" + type + "L") // заканчивается список }
```

Вы можете проходить любое число аргументов этой функции, и затем показывать каждый аргумент как каждый отдельный пункт в обозначенном типе списка. Например, следующий запрос на функцию:

```
list("o", "one", 1967, "three", "etc, etc...")
```

1. o
2. one
3. 1967
4. three
5. etc, etc ...

Определение Методов

Метод - функция, связанная с объектом. Вы определяете метод таким же образом, как вы определяете стандартную функцию. Затем, используйте следующий синтаксис, чтобы связать функцию с существующим бъектом:

```
object.methodname = function_name
```

Где *object* - существующий объект, *methodname* - имя, которое вы присваиваете методу, и *function_name* - имя функции.

Вы можете вызывать метод в контексте объекта следующим образом:

```
object.methodname (params);
```

Использование this для ссылок Объекта

JavaScript имеет специальное ключевое слово **this**, которое вы можете использовать, чтобы обращаться к текущему объекту. Например, пусть у вас есть функция с именем *validate* , которая проверяет правильность свойства значения объекта, данного объект, и high и low значения:

```
function validate(obj, lowval, highval)
 {
  if ((obj.value < lowval) || (obj.value > highval))
    alert("Invalid Value!")
}
```

Вы можете вызывать *validate* в каждом элементе формы обработчика событий onChange, используя **this.** Вообще, метод **this** обращается к вызывающему объекту.

Создание Новых Объектов

И клиент и сервер *JavaScript* имеют строки предопределенных объектов. Кроме того, вы можете создавать ваши собственные объекты. Создание вашего собственного объекта требует двух шагов:

- Определить тип объекта, написанной функции.
- Создать образец объекта с **new**.

Чтобы определять тип объекта, создайте функцию для типа объекта, которая определяет его имя, и его свойства и методы. Например, пусть вы хотите создавать тип объекта для автомобилей. Вы хотите этот тип объектов, который будет назван *car*, и Вы хотите, чтобы он имел свойства для make, model, year. Чтобы сделать это, вы должны написать следующую функцию:

```
 function car(make, model, year)
{
  this.make = make;
  this.model = model;
  this.year = year;
}
```

Замечание, используйте **this**, чтобы присвоить значения свойствам объекта, основанные на значениях функции.

Теперь вы можете создавать объект, с именем *mycar* следующим образом:

```
 mycar = new car("Eagle", "Talon TSi", 1993);
```

Это утверждение создает *mycar* и присваивает ему указанные значения для его свойств. Затем значение `mycar.make` - строка "Eagle", `mycar.year` - целое число 1993, и так далее.

Вы можете создавать любое число объектов *car* запрашивая к **new**. Например,

```
kenscar = new car("Nissan", "300ZX", 1992)
```

Объект может иметь свойство, которое является самостоятельным другим объектом. Например, пусть вы определили объект с именем *person* следующим образом:

```
function person(name, age, sex)
 {
  this.name = name;
  this.age = age;
```

```
  this.sex = sex;
}
```

И затем подтверждаете два новых объектов *person* следующим образом:

```
rand = new person("Rand McNally", 33, "M")
ken = new person("Ken Jones", 39, "M")
```

Затем вы можете перезаписать определение *car*, чтобы включить свойство владельца, которое берет объект *person*, следующим образом:

```
function car(make, model, year, owner)
{
  this.make = make;
  this.model = model;
  this.year = year;
  this.owner = owner;
}
```

Затем вы используете следующее:

```
car1 = new car("Eagle", "Talon TSi", 1993, rand);
car2 = new car("Nissan", "300ZX", 1992, ken)
```

Заметим, что вместо прохождения строкового литерала или целого числа вычисляет при создании новых объектов, вышеупомянутый ход утверждений объектов rand и ken как аргументов владельцев. Затем, если вы хотите выяснять имя владельца car2, вы можете обращаться к следующему свойству:

```
 car2.owner.name
```

Заметьте, что вы можете всегда прибавлять свойства к предопределенному объекту. Например, утверждение:

```
 car1.color = "black"
```

Прибавляет свойство color к car1, и присваивает ему значение " black". Однако, это не воздействует на любые другие объекты. Чтобы прибавить новое свойство ко всем объектам того же самого типа, вы должны прибавить свойство к определению типа объекта *car*.

Определение Методов

Вы можете определять методы для типа объекта включением определение метода на определении типа объекта. Например, пусть у вас есть набор файлов изображений GIF, и вы хотите определить метод, который показывает информацию для car, наряду с соответствующим изображением. Вы можете определить функцию типа:

```
function displayCar()
{
  var result = "A Beautiful " + this.year
           + " " + this.make + " " + this.model;
  pretty_print(result)
}
```

Где *pretty_print* - предопределенная функция, которая показывает строку. Используйте this, чтобы обратиться к объекту, который принадлежит методу.

Вы можете делать функцию методом из *car*, прибавляя утверждение

```
 This.displayCar = displayCar;
```

к определению объекта. Так, полное определение *car* теперь выглядит так:

```
function car(make, model, year, owner)
{
  this.make = make;
  this.model = model;
  this.year = year;
  this.owner = owner;
  this.displayCar = displayCar;
}
```

Вы можете вызывать этот новый метод следующим образом:

```
car1.displayCar ()
car2.displayCar ()
```

Это будет выглядеть подобно следующему выводу:

A Beautiful 1993 Eagle Talon TSi

A Beautiful 1992 Nissan 300ZX

Задание 2

1. Создайте переменную str и присвойте ей значение "Привет, Мир!".
2. Создайте переменную numb и присвойте ей значение 100.
2. Создайте функцию Func(), которая возвращает false.

Глава 4. Вывод текста в окна и фреймы. Изменение некоторых свойств окон

Мы рассмотрели такие понятия, как функции, переменные, комментарии, правила именования переменных и функций, а также перечислили зарезервированные слова. Можно двигаться дальше.

Если предыдущая часть была чисто теоретической, то теперь мы перейдем к практике. И начнем с того, что рассмотрим, как с помощью языка JavaScript вывести что-нибудь на экран. Для этого мы будем использовать метод write объекта document. Собственно говоря, рассмотрение объектов, предоставляемых браузером (будь то Netscape Navigator или Microsoft Internet Explorer), - тема одного из следующих уроков. Поэтому мы будем считать, что есть некоторый объект document, представляющий собой всю HTML-страницу, отображаемую в данный момент в окне браузера, и у этого объекта есть метод write, позволяющий выводить что-нибудь на экран.

Метод **write** имеет всего один аргумент, с помощью которого и задается информация, выводимая на экран. Это может быть строка (заключенная в кавычки), переменная или комбинация строки и переменной. Приведем пример использования всех трех типов аргументов.

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--
  var DemoStr = "Строковая переменная";
  // Сначала выведем строку
  document.write( "Выводим строку");
  // Затем - содержимое строковой переменной
  document.write( DemoStr);
  // А потом - строку и переменную
```

```
  document.write( "<B>"+DemoStr+"</B>");
//-->
</SCRIPT>
</BODY>
</HTML>
```

Отметим, что для объединения строковой переменной и строки мы использовали символ "+", а также поместили в эту строку тэги языка HTML - содержимое строковой переменной DemoStr выводится выделенным шрифтом.

Как вы, наверное, уже догадались, возможность использования метода **write** и комбинации строк и строковых переменных, включая возможность вывода тэгов языка HTML, является первым шагом на пути к созданию динамических страниц - HTML-страниц, содержимое которых может изменяться в зависимости от действий пользователя.

Помимо вывода непосредственно в текущий документ, можно осуществлять вывод в один из фреймов из набора фреймов или в другое окно.

Как видно из результатов работы приведенного примера, все три строки выводятся одна за другой, без разделителей и символов перевода строки. У объекта **document** существует метод **writeln**, который обладает функциональностью, аналогичной методу **write**, за исключением того, что после вывода строки происходит переход на новую строку. Ряд браузеров игнорирует метод **writeln**. В этом случае можно дополнить строку символами "перевод каретки" и "возврат строки" или поместить после строки тэг начала параграфа, как это показано в следующем примере:

```
<HTHL>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--
  var DemoStr = "Строковая переменная";
  var P   = "<P>";
  // Сначала выведем строку
  document.write( "Выводим строку"+P);
  // Затем - содержимое строковой переменной
  document.write( DemoStr+P);
  // А потом - строку и переменную
  document.write( "<B>"+DemoStr+"</B>"+P)
//-->
</SCRIPT>
</BODY>
</HTML>
```

Выше мы упомянули о возможности управления содержимым одного из фреймов. Давайте посмотрим, как это сделать. Сперва создадим страницу, состоящую из двух фреймов:

```
Файл A.HTM
<HTML>
<HEAD>
<TITLE>Frames Page A</TITLE>
</HEAD>
<BODY>
  Page A
</BODY>
</HTML>

Файл B.HTM
```

```
<HTML>
<HEAD>
<TITLE>Frames Page B</TITLE>
</HEAD>
<BODY>
  Page B
</BODY>
</HTML>
```

Затем создадим базовую страницу, на которой будут отображаться наши фреймы, и сохраним ее в файле с именем MAIN.HTM:

```
<HEAD>
<TITLE>Frames Demo</TITLE>
</HEAD>
<FRAMESET COLS="50%,50%">
  <FRAME SRC="A.HTM">
  <FRAME SRC="B.HTM">
</FRAMESET>
</HTML>
```

Предположим, нам нужно, чтобы программа, написанная на JavaScript и расположенная в файле A.HTM, выводила что-либо в фрейм, описанный в файле B.HTM. Для этого необходимо написать следующий код:

```
<HEAD>
<TITLE>Frames Page A</TITLE>
</HEAD>
<BODY>
  Page A
<SCRIPT LANGUAGE = "JavaScript">
<!--
  parent.frames[1].document.write( "Вывод в фрейм");
//-->
</SCRIPT>
</BODY>
</HTML>
```

Здесь мы используем тот факт, что объект **parent** хранит массив фреймов. Мы обращаемся к одному из элементов этого массива (второму в нашем примере, так как массив начинается с 0) и вызываем метод **document.write**.

Остался один нюанс. Отображаемый в правом фрейме, находится в открытом состоянии. Было бы правильнее написать следующий код:

```
<HTML>
<HEAD>
<TITLE>Frames Page A</TITLE>
</HEAD>
<BODY>
  Page A

<SCRIPT LANGUAGE = "JavaScript">
<!--
  parent.frames[1].document.open;
  parent.frames[1].document.write( "Вывод в фрейм");
  parent.frames[1].document.close;
```

```
//-->
</SCRIPT>
</BODY>
</HTML>
```

Завершим наш урок еще несколькими примерами использования методов объекта **document**. Чтобы изменить цвет фона документа, достаточно изменить значение свойства **bgColor** объекта **document**:

```
  parent.frames[1].document.bgColor = "red";
```

или

```
  parent.frames[1].document.bgColor = "#c0c0c0";
```

Точно так же можно задать цвет текста - для этого необходимо изменить значение свойства **fgColor**:

```
  parent.frames[1].document.fgColor = "#707070";
```

Свойства **alinkColor**, **linkColor** и **vlinkColor** позволяют задать соответственно цвет активной ссылки, цвет ссылки и цвет ранее посещавшийся ссылки.

И последнее: как было сказано выше, вы можете активно пользоваться тэгами языка HTML. Поэтому вместо изменения значений атрибутов объекта **document** можно, например, написать следующий код:

```
<HTML>
<HEAD>
<TITLE>Frames Page A</TITLE>
</HEAD>
<BODY>
  Page A
<SCRIPT LANGUAGE = "JavaScript">
<!--
  parent.frames[1].document.open;
  parent.frames[1].document.write( "<BODY FGCOLOR='red'>" +
    "Hello" + "</BODY>");
  parent.frames[1].document.close;
//-->
</SCRIPT>
</BODY>
</HTML>
```

Задание 3

1. Создайте переменную str и присвойте ей значение "Привет, Мир!".
2. Создайте функцию HelloWorld(s), которая примет строку в качестве параметра и выведет ее на экран.

Глава 5. Операторы языка JavaScript

На этом занятии мы рассмотрим операторы языка JavaScript, с помощью которых можно создавать программы, действительно выполняющие полезные действия. Операторы языка тесно связаны с понятием *выражений* - синтаксических конструкций с использованием таких элементов языка, как for, if и while. Каждое выражение состоит из двух частей - *операндов*, задающих значения для *операторов*, с помощью которых указываются производимые над операндами действия.

Выражения в JavaScript могут быть разделены на несколько категорий - выражения присвоения, математические выражения и сравнения. Для каждой категории выражений существует набор операторов, которые и рассматриваются ниже.

Присвоение значений переменным

Самым распространенным выражением в этой категории является обычное присваивание с использованием оператора "=". В следующей таблице перечислены возможные операторы присваивания, поддерживаемые в JavaScript.

Таблица 1. Операторы присваивания в JavaScript

Оператор	**Действие**	**Пример**
=	Присваивает значение переменной	MyVar = 100;
+=	Увеличивает значение переменной на указанную величину	MyVar += 10;
-=	Уменьшает значение переменной на указанную величину	MyVar -= 10;
*=	Умножает значение переменной на указанную величину	MyVar *= 2;
/=	Делит значение переменной на указанную величину	MyVar /= 5;
%=	Делит значение переменной на указанную величину и возвращает остаток	MyVar %= 3;

Первый оператор в данной таблице - оператор обычного присваивания. Он используется для задания значения той или иной переменной. Отметим, что если переменная уже содержала какое-либо значение, то оно теряется и заменяется новым. Рассмотрим несколько примеров использования данного оператора:

```
MyStrVar = "String variable"; //присваивание значения строковой
                              //переменной
MyNumVar = 100;               //присваивание цифрового значения
```

Оператор "+=" может использоваться с цифровыми и строчными операндами. В первом случае происходит сложение значения, содержащегося в переменной с операндом, во втором - объединение двух строк в новую строку. Рассмотрим несколько примеров использования данного оператора:

```
MyVar = 100;
MyVar += 10;            // значение MyVar равно 110
MyStrVar = "Java";
MyStrVar += "Script"    // значение MyStrVar равно "JavaScript"
```

В табл. 2 приведены операции, выполняемые операторами, указанными в табл. 1.

Таблица 2. Операции, выполняемые операторами из табл. 1

Оператор	**Операция**
MyVal += Val	MyVal = MyVal + Val
MyVal -= Val	MyVal = MyVal - Val
MyVal *= Val	MyVal = MyVal * Val
MyVal /= Val	MyVal = MyVal / Val
MyVal %= Val	MyVal= MyVal % Val

Математические операторы

Для выполнения базовых математических операций в JavaScript существует набор операторов, которые перечислены в табл. 3.

Таблица 3. Математические операторы

Оператор	**Описание**	**Пример**
-Значение	Значение рассматривается как отрицательное число	MyVar = -10;
Значение1 +	Выполняется сложение Значение1 и Значение2. Возможно	MyVar = Var2 +

Значение2	использование для объединения строк	100; MySVar = "ABC" + "DEF";
Значение1 - Значение2	Выполняется вычитание Значения1 из Значения2	MyVar = Var2 - 100;
Значение1 * Значение2	Выполняется перемножение Значения1 на Значение2	MyVar = 10 * 12;
Значение1 % Значение2	Выполняется целочисленное деление Значения1 на Значение2 и выводится остаток от деления	MyVar = Var2 % Var1;
Значение1 / Значение2	Выполняется деление с плавающей точкой Значения1 на Значение2	MyVar = Var2 / 3;
Значение1++	Значение1 увеличивается на 1	MyVar++;
Значение1--	Значение1 уменьшается на 1	MyVar--;

Существует два варианта операций инкремента/декремента ("Значение1++/Значение1--"). Первый вариант - постфиксный, второй - префиксный:

```
MyVal = 100;
Val = MyVal++;     // Постфиксный: Val -> 100, MyVal -> 101
MyVal = 100;
Val = ++MyVal;     // Префиксный:  Val -> 101, MyVal -> 101
```

Операторы сравнения

Отдельную группу операторов языка JavaScript составляют операторы сравнения. Они перечислены в табл. 4.

Таблица 4. Операторы сравнения

Оператор	**Операция**
Значение1 == Значение2	Проверяет равенство Значения1 и Значения2
Значение1 != Значение2	Проверяет неравенство Значения1 и Значения2
Значение1 > Значение2	Проверяет больше ли Значение1, чем Значение2
Значение1 >= Значение2	Проверяет больше ли или равно Значение1 Значению2
Значение1 < Значение2	Проверяет меньше ли Значение1, чем Значение2
Значение1 <= Значение2	Проверяет меньше ли или равно Значение1 Значению2
Значение1 && Значение2	Выполняет логическую операцию И над Значением1 и Значением2
Значение1 || Значение2	Выполняет логическую операцию ИЛИ над Значением1 и Значением2
!Значение	Выполняет логическую операцию НЕ - инверсию значения

Результатом использования операторов сравнения всегда является булевская величина (имеющая значение true/false). Поэтому такие операторы обычно используются в конструкциях с if. Например:

```
if( MyVar > 10)
{
  break;
```

```
  }
  else
  {
    alert('MyVal <= 20');
  }
```

Как видно из приведенной выше таблицы, ряд операторов сравнения позволяет выполнить логческие операции над двумя операндами. Правила выполнения логических операций И и ИЛИ приведены в табл. 5.

Таблица 5. Правила выполнения логических операций

Операция И

Операнд1	Операнд2	Результат
false	false	false
false	true	false
true	false	false
true	true	true

Операция ИЛИ

Операнд1	Операнд2	Результат
false	false	false
false	true	true
true	false	true
true	true	true

Выражения с оператором "?"

В языке JavaScript поддерживается довольно удобный метод создания выражений с проверкой, заимствованный из языка C. Вот его синтаксис:

```
  (условие) ? istrue : isfalse;
```

Синтаксис данного выражения очень прост. Условие - выражение, значение которого вы тестируете, istrue - действия, выполняемые при значении выражения, равном true, а isfalse - действия, выполняемые при значении выражения, равном false. Отметим, что необходимо привести и то, и другое действие, разделенные символом ":". Рассмотрим следующий пример. Пусть необходимо, чтобы пользователь ввел какое-нибудь число, а затем нужно проверить - введено ли число или нажата кнопка Cancel. Вот пример программы на JavaScript, в которой для отображения панели ввода используется стандартная функция prompt, а разбор введенной информации происходит с помощью выражения с оператором "?":

```
<html>
<head><title>Пример на JavaScript</title>
<script language="JavaScript">
  function Test()
  {
    Res = prompt('Введите любое число', '');
    (Res == null) ? alert('Нажата кнопка Cancel') :
      alert('Введено число:' + Res);
  }
</script>
</head>
<body onLoad="Test();">
</body>
</html>
```

Битовые операторы

И последняя категория операторов, которую мы рассмотрим на этом занятии, - битовые операторы. Эти операторы используются только с числовыми операндами и выполняют операции над отдельными битами. Битовые операторы, реализованные в языке JavaScript, перечислены в табл. 6.

Таблица 6. Битовые операторы, реализованные в языке JavaScript

Оператор	Название	Выполняемые действия
&	Битовая операция И	Побитовая операция И над операндом
|	Битовая операция ИЛИ	Побитовая операция ИЛИ над операндом
^	Битовая операция И/ИЛИ	Побитовая операция И/ИЛИ над операндом
~	Битовая операция НЕ	Побитовая операция НЕ над операндом
<<	Сдвиг влево	Сдвиг значений битов влево на один или более бит
>>	Сдвиг вправо	Сдвиг значений битов вправо на один или более бит

Рассмотрим пример использования одного из битовых операторов.

```
<html>
<head><title>Пример на JavaScript</title>
<script language="JavaScript">
  function Test()
  {
    var tmp = 2;
    tmp = tmp << 4;    // значение равно 32
    alert(tmp);
  }
</script>
</head>
<body onLoad="Test();">
</body>
</html>
```

Порядок выполнения операторов

Как вы, наверное, смогли догадаться, в языке JavaScript возможно использование нескольких операторов в одном выражении. При этом существует *порядок вычисления* таких выражений. Сначала выполняется умножение и деление, затем - сложение и вычитание и т.п. Выбор операторов происходит слева направо.

Порядок вычисления операторов языка JavaScript приведен в табл. 7.

Таблица 7. Порядок вычисления операторов языка JavaScript

Порядок	Оператор
1	- (унарный минус), + (унарный плюс), ~ (битовое НЕ), ! (логическое НЕ), ++, --
2	* (умножение), / (деление), %
3	+ (сложение), - (вычитание)
4	<< (сдвиг влево), >> (сдвиг вправо)
5	< (меньше чем), <= (меньше или равно), > (больше чем), => (больше или равно), == (равно)
6	& (побитное И)
7	^ (побитное И/НЕ)

8	| (побитное ИЛИ)
9	&& (логическое И)
10	|| (логическое ИЛИ)
11	?:
12	Операторы присваивания
13	, (запятая)

Рассмотрим пример порядка выполнения операторов. Предположим, существует следующее выражение:

```
Res = 10 * ((20 + 30) /2)
```

Оно вычисляется в следующем порядке:

1. К 20 прибавляется 30.
2. Результат, полученный на первом шаге, делится на 2.
3. Результат, полученный на втором шаге, умножается на 10.

Задание 4

1. Создайте три переменные A, B, C. Изначально присвойте им значения 5, 100 и 500 соответственно.
2. Поменяйте знак переменной C.
3. Сложите A+B, при этом результат присвойте переменной D.
4. Поделите C на D, результат запишите в A;
5. Если A равно B выведите на экран "A == B", иначе выведите "A!=B";

Глава 6.Элементы управление логикой программы (ветвления, циклы и тд.)

For

Элемент for используется для организации блоков кода, которые должны выполняться несколько раз. Число повторов (итераций) контролируется с помощью специальной переменной. Синтаксис элемента for выглядит следующим образом:

```
for( StartVal; Condition; Inc)
```

где

- StartVal - начальное значение счетчика цикла. Обычно оно равно 0 или 1, но может принимать и любые другие значения. Например, Count = 0 или i = 1;
- Condition - выражение, используемое для контроля числа итераций цикла. Пока значение данного выражения истинно, цикл продолжается. Например, i<=9 истинно, пока значение i меньше или равно 9;
- Inc - указывает, на сколько должно увеличиваться значение счетчика StartVal. Например, i++ увеличивает значение i на единицу.

Отметим, что для разделения параметров элемента for используются символы ";", а не запятые - такой синтаксис принят также в языках C, C++ и Java.

В следующем примере показан цикл от 0 до 9 с шагом 1. В каждой итерации выводится значение счетчика. Действия, выполняемые в каждой итерации, заключаются в фигурные скобки {} - они отделяют блок кода цикла:

```
for( i = 0; i <= 9; i++)
{
  document.write( "i = " + i + "<br>");
}
```

Приведем еще один пример - в нем выполняется цикл, значение счетчика которого увеличивается на 5 при каждой итерации:

```
for( i = 0; i < 51; i+=5)
{
```

```
    document.write( "i = " + i + "<br>");
  }
```

Данный цикл выполняется от 0 до 50 с шагом пять: 0, 5, 10...50.

Break

Элемент break служит для указания на то, что выполнение кода в данном блоке должно быть завершено и продолжено с первого оператора, находящегося после данного блока. Элемент break используется в блоках, созданных с помощью элементов for, for..in и while. Чаще всего элемент break используется для преждевременного завершения цикла. Например:

```
  for( i=0; i<=19; i++)
  {
    if( i == 10) break;
    document.write( "i = " + i + "<br>");
  }
```

В данном примере показан цикл от 0 до 19, в каждой итерации которого выводится значение счетчика. Элемент if внутри цикла проверяет значение счетчика и, если оно равно 10, завершает выполнение цикла с помощью элемента break.

Continue

Элемент continue в блоках, созданных с помощью элементов for, for..in и while, служит для указания на то, что все следующие за ним выражения должны быть пропущены и должна быть начата следующяя итерация. Например:

```
  for( i=0; i<=19; i++)
  {
    if( i == 10) continue;
    document.write( "i = " + i + "<br>");
  }
```

В данном примере показан цикл от 0 до 19, в каждой итерации которого выводится значение счетчика. Элемент if внутри цикла проверяет значение счетчика, и, если оно равно 10, вывод значения счетчика пропускается. Таким обозом, на экран будут выведены цифры от 0 до 9 и от 11 до 19.

For..in

Элемент for..in представляет собой специальную форму элемента for и используется для получения названий и значений свойств объектов. Этот элемент можно использовать для изучения свойств встроенных объектов и их значений, для отладки приложений, а также для быстрого доступа к значениям таких объектов, как, например, форма. Синтаксис элемента for..in выглядит следующим образом:

```
  for( var in object)
  {
    ...
  }
```

- var - имя переменной, используемой в качестве индекса для доступа к свойствам объекта и их значениям;
- object - объект, к которому осуществляется доступ.

Рассмотрим ряд примеров. Предположим, вы хотите узнать текущие значения свойств объекта Navigator (более подробно об этом объекте мы поговорим на одном из следующих занятий). Вот как это сделать:

```
  for( i in navigator)
  {
```

```
    document.write( i + " = " + navigator[i] + "<br>");
  }
```

Пример работы данной программы для Netscape Communicator 4.01 показан ниже.

```
userAgent = Mozilla/4.01 [en] (Win95; I)
appCodeName = Mozilla
appVersion = 4.01 [en] (Win95; I)
appName = .Netscape
language = en
platform = Win32
plugins = [object PluginArray]
mimeTypes = [object MimeTypeArray]
```

Точно так же мы можем получить текущие значения свойств объекта Screen (объект присутствует в браузере Communicator 4.x):

```
  for( i in screen)
  {
    document.write(i + " = " + screen[1] + "<br>");
  }
```

Вот пример работы данной программы:

```
width = 640
height = 480
pixelDepth = 16
colorDepth = 16
availWidth = 640
availHeight = 480
availLeft = 0
availTop = 0
```

Рассмотрим еще один пример - в нем показано, как использовать элемент for..in для получения значений полей формы:

```
<script language="JavaScript">
  function foreTest()
  {
    for( i in document.demoform)
    {
      alert( i + " " + document.demoform[i]);
    }
    document.close;
  }
</script>
<FORN NAME="demoform">
<INPUT TYPE="text" NAME="txt1"><br>
<INPUT TYPE="text" NAME="txt2"><br>
<INPUT TYPE="tsxt" NAME="txt3"><br>
<INPUT TYPE="button" VALUE="Test" onClick="formTest()">
</FORM>
```

Function

Элемент Function используется для создания собственных функций и объектов. Функции представляют собой подпрограммы, которые могут вызываться из кода на JavaScript. Например, можно создать функцию для проверки правильности заполнения полей формы перед ее отсылкой на сервер - Check. Код такой функции приведен ниже.

```
<script language="JavaScript">
```

```
  // Функция для отсылки формы на сервер
  function SendForm()
  {
    if( Check()) window.document.forms[0].submit;
  }
  // Функция для проверки содержимого формы
  function Check()
  {
    var doc = window.document;
    if( doc.forms[0].elements[0].value == '' ||
        doc.forms[0].elements[1].value == '' ||
        doc.forms[0].elements[2].value == '')
    {
      alert( 'Поля не могут быть пустыми');
      return false;
    }
    else
      return true;
  }
</script>
```

Здесь функция Check выполняет проверку первых трех полей формы и, если одно из них не содержит информации, выводит предупреждающее сообщение.

Функция SendForm (в которой вызывается функция проверки Check) "связывается" с формой следующим образом:

```
<form name="rqform" action="request.dll/doform" method="post">
<font size=4 color="#000090">
  <b>Пожалуйста, введите информацию:</b>
  <br><br>
</font>
<table bgcolor="#fff8bd" border=0 cellspacing=0 cellpadding=1>
  <tr>
   <td><b> Фамилия</b></td>
   <td><input type="text" size=35 value="" name="famname">
  </tr>
  <tr>
   <td><b> Имя</b></td>
   <td><input type="text" size=35 value="" name="mainname">
  </tr>
  <tr>
    <td><b> Отчество</b></td>
    <td><input type="text" size=35 value="" name="fathname">
  </tr>
  <tr>
    <td><b> Как с Вами связаться</b>  </td>
    <td><TEXTAREA             name="contact"             rows="5"
cols="30"></TEXTAREA></td>
  </tr>
  <tr>
    <td width="50%" align="left">
     <input             type="Submit"             value="Послать"
onClick="SendForm()"></td>
```

```
    <td width=503 align right>
     <input              type="Reset"              value="Отменить"
onClick="history.back(-1)"></td>
  </tr>
</table>
</form>
```

В обработчике нажатия кнопки "Послать" указано, что должна вызываться функция SendForm:

```
onClick="SendForm()"
```

Точно так же можно "связывать" функции с обработчиками других событий, например onMouseOver или onMouseOut.

If..else

Элемент if..else используется для создания выражений, зависящих от каких-либо условий. Алгоритм работы данного элемента может выглядеть следующим образом:

- если результат выражения имеет значение true, выполняются инструкции, расположенные в блоке if;
- если результат выражения имеет значение false, выполняются инструкции, расположенные в блоке else. Если блок else не указан, выполнение программы переходит на следующий элемент после всего блока if..else.

Синтаксис элемента if..else выглядит следующим образом:

```
  if( expression)
```

Результат выражения expression всегда имеет либо значение true, либо значение false. Если в блоках if и else присутствует только одно выражение, можно использовать следующую синтаксическую конструкцию:

```
  if( Check())
    Send();
  else
    Clear();
```

Если же в каждом блоке присутствует более одного выражения, то для выделения блоков if и else используются фигурные скобки {}. Использование фигурных скобок указывает на то, что должен выполняться весь код, заключенный между ними:

```
  if( Check())
  {
    alert('Отсылаем форму');
    Send();
  }
  else
  {
    alert('Очищаем форму');
    Clear();
  }
```

Отметим, что в выражении, результат которого проверяется, могут присутствовать любые операторы языка, например:

```
  function Check()
  {
    var doc = window.document;
    if( doc.forms[0].elements[0].value == '')
    {
      alert('Поля не могут быть пустыми');
      return false;
    }
```

```
    else
      return true;
  }
```

В приведенном выше примере выполняется сравнение содержимого первого поля формы с пустой строкой.

New

Элемент new создает новую копию объекта. Он может использоваться двумя способами:

- для создания нового объекта Date, который является встроенным объектом языка JavaScript;
- для создания нового пользовательского объекта.

Синтаксис элемента new выглядит следующим образом:

```
varName = new objectName(params);
```

- varName - имя переменной, в которой будет храниться новая копия объекта;
- objectName - имя объекта. При использовании встроенного объекта Date используется слово Date (с большой буквы!), при использовании пользовательских объектов - имена объектов;
- params - один или более параметров, передаваемых при создании копии объекта.

В следующем примере показано, как использовать элемент new для создания новой копии (экземпляра) встроенного объекта Date:

```
now = new Date();
```

После этого переменная now содержит копию объекта Date, и ее можно использовать для изменения свойств объекта и вызова его методов. Например, чтобы узнать текущий час, мы вызываем метод getHours():

```
now = new Date();
nowHour = now.getHours();
```

В следующем примере показано, как использовать элемент new для создания новой копии (экземпляра) пользовательского объекта. Этот объект присваивает переданную ему строку значению свойства name. Затем значение этого свойства отображается в панели сообщений:

```
user = new someUser("Alex Fedorov");
alert( user.name);
function someUser(nameParam)
{
  this.name = nameParam;
  return (this);
}
```

Return

Выражение return используется для указания на конец функции. Когда интерпретатор JavaScript обнаруживает это выражение, он "возвращается" на строку, следующую после вызова функции. Выражение return может использоваться для возвращения каких-либо значений. Возвратимся к рассмотренному выше примеру:

```
function Check()
{
  var doc = window.document;
  if( doc.forms[0].elements[0].value == '')
  {
    alert('Поля не могут быть пустыми');
    return false;
  }
  else
    return true;
```

```
  }
```

Здесь функция Check возвращает значение false, если выражение в блоке if имеет значение true и true в противном случае.

This

This - это на самом деле не выражение, а ключевое слово. Оно используется для ссылки на текущий объект вместо указания формального имени объекта. Возможны два способа использования ключевого слова this:

- для ссылки на текущую форму или интерфейсный элемент в обработчике события (например, onClick или onSubmit);
- для задания нового свойства в пользовательском объекте.

Ключевое слово this можно использовать как совместно с именем объекта, так и без него, то есть либо просто this, либо this.object. Рассмотрим следующий пример. Предположим, вы хотите, чтобы в обработчике нажатия кнопки возвращалась вся форма:

```
<INPUT TYPE="button" NAME="test" VALUE="Test"
  onClick="test(this.form)">
```

если же использовать просто ключевое слово this, то будет возвращаться не вся форма, а только кнопка:

```
<INPUT TYPE="button" NAME="test" VALUE="Test"
  onClick="test(this)">
```

Вот пример, показывающий различия использования ключевого слова this:

```
<html>
<head><title>JS - CP1197</title>
<script language="JavaScript">
  function myTest(obj)
  {
    alert(obj.name);
    return;
  }
</script>
</head>
<body>
<p align="center">
Пример использования ключевого слова <b>this</b>
</p>
<center>
<form name="ThisTest">
<INPUT TYPE="button" NAME="test1" VALUE="Test1"
  onClick="myTest(this.form)">
<INPUT TYPE="button" NAME="test2" VALUE="Test2"
  onClick="myTest(this)">
</form>
</center>
</body>
</html>
```

Var

Выражение var используется для задания переменных. При этом можно указать и значение переменной. Это выражение также задает область видимости переменной, в том случае, когда переменная описывается внутри функции.

Синтаксис выражения var выглядит следующим образом:

```
  var VariableName;
```

или

```
var VariableName = value;
```

- VariableName - имя переменной;
- value - значение, присваиваемое переменной.

В первом случае объявляется переменная VariableName, но ее значение не задается (пусто). Во втором случае значение переменной задается при ее объявлении. При использовании вне пользовательской функции два следующих выражения эквивалентны:

```
var VariableName = value;
VariableName = value;
```

И в том и в другом случае задается глобальная (доступная из всех функций) переменная. Например, переменная someVar является глобальной:

```
var someVar = 100;
function showVar()
{
  alert( someVar);
}
```

Внутри функции возможно задание локальной (доступной только из этой функции) переменной с тем же именем (если это необходимо):

```
var someVar = 100;
function showLocalVar()
{
  var someVar = 256;
  alert( 'local var = ' + someVar);
  showGlobalVar();
}
function showGlobalVar()
{
  alert( 'global var = ' + someVar);
}
```

While

Выражение while используется для создания цикла, выполняющегося, пока значение выражения, указанного в качестве параметра, имеет значение true. Тело цикла заключается в фигурные скобки.

Синтаксис выражения while выглядит следующим образом:

```
while( exprsssion)
{
  // тело цикла
}
```

В следующем примере тело цикла выполняется 10 раз. В каждой итерации выводится значение счетчика:

```
loopCount = 0;
while( loopCount < 10)
{
  document.write( "LoopCount = " + loopCount + "<BR>");
  loopCount++;
}
```

With

Выражение with предназначено для более (или менее) наглядного и простого использования объектов. Например, для доступа к методам встроенного объекта Math требуется написание следующего кода:

```
alert( Math.PI);
alert( Math.round( 1234.5678));
```

Синтаксис выражения with выглядит следующим образом:

```
with( object)
{
  выражения
}
```

- object - используемый объект;
- выражения - одно и более выражений, в которых объект используется по умолчанию.

Таким образом, используя выражение with, мы можем не указывать объект Math при каждом обращении к его свойствам и методам:

```
with( Math)
{
  alert( PI);
  alert( round( 1234.5678));
}
```

В выражении with возможно использование как встроенных, так и пользовательских объектов. Например, для доступа к первому элементу формы можно написать:

```
function Show()
{
  with( document.forms[0].elements[0])
  {
    alert( name);
    alert( value);
  }
}
```

В данном примере в выражении with указан первый элемент формы, поэтому в блоке with функции alert обращаются к свойствам объекта document.forms[0].elements[0] как к свойствам объекта по умолчанию.

Комментарий //

Мы рассматривали комментарии в первой части нашего курса. Напомним, что комментарии используются для задания текстовых описаний внутри скриптовых программ. Символы //, указывающие на начало комментария, могут располагаться в любом месте строки - интерпретатор JavaScript игнорирует содержимое строки от символов // до конца строки.

Задание 5

1.Используя оператор **for** организуйте цикл по **i** от 0 до 5.

2.На каждом шаге цикла необходимо вывести значение **i** на экран. Подсказка: поскольку в теле цикла выполняется лишь одно действие, то операторные скобки { } открывать не обязательно. Для того, чтобы цифры не выводились слитно, добавляйте пробел после каждой.

3. Перейти на новую строку используя тэг

4. Используя оператор **while** организуйте цикл по **i** от 10 пока **i** больше или равно 0.

5. В теле цикла выводить значение **i** на экран (для того, чтобы значения не выводились слитно, добавляйте пробел после каждого), а затем значение **i** уменьшать на два.

Глава 7. Основные встроенные объекты JavaScript

На этом занятии мы рассмотрим встроенные объекты языка JavaScript. В отличие от объектов, создаваемых пользователями, и объектов, составляющих объектную модель браузера (речь об этой модели пойдет на одном из следующих занятий), встроенные объекты доступны в любом контексте - будь то Microsoft Internet Explorer или Netscape Navigator.

Согласно спецификации ECMAScript (см. ECMAScript Language Specification, Standard ECMA-262), определяющей основные требования к языку JavaScript, в языке должны быть реализованы следующие объекты: Global, Object, Function, Array, String, Boolean, Number, Math и Date.

На данном занятии мы рассмотрим встроенные объекты Array, Boolean, Date, Function, Math, Number и String, а к объектам Global и Object вернемся в следующий раз.

Объект Array

Язык JavaScript не имеет встроенного типа данных для создания массивов, поэтому для решения таких задач вам предоставляется объект Array. Он имеет методы для объединения массивов, их сортировки и перестановки; есть также возможность определения размера массива.

Массив - это упорядоченный набор значений, доступ к которому осуществляется по имени и индексу. Например, в программе может быть создан массив, содержащий набор сообщений - allMsg, каждое из которых имеет свой индекс. Таким образом, allMsg[0] будет первым сообщением, allMsg[1] - вторым и так далее.

Для создания объекта Array можно применить два взаимозаменяемых способа.

Способ 1

Вы вызываете конструктор new и задаете размер (число элементов) массива. Заполнение массива происходит позже. Рассмотрим следующий пример.

```
<html>
<head><title>JavaScript. 12-97</title>
<script language="JavaScript">
// создание нового массива
  allStr = new Array(5);
// заполнение массива
  allStr[0] = "Message #1";
  allStr[1] = "Message #2";
  allStr[2] = "Message #3";
  allStr[3] = "Message #4";
  allStr[4] = "Message #5";
// функция для отображения элемента массива
  function showMsg(ndx)
  {
    alert(allStr[ndx]);
  }
</script>
</head>
<-- При загрузке документа показать сообщение N4 -->
<body onLoad="showMsg(3);">
</body>
</html>
```

В приведенном выше примере создается массив, состоящий из 5 элементов, а затем происходит его заполнение.

Способ 2

Вы вызываете конструктор new и сразу задаете значения всех элементов массива. Размер в данном случае в явном виде не указывается. Рассмотрим пример.

```
<html>
<head><title>JavaScript. 12-97</title>
<script language="JavaScript">
// создание нового массива и его заполнение
  allStr = new Array("Message #1", "Message #2", "Message #3",
    "Message #4", "Message #5");
// функция для отображения элемента массива
  function showMsg(ndx)
  {
    alert(allStr[ndx]);
  }
</script>
</head>
<-- При загрузке документа показать сообщение N4 -->
<body onLoad="showMsg(3);">
</body>
</html>
```

Здесь мы задаем значения всех элементов массива непосредственно при вызове конструктора new.

Методы объекта Array

Объект Array имеет следующие методы.

Метод	Описание
join	Объединяет все элементы массива в одну строку
reverse	Изменяет порядок элементов в массиве - первый элемент становится последним, последний - первым
Sort	Выполняет сортировку элементов массива

Рассмотрим примеры использования методов объекта Array. Пусть имеется массив, содержащий ряд элементов, и две функции - showAll и showElement. Первая функция, которая служит для показа всех элементов массива, вызывает в цикле функцию showElement, показывающую содержимое заданного элемента массива:

```
<html>
<head><title>JavaScript. 12-97</title>
<script language="JavaScript">
  myArray = new Array("Mother", "Father", "Sister", "Brother",
          "Uncle");
  function showElement(ndx)
  {
    alert(myArray[ndx]);
  }
  function showAll()
  {
    for( i = 0; i <= myArray.length-1; i++)
    {
      showElement(i);
    }
  }
</script>
```

```
</head>
<body onLoad="showAll();">
</body>
</html>
```

Если мы загрузим приведенный выше файл, то увидим последовательность панелей сообщений, в каждой из которых будет отображаться один элемент массива myArray - Mother, Father, Sister, Brother и Uncle.

Создадим функцию test, в которой напишем следующий код:

```
  function test()
  {
    alert(myArray.join());
  }
```

и изменим содержимое тэга <body>:

```
  <body onLoad="test();">
```

Метод join имеет необязательный параметр, который позволяет задать разделитель между элементами массива. По умолчанию используется символ ",". Например

```
  function test()
  {
    alert(myArray.join(" _|_ "));
  }
```

Метод reverse используется для перестановки элементов массива. Добавим вызов этого метода в функцию test:

```
  function test()
  {
    myArray.reverse();
    alert(myArray.join(";"));
  }
```

Первый элемент массива занял последнее место, второй - предпоследнее и так далее.

Метод sort используется для сортировки элементов массивов. Добавим вызов этого метода в функцию test:

```
  function test()
  {
    myArray.sort();
    alert(myArray.join(";"));
  }
```

Создание многомерных массивов.

Объект Array позволяет создавать многомерные массивы. Ниже приведен пример создания многомерного массива.

```
<html>
<head><title>JavaScript. 12-97</title></head>
<body>
<center>
<font size=5><b>Multidimentional Array</b></font><p>
<script language="JavaScript">
  a = new Array(4);
  for( i=0; i < 4; i++)
  {
    a[i] = new Array(4);
    for( j=0; j < 4; j++)
    {
      a[i][j] = "["+i+","+j+"]";
```

```
    }
  }
  for( i=0; i < 4; i++)
  {
    str = "Row "+i+":";
    for( j=0; j < 4; j++)
    {
      str += a[i][j];
    }
    document.write( str, "<br>");
  }
</script>
</center>
</body>
</html>
```

В приведенной выше программе создается массив из четырех элементов; каждый элемент также представляет собой массив из четырех элементов. В каждый элемент заносится значение пары i,j, задающей индекс элемента. Затем в цикле происходит отображение содержимого элементов данного массива.

Объект Boolean

Встроенный объект Boolean используется для преобразования небулевых значений в булевы (логические). Для создания экземпляра объекта Boolean используется конструктор new:

```
bfalse = new Boolean(false);
btrue  = new Boolean(true);
```

Метод valueOf возвращает значение булевой переменной в виде булева значения, а метод toString - в виде строчного значения. Пример использования этих методов показан ниже.

```
<html>
<head><title>JavaScript 12.97</title></head>
<body>
<script language="JavaScript">
  // создадим две булевых переменных
  bfalse = new Boolean(false);
  btru = new Boolean(true);
  // выведем их значения (булевы значения)
  document.write(bfalse.valueOf()+"<br>");
  document.write(btrue.valueOf()+"<br>");
  // выведем строчные значения
  document.write(bfalse.toString()+"<br>");
  document.write(btrue.toString()+"<br>");
</script>
</body>
</html>
```

Объект Date

Объект Date и его методы используются для работы с датой и временем в скриптовых программах. Этот объект имеет большой набор методов для установки даты, получения, ее значения и выполнения различных преобразований. Объект Date не имеет свойств.

Дата в языке JavaScript хранится точно так же, как в Java, - она представляет собой число миллисекунд, прошедших с 1 января 1970 года. Таким образом, не поддерживаются даты ранее данной.

Для создания экземпляра объекта Date используется конструктор Date:

```
MyDate = new Date([параметры]);
```

Возможно указание следующих параметров:

- никаких параметров - экземпляр будет содержать текущую дату и время. Например, today = new Date();
- строка, представляющая собой дату в следующем формате: "Месяц день, год часы:минуты:секунды". Например, someDate = new Date("May 15, 1996"). Если число часов, минут или секунд на указано, их значения равны 0;
- набор целочисленных значений для года, месяца и дня. Например, otherDay = new Date(96, 4, 15);
- набор целочисленных значений для года, месяца, дня, часов, минут и секунд. Например, sameDay = new Date(96, 4, 15, 15, 30, 0);

Пример использования различных параметров показан ниже.

```
<html>
<head><title>JavaScript 12.97</title></head>
<body>
<center>
<script language="JavaScript">
  today = new Date();
  document.write("today="+today+"<br>");
  someDate = new Date("May 16, 1996");
  document.write("someDate="+someDate+"<br>");
  otherDay = new Date( 96, 4, 15);
  document.write("otherDay="+otherDay+"<br>");
  sameDay = new Date( 96, 4, 16, 15, 30, 0);
  document.write("sameDay="+sameDay+"<br>");
</script>
</center>
</body>
</html>
```

Методы объекта Date

Методы, предоставляемые объектом Date для управления датой и временем, можно подразделить на следующие категории:

- *методы установки (set)* - методы для установки даты и времени у экземпляров объекта Date;
- *методы определения (get)* - методы для определения даты и времени у экземпляров объекта Date;
- *методы преобразования (to)* - методы для преобразования даты и времени в строки;
- *методы для обработки даты*.

Методы установки и определения могут использоваться для получения/изменения значений секунд, минут, часов, дня в месяце, дня недели, месяца и года. Например, есть метод getDay, позволяющий узнать день недели, но нет метода setDay, так как день недели устанавливается автоматически.

Все эти методы используют целочисленные значения, перечисленные в следующей таблице.

Значение	**Диапазон**
Число секунд и минут	0..59
Число часов	0..23
День недели	0..6
Дата	1..31

Месяц	0..11 (Январь..Декабрь)
Год	Число лет с 1900

Рассмотрим следующий пример. Предположим, вы задали дату 15 мая 1996 года. Ниже показано использование ряда методов объекта Date.

```
<html>
<head><title>JavaScript 12.97</title></head>
<body>
<center>
<p>
<script language="JavaScript">
  someDate = new Date( "May 15, 1996");
  document.write("someDate="+someDate+"<br>");
  document.write("getDay  ="+someDate.getDay()+"<br>");
  document.write("getMonth="+someDate.getMonth()+"<br>");
  document.write("getYear ="+someDate.getYear()+"<br>");
</script>
</center>
</body>
</html>
```

Методы getTime и setTime удобно использовать для сравнения дат. Метод getTime возвращает число миллисекунд. Например, ниже показано, как узнать число дней, оставшихся в этом году.

```
<html>
<head><title>JavaScript 12.97</title></head>
<body>
<center>
<br><br><br>
<script language="JavaScript">
  today = new Date();
  // задать дату
  endYear = new Date("December 31, 1990");
  // поменять год
  endYear.setYear(today.getYear());
  // вычислить число миллисекунд в дне
  msPerDay = 24 * 60 * 60 * 1000;
  // получить число дней
  daysLeft = (endYear.getTime() - today.getTime()) / msPerDay;
  // округлить
  daysLeft = Math.round(daysLeft);
  // показать
  document.write("Number of days left in the year: "+daysLeft);
</script>
</center>
</body>
</html>
```

Метод parse может использоваться для преобразования стройного представления времени. Например,

```
someDate = new Date();
someDate.setTime(Date.parse("May 15, 1996"));
```

Завершим рассмотрение объекта Date небольшим практическим примером. Мы создадим программу, выводящую текущее время, - при желании вы сможете включить ее в состав своей HTML-страницы. Время будет выводиться как заголовок интерфейсного элемента "кнопка". В

качестве упражнения предлагаю вам дополнить программу так, чтобы при нажатии кнопки показывалась текущая дата.
Вот текст программы, отображающей время.

```
<html>
<head><title>JavaScript 12.97</title>
<script language="JavaScript">
  function showTime()
  {
    // получим текущую дату
    var time = new Date();
    // узнаем число часов
    var hour = time.getHours();
    // узнаем число минут
    var minute = time.getMinutes();
    // узнаем число секунд
    var second = time.getSeconds();
    // покажем часы
    var temp = hour;
    // покажем минуты (прибавив 0 к 0-9)
    temp += ((minute < 10) ? ":0" : ":") + minute;
    // покажем секунды (прибавив 0 к 0-9)
    temp += ((second < 10) ? ":0" : ":") + second;
    // отобразим
    document.forms[0].clock.value = temp;
    // будем показывать каждую секунду
    id = setTimeout( "showTime()", 1000);
  }
</script>
</head>
<body onLoad="showTime();">
<center>
<font size=5><b>
 JavaScript Date & Time Demo
</b></font>
<form name="DateDemo">
 <input type="button" name="clock" value="xxxxxxxx">
</form>
</center>
</body>
</html>
```

Функция для отображения даты может выглядеть, например, так:

```
  function showDate()
  {
    // получим текущую дату
    var D = new Date();
    // узнаем дань недели
    var DOW = D.getDay();
    // узнаем дату
    var Day = D.getDate();
    // узнаем месяц
    var Month = D.getMonth();
```

```
    // узнаем год
    var Year = D.getYear();
    // покажем
    temp = Day + "/" + Month + "/" + Year;
    document.forms[0].clock.value = temp;
  }
```

В переменной DOW содержится номер дня недели. Воспользовавшись рассмотренным выше объектом Array, мы можем создать массив названий дней недели:

```
  dayNames = new
          Array("воскресенье","понедельник","вторник","среда",
                                  "четверг","пятница","суббота");
```

и добавить в предпоследней строке функции showDate следующее:

```
  temp = Day + "/" + Month + "/" + Year + dayNames[DOW];
```

Объект Function

Встроенный объект Function позволяет задать строку кода на JavaScript, которая будет расцениваться как функция. Для создания экземпляра объекта Function используется конструктор new:

```
  var newBgColor = new Function("c", "document.bgColor=c");
```

Пример использования данной функции показан ниже.

```
<html>
<head><title>JavaScript 12.97</title></head>
<body>
<center>
<script language="JavaScript">
  var newBgColor = new Function("c", "document.bgColor=c");
  function fnDemo()
  {
    newBgColor("#a7b7c7");
  }
</script>
<form>
<input type="button" value="bgColor" onClick="fnDemo();">
</form>
</center>
</body>
</html>
```

Рассмотрим еще один пример. Предположим, нам нужна функция, перемножающая значения двух аргументов. Назовем такую функцию myMult:

```
  var myMult = new Function("x", "y", "return x*y");
```

Воспользоваться ею можно так:

```
<html>
<head><title>JavaScript 12.97</title></head>
<body>
<center>
<script language="JavaScript">
  var myMult = new Function("x", "y", "return x*y");
  document.write("10*20="+myMult(10, 20));
</script>
</center>
</body>
```

```
</html>
```

Мы даже можем создать небольшую форму и использовать ее для получения произведения различных чисел:

```
<html>
<head><title>JavaScript 12.97</title></head>
<body>
<center>
<script language="JavaScript">
  var myMult = new Function("x", "y", "return x*y");
  function calc()
  {
    document.forms[0].sum.value =
      myMult(document.forms[0].x.value, document.forms[0].y.value);
  }
</script>
</center>
<form>
<input type="text" name="x"   value=10  size=4>
<input type="text" name="y"   value=10  size=4>
<input type="text" name="mul" value=" " size=4>
<input type="button" value="calc" onClick="calc();">
</form>
</body>
</html>
```

Объект Math

Встроенный объект Math предоставляет свойства и методы для получения различных математических констант и выполнения математических функций. Ниже приводится исходный текст программы для вывода этих значений.

```
<html>
<head><title>JavaScript 3 Demo</title></head>
<body bgcolor="#a7b7c7">
<center><font size=4><b>Math</b> Object Properties</font><p>
<script language="JavaScript">
  s = "<TABLE BORDER=2>" +
      "<TR><TD><B>E</B></TD><TD>"             +  Math.E            +
"</TD></TR>" +
      "<TR><TD><B>LN2</B></TD><TD>"           +  Math.LN2          +
"</TD></TR>" +
      "<TR><TD><B>LN10</B></TD><TD>"          +  Math.LN10         +
"</TD></TR>" +
      "<TR><TD><B>LOG2E</B></TD><TD>"         +  Math.LOG2E        +
"</TD></TR>" +
      "<TR><TD><B>LOG10E</B></TD><TD>"        +  Math.LOG10E       +
"</TD></TR>" +
      "<TR><TD><B>PI</B></TD><TD>"            +  Math.PI           +
"</TD></TR>" +
      "<TR><TD><B>SQRT1_2</B></TD><TD>"       +  Math.SQRT1_2      +
"</TD></TR>" +
      "<TR><TD><B>SQRT2</B></TD><TD>"         +  Math.SQRT2        +
"</TD></TR>" +
```

```
      "</TABLE>";
  document.write(s);
</script>
</center>
</body>
</html>
```

В следующей таблице перечислены методы, предоставляемые объектом Math.

Метод	Описание
abs	Возвращает абсолютное значение аргумента
sin, cos, tan	Стандартные тригонометрические функции, аргументы указываются в радианах
acos, asin, atan	Инверсные тригонометрические функции, возвращают значения в радианах
exp, log	Экспонента и натуральный логарифм, основание - e
ceil	Возвращает целое число, большее или равное аргументу
floor	Возвращает целое число, меньшее или равное аргументу
min, max	Возвращает больший или меньший из двух аргументов
pow	Возвращает степень аргумента
round	Округляет аргумент до ближайшего целого
sqrt	Возвращает квадратный корень аргумента

При использовании большого числа свойств и методов объекта Math, например при проведении каких-либо вычислений, удобно воспользоваться оператором with:

```
  with( Math)
  {
    a = PI * r*r;
    b = floor(c);
    x = r * sin( theta);
    y = r * cos( theta);
    z = round( b);
  }
```

Объект Number

Свойства встроенного объекта Number используются для получения значений различных числовых констант типа максимального значения, "not-a-number" (NaN) и бесконечности. Ниже приводится исходный текст программы для вывода значений свойств объекта Number.

```
<html>
<head><title>JavaScript 3 Demo</title></head>
<body bgcolor="#a7b7c7">
<center><font size=4><b>Number</b> Object Properties</font><p>
<script language="JavaScript">
  s = "<TABLE BORDER=2>" +
      "<TR><TD><B>MAX_VALUE</B></TD><TD>"          +
         Number.MAX_VALUE          + "</TD></TR>" +
      "<TR><TD><B>MIN_VALUE</B></TD><TD>"          +
         Number.MIN_VALUE          + "</TD></TR>" +
      "<TR><TD><B>NaN</B></TD><TD>"                +
         Number.NaN                + "</TD></TR>" +
      "<TR><TD><B>NEGATIVE_INFINITY</B></TD><TD>" +
         Number.NEGATIVE_INFINITY + "</TD></TR>" +
```

```
      "<TR><TD><B>POSITIVE_INFINITY</B></TD><TD>" +
         Number.POSITIVE_INFINITY + "</TD></TR>" +
      "</TABLE>";
  document.write(s);
</script>
</center>
</body>
</html>
```

Объект String

Язык JavaScript не имеет встроенного типа данных string. Тем не менее, используя объект String и его методы, вы можете работать со строками в скриптовых программах. Объект String содержит большое число методов для управления строками и одно свойство (length), позволяющее определить длину строки.

Экземпляр объекта String создается с помощью вызова конструктора new:

```
  myString = new String("myString Object");
```

Объект String предоставляет два типа методов. К первому типу относятся методы, выполняющие преобразования или другие операции над строками: substring, toUpperCase и т.п. Отметим, что группа функций, возвращающих HTML-версии строк, не входит в стандарт ECMAScript.

Методы объекта String перечислены в следующей таблице.

Методы объекта String

Метод	**Описание**
anchor	Создает HTML-элемент anchor
big, blink, bold, fixed, italics, small, strike, sub, sup	Создает соответствующий HTML-элемент
charAt	Возвращает символ, находящийся в указанной позиции строки
indexOf, lastIndexOf	Возвращает позицию подстроки в строке или последней подстроки в строке соответственно
link	Создает HTML-ссылку
split	Разделяет объект String на массив строк, преобразуя строку в подстроки
substring	Возвращает указанную подстроку
toLowerCase, toUpperCase	Преобразует символы строки в символы нижнего и верхнего регистра соответственно

Рассмотрим несколько примеров использования методов этого объекта.

```
<html>
<head><title>JavaScript 3 Demo</title></head>
<body bgcolor="#a7b7c7">
<center>
<script language="JavaScript">
  // создадим новую строку
  var s = new String('Netscape Navigator');
  // получим подстроку
  var n = s.substring(0,8);
  // отобразим строку и подстроку
  document.write("string="+s+"<br>substring="+n);
  // преобразуем к нижнему регистру
```

```
  var l = s.toLowerCase(s);
  // покажем
  document.write("<br>"+l);
  // преобразуем к верхнему регистру и покажем
  document.write("<br>"+s.toUpperCase(s));
</script>
</center>
</body>
</html>
```

Это были методы преобразования. Теперь посмотрим на методы, возвращающие HTML-версии строк:

```
<html>
<head><title>JavaScript 3 Demo</title></head>
<body bgcolor="#a7b7c7">
<center>
<script language="JavaScript">
  // создадим новую строку
  var s = new String('Netscape Navigator');
  // добавим <b> и </b>
  document.write(s.bold(s)+"<br>");
  // добавим <i> и </i>
  document.write(s.italics(s)+"<br>");
  // добавим <tt> и </tt>
  document.write(s.fixed(s));
  // добавим <sub> и </sub>
  document.write(s.sub(s)+" ");
  // добавим <sup> и </sup>
  document.write(s.sup(s)+"<br>");
  // добавим <strike> и </strike>
  document.write(s.strike(s)+"<br>");
</script>
</center>
</body>
</html>
```

Для того, чтобы преобразовать строку в ссылку, необходимо вызвать метод link:

```
  var s = new String('Netscape Navigator');
  document.write(s.link("http://www.netscape.com"));
```

В документе для переменной s HTML-текст будет выглядеть следующим образом:

```
<A HREF="http://www.netscape.com">Netscape Navigator</A>
```

Задание 6

1. Создайте массив Arr.
2. Первому элементу присвойте значение 1, второму 8, третьему 4.
3. Отсортируйте массив, используя sort().
4. Выведите значение второго элемента массива после сортировки.

Глава 8. Основные встроенные функции языка JavaScript

JavaScript имеет несколько функций "верхнего уровня", встроенных в язык. Рассмотрим следующие функции:

§ eval
§ parseInt
§ parseFloat

8.1 Функция eval

Аргумент встроенной функции *eval* - строка. Строкой может быть - любая строка, представляющая выражение *JavaScript*, утверждение, или последовательность утверждений. Выражение может включать переменные и свойства существующих объектов.

Если аргумент представляет выражение, eval вычисляет выражение. Если аргумент представляет один или большее количество *JavaScript* утверждений, eval вычисляет утверждения.

Эта функция полезна для оценки строки, представляющей арифметическое выражение.

Функция eval не ограничена оценкой численных выражений. Ее аргумент может включать ссылки объекта или даже *JavaScript* утверждения. Например, вы могли определить функцию setValue, которая принимает два аргумента: объект и значение, и выглядит следующим образом:

```
function setValue (myobj, myvalue) {
   eval ("document.forms[0]." + myobj + ".value") = myvalue; }
```

Затем, например, вы могли вызвать эту функцию, чтобы установить значение элемента формы "text1" следующим образом:

```
 setValue (text1, 42)
```

8.2 Функции parseInt и parseFloat

Эти две встроенные функции возвращают числовое значение когда дана строка как аргумент.

Функция parseFloat анализирует его строковый аргумент, и возвращает число с плавающей точкой, если первый символ переданной строки - знак "плюс", знак "минус", десятичная точка, число "e" (основание натурального логарифма) или цифра. Если parseFloat сталкивается с недопустимым символом, то метод возвращает значение, основанное на подстроке, следующей до этого символа, игнорируя все последующие. Если первый же символ недопустим, parseFloat возвращает одно из двух значений, в зависимости от платформы:

0 на платформах Windows.

"NaN" на любой другой платформе, указывая, что значение - не номер.

Для арифметических целей, значение "NaN" - не число в любом основании системы счисления. Вы можете вызывать функцию isNaN, чтобы определить,является ли результат parseFloat "NaN".

Функция parseInt анализирует первый строковый аргумент, и возвращает целое число, основанное на указанном основании системы счисления. Например, при параметре radix, равном 10, string преобразовывается в десятичное число, при 8 преобразовывается в восьмеричное и при 16 - в шестнадцатеричное. Значения, большие 10, для оснований, превышающих 10, представляются символами от A до F вместо чисел. Использование radix, равного 2, служит для преобразований в двоичные числа. Числа с плавающей запятой будут преобразованы в целые числа. Правила обработки строки идентичны правилам для parseFloat.

Если parseInt сталкивается с символом, который - не символ в указанном основании системы счисления, то игнорирует его и возвращает значение целого числа, анализируемого до этого символа. Если первый символ не может быть преобразован к символу в указанном основании системы счисления, то возвращает *NaN.* ParseInt усекает числа до значения целого числа.

Задание 7

1. Создайте переменную **expr** и присвойте ей строку "10*20/3".
2. Используя функцию **eval()** вычислите значение выражения, которое представлено строкой **expr** и присвойте переменной **x** результат.
3. Выведите переменную **x** на экран.

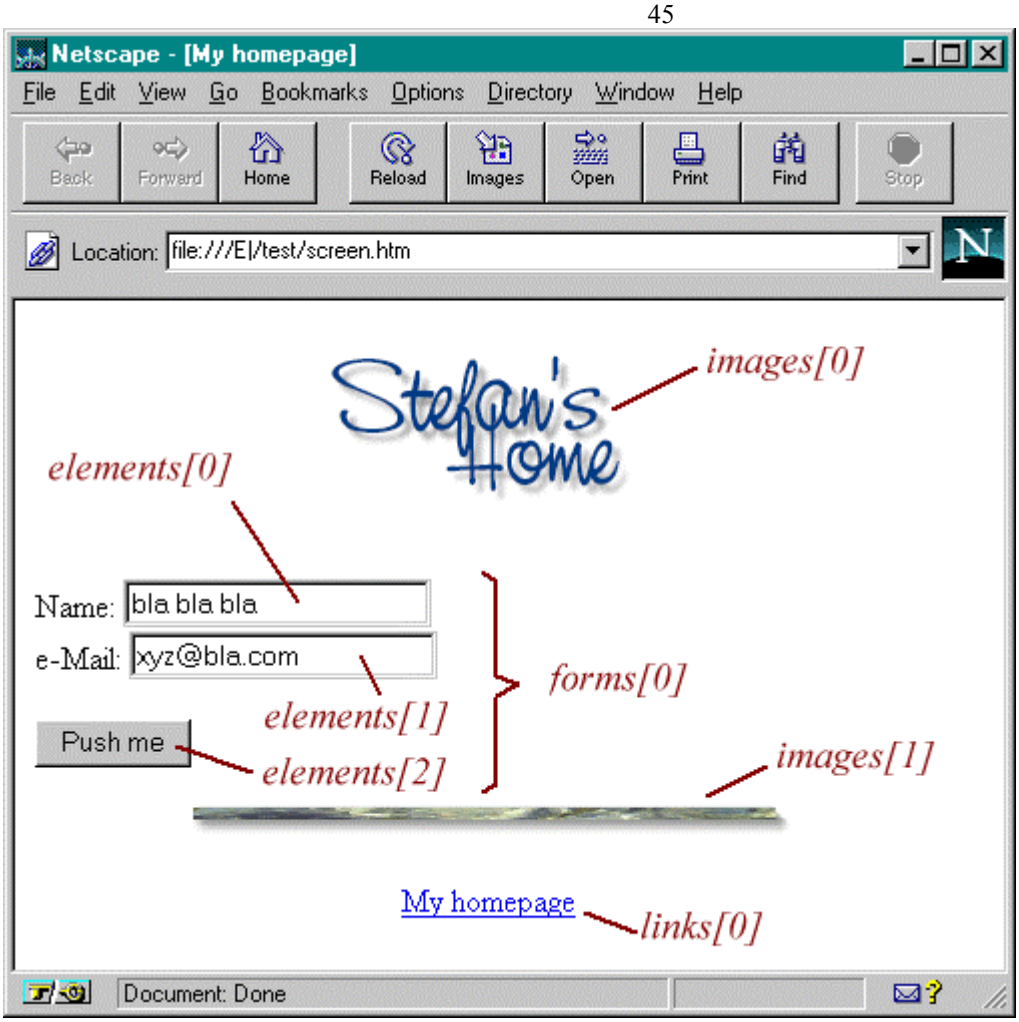
Глава 9. Обработка форм

В языке JavaScript все элементы на web-странице выстраиваются в иерархическую структуру. Каждый элемент предстает в виде объекта. И каждый такой объект может иметь определенные свойства и методы. В свою очередь, язык JavaScript позволит Вам легко управлять объектами

web-страницы, хотя для этого очень важно понимать иерархию объектов, на которые опирается разметка HTML. Как это все действует, Вы сможете быстро понять на следующем примере. Рассмотрим простую HTML-страницу:

```
<html>
<head>
</head>
<body>
<center>
<img src="home.gif" name="pic1" width=200 height=100>
</center>
<p>
<form name="myForm">
Name:
<input type="text" name="name" value=""><br>
e-Mail:
<input type="text" name="email" value=""><br><br>
<input type="button" value="Push me" name="myButton" onClick="alert('Yo')">
</form>
<p>
<center>
<img src="ruler.gif" name="pic4" width=300 height=15>
<p>
<a href="http://rummelplatz.uni-mannheim.de/~skoch/">My homepage</a>
</center>
</body>
</html>
```

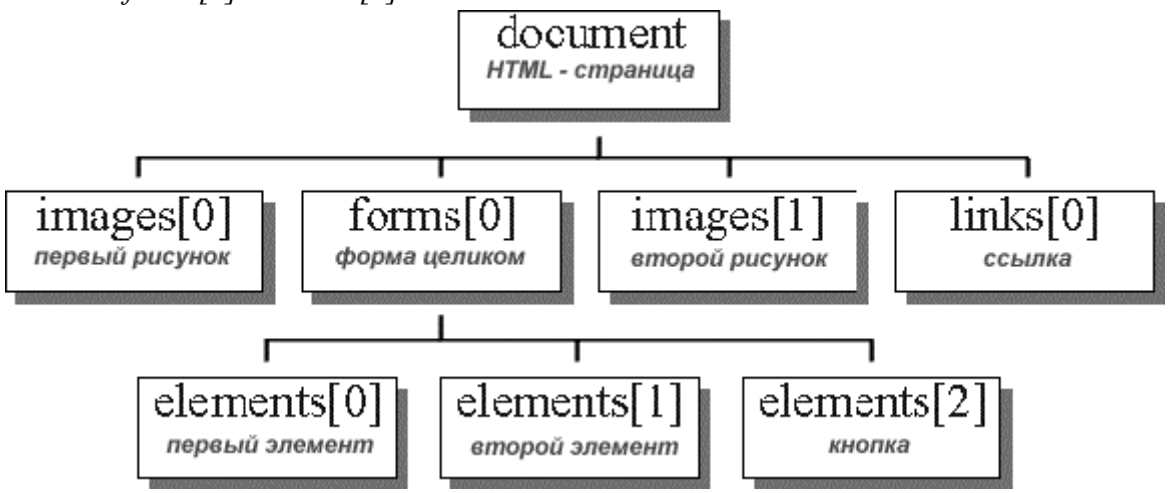
А вот как выглядит эта страница на экране (я добавил к ней еще кое-что):



Итак, мы имеем два рисунка, одну ссылку и некую форму с двумя полями для ввода текста и одной кнопкой. С точки зрения языка JavaScript окно браузера - это некий объект window. Этот объект также содержит в свою очередь некоторые элементы оформления, такие как строка состояния. Внутри окна мы можем разместить документ HTML (или файл какого-либо другого типа - однако пока мы все же ограничимся файлами HTML). Такая страница является ни чем иным, как объектом document. Это означает, что объект document представляет в языке JavaScript загруженный на настоящий момент документ HTML. Объект document является очень важным объектом в языке JavaScript и Вы будете пользоваться им многократно. К свойствам объекта document относятся, например, цвет фона для web-страницы. Однако для нас гораздо важнее то, что все без исключения объекты HTML являются свойствами объекта document. Примерами объекта HTML являются, к примеру, ссылка или заполняемая форма. На следующем рисунке иллюстрируется иерархия объектов, создаваемая HTML-страницей из нашего примера:

Разумеется, мы должны иметь возможность получать информацию о различных объектах в этой иерархии и управлять ею. Для этого мы должны знать, как в языке JavaScript организован доступ к различным объектам. Как видно, каждый объект иерархической структуры имеет свое имя. Следовательно, если Вы хотите узнать, как можно обратиться к первому рисунку на нашей HTML-странице, то обязаны сориентироваться в иерархии объектов. И начать нужно с самой вершины. Первый объект такой структуры называется document. Первый рисунок на странице представлен как объект *images[0]*. Это означает, что отныне мы можем получать доступ к этому объекту, записав в JavaScript *document.images[0]*. Если же, например, Вы хотите знать, какой текст ввел читатель в первый элемент формы, то сперва должны выяснить, как

получить доступ к этому объекту. И снова начинаем мы с вершины нашей иерархии объектов. Затем прослеживаем путь к объекту с именем *elements[0]* и последовательно записываем названия всех объектов, которые минуем. В итоге выясняется, что доступ к первому полю для ввода текста можно получить, записав:
document.forms[0].elements[0]



А теперь как узнать текст, введенный читателем? Чтобы выяснить, которое из свойств и методов объекта позволяют получить доступ к этой информации, необходимо обратиться к какому-либо справочнику по JavaScript (например, это может быть документация, предоставляемая фирмой Netscape, либо моя книга). Там Вы найдете, что элемент, соответствующий полю для ввода текста, имеет свойство value, которое как раз и соответствует введенному тексту. Итак, теперь мы имеем все необходимое, чтобы прочитать искомое значение. Для этого нужно написать на языке JavaScript строку:
name= document.forms[0].elements[0].value;
Полученная строка заносится в переменную name. Следовательно, теперь мы можем работать с этой переменной, как нам необходимо. Например, мы можем создать выпадающее окно, воспользовавшись командой *alert("Hi " + name)*. В результате, если читатель введет в это поле слово 'Stefan', то по команде *alert("Hi " + name)* будет открыто выпадающее окно с приветствием ' Hi Stefan '.
Если Вы имеете дело с большими страницами, то процедура адресации к различным объектам по номеру может стать весьма запутанной. Например, придется решать, как следует обратиться к объекту *document.forms[3].elements[17]* или *document.forms[2].elements[18]*? Во избежание подобной проблемы, Вы можете сами присваивать различным объектам уникальные имена. Как это делается, Вы можете увидеть опять же в нашем примере:
<form name="myForm">
Name:
*<input type="text" name="name" value="">
*
...
Эта запись означает, что объект *forms[0]* получает теперь еще и второе имя - *myForm*. Точно так же вместо *elements[0]* Вы можете писать name (последнее было указано в атрибуте name тэга <*input*>). Таким образом, вместо
name= document.forms[0].elements[0].value;
Вы можете записать
name= document.myForm.name.value;

Это значительно упрощает программирование на JavaScript, особенно в случае с большими web-страницами, содержащими множество объектов. (Обратите внимание, что при написании имен Вы должны еще следить и за положением регистра - то есть Вы не имеете права написать

myform вместо *myForm*). В JavaScript многие свойства объектов доступны не только для чтения. Вы также имеете возможность записывать в них новые значения. Например, посредством JavaScript Вы можете записать в упоминавшееся поле новую строку.
Пример кода на JavaScript, иллюстрирующего такую возможность - интересующий нас фрагмент записан как свойство onClick второго тэга *<input>*:

```
<form name="myForm">
<input type="text" name="input" value="bla bla bla">
<input type="button" value="Write"
onClick="document.myForm.input.value= 'Yo!'; ">
```

Исходный код скрипта:

```
<html>
<head>
<title>Objects</title>
<script language="JavaScript">
<!-- hide
function first() {
// создает выпадающее окно, где размещается
// текст, введенный в поле формы
alert("The value of the textelement is: " +
document.myForm.myText.value);
}
function second() {
// данная функция проверяет состояние переключателей
var myString= "The checkbox is ";
// переключатель включен, или нет?
if (document.myForm.myCheckbox.checked) myString+="checked"
else myString+= "not checked";
// вывод строки на экран
alert(myString);
}
// -->
</script>
</head>
<body bgcolor=lightblue>
<form name="myForm">
<input type="text" name="myText" value="bla bla bla">
<input type="button" name="button1" value="Button1"
onClick="first()">
<br>
<input type="checkbox" name="myCheckbox" CHECKED>
<input type="button" name="button2" value="Button2"
onClick="second()">
</form>
<p><br><br>
<script language="JavaScript">
<!-- hide
document.write("The background color is: ");
document.write(document.bgColor + "<br>");
document.write("The text on the second button is:");
document.write(document.myForm.button2.value);
// -->
```

</script>
</body>
</html>

Задание 8

1. Дана форма, содержащая 2 элемента: текстовое поле и кнопку. (См. рисунок ниже)

2. Известно, что кнопка при наступлении события onClick вызывает функцию TestForm();
3. Создайте функцию TestForm(), которая анализирует первый элемент формы (текстовое поле)

и если поле не пусто, то выводит значение поля при помощи alert(), иначе этим же способом выводит "Поле пусто";

Глава 10. Программирование свойств окна браузера

Класс объектов Window - это самый старший класс в иерархии объектов JavaScript. К этому классу объектов относятся объект window и объект frame. Объект window ассоциируется с окном программы-браузера, а объект frame с окнами внутри окна браузера, которые порождаются последним при использовании автором HTML-страниц контейнеров FRAMESET и FRAME.

При программировании на JavaScript наиболее часто используют следующие свойства и методы объектов типа window:

Свойства	Методы
status	open
location	close
history	focus
navigator	

Объект window порождается только в момент открытия окна. Все остальные объекты, которые порождаются при загрузке страницы в окно, есть свойства объекта window. Таким образом, у window могут быть разные свойства при загрузке разных страниц.

Поле статуса (строка статуса, status bar)

Полем статуса (status bar) называют среднее поле нижней части окна браузера сразу под областью отображения HTML-страницы. В поле статуса отображается информация о состоянии работы браузера (загрузка документа, загрузка графики, завершение загрузки, запуск апплета и т.п.). Программа на JavaScript имеет возможность работать с этим полем как с изменяемым свойством окна. При этом фактически с этим полем связаны два разных свойства:

- window.status
- window.defaultStatus

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

Программируем status

Свойство status связано с отображением сообщений о событиях, отличных от простой загрузки страницы. Например, когда мышь проходит над гипертекстовой ссылкой, то URL, указанный в атрибуте href, отображается в поле статуса. При попадании мыши на поле свободное от ссылок, в поле статуса восстанавливается сообщение умолчания (Document:Done). Эта техника реализована на данной странице при переходе на описание свойств status и defaultStatus:

window.status

В документации по JavaScript указано, что обработчик событий Mouseover и Mouseout должен возвращать значение true. Это нужно для того, чтобы браузер не выполнял действий по умолчанию. Проверка показывает, что Navigator 4.0 прекрасно обходится и без возврата значения true.

Программируем defaultStatus

Свойство defaultStatus определяет текст, отображаемый в поле статуса, когда никаких событий не происходит. В нашем документе мы определили его в момент загрузки документа:

<body onLoad="window.defaultStatus='Status bar programming';">

Это сообщение появляется в момент, когда загружены все компоненты страницы (текст, графика, апплеты и т.п.). Оно восстанавливается в строке статуса после возврата из любого события, которое может произойти в момент просмотра документа. Любопытно, что движение мыши по свободному от гипертекстовых ссылок полю страницы приводит к постоянному отображению defaultStatus.

Поле location

Поле location отображает URL загруженного документа. Если пользователь хочет вручную перейти к какой-либо странице (набрать ее URL), то он делает это в поле location. Поле располагается в верхней части окна браузера ниже панели инструментов, но выше панели личных предпочтений.

Вообще говоря, location - это объект. Из-за изменений в версиях JavaScript класс location входит как подкласс и в класс window и в класс document. Мы будем рассматривать location только как window.location. Кроме того, location - это еще и подкласс класса URL, к которому относятся также объекты классов Area и Link. Location наследует все свойства URL, что позволяет легко получить доступ к любой части схемы URL.

Рассмотрим характеристики и способы использования объекта location:

Следующий код страницы автоматически загружает страницу http://google.com

```
<html>
<head>
</head>
<body>
<script>
document.location="http://google.com";
</script>
</body>
</html>
```

История посещений (History)

История посещений (трасса) страниц World Wide Web позволяет пользователю вернуться к странице, которую он просматривал несколько минут (или часов, или дней) назад. История посещений в JavaScript трансформируется в объект класса History. Этот объект указывает на массив URL страниц, которые пользователь посещал и которые он может получить, выбрав из меню браузера режим GO. Методы объекта history позволяют пользователю загружать страницы, используя URL из этого массива.

Для того чтобы не создать проблем с безопасностью браузера, путешествовать по history можно только используя индекс URL. При этом URL, как текстовая строка программисту не доступен. Наиболее популярным способом использования этого объекта является его применение в примерах или страницах, на которые могут ссылаться из нескольких разных страниц, предполагая, что вернуться к странице, из которой пример будет загружен:

<form><input type=button value=Назад onClick=history.back()></form>

Если нажать на кнопочку "Назад", то вернемся на страницу, которую просматривали до этого (Можно, конечно, прыгнуть и дальше, но вы сами-то знаете куда попадете?*:-)*)

Тип браузера (Объект Navigator)

В связи с войной браузеров стала актуальной задача настройки страницы на конкретную программу просмотра. При этом возможны два варианта: определение типа браузера на стороне сервера и определение типа браузера на стороне клиента. Для последнего варианта в арсенале объектов JavaScript есть объект Navigator. Этот объект - свойство объекта window.

Рассмотрим простой пример определения типа программы просмотра:

```
<form><input type=button value="Тип навигатора" onClick="window.alert(window.navigator.userAgent);"></form>
```

При нажатии на кнопку отображается окно предупреждения. В нем (окне) - строка userAgent, которую браузер помещает в соответствующий HTTP-заголовок.

Эту строку можно разобрать по составным компонентам:

Список свойств Navigator:

У объекта navigator есть еще несколько интересных с точки зрения программирования применений. Например, проверка поддержки Java:

Измените теперь настройки поддержки Java в вашем браузере и перезагрузите страницу. После этого обратите внимание на последнее предложение предыдущего параграфа.

Аналогично можно проверить форматы графических файлов, которые поддерживает ваш браузер:

```
<script>
if(navigator.mimeTypes['image/gif']!=null) document.write("Ваш браузер поддерживает GIF<br>");
if(navigator.mimeTypes['image/kuku']==null) document.write("Ваш браузер не поддерживает KUKU");
</script>
```

Задание 9

1. При помощи alert() вывести на экран название браузера
2. При помощи alert() вывести на экран версию браузера
3. Загрузить в окно страницу "index.htm"

Глава 11. Программирование графики

Рассмотрим теперь объект Image, который стал доступен, начиная с версии с 1.1 языка JavaScript (то есть с Netscape Navigator 3.0). С помощью объекта Image Вы можете вносить изменения в графические образы, присутствующие на web-странице. В частности, это позволяет нам создавать мультипликацию.

Заметим, что пользователи браузеров более старых версий (таких как Netscape Navigator 2.0 или Microsoft Internet Explorer 3.0 - т.е. использующих версию 1.0 языка JavaScript) не смогут запускать скрипты, приведенные в этой части описания. Или, в лучшем случае, на них нельзя будет получить полный эффект. Давайте сначала рассмотрим, как из JavaScript можно адресоваться к изображениям, представленным на web-странице. В рассматриваемом языке все изображения предстают в виде массива. Массив этот называется images и является свойством объекта document. Каждое изображение на web-странице получает порядковый номер: первое изображение получает номер 0, второе - номер 1 и т.д. Таким образом, к первому изображению мы можем адресоваться записав document.images[0].

Каждое изображение в HTML-документе рассматривается в качестве объекта Image. Объект Image имеет определенные свойства, к которым и можно обращаться из языка JavaScript. Например, Вы можете определить, какой размер имеет изображение, обратившись к его свойствам width и height. То есть по записи *document.images[0].width* Вы можете определить ширину первого изображения на web-странице (в пикселах).

К сожалению, отслеживать индекс всех изображений может оказаться затруднительным, особенно если на одной странице у Вас их довольно много. Эта проблема решается

назначением изображениям своих собственных имен. Так, если Вы заводите изображение с помощью тэга
<img src="img.gif" name="myImage" width=100 height=100>
то Вы сможете обращаться к нему, написав document.myImage или document.images["myImage"].

Загрузка новых изображений

Хотя, конечно, и хорошо знать, как можно получить размер изображения на web-странице, это не совсем то, чего бы мы хотели. Мы хотим осуществлять смену изображений на web-странице и для этого нам понадобится атрибут src. Как и в случае тэга <img>, атрибут src содержит адрес представленного изображения. Теперь - в языке JavaScript 1.1 - Вы имеете возможность назначать новый адрес изображению, уже загруженному в web-страницу. И в результате, изображение будет загружено с этого нового адреса, заменив на web-странице старое. Рассмотрим к примеру запись:
<img src="img1.gif" name="myImage" width=100 height=100>
Здесь загружается изображение img1.gif и получает имя myImage. В следующей строке прежнее изображение img1.gif заменяется уже на новое - img2.gif:
document.myImage.src= "img2.src";
При этом новое изображение всегда получает тот же размер, что был у старого. И Вы уже не можете изменить размер поля, в котором это изображение размещается.

Упреждающая загрузка изображения

Один из недостатков такого подхода может заключаться в том, что после записи в src нового адреса, начинается процесс загрузки соответствующего изображения. И поскольку этого не было сделано заранее, то еще пройдет некоторое время, прежде чем новое изображение будет передано через Интернет и встанет на свое место. В некоторых ситуациях это допустимо, однако часто подобные задержки неприемлемы. И что же мы можем сделать с этим? Конечно, решением проблемы была бы упреждающая загрузка изображения. Для этого мы должны создать новый объект Image. Рассмотрим следующие строки:
hiddenImg= new Image();
hiddenImg.src= "img3.gif";
В первой строке создается новый объект Image. Во второй строке указывается адрес изображения, которое в дальнейшем будет представлено с помощью объекта hiddenImg. Как мы уже видели, запись нового адреса в атрибуте src заставляет браузер загружать изображение с указанного адреса. Поэтому, когда выполняется вторая строка нашего примера, начинает загружаться изображение img2.gif. Но как подразумевается самим названием hiddenImg ("скрытая картинка"), после того, как браузер закончит загрузку, изображение на экране не появится. Оно будет лишь сохранено в памяти компьютера (или точнее в кэше) для последующего использования. Чтобы вызвать изображение на экран, мы можем воспользоваться строкой:
document.myImage.src= hiddenImg.src;
Но теперь изображение уже немедленно извлекается из кэша и показывается на экране. Таким образом, сейчас мы управляли упреждающей загрузкой изображения. Конечно браузер должен был к моменту запроса закончить упреждающую загрузку, чтобы необходимое изображение было показано без задержки. Поэтому, если Вы должны предварительно загрузить большое количество изображений, то может иметь место задержка, поскольку браузер будет занят загрузкой всех картинок. Вы всегда должны учитывать скорость связи с Интернет - загрузка изображений не станет быстрее, если пользоваться только что показанными командами. Мы лишь пытаемся чуть раньше загрузить изображение - поэтому и пользователь может увидеть их раньше. В результате и весь процесс пройдет более гладко.

Если у Вас есть быстрая связь с Интернет, то Вы можете не понять, к чему весь этот разговор. О какой задержке все время говорит этот парень? Прекрасно, но еще остаются люди, имеющие более медленный модем, чем 14.4 (Нет, это не я. Я только что заменил свой на 33.6, да ...).

Изменение изображений в соответствии с событиями, инициируемыми самим читателем

Вы можете создать красивые эффекты, используя смену изображений в качестве реакции на определенные события. Например, Вы можете изменять изображения в тот момент, когда курсор мыши попадает на определенную часть страницы.
Исходный код этого примера выглядит следующим образом:

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

При этом могут возникнуть следующие проблемы:
- Читатель пользуется браузером, не имеющим поддержки JavaScript 1.1.
- Второе изображение не было загружено.
- Для этого мы должны писать новые команды для каждого изображения на web-странице.
- Мы хотели бы иметь такой скрипт, который можно было бы использовать во многих web-страницах вновь и вновь, и без больших переделок.

Теперь мы рассмотрим полный вариант скрипта, который мог бы решить эти проблемы. Хотя скрипт и стал намного длиннее - но написав его один раз, Вы не будете больше беспокоиться об этих проблемах. Чтобы этот скрипт сохранял свою гибкость, следует соблюдать два условия:
- Не оговоривается количество изображений - не должно иметь значения, сколько их используется, 10 или 100
- Не оговоривается порядок следования изображений - должна существовать возможность изменять этот порядок без изменения самого кода

Рассмотрим скрипт (я внес туда некоторые комментарии):

```
<html>
<head>
<script language="JavaScript">
<!-- hide
// ok, у нас браузер с поддержкой JavaScript
var browserOK = false;
var pics;
// -->
</script>
<script language="JavaScript1.1">
<!-- hide
// браузер с поддержкой JavaScript 1.1!
browserOK = true;
pics = new Array();
// -->
</script>
<script language="JavaScript">
<!-- hide
var objCount = 0; // количество изображений на web-странице
function preload(name, first, second)
```

```
{
// предварительная загрузка изображений и размещение их в массиве
if (browserOK)
{
pics[objCount] = new Array(3);
pics[objCount][0] = new Image();
pics[objCount][0].src = first;
pics[objCount][1] = new Image();
pics[objCount][1].src = second;
pics[objCount][2] = name;
objCount++;
}
}
function on(name)
{
if (browserOK)
{
for (i = 0; i < objCount; i++)
{
if (document.images[pics[i][2]] != null)
if (name != pics[i][2])
{
// вернуть в исходное состояние все другие изображения
document.images[pics[i][2]].src = pics[i][0].src;
} else
{
// показывать вторую картинку, поскольку курсор пересекает данное изображение
document.images[pics[i][2]].src = pics[i][1].src;
}
}
}
}
function off()
{
if (browserOK)
{
for (i = 0; i < objCount; i++)
{
// вернуть в исходное состояние все изображения
if (document.images[pics[i][2]] != null)
document.images[pics[i][2]].src = pics[i][0].src;
}
}
}
/* заранее загружаемые изображения - Вы должны здесь указать
изображения, которые нужно загрузить заранее, а также объект Image,
к которому они относятся (первый аргумент). Именно эту часть
нужно корректировать, если Вы хотите использовать скрипт
применительно к другим изображениям (конечно это не освобождает
Вас от обязанности подредактировать в документе также и раздел body)*/
preload("link1", "img1f.gif", "img1t.gif");
```

```
preload("link2", "img2f.gif", "img2t.gif");
preload("link3", "img3f.gif", "img3t.gif");
// -->
</script>
</head>

<body>
<a href="link1.htm" onMouseOver="on('link1')"
onMouseOut="off()">
<img name="link1" src="img1f.gif"
width="140" height="50" border="0"></a>
<a href="link2.htm" onMouseOver="on('link2')"
onMouseOut="off()">
<img name="link2" src="img2f.gif"
width="140" height="50" border="0"></a>
<a href="link3.htm" onMouseOver="on('link3')"
onMouseOut="off()">
<img name="link3" src="img3f.gif"
width="140" height="50" border="0"></a>
</body>
</html>
```

Данный скрипт помещает все изображения в массив pics. Создает этот массив функция preload(), которая вызвается в самом начале. Вызов функции preload() выглядит просто как:

```
preload("link1", "img1f.gif", "img1t.gif");
```

Это означает, что скрипт должен загрузить с сервера два изображения: img1f.gif и img1t.gif. Первое из них - это та картинка, которая будет представлена, пока курсор мыши не попадает в область изображения. Когда же пользователь помещает курсор мыши на изображение, то появляется вторая картинка. При вызове функции preload() в качестве первого аргумента мы указываем слово "link1" и тем самым задаем на web-странице объект Image, которому и будут принадлежать оба предварительно загруженных изображения. Если Вы посмотрите в нашем примере в раздел<body>, то обнаружите изображение с тем же именем link1. Мы используем не порядковый номер, а именно имя изображения для того, чтобы иметь возможность переставлять изображения на web-странице, не переписывая при этом сам скрипт.

Обе функции on() и off() вызываются посредством программ обработки событий onMouseOver и onMouseOut. Поскольку сам элемент image не может отслеживать события MouseOver и MouseOut, то мы обязаны сделать на этих изображениях еще и ссылки. Можно видеть, что функция on() возвращает все изображения, кроме указанного, в исходное состояние. Делать это необходимо потому, что в противном случае выделеными могут оказаться сразу несколько изображений (дело в том, что событие MouseOut не будет зарегистрировано, если пользователь переместит курсор с изображения сразу за пределы окна).

Изображения - без сомнения могучее средство уличшения Вашей web-страницы. Объект Image дает Вам возможность создавать действительно сложные эффекты. Однако заметим, что не каждое изображение или программа JavaScript способны улучшить Вашу страницу. Если Вы пройдетесь по Сети, то сможете увидеть множество примеров, где изображения использованы самым ужасным способом. Не количество изображений делает Вашу web-страницу привлекательной, а их качество. Сама загрузка 50 килобайт плохой графики способна вызвать раздражение.

При создании специальных эффектов с изображениями с помощью JavaScript помните об этом и ваши посетители/клиенты будут чаще возвращаться на Ваши страницы.

Задание 10

1. В документе имеется гиперссылка содержащая рисунок:

2. Гиперссылка имеет обработчики событий OnMouseOver="ChImg('img2.gif')" и OnMouseOut="ChImg('img3.gif')";
3. Напишите функцию ChImg(s), которая изменяет рисунок на другой, указанный в s.

Глава 12. Создание новых окон и некоторые манипуляции сними

Открытие новых окон в браузере - грандиозная возможность языка JavaScript. Вы можете либо загружать в новое окно новые документы (например, те же документы HTML), либо (динамически) создавать новые материалы. Посмотрим сначала, как можно открыть новое окно, потом как загрузить в это окно HTML-страницу и, наконец, как его закрыть. Приводимый далее скрипт открывает новое окно браузера и загружает в него некую web-страничку:

```
<html>
<head>
<script language="JavaScript">
<!-- hide
function openWin() {
myWin= open("bla.htm");
}
// -->
</script>
</head>
<body>
<form>
<input type="button" value="Открыть новое окно" onClick="openWin()">
</form>
</body>
</html>
```

В представленном примере в новое окно с помощью метода *open()* записывается страница *bla.htm*.

Заметим, что Вы имеете возможность управлять самим процессом создания окна. Например, Вы можете указать, должно ли новое окно иметь строку статуса, панель инструментов или меню. Кроме того Вы можете задать размер окна. Например, в следующем скрипте открывается новое окно размером 400x300 пикселов. Оно не имеет ни строки статуса, ни панели инструментов, ни меню.

```
<html>
<head>
<script language="JavaScript">
<!-- hide
function openWin2() {
myWin= open("bla.htm", "displayWindow",
"width=400,height=300,status=no,toolbar=no,menubar=no");
}
// -->
</script>
</head>
<body>
```

```
<form>
<input type="button" value=" Открыть новое окно" onClick="openWin2()">
</form>
</body>
</html>
```

Как видите, свойства окна мы формулируем в строке: *"width=400,height=300,status=no,toolbar=no,menubar=no".*

Обратите внимание также и на то, что Вам не следует помещать в этой строке символы пробела!

Список свойств окна, которыми Вы можете управлять:

directories	yes|no
height	*количество пикселов*
location	yes|no
menubar	yes|no
resizable	yes|no
scrollbars	yes|no
status	yes|no
toolbar	yes|no
width	*количество пикселов*

В версии 1.2 языка JavaScript были добавлены некоторые новые свойства (то есть в Netscape Navigator 4.0). Вам не следует пользоваться этими свойствами, готовя материалы для Netscape 2.x, 3.x или Microsoft Internet Explorer 3.x, поскольку эти браузеры не понимают языка 1.2 JavaScript.

Новые свойства окон:

alwaysLowered	yes|no
alwaysRaised	yes|no
dependent	yes|no
hotkeys	yes|no
innerWidth	*количество пикселов* (заменяет width)
innerHeight	*количество пикселов* (заменяет height)
outerWidth	*количество пикселов*
outerHeight	*количество пикселов*
screenX	*количество пикселов*
screenY	*количество пикселов*
titlebar	yes|no
z-lock	yes|no

Вы можете найти толкование этих свойств в описании языка JavaScript 1.2. В дальнейшем я для некоторых из них дам разъяснение и примеры использования.

Например, теперь с помощью этих свойств Вы можете определить, в каком месте экрана должно находиться вновь открываемое окно. Работая со старой версией языка JavaScript, Вы не смогли бы этого сделать.

Имя окна

Как видите, открывая окна, мы должны использовать три аргумента:

```
myWin= open("bla.htm", "displayWindow",
"width=400,height=300,status=no,toolbar=no,menubar=no");
```

А для чего нужен второй аргумент? Это имя окна. Ранее мы видели, как оно использовалось в параметре target. Так, если Вы знаете имя окна, то можете загрузить туда новую страницу с помощью записи

<a href="bla.html" target="displayWindow">
При этом Вам необходимо указать имя соответствующего окна (если же такого окна не существует, то с этим именем будет создано новое). Обратите внимание, что *myWin* - это вовсе не имя окна. Но только с помощью этой переменной Вы можете получить доступ к окну. И поскольку это обычная переменная, то область ее действия - лишь тот скрипт, в котором она определена. А между тем, имя окна (в данном случае это *displayWindow*) - уникальный идентификатор, которым можно пользоваться с любого из окон браузера.

Создание окон

Вы можете также закрывать окна с помощью языка JavaScript. Чтобы сделать это, Вам понадобится метод close(). Давайте, как было показано ранее, откроем новое окно. И загрузим туда очередную страницу:

```
<html>
<script language="JavaScript">
<!-- hide
function closeIt() {
close();
}
// -->
</script>
<center>
<form>
<input type=button value="Close it" onClick="closeIt()">
</form>
</center>
</html>
```

Если теперь в новом окне Вы нажмете кнопку, то оно будет закрыто. *open()* и *close()* - это методы объекта window. Мы должны помнить, что следует писать не просто *open()* и *close(),* а *window.open()* и *window.close()*. Однако в нашем случае объект window можно опустить - Вам нет необходимости писать префикс *window*, если Вы хотите всего лишь вызвать один из методов этого объекта (и такое возможно только для этого объекта).

Задание 11

1. Создайте новое окно (идентификатор окна MyWin) с именем "Window1" размером 400x300, содержащее файл "index.htm", с полосами прокрутки. Все остальные параметры оставить по умолчанию.

Глава 13. Динамическое создание документов

Теперь мы готовы к рассмотрению такой замечательной возможности JavaScript, как динамическое создание документов. То есть Вы можете разрешить Вашему скрипту на языке JavaScript самому создавать новые HTML-страницы. Более того, Вы можете таким же образом создавать и другие документы Web, такие как VRML-сцены и т.д. Для удобства Вы можете размещать эти документы в отдельном окне или фрейме.

Для начала мы создадим простой HTML-документ, который покажем в новом окне. Рассмотрим следующий скрипт.

```
<html>
<head>
<script language="JavaScript">
<!-- hide
function openWin3() {
myWin= open("", "displayWindow",
"width=500,height=400,status=yes,toolbar=yes,menubar=yes");
// открыть объект document для последующей печати
```

```
myWin.document.open();
// генерировать новый документ
myWin.document.write("<html><head><title>On-the-fly");
myWin.document.write("</title></head><body>");
myWin.document.write("<center><font size=+3>");
myWin.document.write("Данный документ HTML был создан ");
myWin.document.write("с помощью JavaScript!");
myWin.document.write("</font></center>");
myWin.document.write("</body></html>");
// закрыть документ - (но не окно!)
myWin.document.close();
}
// -->
</script>
</head>
<body>
<form>
<input type=button value="On-the-fly" onClick="openWin3()">
</form>
</body>
</html>
```

Давайте рассмотрим функцию *winOpen3 ()*. Очевидно, мы сначала открываем новое окно браузера. Поскольку первый аргумент функции *open()* - пустая строка (*""*), то это значит, что мы не желаем в данном случае указывать конкретный адрес URL. Браузер должен не только обработать имеющийся документ - JavaScript обязан создать дополнительно новый документ.

В скрипте мы определяем переменную myWin. И с ее помощью можем получать доступ к новому окну. Обратите пожалуйста внимание, что в данном случае мы не можем воспользоваться для этой цели именем окна (*displayWindow*). После того, как мы открыли окно, наступает очередь открыть для записи объект document. Делается это с помощью команды:

```
// открыть объект document для последующей печати
myWin.document.open();
```

Здесь мы обращаемся к *open()* - методу объекта *document.* Однако это совсем не то же самое, что метод *open()* объекта window! Эта команда не открывает нового окна - она лишь готовит *document* к предстоящей печати. Кроме того, мы должны поставить перед *document.open()* приставку *myWin*, чтобы получить возможность писать в новом окне.

В последующих строках скрипта с помощью вызова *document.write()* формируется текст нового документа:

```
// генерировать новый документ
myWin.document.write("<html><head><title>On-the-fly");
myWin.document.write("</title></head><body>");
myWin.document.write("<center><font size=+3>");
myWin.document.write("This HTML-document has been created ");
myWin.document.write("with the help of JavaScript!");
myWin.document.write("</font></center>");
myWin.document.write("</body></html>");
```

Как видно, здесь мы записываем в документ обычные тэги языка HTML. То есть мы фактически генерируем разметку HTML! При этом Вы можете использовать абсолютно любые тэги HTML.

По завершении этого мы обязаны вновь закрыть документ. Это делается следующей командой:

```
// закрыть документ - (но не окно!)
myWin.document.close();
```

Как я уже говорил, Вы можете не только динамически создавать документы, но и по своему выбору размещать их в в том или ином фрейме. Например, если Вы получили два фрейма с именами *frame1* и *frame2*, а теперь во *frame2* хотите сгенерировать новый документ, то для этого в *frame1* Вам достаточно будет написать следующее:

```
parent.frame2.document.open();
parent.frame2.document.write("Here goes your HTML-code");
parent.frame2.document.close();
```

Задание 12

1.Создайте новое окно с идентификатором MyWin без всяких параметров.
2.Сгенерируйте HTML-document который будет содержать строку "JavaScript Window".

Глава 14. Отправка сообщений по электронной почте

Данная глава продолжает тему, касающуюся работы с формами.
Формы широко используются в Интернете. Информация, введенная в форму, часто посылается обратно на сервер или отправляется по электронной почте на некоторый адрес. Проблема состоит в том, чтобы убедиться, что введеная пользователем в форму информация корректна. Легко проверить ее перед пересылкой в Интернет можно с помощью языка JavaScript. Сначала я бы хотел продемонстрировать, как можно выполнить проверку формы. А затем мы рассмотрим, какие есть возможности для пересылки информации по Интернет.
Сперва нам необходимо создать простой скрипт. Допустим, HTML-страница содержит два элемента для ввода текста. В первый из них пользователь должен вписать свое имя, во второй элемент - адрес для электронной почты. Если пользователь ввел свое имя (например, 'Stefan') в первый элемент, то скрипт создает выпадающее окно с сообщением 'Hi Stefan!'.
Что касается информации, введенной в первый элемент, то Вы будете получать сообщение об ошибке, если туда ничего не было введено. Любая представленная в элементе информация будет рассматриваться на предмет корректности. Конечно, это не гарантирует, что пользователь введет не то имя. Браузер даже не будет возражать против чисел. Например, если Вы ввведете '17', то получите приглашение 'Hi 17!'. Так что эта проверка не может быть идеальна. Второй элемент формы несколько более сложнее. Попробуйте ввести простую строку - например Ваше имя. Сделать это не удастся до тех пор, пока Вы не укажите @ в Вашем имени... Признаком того, что пользователь правильно ввел адрес электронной почты служит наличие символа @. Этому условию будет отвечать и одиночный символ @, даже несмотря на то, что это бессмысленно. В Интернет каждый адрес электронной почты содержит символ @, так что проверка на этот символ здесь уместна.
Как скрипт работает с этими двумя элементами формы и как выглядит проверка?

```
<html>
<head>
<script language="JavaScript">
<!-- Скрыть
function test1(form) {
if (form.text1.value == "")
alert("Пожалуйста, введите строку!")
else {
alert("Hi "+form.text1.value+"! Форма заполнена корректно!");
}
}
function test2(form) {
if (form.text2.value == "" ||
form.text2.value.indexOf('@', 0) == -1)
alert("Неверно введен адрес e-mail!");
else alert("OK!");
```

```
}
// -->
</script>
</head>
<body>
<form name="first">
Введите Ваше имя:<br>
<input type="text" name="text1">
<input type="button" name="button1" value="Проверка" onClick="test1(this.form)">
<P>
Введите Ваш адрес e-mail:<br>
<input type="text" name="text2">
<input type="button" name="button2" value="Проверка" onClick="test2(this.form)">
</body>
</html>
```

Рассмотрим сначала HTML-код в разделе body. Здесь мы создаем лишь два элемента для ввода текста и две кнопки. Кнопки вызывают функции test1(...) или test2(...), в зависимости от того, которая из них была нажата. В качестве аргумента к этим функциям мы передаем комбинацию this.form, что позже позволит нам адресоваться в самой функции именно к тем элементам, которые нам нужны. Функция test1(form) проверяет, является ли данная строка пустой. Это делается посредством if (form.text1.value == "")... . Здесь 'form' - это переменная, куда заносится значение, полученное при вызове функции от 'this.form'. Мы можем извлечь строку, введенную в рассматриваемый элемент, если к form.text1 припишем 'value'. Чтобы убедиться, что строка не является пустой, мы сравниваем ее с "". Если же окажется, что введенная строка соответствует "", то это значит, что на самом деле ничего введено не было. И наш пользователь получит сообщение об ошибке. Если же что-то было введено верно, пользователь получит подтверждение - ok.

Следующая проблема заключается в том, что пользователь может вписать в поле формы одни пробелы. И это будет принято, как корректно введенная информация! Если есть желание, то Вы конечно можете добавить проверку такой возможности и исключить ее. Я полагаю, что это будет сделать легко, опираясь лишь на представленную здесь информацию. Рассмотрим теперь функцию test2(form). Здесь вновь сравнивается введенная строка с пустой - "" (чтобы удостовериться, что что-то действительно было введено читателем). Однако к команде if мы добавили еще кое-чего. Комбинация символов || называется оператором OR (ИЛИ). С ним Вы уже знакомились в шестой части учебника. Команда if проверяет, чем заканчивается первое или второе сравнения. Если хотя бы одно из них выполняется, то и в целом команда if имеет результатом true, а стало быть будет выполняться следующая команда скрипта. Словом, Вы получите сообщение об ошибке, если либо предоставленная Вами строка пуста, либо в ней отсутствует символ @. (Второй оператор в команде if следит за тем, чтобы введенная строка содержала @.)

Проверка на присутствие определенных символов

В некоторых случаях Вам понадобится ограничивать информацию, вводимую в форму, лишь некоторым набором символов или чисел. Достаточно вспомнить о телефонных номерах - представленная информация должна содержать лишь цифры (предполагается, что номер телефона, как таковой, не содержит никаких символов). Нам необходимо проверять, являются ли введенные данные числом. Сложность ситуации состоит в том, что большинство людей вставляют в номер телефона еще и разные символы - например: 01234-56789, 01234/56789 или 01234 56789 (с символом пробела внутри). Не следует принуждать пользователя отказываться от таких символов в телефонном номере. А потому мы должны дополнить наш скрипт процедурой проверки цифр и некоторых символов. Решение задачи продемонстрировано в следующем примере:

```
<html>
<head>
<script language="JavaScript">
<!-- hide
function check(input) {
var ok = true;
for (var i = 0; i < input.length; i++) {
var chr = input.charAt(i);
var found = false;
for (var j = 1; j < check.length; j++) {
if (chr == check[j]) found = true;
}
if (!found) ok = false;
}

return ok;
}
function test(input) {
if (!check(input, "1", "2", "3", "4",
"5", "6", "7", "8", "9", "0", "/", "-", " ")) {
alert("Input not ok.");
}
else {
alert("Input ok!");
}
}
// -->
</script>
</head>
<body>
<form>
Telephone:
<input type="text" name="telephone" value="">
<input type="button" value="Check"
onClick="test(this.form.telephone.value)">
</form>
</body>
</html>
```

Функция test() определяет, какие из введенных символов признаются корректными.

Предоставление информации, введенной в форму

Какие существуют возможности для передачи информации, внесенной в форму? Самый простой способ состоит в передаче данных формы по электронной почте (этот метод мы рассмотрим поподробнее). Если Вы хотите, чтобы за вносимыми в форму данными следил сервер, то Вы должны использовать интерфейс CGI (Common Gateway Interface). Последнее позволяет Вам автоматически обрабатывать данные. Например, сервер мог бы создавать базу данных со сведениями, доступную для некоторых из клиентов. Другой пример - поисковые страницы, такие как Yahoo. Обычно в них представлена форма, позволяющая создавать запрос для поиска в собственной базе данных. В результате пользователь получает ответ вскоре после того, как нажимает на соответствующую кнопку. Ему не приходится ждать, пока люди, отвечающие за поддержание данного сервера, прочтут указанные им данные и отыщут требуемую информацию. Все это автоматически выполняет сам сервер. JavaScript не позволяет

делать таких вещей. С помощью JavaScript Вы не сможете создать книгу читательских отзывов, поскольку JavaScript лишен возможности записывать данные в какой-либо файл на сервере. Делать это Вы можете только через интерфейс CGI. Конечно, Вы можете создать книгу отзывов, для которой пользователи присылали сведения по электронной почте. Однако в этом случае Вы должны заносить отзывы вручную. Так можно делать, если Вы не предполагаете получать ежедневно по 1000 отзывов. Соответствующий скрипт будет простым текстом HTML. И никакого программирования на JavaScript здесь вовсе не нужно! Конечно за исключением того случая, если Вам понадобится перед пересылкой проверить данные, занесенные в форму - и здесь уже Вам действительно понадобится JavaScript. Я должен лишь добавить, что команда mailto работает не повсюду - например, поддержка для нее отсутствует в Microsoft Internet Explorer 3.0.

```
<form method=post action="mailto:your.address@goes.here" enctype="text/plain">
Нравится ли Вам эта страница?
<input name="choice" type="radio" value="1">Вовсе нет.<br>
<input name="choice" type="radio" value="2" CHECKED>Напрасная трата времени.<br>
<input name="choice" type="radio" value="3">Самый плохой сайт в Сети.<br>
<input name="submit" type="submit" value="Send">
</form>
```

Параметр enctype="text/plain" используется для того, чтобы пересылать именно простой текст без каких-либо кодируемых частей. Это значительно упрощает чтение такой почты.

Если Вы хотите проверить форму прежде, чем она будет передана в сеть, то для этого можете воспользоваться программой обработки событий onSubmit. Вы должны поместить вызов этой программы в тэг <form>. Например:

```
function validate() {
// check if input ok
// ...
if (inputOK) return true
else return false;
}
...
<form ... onSubmit="return validate()">
...
```

Форма, составленная таким образом, не будет послана в Интернет, если в нее внесены некорректные данные.

Выделение определенного элемента формы

С помощью метода focus() Вы можете сделать вашу форму более дружественной. Так, Вы можете выбрать, который элемент будет выделен в первую очередь. Либо Вы можете приказать браузеру выделить ту форму, куда были введены неверные данные. Можно сделать так, что браузер сам установит курсор на указанный пользователем элемент формы, так что ему не придется щелкать по форме, прежде чем что-либо занести туда. Сделать это Вы можете с помощью следующего фрагмента скрипта:

```
function setfocus() {
document.first.text1.focus();
}
```

Эта запись могла бы выделить первый элемент для ввода текста в скрипте, который я уже показывал. Вы должны указать имя для всей формы - в данном случае она называется first - и имя одного элемента формы - text1. Если Вы хотите, чтобы при загрузке страницы данный элемент выделялся, то для этого Вы можете дополнить Ваш тэг <body> атрибутом onLoad. Это будет выглядеть как:

```
<body onLoad="setfocus()">
```

Остается еще дополнить пример следующим образом:
```
function setfocus() {
document.first.text1.focus();
document.first.text1.select();
}
```
Задание 13
1.Создайте форму, содержащую текстовое поле и кнопку submit с надписью Send.
2.При нажатии на кнопку, форма отправляет данные по адресу "name@mailserver.com"
Примечание: используйте только необходимые атрибуты html-тэгов, не используйте в тэгах кавычки, тэги описывайте символами нижнего регистра.

Глава 15. Передача данных для HTML-файлов и их обработка без привлечения CGI
Мало где вы сможете встретить рассматриваемые здесь вопросы. Почему-то большинство создателей учебников по JavaScript и DHTML старательно умалчивают данный аспект. Меж тем тема интересная, не сложная и весьма полезная.

Как передать данные в *.html-файл.
Очень просто - через его адрес (URL). После адреса ставите знак вопроса и после него задаете параметр, который хотите сообщить файлу. Если нужно передать несколько параметров, разделяете их амперсандом. Например, так
"file.html?1234" или
"file.html?12&42&param&78456"
Можно передавать данные из формы, используя метод *get*. В этом случае вызов файла будет выглядеть как
"file.html?name1=value1&...nameX=valueX"

Как получить переданные данные
Как вы догадываетесь, обрабатывается все с использованием JavaScript.
```
var ex_url=location.search.substring(1);
```
То бишь мы записываем в переменную *ex_url* все то, что находится после знака вопроса. Если вы передаете один параметр, то все - обрабатывайте *ex_url* (кстати, ее тип - строка) и в зависимости от ее значения что-либо делайте. Если вы передавали несколько параметров, надо их разделить.
```
var param=ex_url.split('&');
```
Теперь мы получили массив *param*, содержащий переданные значения. Если вы передавали данные через форму, то надо еще избавиться от знаков равенства.
```
var values = new Array();
for(i=0; i<param.length;i++) {
var b = param[i].split('=');
values[b[0]] = unescape(b[1]);
}
```
Теперь все данные занесены в хеш. Если строка запроса была, например,
"file.html?name=alex&age=28&left=right"
то получили массив
```
values[name]="alex";
values[age]="28";
values[left]="right";
```

или
```
values[0]="alex";
values[1]="28";
```

values[2]="right";
кому как больше нравится.

Пример использования

В качестве примера рассмотрим такую ситуацию. Есть набор фотографий и, помимо обычной галереи с предпросмотром, нужно организовать слайд-шоу. Можно, конечно, изменять свойство *src* картинок, но Netscape, например, не позволяет при этом изменять размеры и все картинки будут втиснуты в рамки самой первой, следовательно, искаженными. К тому же, желательно, чтобы при просмотре новой фотографии обновлялись и баннеры, а вставлять для каждого свой скрипт неохота. Поэтому сделаем следующее:
Для простоты предположим, что все фотографии сохранены в файлах *1.jpg, 2.jpg, 3.jpg* и так далее. Делаем страничку, а в том месте, где должна быть фотография, вставляем следующий скрипт.

```
<table>
<script language=javascript>
max_num=100;
ex_url=location.search.substring(1);
if (ex_url.length==0) {
number=1;
}
else {
number=parseInt(ex_url,10);
}
prev=number-1;
next=number+1;
if (number<2) {
}
else {
document.write("<a href=file.html?"+prev+">back</a>");
}
if (number==max_num) {
}
else {
document.write("<a href=file.html?"+next+">next</a><br>");
}
document.write("<img src="+number+".jpg>");
</script></table>
```

Комментарии: max_num - число фотографий. Я написал 100, но в принципе их число неограничено, вставьте свое значение. над картинкой выводятся ссылки на предыдущую и последующую. Для первой нет предыдущей (но вы можете вставить ссылку на другой файл), для последней нет следующей (и опять вы можете вставить ссылку на другой файл). Если никакие параметры не передаются, то есть не указан номер просматриваемой фотографии, показывается первая. Для простоты примера я не вставлял "защиту от дураков" - то есть если кто-то решит передать в файл не число, а просто набор символов. В этом случае просто не будет никакой картинки.

Глава 16. Модель событий в JavaScript 1.2

Новые события

Наступило время, рассмотреть одну из новых особенностей Netscape Navigator 4.x - модель событий JavaScript 1.2. Приведенные здесь примеры будут работать только в Netscape Navigator 4.x (хотя большинство из них работают также и в предварительных версиях этого браузера).

В JavaScript 1.2 поддерживается обработка следующих событий (если Вы хотите узнать побольше об этих событиях, обратитесь к документации JS 1.2 от фирмы Netscape - http://developer.netscape.com/library/documentation/communicator/jsguide/js1_2.htm):
Abort Focus MouseOut Submit
Blur KeyDown MouseOver Unload
Click KeyPress MouseUp
Change KeyUp Move
DblClick Load Reset
DragDrop MouseDown Resize
Error MouseMove Select

Изучая таблицу, можете увидеть, что была реализована обработка некоторых новых событий. На этом уроке мы и рассмотрим некоторые из них.

Resize

Сперва давайте рассмотрим событие Resize. С помощью этого события Вы можете определить, был ли размер окна изменен читателем. Следующий скрипт демонстрирует, как это делается:

```
<html>
<head>
<script language="JavaScript">
window.onresize= message;
function message() {
alert("The window has been resized!");
}
</script>
</head>
<body>
Пожалуйста, измените размер этого окна.
</body>
</html>
```

В строке

```
window.onresize= message;
```

мы задаем процедуру обработки такого события. Точнее, функция message() будет вызываться всякий раз, как только пользователь изменит размер окна. Возможно, Вы не знакомы с таким способом назначения программ, обрабатывающих события. Однако JavaScript 1.2 ничего нового здесь не привносит. Например, если у Вас есть объект button, то Вы можете определить процедуру обработки события следующим образом:

```
<form name="myForm">
<input type="button" name="myButton" onClick="alert('Click event occured!')">
</form>
```

Однако Вы можете написать это и по другому:

```
<form name="myForm">
<input type="button" name="myButton">
</form>
...
<script language="JavaScript>
document.myForm.myButton.onclick= message;
function message() {
alert('Click event occured!');
}
</script>
```

Можно подумать, что вторая альтернатива немного сложнее. Однако почему тогда именно ее мы используем в первом скрипте? Причина состоит в том, что объект window нельзя определить через какой-либо определенный тэг - поэтому нам и приходится использовать второй вариант.
Два важных замечания: во-первых, Вам не следует писать window.onResize - я имею в виду, что Вы должны писать все прописными буквами. Во-вторых, Вы не должны ставить после сообщения никаких скобок. Если Вы напишете window.onresize= message(), то браузер интерпретирует message() как вызов функции. Однако в нашем случае мы не хотим напрямую вызывать эту функцию - мы лишь хотим определить обработчик события.

Объект Event

В язык JavaScript 1.2 добавлен новый объект Event. Он содержит свойства, описывающие некое событие. Каждый раз, когда происходит какое-либо событие, объект Event передается соответствующей программе обработки.
В следующем примере на экран выводится некое изображение. Вы можете щелкнуть где-нибудь над ним клавишей мыши. В результате появится окошко сообщений, где будут показаны координаты той точки, где в этот момент находилась мышь.
Код скрипта:

```
<layer>
<a href="#" onClick="alert('x: ' + event.x + ' y: ' + event.y); return false;">
<img src="davinci.jpg" width=209 height=264 border=0></a>
</layer>
```

Как видите, в тэг <a> мы поместили программу обработки событий onClick, как это мы уже делали в предшествующих версиях JavaScript. Новое здесь заключается в том, что для создания окошка с сообщением мы используем event.x и event.y. А это как раз и есть объект Event, который здесь нам нужен, чтобы узнать координаты мыши.
К тому же я поместил все команды в тэг <layer>. Благодаря этому мы получаем в сообщении координаты относительно данного слоя, т.е. в нашем случае относительно самого изображения. В противном же случае мы получили бы координаты относительно окна браузера. (инструкция return false; используется здесь для того, чтобы браузер обрабатывал далее данную ссылку)
Объект Event получил следующие свойства (их мы рассмотрим в следующих примерах):
Data - Массив адресов URL оставленных объектов, когда происходит событие DragDrop.
LayerX - Горизонтальное положение курсора (в пикселах) относительно слоя. В комбинации с событием Resize это свойство представляет ширину окна браузера.
LayerY - Вертикальное положение курсора (в пикселах) относительно слоя. В комбинации с событием Resize это свойство представляет высоту окна браузера.
modifiers - Строка, задающая ключи модификатора - ALT_MASK, CONTROL_MASK, META_MASK or SHIFT_MASK
pageX - Горизонтальное положение курсора (в пикселах) относительно окна браузера.
pageY - Вертикальное положение курсора (в пикселах) относительно окна браузера.
screenX - Горизонтальное положение курсора (в пикселах) относительно экрана.
screenY - Вертикальное положение курсора (в пикселах) относительно экрана.
target - Строка, представляющая объект, которому исходно было послано событие.
type - Строка, указывающая тип события.
which - ASCII-значение нажатой клавиши или номер клавиши мыши.
x - Синоним layerX.
y - Синоним layerY.

Перехват события

Одна из важных особенностей языка - перехват события. Если кто-то, к примеру, щелкает на кнопке, то вызывается программа обработки события onClick, соответствующая этой кнопке. С помощью обработки событий Вы можете добиться того, чтобы объект, соответсвующий

вашему окну, документу или слою, перехватывал и обрабатывал событие еще до того, как для этой цели объектом указанной кнопки будет вызван обработчик событий. Точно так же объект вашего окна, документа или слоя может обрабатывать сигнал о событии еще до того, как он достигает своего обычного адресата.
Чтобы увидеть, для чего это может пригодиться, давайте рассмотрим следующий пример:

```
<html>
<head>
<script language="JavaScript">
window.captureEvents(Event.CLICK);
window.onclick= handle;
function handle(e) {
alert("Объект window перехватывает это событие!");
return true; // т.е. проследить ссылку
}
</script>
</head>
<body>
<a href="test.htm">Click on this link</a>
</body>
</html>
```

Как видно, мы не указываем программы обработки событий в тэге <a>. Вместо этого мы пишем

```
window.captureEvents(Event.CLICK);
```

с тем, чтобы перехватить событие Click объектом window. Обычно объект window не работает с событием Click. Однако, перехватив, мы затем его переадресуем в объект window. Заметим, что в Event.CLICK фрагмент CLICK должен писаться заглавными буквами. Если же Вы хотите перехватывать несколько событий, то Вам следует отделить их друг от друга символами |. Например:

```
window.captureEvents(Event.CLICK | Event.MOVE);
```

Помимо этого в функции handle(), назначенной нами на роль обработчика событий, мы пользуемся инструкцией return true;. В действительности это означает, что браузер должен обработать и саму ссылку, после того, как завершится выполнение функции handle(). Если же Вы напишете вместо этого return false;, то на этом все и закончится.
Если теперь в тэге <a> Вы зададите программу обработки события onClick, то поймете, что данная программа при возникновении данного события вызвана уже не будет. И это не удивительно, поскольку объект window перехватывает сигнал о событии еще до того, как он достигает объекта link. Если же Вы определите функцию handle() как

```
function handle(e) {
alert("The window object captured this event!");
window.routeEvent(e);
return true;
}
```

то компьютер будет проверять, определены ли другие программы обработки событий для данного объекта. Переменная *e* - это наш объект Event, передаваемый функции обработки событий в виде аргумента.
Кроме того, Вы можете непосредственно послать сигнал о событии какому-либо объекту. Для этого Вы можете воспользоваться методом handleEvent(). Это выглядит следующим образом:

```
<html>
<script language="JavaScript">
window.captureEvents(Event.CLICK);
window.onclick= handle;
```

```
function handle(e) {
document.links[1].handleEvent(e);
}
</script>
<a href="test.htm">"Кликните" по этой ссылке</a><br>
<a href="test.htm"
onClick="alert('Обработчик событий для второй ссылки!');">Вторая ссылка</a>
</html>
```

Все сигналы о событиях Click, посылаются на обработку по второй ссылке - даже если Вы вовсе и не щелкнули ни по одной из ссылок!

Следующий скрипт демонстрирует, как Ваш скрипт может реагировать на сигналы о нажатии клавиш. Нажмите на какую-либо клавишу и посмотрите, как работает этот скрипт.

```
<html>
<script language="JavaScript">
window.captureEvents(Event.KEYPRESS);
window.onkeypress= pressed;
function pressed(e) {
alert("Key pressed! ASCII-value: " + e.which);
}
</script>
</html>
```

работоспособности описанного выше метода в домашних условиях, у вас возникнут проблемы. Дело в том, что для приема данных нужен сервер. Windows без наворотов такие запросы не понимает – выдает сообщение об ошибке. Если у вас не стоит Apache или сервер Windows NT, и вы не хотите с ними возиться - и не надо, установите Small HTTP Server. Эта манюсенькая программа позволит вам тестировать в домашних условиях CGI, SSI, PHP и т.д.

Объекты JavaScript

anchor (массив *anchors*)	location
button	Math
checkbox	navigator
Date	password
document	radio
массив *elements*	reset
form (массив *forms*)	string
frame (массив *frames*)	submit
hidden	text
history	textarea
link (массив *links*)	window

Объект anchor (массив anchors)

Фрагмент текста, который может быть помещен в гиперссылку.

Синтаксис:

Для определения anchor используется стандартный HTML синтаксис.

```
<A [HREF=locationorURL]
NAME="anchorName"
[TARGET="windowName"]>
anchorText
</a>
```

HREF=*locationorURL* идентифицирует назначение якоря или URL. Если этот атрибут представлен, то объект anchor также является объектом link.

NAME=*"anchorName"* определяет таг, который является доступной гипертекстовой ссылкой внутри текущего документа.

TARGET=*"windowName"* определяет окно, в которое будет загружаться ссылка. Этот атрибут имеет смысл, только если представлен HREF=*locationorURL*. Смотрите также link.

anchorText определяет текст, отображаемый якорем.

Вы можете также определить якорь, используя метод anchors.

Свойство:

- document

Описание:

Если объект anchor является также объектом link, то объект входит в массивы anchors и links.

Массив anchors

Вы можете ссылаться на объекты anchor в вашей программе, используя массив anchors. Этот массив содержит запись для каждого тага <a>, содержащего атрибут NAME по порядку встречаемости в документе. Например, если документ содержит три поименованных якоря, то эти якоря представлены как document.anchor[0], document.anchor[1], document.anchor[2].

Использование массива anchors:

1. document.anchors[*index*]
2. document.anchors.length

index целое число, представляющее якорь в документе.

Для получения количества якорей в документе используется свойство length: document.anchors.length.

Хотя массив *anchors* представляет собой поименованные якоря, значение anchors[*index*] является всегда нулевым. Но если в документе якоря именуются по порядку натуральными числами, вы можете использовать массив *anchors* и его свойство length для употребления имени якоря перед использованием его в операторах, таких как установка location.hash.

Элементы массива anchors окрыты открыты только для чтения. Например, выражение document.anchors[0]="anchor1" не имеет эффекта.

Свойства:

Объект anchor не имеет свойств.

Массив anchors имеет следующие свойства:

- length определяет число поименованных якорей в документе.

Методы:

- нет

События:

- нет

Объект button

Изменен в Navigator 3.0.

Нажимаемая кнопка в HTML форме.

Синтаксис:

Определение кнопки:

```
<INPUT
TYPE="button"
NAME="buttonName"
VALUE="buttonText"
[onClick="handlerText"]>
```

NAME=*"buttonName"* определяет имя объекта button. Вы можете получить это значение, используя свойство name.

VALUE=*"buttonText"* определяет текст, отображаемый на кнопке. Вы можете получить это значение, используя свойство value.

1. *buttonName.propertyName*
2. *buttonName.methodName(parameters)*
3. *formName*.elements[*index*].*propertyName*
4. *formName*.elements[*index*].*methodName(parameters)*

buttonName значение атрибута NAME объекта button.
formName значение атрибута NAME объекта form или элемента в массиве *forms*.
index целое число, представляющее объект button в форме.
propertyName одно из свойств, описанных ниже.
methodName один из методов, описанных ниже.

Свойство:

- form

Описание:

Объект button в форме выглядит следующим образом:
Объект button является элементом формы и должен быть определен внутри тага <FORM>.
Объект button является обычной кнопкой, которую вы можете использовать для выполнения действия, определенного вами. Кнопка выполняет скрипт, определенный событием onClick.

Свойства:

- name отражает атрибут NAME
- value отражает атрибут VALUE

Методы:

- click

События:

- onClick

Смотрите также:

- объекты form,reset и submit.

Объект checkbox

Изменен в Navigator 3.0.
Контрольный переключатель (checkbox) в HTML форме. checkbox является сенсорным переключателем, позволяющим пользователю устанавливать значение on или off.

Синтаксис:

```
TYPE="checkbox"
NAME="checkboxName"
VALUE="checkboxValue"
[CHECKED]
[onClick="handlerText"]>
textToDisplay
```

NAME="*checkboxName*" определяет имя объекта checkbox. Вы можете получить это значение, используя свойство name.
VALUE="*checkboxValue*" определяет значение, которое посылается серверу при выборе checkbox и отправке формы. По умолчанию это "on". Вы можете получить это значение, используя свойство value.
CHECKED определяет checkbox, отображаемый помеченным галочкой. Вы можете получить это значение, используя свойство defaultChecked.
textToDisplay определяет текст, отображаемый рядом с checkbox.
Использование свойств и методов объекта checkbox:

1. *checkboxName.propertyName*
2. *checkboxName.methodName(parameters)*

3. *formName*.elements[*index*].*propertyName*
4. *formName*.elements[*index*].*methodName(parameters)*

checkboxName значение атрибута NAME объекта checkbox.
formName любое значение атрибута NAME объекта form или элемента в массиве *forms*.
index целое число, представляющее объект checkbox в форме.
propertyName одно из свойств, описанных ниже.
methodName один из методов, описанных ниже.

Свойство:

- form

Описание:

Объект checkbox в форме выглядит следующим образом:
Объект checkbox является элементом формы и должен быть определен внутри тага <FORM>.
Свойство checked используется для определения checkbox, помеченного галочкой в настоящий момент. Свойство defaultChecked используется для определения checkbox, помеченного галочкой при загрузке формы.

Свойства:

- checked позволяет вам в программе установить какой checkbox будет помечен галочкой.
- defaultChecked отражает атрибут CHECKED.
- name отражает атрибут NAME.
- value отражает атрибут VALUE.

Методы:

- click

События:

- onClick

Смотрите также:

- объекты form и radio.

Объект Date

Изменен в Navigator 3.0.
Позволяет вам работать с датой и временем.

Синтаксис:

Определение объекта Date:

1. *dateObjectName* = new Date()
2. *dateObjectName* = new Date(*"month day, year hours:minutes:seconds"*)
3. *dateObjectName* = new Date(*year, month, day*)
4. *dateObjectName* = new Date(*year, month, day, hours, minutes, seconds*)

dateObjectName любое имя нового объекта или свойство существующего объекта.
year, month, day, hours, minutes, seconds строковые значения для 2 формы синтаксиса. Для 3 и 4 - целочисленные значения.
Использование методов Date:
`dateObjectName.methodName(parameters)`
dateObjectName любое имя существующего объекта Date или свойство существующего объекта.
methodName один из методов, описанных ниже.
Исключение: методы parse и UTC объекта Date являются статическими методами, которые вы используются следующим образом:
`Date.UTC(parameters)`
`Date.parse(parameters)`

Свойство:

- нет

Описание:

Объект Date является встроенным объектом JavaScript.

Формой 1 синтаксиса создаются текущие дата и время. Если вы пропускаете часы, минуты или секунды в формах 2 и 4 синтаксиса, то будет установлено нулевое значение.

Способ обращения к датам JavaScript очень похож на способ Java: оба языка имеют много одинаковых методов date и оба хранят даты внутренне как количество миллисекунд с 1 января 1970 00:00:00. Даты, предшествующие 1970 г. не допускаются.

Свойства:

- нет

Методы:

- getDate
- getDay
- getHours
- getMinutes
- getMonth
- getSeconds
- getTime
- getTimezoneOffset
- getYear
- parse
- setDate
- setHours
- setMinutes
- setMonth
- setSeconds
- setTime
- setYear
- toGMTString
- toLocaleString
- UTC

События:

- нет. Встроенные объекты не имеют событий.

Объект document

Изменен в Navigator 3.0.

Содержит информацию о текущем документе и обеспечен методами отображения HTML-документа.

Синтаксис:

Для определения объекта document используется стандартный HTML синтаксис:

```
<BODY
BACKGROUND="backgroundImage"
BGCOLOR="backgroundColor"
TEXT="foregroundColor"
LINK="unfollowedLinkColor"
ALINK="activatedLinkColor"
VLINK="followedLinkColor"
[onLoad="handlerText"]
[onUnload="handlerText"]>
```

```
</BODY>
```

BACKGROUND определяет картинку, которая выполняет роль фона документа.
BGCOLOR, TEXT, LINK, ALINK, VLINK определяет цвет как шестиразрядное шестнадцатиричное число (в формате "rrggbb" или "#rrggbb") или как одно из строковых названий в Color Value.
Использование свойств и методов объекта document:

1. document.*propertyName*
2. document.*methodName(parameters)*

propertyName одно из свойств, описанных ниже.
methodName один из методов, описанных ниже.
Свойство:

- window

Описание:

HTML документ состоит из тагов <HEAD> и <BODY>. <HEAD> содержит информацию о заголовке документа и основании (абсолютный URL основания, используемый для относительных URL ссылок в документе). Таг <BODY> заключает в себе тело документа, который определен текущим URL. Все тело документа (все другие элементы HTML документа) находятся внутри тага <BODY>.
Вы можете загрузить новый документ, используя объект location.
Вы можете ссылаться на якоря, формы и ссылки документа, используя массивы *anchors, forms* и *links*. Эти массивы содержат запись для каждого якоря, формы и ссылки в документе.
Свойства:

- alinkColor отражает атрибут ALINK
- anchors массив, отражающий все якоря в документе
- bgColor отражает атрибут BGCOLOR
- cookie определяет "ключик"
- fgColor отражает атрибут TEXT
- forms массив, отражающий все формы в документе
- lastModified отражает дату последней модификации документа
- linkColor отражает атрибут LINK
- links массив, отражающий все ссылки в документе
- referrer отражает URL документа, из которого вызван текущий документ
- title отражает содержание тага <TITLE>
- URL отражает полный URL документа
- vlinkColor отражает атрибут VLINK

Следующие объекты также являются свойствами объекта document:

- anchor
- form
- history
- link

Методы:

- close
- open
- write
- writeln

События:

- нет. События onLoad и onUnload определяются в таге <BODY>, но являются событиями объекта window.

Смотрите также:

- объекты frame и window

Массив elements

Массив объектов, содержащий элементы формы (такие как объекты checkbox, radio и text) по порядку встречаемости.

Синтаксис:

1. *formName*.elements[*index*]
2. *formName*.elements.length

formName любое имя формы или элемента в массиве forms.
index целое число, представляющее объект в форме.

Свойство:

- form

Описание:

Вы можете ссылаться на элементы формы в вашей программе, используя массив *elements*. Этот массив содержит запись для каждого объекта (button, checkbox, password, radio, select, submit, text, textarea) в форме по порядку встречаемости. Например, если форма содержит поле text и два элемента checkbox, то эти элементы выглядят так *formName*elements[0], *formName*elements[1], *formName*elements[2].

Хотя вы можете также ссылаться на элементы формы, используя имя элемента (из атрибута NAME), массив elements позволяет ссылаться на объекты формы без использования их имен. Например, если первый объект в форме *userInfo* является объектом text *userName*, вы можете получить его значение любым из следующих способов:

```
userInfo.userName.value
userInfo.userName[0].value
```

Для получения количества элементов формы, используется свойство length:

```
formName.elements.length
```

Каждая кнопка radio в объекте radio представляется как отдельный элемент в массиве *elements*.

Элементы в массиве *elements* открыты только для чтения. Например, выражение *formName*.elements[0]="music" не имеет эффекта.

Значение каждого элемента в массиве *elements* является полным HTML выражением для объекта.

Свойства:

- length отражает количество элементов формы

Смотрите также:

- объект form

Объект form (массив forms)

Изменен в Navigator 3.0.

Позволяет пользователям вставлять текст и делать изменения из объектов формы таких как графические опции, селекторные кнопки и списки элементов. Вы можете также использовать форму для отправки данных серверу.

Синтаксис:

Для определения формы используется стандартный синтаксис HTML с добавлением события onSubmit:

```
<FORM
NAME="formName"
TARGET="windowName"
```

```
ACTION="serverURL"
METHOD=GET | POST
ENCTYPE="encodingType"
[onSubmit="handlerText"]>
</FORM>
```

NAME="*formName*" определяет имя объекта form.

TARGET="*windowName*" определяет окно, в которое загружается результат передачи формы. Когда вы используете форму с атрибутом TARGET, сервер показывает ответы в окне *windowName* вместо окна, содержащего форму. *windowName* может быть существующим окном, именем фрейма, определенного в таге <FRAMESET> или одним из имен фрейма _top, _parent, _self или _blank; оно не может быть выражением JavaScript (например, parent.*frameName* или *windowName*.*frameName*). Некоторые значения для этого атрибута могут требовать определенных значений для других атрибутов. Смотрите RFC 1867. Вы можете получить это значение, используя свойство target.

ACTION="*serverURL*" определяет URL сервера, для которого поле формы вводит информацию is sent. Этот атрибут может указывать приложения CGI или LiveWire на сервере, это может также быть mailto: URL если форма отправляет почту. Смотрите объект location, где описаны компоненты URL. Некоторые значения для этого атрибута могут требовать определенных значений для других атрибутов. Смотрите RFC 1867. Вы можете получить это значение, используя свойство action.

METHOD=GET | POST определяет метод передачи информации серверу, определенному ACTION. GET (по умолчанию) добавляет введенную информацию к URL, которая в большинстве принимающих систем становится значением переменной окружения QUERY_STRING. POST отправляет вводимую информацию в теле данных, которое является доступным на *stdin* с длинной данных в переменной окружения CONTENT_LENGTH. Некоторые значения для этого атрибута могут требовать определенных значений для других атрибутов. Смотрите RFC 1867. Вы можете получить это значение, используя свойство method.

ENCTYPE="*encodingType*" определяет MIME кодировку данных, установленную: "application/x-www-forum-urlencoded" (по умолчанию) или "multipart/form-data". Некоторые значения для этого атрибута могут требовать определенных значений для других атрибутов. Смотрите RFC 1867. Вы можете получить это значение, используя свойство encoding.

Использование свойств и методов объекта form:

1. *formName.propertyName*
2. *formName.methodName(parameters)*
3. forms[*index*].*propertyName*
4. forms[*index*].*methodName(parameters)*

formName значение атрибута NAME объекта form.
propertyName одно из свойств, описанных ниже.
methodName один из методов, описанных ниже.
index целое число, представляющее объект form.

Свойство:

- document

Описание:

Каждая форма в документе является отдельным объектом.

Вы можете ссылаться на элементы формы в вашей программе, используя имя элемента (из атрибута NAME) или массив *elements*. Массив *elements* содержит запись для каждого элемента (таких как объекты checkbox, radio или text) в форме.

Массив forms

Вы можете ссылаться на формы в вашей программе, используя массив *forms* (вы можете также использовать имя формы). Этот массив содержит запись для каждого объекта form (тага <FORM>) по порядку встречаемости в документе. Например, если документ содержит три формы, то эти формы представлены так document.forms[0], document.forms[1] и document.forms[2].

Использование массива *forms*:

1. document.forms[*index*]
2. document.forms.length

index целое число, представляющее форму в документе.

Для получения количества форм в документе используется свойство length: document.forms.length.

Вы можете также обращаться к элементам формы, используя массив *forms*. Например, вы обращаетесь к объекту text с именем quantity во второй форме так:

```
document.forms[1].quantity.
```

Элементы массива *forms* открыты только для чтения. Например, выражение document.forms[0]="music" не имеет эффекта.

Значение каждого элемента в массиве *forms* является <object *nameAttribute*>, где *nameAttribute* является атрибутом NAME формы.

Свойства:

Объект form имеет следующие свойства:

- action отражает атрибут ACTION
- elements массив, отражающий все элементы в форме
- encoding отражает атрибут ENCTYPE
- length отражает количество элементов в форме
- method отражает атрибут METHOD
- target отражает атрибут TARGET

Следующие объекты являются также свойствами объекта form:

- button
- checkbox
- hidden
- password
- radio
- reset
- select
- submit
- text
- textarea

Массив *forms* имеет следующие свойства:

- length отражает количество форм в документе

Методы:

- submit

События:

- onSubmit

Смотрите также:

- объекты button, checkbox, hidden, password, radio, reset, select, submit, text, textarea.

Объект frame (массив frames)

Изменен в Navigator 3.0.

Окно, которое может показывать на одном экране несколько независимо прокручиваемых фреймов, каждый из которых имеет свой собственный URL. Фреймы могут указывать на различные URL'и и быть ссылкой других URL'ей, все внутри одного экрана.

Синтаксис:

Для определения объекта frame используется стандартный HTML синтаксис. События onLoad и onUnload определяются в таге <FRAMESET>, но являются событиями объекта window:

```
<FRAMESET
ROWS="rowHeightList"
COLS="columnWidthList"
[onLoad="handlerText"]
[onUnload="handlerText"]>
[<FRAME SRC="locationorURL" NAME="frameName">]
</FRAMESET>
```

ROWS="*rowHeightList*" через запятую указывается набор значений, определяющих высоту фрейма. Можно определить единицу измерения, по умолчанию это пиксели.

COLS="*columnWidthList*" через запятую указывается набор значений, определяющих ширину фрейма. Можно определить единицу измерения, по умолчанию это пиксели. <FRAME> определяет фрейм

SRC="*locationorURL*" определяет URL документа, показываемого во фрейме. URL не может включать имя якоря, например, <FRAME SRC="doc2.html#colors" NAME="frame2" - это не правильно. Смотрите объект location, где описаны компоненты URL.

NAME="*frameName*" определяет имя, используемое как ссылка для перехода по гиперссылкам.

Использование свойств объекта frame:

1. [*windowReference*.]*frameName.propertyName*
2. [*windowReference*.]frames[*index*].*propertyName*
3. window.*propertyName*
4. self.*propertyName*
5. parent.*propertyName*

windowReference переменная *windowVar* из определения окна (смотрите объект window) или один из синонимов top или parent.

frameName значение атрибута NAME в таге <FRAME> объекта frame.

index целое число, представляющее объект frame.

propertyName одно из свойств, описанных ниже.

Свойство:

- Объект frame является свойством window
- Массив frames является свойством frame и window

Описание:

Таг <FRAMESET> используется в HTML документе, единственная его цель - определить расположение фреймов, составляющих страницу. Каждый фрейм является объектом window.

Если таг <FRAME> содержит атрибуты SRC и NAME, вы можете ссылаться на этот фрейм из фрейма, находящегося на том же уровне иерархии, используя parent.*frameName* или parent.frames[*index*]. Например, если четвертый фрейм в установке имеет NAME="homeFrame", то фреймы, находящиеся на том же уровне иерархии, могут ссылаться на этот фрейм, используя parent.homeFrame или parent.frames[3].

Свойства self и window являются синонимами для текущего фрейма, вы можете использовать их для ссылок в текущем фрейме.

Свойства top и parent являются также синонимами, которые могут использоваться вместо имени фрейма. top ссылается на самое верхнее окно, содержащее фреймы или nested framesets, и parent ссылается на окно, содержащее текущий frameset. Смотрите свойства top и parent.

Массив frames

Вы можете ссылаться на объекты frame в вашей программе, используя массив *frames*. Этот массив содержит запись для каждого фрейма-потомка (тага <FRAME>) в окне, содержащем таг <FRAMESET> по порядку встречаемости. Например, если окно содержит три фрейма-потомка, эти фреймы отображаются как parent.frames[0], parent.frames[1], parent.frames[2].

Использование массива frames:

1. [*frameReference*.]frames[*index*]
2. [*frameReference*.]frames.length
3. [*windowReference*.]frames[*index*]
4. [*windowReference*.]frames.length

frameReference действительный путь ссылки на фрейм, описанный в объекте frame.

windowReference переменная *windowVar* из определения окна (смотрите объект window) или один из синонимов top или parent.

index целое число, представляющее количество фреймов в родительском окне.

Для получения количества фреймов-потомков в окне или фрейме используется свойство length:

```
[windowReference.]frames.length
[frameReference.]frames.length
```

Элементы в массиве *frames* открыты только для чтения. Например, выражение [*windowReference*.]frames[0]="frame1" не имеет эффекта.

Значение каждого элемента в массиве *frames* является <object *nameAttribute*>, *nameAttribute* является атрибутом NAME фрейма.

Свойства:

Объект frame имеет следующие свойства:

- frames массив, отражающий все фреймы окна
- name отражает атрибут NAME тага <FRAME>
- length отражает количество фреймов-потомков внутри фрейма
- parent синоним для окна или фрейма, содержащего текущий фрейм
- self синоним для текущего фрейма
- window синоним для текущего фрейма

Массив *frames* имеет следующие свойства:

- length отражает количество фреймов-потомков внутри фрейма

Методы:

- clearTimeout
- setTimeout

События:

- нет. События onLoad и onUnload определяются в таге <FRAMESET>, но являются событиями для объекта window.

Смотрите также:

- объекты document и window.

Объект hidden

Изменен в Navigator 3.0.

Текстовый объект формы, который не отображается в HTML форме. Объект hidden используется для передачи пар имя/значение при загрузке формы.

Синтаксис:

Определение объекта hidden:

```
<INPUT
TYPE="hidden"
NAME="hiddenName"
[VALUE="textValue"]>
```

NAME="*hiddenName*" определяет имя объекта hidden. Вы можете получить это значение, используя свойство name.

VALUE="*textValue*" определяет начальное значение объекта hidden.

Использование свойств объекта hidden:

1. *hiddenName.propertyName*
2. *formName*.elements.[*index*].*propertyName*

hiddenName значение атрибута NAME объекта hidden.

formName любое значение атрибута NAME объекта form или элемента массива *forms*.

index целое число, представляющее объект hidden в форме.

propertyName одно из свойств, описанных ниже.

Свойство:

- form

Описание:

Объект hidden является элементом формы и должен быть определен внутри тага <FORM>.

Объект hidden не может быть увиден и изменен пользователем, но вы можете запрограммировать изменение значения объекта, изменяя свойство value. Вы можете использовать объекты hidden для коммуникаций клиент/сервер.

Свойства:

- name отражает атрибут NAME
- value отражает текущее значение объекта hidden

Методы:

- нет

События:

- нет

Смотрите также:

- свойство cookie

Объект history

Содержит информацию о URL'ях, которые клиент посещал внутри окна. Эта информация сохраняется и доступна через меню Go Navigator'a.

Синтаксис:

Использование объекта history:

1. history.*propertyName*
2. history.*methodName(parameters)*

propertyName одно из свойств, описанных ниже.

methodName один из методов, описанных ниже.

Свойство:

- document

Описание:

Объект history связанным списком URL'ей, посещенных пользователем, как показано в меню Go Navigator'a.

Свойства:

- length отражает количество записей в объекте history

Методы:

- back
- forward
- go

События:

- нет

Смотрите также:

- свойство location

Объект link (массив links)

Изменен в Navigator 3.0.

Кусок текста или картинка, определенные как гипертекстовая ссылка. При выборе пользователем ссылки в тексте, в окно загружается документ, соответствующий этой гипертекстовой ссылке.

Синтаксис:

Для определения ссылки используется стандартный HTML синтаксис с добавлением событий onClick и onMouseOver:

```
<A HREF=locationorURL
[NAME="anchorName"]
[TARGET="windowName"]
[onClick="handlerText"]
[onMouseOver="handlerText"]>
linkText
</A>
```

A HREF=*locationorURL* идентифицирует место назначение якоря или URL. Смотрите объект location, где описаны компоненты URL.

NAME="*anchorName*" определяет таг, который становиться доступной гипертекстовой ссылкой внутри текущего документа. Если этот атрибут представлен, объект link является также объектом anchor. Смотрите anchor.

TARGET="*windowName*" определяет окно, в которое загружается обозначенный ссылкой документ. *windowName* может быть существующим окном, это также может быть имя фрейма, определенного в таге <FRAMESET>, или одно из literal имен фреймов _top, _parent, _self или _blank, это не может быть выражением JavaScript (например, это не может быть parent.*frameName* или *windowName.frameName*).

linkText отображается как гипертекстовая ссылка на URL.

Вы можете также определить ссылку, используя метод link.

Использование свойств объекта link:

```
document.links[index].propertyName
```

index целое число, отражающее объект link.

propertyName одно из свойств, описанных ниже.

Свойство:

- document

Описание:

Каждый объект link является объектом location и имеет те же свойства как и объект location.

Если объект link также является объектом anchor, то объект записан в массивах *anchors* и *links*.

Когда пользователь выбирает объект link и переходит в документ, обозначенный ссылкой (определенный HREF=*locationorURL*), то этот документ содержит URL документа источника.

Массив links

Вы можете ссылаться на объекты link в вашей программе, используя массив *links*. Этот массив содержит запись для каждого объекта link (тага <A HREF="">) по порядку встречаемости в документе. Например, если документ содержит три объекта link, то эти ссылки представлены так document.links[0], document.links[1] и document.links[2].

Использование массива *links*:

1. document.links[*index*]
2. document.links.length

index целое число, представляющее ссылку в документе.

Для получения количества ссылок в документе используется свойство length: document.links.length.

Элементы в массиве *links* открыты только для чтения. Например, выражение document.links[0]="link1" не имеет эффекта.

Свойства:

Объект link имеет следующие свойства:

- hash определяет имя якоря в URL
- host определяет hostname:port часть URL'a
- hostname определяет хост и доменное имя или IP адрес сетевого хоста
- href определяет запись URL
- pathname определяет url-path часть URL'a
- port определяет коммуникационный порт, который сервер использует для коммуникаций
- protocol определяет начало URL, включая двоеточие
- search определяет запрос
- target отражает атрибут TARGET

Массив *links* имеет следующие свойства:

- length отражает количество ссылок в документе

Методы:

- нет

События:

- onClick
- onMouseOver

Смотрите также:

- объект anchor
- метод link

Объект location

Изменен в Navigator 3.0.

Содержит информацию о текущем URL.

Синтаксис:

Использование объекта location:

`[`*`windowReference`*`.]location[.`*`propertyName`*`]`

windowReference переменная *windowVar* из определения окна (смотрите объект window) или один из синонимов top или parent.

propertyName одно из свойств, описанных ниже. Пропуск имени свойства является равносильным определению свойства href (полный URL).

Свойство:

- window

Описание:

Объект location представляет собой полный URL. Каждое свойство объекта location представляет собой отдельную часть URL.

Следующий формат URL показывает связь между location свойствами:

protocol//hostname:port pathname search hash

protocol представляет собой начало URL, включая первое двоеточие.

hostname представляет хост и доменное имя или IP адрес сетевого хоста.

port представляет коммуникационный порт, который сервер использует для коммуникаций.

pathname представляет url-path часть URL'a.

search представляет любой запрос в URL'e, начинающийся со знака вопроса.

hash представляет имя якоря фрагмент в URL'e, начинающийся со знака #.

Смотрите описание свойств ниже, где более детально описаны различные части URL, или свойство href.
Объект location имеет еще два свойства, не показанных в формате:
href представляет полный URL.
host представляет набор *hostname:port*.
Объект location содержится в объекте window. Если вы ссылаетесь на объект location без определения окна, то объект location представляется как текущий location.
Если вы ссылаетесь на объект location и определяете имя окна, например, *windowReference*.location.*propertyName*, то объект location представляется как location определенного окна.
Не путайте объект location со свойством location объекта document. Вы не можете изменить значение свойства location (document.location), но вы можете изменить значение свойств объекта location (window.location.*propertyName*). document.location является строковым значением, которое обычно равно window.location.href, который устанавливается когда вы загружаете документ, но перенаправление может изменить его.
Синтаксис для общеизвестных типов URL:

URL type	**Protocol**	**Example**
JavaScript	javascript:	javascript:history.go(-1)
Navigator info	about:	about:cache
Word Wide Web	http:	http://www.netscape.com/
File	file:	file:///javascript/methods.html
FTP	ftp:	ftp://ftp.mine.com/home/mine
MailTo	mailto:	mailto:info@netscape.com
Usenet	news:	news://news.scruznet.com/comp.la ng.javascript
Gopher	gopher:	gopher.myhost.com

javascript:protocol оценивает выражение после двоеточия (:), если оно есть, и загружает страницу, содержащую строковое значение выражения, если оно не определено. Если выражение не определено, то новая страница не загружается.
about:protocol обеспечивает информацией Navigator и имеет следующий синтаксис:

```
about:[cache|plugins]
```

about: является равносильным выбору About Netscape из Help меню Navigator'a.
about:cache показывает disk cache статистики.
about:plug-ins показывает информацию о сконфигурированных вами plug-ins'ax. Это равносильно выбору About Plug-ins из Help меню Navigator'a.

Свойства:

- hash определяет имя якоря в URL
- host определяет hostname:port часть URL'a
- hostname определяет хост и доменное имя или IP адрес сетевого хоста
- href определяет запись URL
- pathname определяет url-path часть URL'a
- port определяет коммуникационный порт, который сервер использует для коммуникаций
- protocol определяет начало URL, включая двоеточие
- search определяет запрос

Методы:

- нет

События:

- нет

Смотрите также:

- объект history
- свойство URL

Объект Math

Изменен в Navigator 3.0.

Встроенный объект, имеющий свойства и методы для математических констант и функций. Например, свойство PI объекта Math имеет значение Пи.

Синтаксис:

Использование объекта Math:

1. Math.*propertyName*
2. Math.*methodName(parameters)*

propertyName одно из свойств, описанных ниже.
methodName один из методов, описанных ниже.

Свойство:

- нет

Описание:

Объект Math является встроенным объектом JavaScript.

Вы ссылаетесь на константу PI как Math.PI. Константы определены в JavaScript с точностью до действительных чисел.

Аналогично, вы ссылаетесь на функции Math как на методы. Например, функция синуса - Math.sin(*argument*), где *argument* является аргументом функции.

Выражение with удобно при использовании нескольких констант и методов Math, так как не нужно указывать тип "Math" для каждой константы или метода. Например,

```
with (Math) {
a = PI*r*r
y = r*sin(theta)
x = r*cos(theta)
}
```

Свойства:

- E
- LN2
- LN10
- LOG2E
- LOG10E
- PI
- SQRT1_2
- SQRT2

Методы:

- abs
- acos
- asin
- atan
- ceil
- cos
- exp
- floor

- log
- max
- min
- pow
- random
- round
- sin
- sqrt
- tan

События:

- нет. Встроенные объекты не имеют событий.

Объект navigator

Изменен в Navigator 3.0.
Содержит информацию о используемой версии Navigator'a.
Синтаксис:
Использование объекта navigator:
`navigator.`*`propertyName`*

propertyName одно из свойств, описанных ниже.
Свойство:

- нет

Описание:
Объект navigator используется для определения версии Navigator'a ваших пользователей.
Свойства:

- appCodeName определяет кодовое имя броузера
- appName определяет имя броузера
- appVersion определяет версию броузера
- userAgent определяет заголовок пользовательского агента

Методы:

- нет

События:

- нет

Смотрите также:

- объект link
- объект anchor

Объект password

Изменен в Navigator 3.0.
Текстовое поле в HTML форме, значение которого на экране отображается звездочками (*). Когда пользователь вводит текст в это поле, звездочки (*) скрывают введенное значение.
Синтаксис:
Для определения объекта password используется стандартный HTML синтаксис:
```
<INPUT
TYPE="password"
NAME="passwordName"
[VALUE="textValue"]
SIZE=integer>
```
NAME="*passwordName*" определяет имя объекта password. Вы можете получить это значение, используя свойство name.

VALUE="*textValue*" определяет первоначальное значение объекта password. Вы можете получить это значение, используя свойство defaultValue.

SIZE=*integer* определяет количество символов объекта password, вмещающихся без прокрутки.

Использование свойств и методов объекта password:

1. *passwordName.propertyName*
2. *passwordName.methodName*
3. *formName*.elements[*index*].*propertyName*
4. *formName*.elements[*index*].*methodName(patameters)*

passwordName значение атрибута NAME объекта password.

formName любое значение атрибута NAME объекта form или элемента в массиве *forms*.

index целое число, представляющее объект password в форме.

propertyName одно из свойств, описанных ниже.

methodName один из методов, описанных ниже.

Свойство:

- form

Описание:

Объект password в форме выглядит следующим образом:

Объект password является элементом формы и должен быть определен внутри тага <FORM>.

Свойства:

- defaultValue отражает атрибут VALUE
- name отражает атрибут NAME
- value отражает текущее значение поля объекта password

Методы:

- focus
- blur
- select

События:

- нет

Смотрите также:

- объекты form и text

Объект radio

Изменен в Navigator 3.0.

Установка статических кнопок (кнопок radio) в HTML форме. Установка кнопок radio позволяет пользователю выбрать один пункт из списка.

Синтаксис:

Для определения установки кнопок radio используется стандартный HTML синтаксис с добавлением события onClick:

```
<INPUT
TYPE="radio"
NAME="radioName"
VALUE="buttonValue"
[CHECKED]
[onClick="handlerText"]>
textToDisplay
```

NAME="*radioName*" определяет имя объекта radio. Все кнопки radio в группе имеют одинаковый атрибут NAME. Вы можете получить это значение, используя свойство name.

VALUE="*buttonValue*" определяет значение, которое возвращается серверу, когда radio кнопка выбирается и форма утверждается. По умолчанию это "нет". Вы можете получить это значение, используя свойство value.

CHECKED определяет, что кнопка radio выбрана. Вы можете получить это значение, используя свойство defaultChecked.

textToDisplay определяет текст, отображаемый рядом с кнопкой radio.

Использование свойств и методов объекта radio:

1. *radioName*[*index1*].*propertyName*
2. *radioName*[*index1*].*methodName(parameters)*
3. *formName*.elements[*index2*].*propertyName*
4. *formName*.elements[*index2*].*methodName(parameters)*

radioName значение атрибута NAME объекта radio.

index1 целое число, представляющее кнопку radio в объекте radio.

formName любое значение атрибута NAME объекта form или элемента в массиве *forms*.

index2 целое число, представляющее кнопку radio в форму. Массив

elements содержит записи для каждой кнопке radio в объекте radio.

propertyName одно из свойств, описанных ниже.

methodName(parameters) один из методов, описанных ниже.

Свойство:

- нет

Описание:

Объект radio в форме выглядит следующим образом:

Объект radio является элементом формы и должен быть определен внутри тага <FORM>.

Все кнопки radio в группе кнопок radio используют одинаковое свойство name. Для обращения к отдельным кнопкам radio в вашей программе, используйте имя объекта с индексом, начинающимся с нуля, для каждой кнопки, также как вы это делали для массива, *forms*: document.forms[0].*radioName*[0] это первая, document.forms[0].*radioName*[1] это вторая и так далее.

Свойства:

- checked позволяет вам программно выбирать кнопку radio
- defaultChecked отражает атрибут CHECKED
- length отражает количество кнопок radio в объекте radio
- name отражает атрибут NAME
- value отражает атрибут VALUE

Методы:

- click

События:

- onClick

Смотрите также:

- объекты checkbox, form и select

Объект reset

Изменен в Navigator 3.0.

Кнопка сброса (кнопка reset)в HTML форме. Кнопка reset сбрасывает все элементы в форме в их значения, установленные по умолчанию.

Синтаксис:

Для определение кнопки reset используется стандартный HTML синтаксис с добавлением события onClick:

```
<INPUT
TYPE="reset"
NAME="resetName"
VALUE="buttonText"
[onClick="handlerText"]
```

NAME="*resetName*" определяет имя объекта reset. Вы можете получить это значение, используя свойство name.

VALUE="*buttonText*" определяет текст, отображаемый на кнопке. Вы можете получить это значение, используя свойство value.

Использование свойств и методов объекта reset:

1. *resetName.propertyName*
2. *resetName.methodName(parameters)*
3. *formName*.elements[*index*].*propertyName*
4. *formName*.elements[*index*].*methodName(parameters)*

resetName значение атрибута NAME объекта reset.

formName любое значение атрибута NAME объекта form или элемента в массиве *forms*.

index целое число, представляющее объект reset в форме.

propertyName одно из свойств, описанных ниже.

methodName один из методов, описанных ниже.

Свойство:

- form

Описание:

Объект reset в форме выглядит следующим образом:

Объект reset является элементом формы и должен быть описан внутри тага <FORM>.

Событие onClick кнопки reset не может предотвратить сброса формы; если вы нажали кнопку, сброс не может быть отменен.

Свойства:

- name отражает атрибут NAME
- value отражает атрибут VALUE

Методы:

- click

События:

- onClick

Объект string

Изменен в Navigator 3.0.

Ряд символов.

Синтаксис:

Использование объекта string:

1. *stringName.propertyName*
2. *stringName.methodName(parameters)*

stringName имя строковой переменной.

propertyName одно из свойств, описанных ниже.

methodName один из методов, описанных ниже.

Свойство:

- нет

Описание:

Объект string является встроенным объектом JavaScript.

Строка может быть представлена как литерал, заключенный в одинарные или двойные кавычки; например, "Netscape" или 'Netscape'.

Свойства:

- length определяет длину ряда

Методы:

- anchor
- big
- blink

- bold
- charAt
- fixed
- fontcolor
- fontsize
- indexOf
- italics
- lastIntexOf
- link
- small
- strike
- sub
- substring
- sup
- toLowerCase
- toUpperCase

События:

- нет. Встроенные объекты не имеют событий.

Смотрите также:

- объекты text и textarea

Объект submit

Изменен в Navigator 3.0.

Кнопка передачи данных (кнопка submit) в HTML форме. Кнопка submit вызывает передачу формы.

Синтаксис:

Для определения кнопки submit используется стандартный HTML синтаксис с добавлением события onClick:

```
<INPUT
TYPE="submit"
NAME="submitName"
VALUE="buttonText"
[onClick="handlerText"]>
```

NAME="*submitName*" определяет имя объекта submit. Вы можете получить это значение, используя свойство name.

VALUE="*buttonText*" определяет текст, отображаемый на кнопке. Вы можете получить это значение, используя свойство value.

Использование свойств и методов объекта submit:

1. *sibmitName.propertyName*
2. *submitName.methodName(parameters)*
3. *formName*.elements[*index*].*propertyName*
4. *formName*.elements[*index*].*methodName(parameters)*

sibmitName значение атрибута NAME объекта submit.
formName значение атрибута NAME объекта form или элемента в массиве *forms*.
index целое число, представляющее объект submit в форме.
propertyName одно из свойств, описанных ниже.
methodName(parameters) один из методов, описанных ниже.

Свойство:

- form

Описание:

Объект submit в форме выглядит следующим образом:

Объект submit является элементом формы и должен быть определен внутри тага <FORM>.

При щелчке на кнопке submit форма передается по URL, определенному в свойстве формы action. Этот action всегда загружает новую страницу клиенту; это может быть текущая страница, если action так определен или не определен вообще.

Событие onClick кнопки submit не может предотвратить передачу формы; вместо этого используйте событие onSubmit формы или вместо объекта submit используйте метод submit.

Свойства:

- name отражает атрибут NAME
- value отражает атрибут VALUE

Методы:

- click

События:

- onClick

Смотрите также:

- объекты button, form и reset
- метод submit
- событие onSubmit

Объект text

Изменен в Navigator 3.0.

Поле ввода текста в HTML форме. Текстовое поле позволяет пользователю вводить слова, фразы или числовой ряд.

Синтаксис:

Для определения объекта text используется стандартный HTML синтаксис с добавлением событий onBlur, onChange, onFocus, onSelect:

```
<INPUT
TYPE="text"
NAME="textName"
VALUE="textValue"
SIZE=integer
[onBlur="handlerText"]
[onChange="handlerText"]
[onFocus="handlerText"]
[onSelect="handlerText"]>
```

NAME="*textName*" определяет имя объекта text. Вы можете получить это значение, используя свойство name.

VALUE="*textValue*" определяет первоначальное значение объекта text. Вы можете получить это значение, используя свойство value.

SIZE=integer определяет количество символов объекта text, помещающихся без прокрутки.

Использование свойств и методов объекта text:

1. *textName.propertyName*
2. *textName.methodName(parameters)*
3. *formName*.elements[*index*].*propertyName*
4. *formName*.elements[*index*].*methodName(parameters)*

textName значение атрибута NAME объекта text.

formName значение атрибута NAME объекта form или элемента в массиве *forms*.

index целое число, представляющее объект text в форме.

propertyName одно из свойств, описанных ниже.

methodName один из методов, описанных ниже.

Свойство:

- form

Описание:

Объект text в форме выглядит следующим образом:

Объект text является элементом формы и должен быть описан внутри тага <FORM>.

Объект text может быть обновлен динамично установкой свойства value (this.value).

Свойства:

- defaultValue отражает атрибут VALUE
- name отражает атрибут NAME
- value отражает текущее значение поля объекта text

Методы:

- blur
- focus
- select

События:

- onBlur
- onChange
- onFocus
- onSelect

Смотрите также:

- объекты form, password, string и textarea

Объект textarea

Изменен в Navigator 3.0.

Многострочное поле ввода текста в HTML форме. Поле textarea позволяет пользователю вводить слова, фразы или числа.

Синтаксис:

Для определения текстовой области используется стандартный HTML синтаксис с добавлением событий onBlur, onChange, onFocus и onSelect:

```
<TEXTAREA
NAME="textareaName"
ROWS="integer"
COLS="integer"
WRAP="off|virtual|physical"
[onBlur="handlerText"]
[onChange="handlerText"]
[onFocus="handlerText"]
[onSelect="handlerText"]>
textToDisplay
</TAXTAREA>
```

NAME="*textareaName*" определяет имя объекта textarea. Вы можете получить это значение, используя свойство name.

ROWS="*integer*" и COLS="*integer*" устанавливает размер в символах отображаемого поля вода.

textToDisplay определяет первоначальное значение объекта textarea. textarea всегда только ASCII текст с разделением на строки.

Атрибут WRAP контролирует длину обрабатываемых строк в TEXTAREA. Значение "off" установлено по умолчанию - строки посылаются так, как они введены. Значение "virtual" отображает строки с переносами, но они посылаются так, как введены. Значение "physical" отображает строки с переносами и посылаются они с установленными переносами.

Использование свойств и методов textarea:

1. *textareaName.propertyName*
2. *textareaName.methodName(property)*
3. *formName*.elements[*index*].*propertyName*
4. *formName*.elements[*index*].*methodName(parameters)*

textareaName значение атрибута NAME объекта textarea.
formName значение атрибута NAME объекта form или элемента в массиве *forms*.
index целое число, представляющее объект textarea в форме.
propertyName одно из свойств, описанных ниже.
methodName один из методов, описанных ниже.

Свойство:

- form

Описание:

Объект textarea в форме выглядит следующим образом:
Объект textarea является элементом формы и должен быть определен внутри тага <FORM>.
Для начала новой строки в объекте textare вы можете использовать символ новой строки. Этот символ различен для разных платформ: в Unix - это \n, Windows - \r\n, Macintosh - \n. Одним из способов ввода символа новой строки программно является тестирование свойством appVersion для определения текущей платформы и установки символа новой строки таким образом. Смотрите примеры свойства appVersion.

Свойства:

- defaultValue отражает атрибут VALUE.
- name отражает атрибут NAME.
- value отражает текущее значение объекта textarea.

Методы:

- blur
- focus
- select

События:

- onBlur
- onChange
- onFocus
- onSelect

Смотрите также:

- объекты form, password, string и text.

Объект window

Изменен в Navigator 3.0.
Объект верхнего уровня для групп объектов document, location и history.

Синтаксис:

Для определения окна используется метод open:

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

windowVar имя нового окна. Эта переменная используется при ссылках на свойства, методы и контейнеры окна.
windowName имя окна, используемое в атрибуте TARGET тагов <FORM> и <A>.
Более подробное определение окна смотрите в методе open.
Использование свойств и методов window:

1. window.*propertyName*
2. window.*methodName(parameters)*
3. self.*propertyName*
4. self.*methodName(parameters)*
5. top.*propertyName*

6. top.*methodName(parameters)*
7. parent.*propertyName*
8. parent.*methodName(parameters)*
9. *windowVar.propertyName*
10. *windowVar.methodName(parameters)*
11. *propertyName*
12. *methodName(parameters)*

windowVar переменная, ссылающаяся на обект window. Смотрите синтаксис определения окна.
propertyName одно из свойств, описаныых ниже.
methodName один из методов, описанных ниже.
Для определения событий onLoad и onUnload для объекта window используются таги <BODY> и <FRAMESET>:

```
<BODY
...
[onLoad="handlerText"]
[onUnload="handlerText"]>
</BODY>
<FRAMESET
ROWS="rowHeightList"
COLS="columnWidthList"
[onLoad="handlerText"]
[onUnload="handlerText"]
[<FRAME SRC="locationorURL" NAME="frameName"]>
</FRAMESET>
```

Информацию о тагах <FRAMESET> и <BODY> смотрите в объектах document и frame.

Свойства:

- нет

Описание:

Объект window является объектом верхнего уровня в клиентской иерархии JavaScript. Объекты frame также являются окнами.

Свойства self и window являются синонимами для текущего окна, и вы можете использовать их для ссылки на текущее окно. Например, вы можете закрыть текущее окно, используя window.close() или self.close(). Вы можете использовать эти свойства для однозначного определения свойства self.status из формы, называемой status.

Свойства top и parent также являются синонимами и могут быть использованы вместо имени окна. top ссылается на самое верхнее окно Navigator-a, а parent ссылается на окно, содержащее frameset. Смотрите свойства top и parent. Поскольку допускается существование текущего окна, вам не нужно ссылаться на имя окна, когда вы объявляете его методы или назначаете свойства. Например, status="Jump to a new location" является действительным назначением свойства и close() является действительным вызовом метода. Однако, когда вы открываете или закрываете окно внутри события, вы должны определить window.open() или window.close() вместо того, чтобы использовать просто open() или close(). Благодаря to the scoping статических объектов в JavaScript, объявление close() без определения имени объекта равносильно document.close().

Вы можете ссылаться на объекты frame окна, используя массив frames. Массив frames содержит запись для каждого фрейма в окне с тагом <FRAMESET>.

У окон отсутствуют события пока в них не загружен некоторый HTML-документ, содержащий таги <BODY> или <FRAMESET>.

Свойства:

- defaultStatus отражает сообщение по умолчанию, отображаемое в строке состояния окна
- frames массив, отражающий все фреймы окна

- length отражает количество фреймов в родительском окне
- name отражает аргумент *windowName*
- parent является синонимом аргумента *windowName* и ссылается на окно, содержащее frameset
- self является синонимом аргумента *windowName* и ссылается на текущее окно
- status определяет текущее сообщение строки состояния окна
- top является синонимом аргумента *windowName* и ссылается на самое верхнее окно Navigator-а
- window является синонимом аргумента *windowName* и ссылается на текущее окно

Следующие объекты являются также свойствами объекта window:

- document
- frame
- location

Методы:

- alert
- close
- confirm
- open
- prompt
- setTimeout
- clearTimeout

События:

- onLoad
- onUnload

Смотрите также:

- объекты document и frame

1. Напишите скрипт, перехватывающий прерывания от клавиатуры.
2. Если был нажат пробел (код символа 32), произвести загрузку файла "index.htm";

Примечание: скрипт работает только в Нэтскейп 4.0 и выше.
Проверка скрипта: щелкнуть мышью в нижнем окне и нажать на клавишу пробел

Методы и функции JavaScript

abs	forward	setDate
acos	getDate	setHours
alert	getDay	setMinutes
anchor	getHours	setMonth
asin	getMinures	setSeconds
atan	getMonth	setTime
back	getSeconds	setTimeout
big	getTime	setYear
blink	getTimezoneOffset	sin
blur	getYear	small
bold	go	sqrt
ceil	indexOf	strike
charAt	isNaN	sub
clearTimeout	italics	submit
click	lastIndexOf	substring

close (объект document)	link	sup
close (объект window)	log	tan
confirm	max	toGMTString
cos	min	toLocaleString
escape	open (объект document)	toLowerCase
eval	open (объект window)	toUpperCase
exp	parse	unescape
fixed	parseFloat	UTC
floor	parseInt	write
focus	pow	writeln
fontcolor	prompt	
fontsize	random	

Метод abs

Возвращает абсолютное значение числа.

Синтаксис:

`Math.abs(`*`number`*`)`

number любое числовое выражение или свойство существующего объекта.

Метод:

Math

Метод acos

Возвращает арккосинус числа (в радианах).

Синтаксис:

`Math.acos(`*`number`*`)`

number числовое выражение между -1 и 1 или свойство существующего объекта.

Метод:

Math

Описание:

Метод acos возвращает числовое значение между 0 и Пи. Если значение *number* находится за пределами данного диапазона, возвращаемое значение всегда будет 0.

Смотрите также:

- методы asin, atan, cos, sin и tan.

Метод alert

Отображает диалоговое окно Alert с сообщением и кнопкой OK.

Синтаксис:

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

Метод:

window

Описание:

Метод alert используется для отображения сообщения, не требующего решения пользователя. Аргумент *message* определяет сообщение, которое содержит диалоговое окно.

Хотя alert является методом объекта window вам не нужно определять *windowReference*, при его вызове. Например, *windowReference*.alert() необязательно.

Смотрите также:

- методы confirm, prompt.

Метод anchor

Создает HTML якорь, который используется как гипертекстовая ссылка.

Синтаксис:

`text.anchor(nameAttribute)`

text любая строка или свойство существующего объекта.

nameAttribute любая строка или свойство существующего объекта.

Метод:

string

Описание:

Метод anchor используется с методами write или writeln для программного создания и отображения якоря в документе. Якорь создается с помощью метода anchor, а write или writeln используется для отображения якоря в документе.

В синтаксисе строка *text* представляет собой текст, который увидит пользователь. Строка *nameAttribute* представляет собой атрибут NAME тага <A>.

Якоря, созданные с помощью метода anchor становятся элементами массива anchors. Информацию о массиве anchors смотрите в объекте anchor.

Смотрите также:

- метод link

Метод asin

Возвращает арксинус числа (в радианах).

Синтаксис:

`Math.asin(number)`

number числовое выражение между -1 и 1 или свойство существующего объекта.

Метод:

Math

Описание:

Метод asin возвращает числовое значение между -Пи/2 и Пи/2. Если значение *number* находится за пределами данного диапазона, возвращаемое значение всегда будет 0.

Смотрите также:

- методы acos, atan, cos, sin, tan.

Метод atan

Возвращает арктангенс числа (в радианах).

Синтаксис:

`Math.atan(number)`

number любое числовое выражение или свойство существующего объекта, представляющее собой тангенс угла.

Метод:

Math

Описание:

Метод atan возвращает числовое выражение между -Пи/2 и Пи/2.

Смотрите также:

- методы acos, asin, cos, sin, tan.

Метод back

Позволяет вернуться на предыдущий URL в списке посещенных URL'ей.

Синтаксис:

`history.back()`

Метод:

history

Описание:

Этот метод выполняет действие равносильное выбору пользователем кнопки Back в окне Navigator'a. Метод back также равносилен history.go(-1).

Смотрите также:

- методы forward, go.

Метод big

Вызывает строку, отображаемою большим шрифтом, как если установить ей таг <BIG>.

Синтаксис:

```
stringName.big()
```

stringName любая строка или свойство существующего объекта.

Метод:

string

Описание:

Для форматирования и отображения строки в документе метод big используется с методами write или writeln.

Смотрите также:

- методы fontsize, small.

Метод blink

Вызывает мигающую строку, как если установить ей таг <BLINK>.

Синтаксис:

```
stringName.blink()
```

stringName любая строка или свойство существующего объекта.

Метод:

string

Описание:

Для форматирования и отображения строки в документе метод blink используется с методами write или writeln.

Смотрите также:

- методы bold, italics, strike.

Метод blur

Изменен в Navigator 3.0.

Убирает фокус с указанного объекта.

Синтаксис:

```
1. password.blur()
2. select.blur()
3. textName.blur()
4. textareaName.blur()
```

password любое значение атрибута NAME объекта password или элемент массива *elements*.
select любое значение атрибута NAME объекта select или элемент массива *elements*.
textName любое значение атрибута NAME объекта text или элемент массива *elements*.
textareaName любое значение атрибута NAME объекта textarea или элемент массива *elements*.

Метод:

password, select, text, textarea.

Описание:

Метод blur используется для удаления фокуса с указанного элемента формы.

Смотрите также:

- методы focus, select.

Метод bold

Вызывает строку, отображаемую жирным шрифтом, как если установить ей таг <B>.

Синтаксис:

```
stringName.bold()
```

stringName любая строка или свойство существующего объекта.

Метод:
string
Описание:
Для форматирования и отображения строки в документе метод bold используется с методами write или writeln.
Смотрите также:

- методы blink, italics, strike.

Метод ceil

Возвращает ближайшее целое числа, округленного в большую сторону или равное числу.
Синтаксис:

`Math.ceil(`*`number`*`)`
number любое числовое выражение или свойство существующего объекта.
Метод:
Math
Смотрите также:

- метод floor.

Метод charAt

Возвращает символ указанный в *index*.
Синтаксис:
`stringName``.charAt(`*`index`*`)`
stringName любая строка или свойство существующего объекта.
index любое целое число от 0 до *stringName*.length-1 или свойство существующего объекта.
Метод:
string
Описание:
Символы в строке индексируются слева направо. Индексом первого символа является 0, индексом последнего символа - *stringName*.length-1. Если вы указали *index* превышающий количество символов в строке, JavaScript возвратит пустую строку.
Смотрите также:

- методы indexOf, lastindexOf.

Метод clearTimeout

Окончание задержки, установленной методом setTimeout.
Синтаксис:
`clearTimeout(`*`timeoutID`*`)`
timeoutID задержка, установка которой была возвращена предыдущим вызовом метода setTimeout.
Метод:
frame, window
Описание:
Смотрите описание метода setTimeout
Смотрите также:

- метод setTimeout

Метод click

Имитирует щелчок мыши на выбранном элементе формы.
Синтаксис:

```
1. buttonName.click()
2. radioName[index].click()
3. checkboxName.click()
```

buttonName любое значение атрибута NAME объектов button, reset или submit или элемент массива *elements*.
radioName значение атрибута NAME объекта radio или элемент массива *elements*.
index целое число, представляющее кнопку radio в объекте radio.
checkboxName любое значение атрибута NAME объекта checkbox или элемент массива *elements*.

Метод:

button, checkbox, radio, reset, submit.

Описание:

Результат действия метода click изменяется в зависимости от вызываемого элемента:

- для button, reset и submit выполняется одинаковое действие - нажатие кнопки.
- для radio - выбор кнопки radio.
- для checkbox - отметка галочкой checkbox и установка значения на on.

Метод close (объект document)

Закрывает поток вывода и завершает вывод данных в рабочую область Navigator'a для отображения.

Синтаксис:

```
document.close()
```

Метод:

document

Описание:

Метод close закрывает поток вывода, открытый методом document.open(). Если поток был открыт для рабочей области Navigator'a, метод close завершает вывод содержимого потока на экран. Таги стиля шрифта, такие как <BIG> и <CENTER>, автоматически закрывают поток вывода. Метод close также останавливает "meteor shower" в иконе Navigator'a и отображает "Document: Done" в строке состояния.

Смотрите также:

- методы open, write, writeln.

Метод close (объект window)

Изменен в Navigator 3.0.
Закрывает указанное окно.

Синтаксис:

```
windowReference.close()
```

windowReference ссылка на окно, как описано в объекте window.

Метод:

window

Описание:

Метод close закрывает указанное окно. Если вы объявляете close без указания *windowReference*, то JavaScript закрывает текущее окно.
В событиях вы должны указывать window.close() вместо обычно используемого close(). Объявление close() без определения имени объекта равносильно document.close().

Смотрите также:

- метод open

Метод confirm

Отображает диалоговое окно с указанным сообщением и кнопками OK и Cancel.

Синтаксис:

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

message любая строка или свойство существующего объекта.

Метод:

window

Описание:

Метод confirm используется для принятия пользователем решения, требующего выбора OK или Cancel. Аргумент *message* определяет сообщение, которое требует решения пользователя. Метод confirm возвращает true, если пользователь выбрал OK, и false, если пользователь выбрал Cancel.

Хотя confirm является методом объекта window, вам не нужно указывать *windowReference* при его вызове. Например, *windowReference*.confirm() является необязательным.

Смотрите также:

- методы alert, prompt

Метод cos

Возвращает косинус числа.

Синтаксис:

`Math.cos(`*`number`*`)`

number числовое выражение, представляющее собой размер угла в радианах или свойство существующего объекта.

Метод:

Math

Описание:

Метод cos возвращает числовое значение между -1 и 1, которое представляет собой косинус угла.

Смотрите также:

- методы acos, asin, atan, sin, tan.

Функция escape

Возвращает ASCII значение аргумента, закодированного в ISO Latin-1.

Синтаксис:

`escape("`*`string`*`")`

string не буквенно-числовая строка в ISO Latin-1 кодировке или свойство существующего объекта.

Описание:

Функция escape не является методом, связанным с любым объектом, но является частью самого языка.

Значение, возвращаемое функцией escape, является строкой вида "%xx", где *xx* является ASCII кодировкой символа в аргументе. Если аргументом функции escape является буквенно-числовым символом, то функция escape возвращает тот же символ.

Смотрите также:

- функцию unescape.

Функция eval

Функция eval выполняет строку-аргумент и подставлает полученное значение вместо себя.

Синтаксис:

`eval("`*`string`*`")`

string любая строка, представляющая собой JavaScript выражение, команду или последовательность команд. Выражение может включать переменные и свойства существующего объекта.

Описание:

Функция eval является встроенной функцией JavaScript. Она не является методом, связанным с любым объектом, но является частью самого языка.

Аргументом функции eval является строка. Не используйте eval для вычислений арифметических выражений. JavaScript вычисляет арифметические выражения автоматически. Если аргумент представляет собой выражение, eval вычисляет выражение. Если аргумент представляет собой одно или более JavaScript команд, то eval выполняет команды.

Если вы построили арифметическое выражение как строку, вы можете использовать eval для ее вычисления.

Метод exp

Возвращает e^{number}, где *number* является аргументом, а *e* является экспонентой, основанием натурального логарифма.

Синтаксис:

`Math.exp(`*`number`*`)`

number любое числовое выражение или свойство существующего объекта.

Метод:

Math

Смотрите также:

- методы log, pow.

Метод fixed

Вызывает строку, отображаемую моноширинным шрифтом, как если установить ей таг <TT>.

Синтаксис:

`stringName``.fixed()`

stringName любая строка или свойство существующего объекта.

Метод:

string

Описание:

Для форматирования и отображения строки в документе метод fixed используется с методами write и writeln.

Метод floor

Возвращает ближайшее целое числа, округленного в меньшую сторону или равное числу.

Синтаксис:

`Math.floor(`*`number`*`)`

number любое числовое выражение или свойство существующего объекта.

Метод:

Math

Смотрите также:

- метод ceil.

Метод focus

Изменен в Navigator 3.0.

Устанавливает фокус на определенный объект.

Синтаксис:

```
1. password.focus()
2. select.focus()
3. textName.focus()
4. textareaName.focus()
```

password любое значение атрибута NAME объекта password или элемент массива *elements*.
select любое значение атрибута NAME объекта select или элемент массива *elements*.
textName любое значение атрибута NAME объекта text или элемент массива *elements*.
textareaName любое значение атрибута NAME объекта textarea или элемент массива *elements*.

Метод:

password, select, text, textarea.

Описание:

Метод focus используется для установки фокуса на указанный элемент формы. Вы можете затем программно ввести значение в элемент или позволить пользователю ввести значение.

Смотрите также:

- методы blur, select.

Метод fontcolor

Вызывает строку, отображаемую установленным цветом, как если поместить ее в таг <FONT COLOR=*color*>.

Синтаксис:

`stringName.fontcolor(color)`

stringName любая строка или свойство существующего объекта.

color строка или свойство существующего объекта, определяющая цвет как шестиразрядное шестнадцатиричное число (RGB) или как одно из строковых названий в списке Color Value.

Метод:

string

Описание:

Для форматирования и отображения строки в документе метод fontcolor используется с методами write и writeln.

Если вы определяете *color* как шестиразрядное шестнадцатиричное число вы должны использовать формат rrggbb.

Метод fontcolor анулирует значение, установленное в свойстве fgColor.

Метод fontsize

Вызывает строку, отображаемую установленным размером шрифта, как если поместить ее в таг <FONT SIZE=*size*>.

Синтаксис:

`stringName.fontsize(size)`

stringName любая строка или свойство существующего объекта.

size целое число от 1 до 7 или строка, представляющая собой целое со знаком (+ или -) от 1 до 7, или свойство существующего объекта.

Описание:

Для форматирования и отображения строки в документе метод fontsize используется с методами write и writeln.

Когда вы определяете *size* как целое, вы устанавливаете размер *stringName* в один из семи специфицированных размеров. Когда вы определяете *size* как "-2", вы устанавливаете размер шрифта *stringName* относительно размера, установленного в таге .

Смотрите также:

- методы big, small.

Метод forward

Загружает следующий URL в списке посещенных URL'ей.

Синтаксис:

`history.forward()`

Метод:

history

Описание:

Этот метод выполняет действие равносильное выбору пользователем кнопки Forward в окне Navigator'a. Метод forward также равносилен history.go(1).

Смотрите также:

- методы back, go.

Метод getDate

Возвращает число месяца для указанной даты.

Синтаксис:

`dateObjectName.getDate()`

dateObjectName любое имя объекта date или свойство существующего объекта.

Метод:
Date
Описание:
Значение, возвращаемое getDate, является целым числом от 1 до 31.
Смотрите также:

- метод setDate

Метод getDay

Возвращает день недели для указанной даты.
Синтаксис:
`dateObjectName.getDay()`
dateObjectName любое имя объекта date или свойство существующего объекта.
Метод:
Date
Описание:
Значение, возвращаемое getDay, является целым числом, соответствующим дню недели: ноль для воскресенья, один для понедельника, два для вторника и так далее.

Метод getHours

Возвращает часы для указанной даты.
Синтаксис:
`dateObjectName.getHours()`
dateObjectName любое имя объекта date или свойство существующего объекта.
Метод:
Date
Описание:
Значение, возвращаемое getHours, является целым числом от 0 до 23.
Смотрите также:

- метод setHours.

Метод getMinutes

Возвращает минуты для указанной даты.
Синтаксис:
`dateObjectName.getMinutes()`
dateObjectName любое имя объекта date или свойство существующего объекта.
Метод:
Date
Описание:
Значение, возвращаемое getMinutes, является целым числом от 0 до 59.
Смотрите также:

- метод setMinutes.

Метод getMonth

Возвращает месяц для указанной даты.
Синтаксис:
`dateObjectName.getMonth()`
dateObjectName любое имя объекта date или свойство существующего объекта.
Метод:
Date
Описание:
Значение, возвращаемое getMonth, является целым числом от 0 до 11. Ноль соответствует январю, один - февралю и так далее.
Смотрите также:

- метод setMonth.

Метод getSeconds

Возвращает секунды в текущем времени.

Синтаксис:

dateObjectName.getSeconds()

dateObjectName любое имя объекта date или свойство существующего объекта.

Метод:

Date

Описание:

Значение, возвращаемое getSeconds, является целым числом от 0 до 59.

Смотрите также:

- метод setSeconds.

Метод getTime

Возвращает числовое значение, соответствующее времени для указанной даты.

Синтаксис:

dateObjectName.getTime()

dateObjectName любое имя объекта date или свойство существующего объекта.

Метод:

Date

Описание:

Значение, возвращаемое методом getTime, является числом миллисекунд, начиная с 1 января 1970 00:00:00. Вы можете использовать этот метод для назначения даты и времени другому объекту date.

Смотрите также:

- метод setTime.

Метод getTimezoneOffset

Возвращает смещение временной зоны в минутах относительно гринвичского меридиана.

Синтаксис:

dateObjectName.getTimezoneOffset()

dateObjectName любое имя объекта date или свойство существующего объекта.

Метод:

Date

Описание:

Смещение временной зоны является разницей между местным временем и GMT (гринвичским временем). Сезонное время (зимнее, летнее) не дает возможности говорить об этом смещении как о константе.

Метод getYear

Возвращает год для указанной даты.

Синтаксис:

dateObjectName.getYear()

dateObjectName любое имя объекта date или свойство существующего объекта.

Метод:

Date

Описание:

Значение, возвращаемое getYear, равно году минус 1900. Например, если год равен 1976, то возвращаемое значение равно 76.

Смотрите также:

- метод setYear.

Метод go

Загружает URL из списка посещенных URL'ей.

Синтаксис:

`history.go(`*`delta | location`*`)`

delta целое число или свойство существующего объекта, представляющее собой относительную позицию в списке посещенных URL'ей.

location строка или свойство существующего объекта, представляющая собой URL или его часть из списка посещенных URL'ей.

Метод:

history

Описание:

Метод go позволяет перейти на адрес, содержащийся в списке посещенных URL'ей, который указан вами в качестве аргумента метода go. Вы можете посмотреть этот список, выбрав History в меню Window. Последние 10 позиций списка также отображаются в меню Go.

Аргумент *delta* может быть положительным и отрицательным числом. Если *delta* больше нуля, то метод go переходит на URL вперед в списке посещенных URL'ей; в противном случае переход осуществляется на URL назад. Если *delta* равна 0, то Navigator перезагружает текущую страницу.

Аргумент *location* является строкой. *location* выбирает для загрузки ближайший адрес в списке посещенных URL'ей, содержащий подстроку *location*, указанную вами в качестве аргумента. Каждая часть URL содержит определенную информацию. Смотрите объект location, где описаны компоненты URL.

Смотрите также:

- методы back, forward.

Метод indexOf

Возвращает индекс позиции впервые встреченного искомого значения в вызванном объекте string. Поиск начинается с *fromIndex*.

синтаксис:

`stringName``.indexOf(`*`searchValue, [fromIndex]`*`)`

stringName любая строка или свойство существующего объекта.

searchValue строка или свойство существующего объекта, представляющая собой искомое значение.

fromIndex место в вызванной строке, с которого начинается поиск. Это может быть любое целое число от 0 до *stringName*.length-1 или свойство существующего объекта.

Метод:

string

Описание:

Символы в строке индексируются слева направо. Индекс первого символа равен 0, индекс последнего - *stringName*.length-1.

Если вы не указываете значение *fromIndex*, JavaScript принимает по умолчанию 0. Если *searchValue* не найден, JavaScript возвращает -1.

Смотрите также:

- методы charAt, lastIndexOf.

Функция isNaN

Изменена в Navigator 3.0.

На UNIX платформах проверяет аргумент, является ли он "NaN" (не числом).

Синтаксис:

`isNaN(`*`testValue`*`)`

testValue значение, которое вы хотите проверить.

Описание:

Функция isNaN является встроенной функцией JavaScript. Она не является методом, связанным с любым объектом, но является частью самого языка. Функция isNaN применяется только на UNIX платформах.

На всех платформах, за исключением Windows, функции parseFloat и parseInt возвращают "NaN", когда они принимают нечисловое значение. Значение "NaN" не является числом в любом случае. Вы можете вызывать функцию NaN для того, чтобы определить является ли результат parseFloat или parseInt "NaN". Если над "NaN" совершаются арифметические операции, то их результатами также будет "NaN".

Функция isNaN возвращает true или false.

Смотрите также:

- функции parseFloat, parseInt.

Метод italics

Вызывает строку, отображаемую курсивом, как если установить ей таг <I>.

Синтаксис:

stringName.italics()

stringName любая строка или свойство существующего объекта.

Метод:

string

Описание:

Для форматирования и отображения строки в документе метод italics используется с методами write или writeln.

Смотрите также:

- методы blink, bold, strike.

Метод lastIndexOf

Возвращает индекс впервые встреченного искомого значения в вызванном объекте string. Поиск по строке осуществляется в обратном направлении, начиная с *fromIndex*.

Синтаксис:

stringName.lastindexOf(*searchValue, [fromIndex]*)

stringName любая строка или свойство существующего объекта.

searchValue строка или свойство существующего объекта, представляющая собой искомое значение.

fromIndex место в вызванной строке, с которого начинается поиск. Это может быть любое целое число от 0 до *stringName*.length-1 или свойство существующего объекта.

Метод:

string

Описание:

Символы в строке индексируются слева направо. Индекс первого символа равен 0, индекс последнего - *stringName*.length-1.

Если вы не указываете значение *fromIndex*, JavaScript принимает по умолчанию *stringName*.length-1 (конец строки). Если *searchValue* не найден, JavaScript возвращает -1.

Смотрите также:

- методы charAt, IndexOf.

Метод link

Создает гипертекстовую ссылку HTML, по которой можно перейти на другой URL.

Синтаксис:

linkText.link(*hrefAttribute*)

Метод:

string

Описание:

Для создания и отображения гипертекстовой ссылки в документе метод link используется с методами write или writeln. Создайте ссылку методом link, затем вызовите write или writeln для отображения ссылки в документе.

В синтаксисе строка *linkText* представляет собой текст, который увидит пользователь. Строка *hrefAttribute* представляет собой атрибут HREF тага <A>, это будет целевой URL. Каждая часть URL содержит определенную информацию. Смотрите объект location, где описаны компоненты URL.

Ссылки, созданные методом link, становятся элементами массива links.

Смотрите также:

- метод anchor.

Метод log

Возвращает натуральный логарифм числа (по основанию *e*).

Синтаксис:

`Math.log(`*`number`*`)`

number любое положительное числовое выражение или свойство существующего объекта.

Метод:

Math

Описание:

Если значение *number* находится за пределами диапазона, возвращенное значение всегда будет -1.797693134862316e+308.

Смотрите также:

- методы exp, pow.

Метод max

Возвращает большее число из двух.

Синтаксис:

`Math.max(`*`number1, number2`*`)`

number1 и *number2* любые числовые аргументы или свойства существующих объектов.

Метод:

Math

Смотрите также:

- метод min.

Метод min

Возвращает меньшее число из двух.

Синтаксис:

`Math.min(`*`number1, number2`*`)`

number1 и *number2* любые числовые аргументы или свойства существующих объектов.

Метод:

Math

Смотрите также:

- метод max.

Метод open (объект document)

Открывает поток для получения вывода методами write и writeln.

Синтаксис:

`document.open(["`*`mimeType`*`"])`

mimeType устанавливает любой из следующих типов документа:

```
text/html
text/plain
image/gif
image/jpeg
```

`image/x-bitmap`
`plug-In`
plug-In любой составной plug-in MIME тип, поддерживаемый Netscape'ом.
Метод:
document
Описание:
Метод open открывает поток для получения вывода методами write и writeln. Если *mimeType* является текстом или картинкой, то поток открыт в рабочую область Navigator'a; иначе, поток открыт на plug-in. Если документ уже существует в целевом окне, то метод open очищает его.
Для закрытия потока используйте метод document.close(). Метод close вызывает текст или картинку, которые были отправлены в рабочую область Navigator'a для отображения. После использования document.close(), введите document.open() снова, когда вы захотите начать вывод другого потока.
mimeType является необязательным аргументом, определяющим тип документа. Если вы не указываете *mimeType*, то метод open принимает по умолчанию text/html.
Описание *mimeType*:

- *text/html* определяет текст, содержащий ASCII текст в HTML формате.
- *text/plain* определяет текст, содержащий ASCII текст с символами конца строки, для ограничения отображаемых строк.
- *image/gif* определяет документ с закодированными байтами, содержащий GIF заголовок и размеры в пикселях.
- *image/jpeg* определяет документ с закодированными байтами, содержащий JPEG заголовок и размеры в пикселях.
- *image/x-bitmap* определяет документ с закодированными байтами, содержащий bitmap заголовок и размеры в пикселях.
- *plug-in* загружает определенный plug-in и использует его как место назначения для методов write и writeln. Например, "x-world/vrtml" загружет VR Scout VRML plug-in из Chaco Communications, a "aplication/x-director" загружает Macromedia Shockware plug-in.

Смотрите также:

- методы close, write, writeln.

Метод open (объект window)

Открывает новое окно web-броузера.
Синтаксис:
`[windowVar=][window].open("URL", "windowName", ["windowFeatures"])`
windowVar имя нового окна. Эта переменная используется при ссылках на свойства, методы и контейнеры окна.
URL определяет URL, открываемый в новом окне. Смотрите объект location, где описаны компоненты URL.
windowName имя окна, используемое в атрибуте TARGET тага <FORM> или <A>. *windowName* может содержать только буквенно-цифровые символы или символ подчеркивания (_).
windowFeatures список через запятую любых из следующих опций или значений:

```
toolbar[=yes | no] | [=1 | 0]
location[=yes | no] | [=1 | 0]
directoties[=yes | no] | [=1 | 0]
status[=yes | no] | [=1 | 0]
menubar[=yes | no] | [=1 | 0]
scrollbars[=yes | no] | [=1 | 0]
resizable[=yes | no] | [=1 | 0]
width=pixels
height=pixels
```

Вы можете использовать любой набор этих опций. Опции разделяются запятой. Не делайте пробелов между опциями.
pixels положительное целое число, определяющее размеры окна в пикселях.
Метод:
window
Описание:
Метод open открывает новое окно web-броузера клиента, что равносильно выбору New WebBrowser из меню File Navigator'a. Аргумент *URL* определяет URL, содержащийся в новом окне. Если *URL* является пустой строкой, то создастся пустое окно.
В событиях вы должны указывать window.open() вместо обычно используемого open(). Объявление open() без определения имени объекта равносильно document.open().
windowFeatures является необязательным списком перечисленных через запятую опций для нового окна. Булевы опции *windowFeatures* принимают значение true, если они определены без значений, или как yes или 1. Например, open("", "messageWindow", "toolbar") и open("", "messageWindow", "toolbar=1") как в первом, так и во втором случае опция toolbar принимает значение true. Если *windowName* не определяет существующего окна и вы не определяете *windowFeatures*, то все булевы опции *windowFeatures* принимают по умолчанию значение true. Если вы определяете любую из опций *windowFeatures*, то все остальные опции принимают значение false, если вы их не определите дополнительно.
Описание *windowFeatures*:

- *toolbar* создает стандартные рабочие инструменты Navigator'a, с такими кнопками как "Back" и "Forward".
- *location* создает поле ввода Location.
- *directories* создает кнопки стандартных директорий Navigator'a, такие как "What's New" и "What's Coll".
- *status* создает строку состояния внизу окна.
- *menubar* создает меню вверху окна.
- *scrollbars* создает горизонтальную и вертикальную прокрутки, когда документ больше, чем размер окна.
- *resizable* позволяет пользователю изменять размер окна.
- *width* определяет ширину окна в пикселях.
- *height* определяет высоту окна в пикселях.

Смотрите также:

- метод close.

Метод parse

Возвращает количество миллисекунд в строковом представлении даты, начиная с 1 января 1970 00:00:00, по местному времени.
Синтаксис:
`Date.parse(`*`dateString`*`)`
Метод:
Date
Описание:
Метод parse выдает дату в строковом представлении (например, "Dec 25, 1995") и возвращает количество миллисекунд, начиная с 1 января 1970 00:00:00 (по местному времени). Эта функция используется для установки значений даты, основанных на строковом значении, например, в сочетании с методом setTime и объектом Date.
Полученная строка представляет собой время, parse возвращает значение времени. Она принимается в стандартном синтаксисе даты IETF: "Mon, 25 Dec 1995 13:30:00 GMT". Она понимает континентальную US временную зону, но в основном, используется временная зона смещения, например "Mon, 25 Dec 1995 13:30:00 GMT+0430" (4 часа, 30 минут западнее

Гринвича). Если вы не указали временной зоны, принимается местная временная зона. GMT и UTC считаются эквивалентными.

Так как функция parse является статическим методом Date, вы всегда используете ее как Date.parse(), а не как метод созданного вами объекта date.

Смотрите также:

- метод UTC

Функция parseFloat

Анализирует строковый аргумент и возвращает число с плавающей точкой.

Синтаксис:

`parseFloat(`*`string`*`)`

string строка, представляющая собой значение, которое вы хотите проанализировать.

Описание:

Функция parseFloat является встроенным объектом JavaScript. Она не является методом, связанным с любым объектом, но является частью самого языка.

Функция parseFloat анализирует строку-аргумент и возвращает число с плавающей точкой. Если встреченный им символ отличается от знака (+ или -), цифры (0-9), десятичной точки или экспоненты, то он возвращает значение до этой точки, игнорируя этот символ и все последующие символы.

Если первый символ не может быть конвертирован в число, parseFloat возвращает одно из следующих значений:

- "пусто" на Windows платформах.
- "NaN" на любых других платформах указывает на то, что значение не является числом.

Смотрите также:

- методы isNaN, parseInt.

Функция parseInt

Анализирует строковый аргумент и возвращает целое число, определенное как основание.

Синтаксис:

`parseInt(`*`string [,radix]`*`)`

string строка, которая представляет собой значение, которое вы хотите проанализировать.

radix целое число, представляющее собой основание, возвращаемого значения.

Описание:

Функция parseFloat является встроенным объектом JavaScript. Она не является методом, связанным с любым объектом, но является частью самого языка.

Функция parseFloat анализирует его первый аргумент-строку и пытается возвратить целое число, определенное как основание. Например, основание 10 означает перевод в десятичное число, 8 - восьмеричное, 16 - шестнадцатиричное, и т.д.

Если parseInt в указанном основании встречает символ, не являющийся числом, то он пропускает его и все следующие символы и возвращает целочисленное значение разобранное до точки. ParseInt усекает числа до целочисленных значений.

Если основание не определено или определено как 0, JavaScript принимает следующее:

- если ввод *string* начинается с "0x", то основание равно 16 (шестнадцатиричное).
- если ввод *string* начинается с "0", то основание равно 8 (восьмиричное).
- если ввод *string* начинается с любого другого значения, то основание равно 10 (десятичное).
- если первый символ не может быть конвертирован в число, parseFloat возвращает одно из следующих значений:
- "пусто" на Windows платформах.
- "NaN" на любых других платформах указывает на то, что значение не является числом.

Для арифметических целей значение "NaN" не является числом в любом случае. Вы можете вызвать функцию isNaN для того, чтобы определить является ли результат parseInt "NaN". Если "NaN" применить в арифметических операциях, то их результатами также будут "NaN".

Смотрите также:

- функции isNaN, parseFloat.

Метод pow

Возвращает *base* в степени *exponent*, т.е. $base^{exponent}$.

Синтаксис:

`Math.pow(`*`base, exponent`*`)`

base числовое выражение или свойство существующего объекта.

exponent числовое выражение или свойство существующего объекта. Если результат может оказаться недопустимым значением (например, pow(-1, 0.5), то возвращенное значение равно нулю.

Метод:

Math

Смотрите также:

- методы exp, log.

Метод prompt

Отображает диалоговое окно с сообщением и полем ввода.

Синтаксис:

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

message любая строка или свойство существующего объекта; строка отображается как сообщение.

inputDefault строка, целое число или свойство существующего объекта, представляющая собой значение вводимое в поле по умолчанию.

Метод:

window

Описание:

Метод prompt используется для отображения диалогового окна, требующего ввода текста пользователем. Если вы не определяете первоначальное значение для *inputDefault*, то диалоговое окно отображает значение <undefined>.

Хотя prompt является методом объекта window, вам не нужно определять *windowReference*, при его вызове. Например, *windowReference*.prompt() является не обязательным.

Смотрите также:

- методы alert, confirm.

Метод random

Изменен в Navigator 3.0.

Возвращает случайное число между нулем и единицей. Этот метод применяется только на UNIX платфомах.

Синтаксис:

`Math.random()`

Метод:

Math

Метод setDate

Устанавливает число месяца для указанной даты.

Синтаксис:

`dateObjectName``.setDate(`*`dayValue`*`)`

dateObjectName любое имя объекта date или свойство существующего объекта.

dayValue целое число от 1 до 31 или свойство существующего объекта, представляющего собой число месяца.

Метод:

Date

Смотрите также:

- метод getDate.

Метод setHours

Устанавливает часы для указанной даты.

Синтаксис:

`dateObjectName.setHours(hoursValue)`

dateObjectName любое имя объекта date или свойство существующего объекта.

hoursValue целое число от 0 до 23 или свойство существующего объекта, представляющее собой часы.

Метод:

Date

Смотрите также:

- метод getHours.

Метод setMinutes

Устанавливает минуты для указанной даты.

Синтаксис:

`dateObjectName.setMinutes(minutesValue)`

dateObjectName любое имя объекта date или свойство существующего объекта.

minutesValue целое число от 0 до 59 или свойство существующего объекта, представляющее собой минуты.

Метод:

Date

Смотрите также:

- метод getMinutes.

Метод setMonth

Устанавливает месяц для указанной даты.

Синтаксис:

`dateObjectName.setMonth(month Value)`

dateObjectName любое имя объекта date или свойство существующего объекта.

monthValue целое число от 0 до 11 (представляющее собой месяцы с января по декабрь) или свойство существующего объекта.

Метод:

Date

Смотрите также:

- метод getMonth.

Метод setSeconds

Устанавливает секунды для указанной даты.

Синтаксис:

`dateObjectName.setSeconds(secondsValue)`

dateObjectName любое имя объекта date или свойство существующего объекта.

secondsValue целое число от 0 до 59 или свойство существующего объекта.

Метод:

Date

Смотрите также:

- метод getSeconds.

Метод setTime

Устанавливает значение объекта date.

Синтаксис:

`dateObjectName.setTime(timevalue)`

dateObjectName любое имя объекта date или свойство существующего объекта.

timevalue целое число или свойство существующего объекта, представляющее собой количество миллисекунд, начиная с 1 января 1970 00:00:00.

Метод:

Date

Описание:

Метод setTime используется для добавления даты и времени другому объекту.

Смотрите также:

- метод getTime.

Метод setTimeout

Выполняет выражение по истечении установленного количества миллисекунд.

Синтаксис:

`timeoutID=setTimeout(expression, msec)`

timeoutID идентификатор, который используется только для окончания выполнения, используя метод clearTimeout.

expression строковое выражение или свойство существующего объекта.

msec числовое значение, числовой ряд или свойство существующего объекта в миллисекундах.

Метод:

frame, window

Описание:

Метод setTimeout выполняет выражение после установленного количества времени. Он не выполняет выражение многократно. Например, если метод setTimeout установлен на 5 секунд, то выражение выполнится через 5 секунд, но не каждые 5 секунд.

Смотрите также:

- метод clearTimeout.

Метод setYear

Устанавливает год для указанной даты.

Синтаксис:

`dateObjectName.setYear(yearValue)`

dateObjectName любое имя объекта date или свойство существующего объекта.

timevalue целое число больше чем 1900 или свойство существующего объекта.

Метод:

Date

Смотрите также:

- метод getYear.

Метод sin

Возвращает синус числа.

Синтаксис:

`Math.sin(number)`

number числовое выражение или свойство существующего объекта, представляющее собой величину угла в радианах.

Метод:

Math

Описание:

Метод sin возвращает числовое значение между -1 и 1, представляющее собой синус угла.

Смотрите также:

- методы acos, asin, atan, cos, tan.

Метод small

Выводит строку, отображаемую маленьким шрифтом, как если установить ей таг <SMALL>.

Синтаксис:

`stringName.small()`

stringName любая строка или свойство существующего объекта.

Метод:

string

Описание:

Для форматирования и отображения строки в документе метод small используется с методами write или writeln.

Смотрите также:

- методы big, fontsize.

Метод sqrt

Возвращает квадратный корень числа.

Синтаксис:

`Math.sqrt(number)`

number любое неотрицательное числовое выражение или свойство существующего объекта.

Метод:

Math

Описание:

Если значение *number* находится за пределами данного диапазона, возвращенное значение всегда будет 0.

Метод strike

Выводит строку, отображаемую как перечеркнутый текст, как если установить ей таг <STRIKE>.

Синтаксис:

`stringName.strike()`

stringName любая строка или свойство существующего объекта.

Метод:

string

Описание:

Для форматирования и отображения строки в документе метод strike используется с методами write или writeln.

Смотрите также:

- методы blink, bold, italics.

Метод sub

Выводит строку, отображаемую как нижний индекс, как если установить ей таг <SUB>.

Синтаксис:

`stringName.sub()`

stringName любая строка или свойство существующего объекта.

Метод:

string

Описание:

Для форматирования и отображения строки в документе метод sub используется с методами write или writeln.

Смотрите также:

- методы sup.

Метод submit

Передает форму.

Синтаксис:

formName.submit()

formName любая строка или свойство существующего объекта.

Метод:

form

Описание:

Метод submit передает указанную форму. Он выполняет такое же действие как кнопка submit.

Метод submit используется для передачи данных http-серверу. Метод submit возвращает данные, используя методы "get" или "post", определенные в свойстве method.

Смотрите также:

- объект submit.
- свойство onSubmit.

Метод substring

Возвращает подстроку объекта string.

Синтаксис:

stringName.substring(*indexA, indexB*)

stringName любая строка или свойство существующего объекта.

indexA любое целое число от 0 до *stringName*.length-1 или свойство существующего объекта.

indexB любое целое число от 0 до *stringName*.length-1 или свойство существующего объекта.

Метод:

string

Описание:

Символы в строке индексируются слева направо. Индекс первого символа равен 0, индекс последнего - *stringName*.length-1.

Если *indexA* меньше чем *indexB*, то метод substring возвращает подстроку, начиная с символа *indexA* и заканчивая символом перед *indexB*. Если *indexA* больше чем *indexB*, то метод substring возвращает подстроку, начиная с символа *indexB* и заканчивая символом перед *indexA*. Если *indexA* равен *indexB*, то метод substring возвращает пустую строку.

Метод sup

Выводит строку, отображаемую как нижний индекс, как если установить ей таг <SUP>.

Синтаксис:

stringName.sup()

*stringName*любая строка или свойство существующего объекта.

Метод:

string

Описание:

Для форматирования и отображения строки в документе метод sup используется с методами write или writeln.

Смотрите также:

- методы sub.

Метод tan

Возвращает тангенс числа.

Синтаксис:

Math.tan(*number*)

number числовое выражение, представляющее собой величину угла в радианах, или свойство существующего объекта.

Метод:

Math

Описание:

Метод tan возвращает числовое значение, представляющее собой тангенс угла.

Смотрите также:

- методы acos, asin, atan, cos, sin.

Метод toGMTString

Переводит дату в строку, используя среднее гринвичское время (GMT).

Синтаксис:

`dateObjectName.toGMTString()`

dateObjectName любое имя объекта date или свойство существующего объекта.

Метод:

Date

Описание:

Точный формат значения возвращаемого toGMTString зависит от платформы.

Смотрите также:

- методы toLocaleString.

Метод toLocaleString

Переводит дату в строку, используя местный часовой пояс.

Синтаксис:

`dateObjectName.toLocaleString()`

dateObjectName любое имя объекта date или свойство существующего объекта.

Метод:

Date

Описание:

Если вы для перевода даты используете toLocaleString, помните, что различные locales собирают строку в различных путях. Используйте методы getHours, getMinutes, getSeconds для получения более переносимых результатов.

Смотрите также:

- методы toGMTString.

Метод toLowerCase

Возвращает значение вызванной строки, переведенной в нижний регистр.

Синтаксис:

`stringName.toLowerCase()`

stringName любая строка или свойство существующего объекта.

Метод:

string

Описание:

Метод toLowerCase возвращает значение *stringName*, переведенное в нижний регистр. toLowerCase не изменяет значения *stringName*.

Смотрите также:

- методы toUpperCase.

Метод toUpperCase

Возвращает значение вызванной строки, переведенной в верхний регистр.

Синтаксис:

`stringName.toUpperCase()`

stringName любая строка или свойство существующего объекта.

Метод:

string

Описание:

Метод toUpperCase возвращает значение *stringName*, переведенное в верхний регистр. toUpperCase не изменяет значения *stringName*.

Смотрите также:

- методы toLowerCase.

Функция unescape

Возвращает ASCII строку для указанного значения.

Синтаксис:

`unescape("string")`

string строка или свойство существующего объекта, содержащие символы в любой из следующих форм:

- *"%integer"*, где *integer* - число между 0 и 255 (десятичное)
- *"hex"*, где *hex* - число между 0x0 и 0xFF (шестнадцатиричное)

Описание:

Функция unescape не является методом, связанным с каким-либо объектом, но является частью самого языка. Строка, возвращаемая функцией unescape, является рядом символов в ISO Latin-1 кодировке.

Смотрите также:

- функцию escape.

Метод UTC

Возвращает количество миллисекунд в объект date, начиная с 1 января 1970 00:00:00, GMT.

Синтаксис:

`Date.UTC(year, month, day, [, hrs] [, min] [, sec])`

year год после 1990.
month месяц между 0-11.
day день месяца между 1-31.
hrs часы между 0-23.
min минуты между 0-59.
sec секунды между 0-59.

Метод:

Date

Описание:

UTC берет параметры даты, разделенные запятой, и возвращает количество миллисекунд, начиная с 1 января 1970 00:00:00, GMT.

Так как UTC является статическим методом Date, используйте его как Date.UTC(), а не как метод созданного вами объекта date.

Смотрите также:

- метод parse.

Метод write

Пишет одно или более HTML выражений в документ в указанном окне.

Синтаксис:

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

с *expression1* по *expressionN* любое JavaScript выражение или свойство существующего объекта.

Метод:

document

Описание:

Метод write отображает любое количество выражений в окне документа. Вы можете определить любое JavaScript выражение методом write, включая числовое, строковое или логическое.

Метод write является таким же как метод writeln, но метод write не добавляет символа перевода на новую строку в конец выходной информации.

Метод write используется внутри тага <SCRIPT> или внутри события. События выполняются после закрытия документа, поэтому метод write по умолчанию откроет новый документ с *mimeType* text/html, если вы не укажете метод document.open() в событии.

Смотрите также:

- методы close, open, writeln.

Метод writeln

Пишет одно или более HTML выражений в документ в указанном окне, добавляя символ перевода на новую строку в конец выходной информации.

Синтаксис:

`document.writeln(`*`expression1 [,expression2], ... [,expressionN]`*`)`

с *expression1* по *expressionN* любое JavaScript выражение или свойство существующего объекта.

Метод:

document

Описание:

Метод writeln отображает любое количество выражений в окне документа. Вы можете определить любое JavaScript выражение методом write, включая числовое, строковое или логическое.

Метод writeln является таким же как метод write, но метод writeln добавляет символ перехода на новую строку в конец выходной информации. HTML игнорирует символ новой строки, за исключением определенных тагов, таких как <PRE>.

Метод writeln используется внутри любого тага <SCRIPT> или внутри события. События выполняются после закрытия документа, поэтому метод writeln по умолчанию откроет новый документ с *mimeType* text/html, если вы не укажете метод document.open() в событии.

Смотрите также:

- методы close, open, write.