1. Основы языка JavaScript

1. *Основные особенности JavaScript*

JavaScript — это относительно простой объектно-ориентированный язык, предназначенный для создания небольших клиентских и серверных приложений для Internet. Программы, написанные на языке JavaScript, включаются в состав HTML-документов и распространяются вместе с ними. Программы просмотра (браузеры – от англ. browser) типа Netscape Navigator и Microsoft Internet Explorer распознают встроенные в текст документа программы-вставки (script-коды) и выполняют их. Таким образом, JavaScript — интерпретируемый язык программирования. Примерами программ на JavaScript могут служить программы, проверяющие введенные пользователем данные или выполняющие какие-то действия при открытии или закрытии документа. Такие программы могут реагировать на действия пользователя — нажатие кнопок "мыши", ввод данных в экранной форме или перемещение "мыши" по странице. Более того, JavaScript-программы могут управлять самим браузером и атрибутами документа.

Язык JavaScript, будучи схожим по синтаксису с языком Java, за исключением объектной модели, в то же время не обладает такими свойствами, как статические типы данных и строгой типизацией. В JavaScript, в отличие от Java, понятие классов не является основой синтаксических конструкций языка. Такой основой является небольшой набор предопределенных типов данных, поддерживаемых исполняемой системой: числовые, булевские и строковые; функции, которые могут быть как самостоятельными, так и методами объектов (метод в терминологии JavaScript — не что иное, как функция/подпрограмма); объектная модель с большим набором предопределенных объектов со своими свойствами и методами, а также правилами задания в программе пользователя новых объектов.

Для создания программ на JavaScript не требуется никаких дополнительных средств— необходим лишь браузер, поддерживающий язык JavaScript соответствующей версии и текстовый редактор, позволяющий создавать HTML-документы. Так как программа на JavaScript встраивается

непосредственно в текст HTML-документа, вы можете немедленно увидеть результаты своей работы во время просмотра документа браузером и при необходимости внести изменения.

2. *Возможности языка JavaScript*

С его помощью можно динамически управлять отображением и содержимым HTML-документов. Можно записывать в отображаемый на экран документ произвольный HTML-код в процессе синтаксического анализа загруженного браузером документа. С помощью объекта Document можно генерировать документы "с нуля" в зависимости от предыдущих действий пользователя или каких-либо иных факторов.

JavaScript позволяет контролировать работу браузера. Например, объект Window поддерживает методы, позволяющие выводить на экран всплывающие диалоговые окна, создавать, открывать и закрывать новые окна браузера, задавать режимы прокрутки и размеры окон и т.д.

JavaScript позволяет взаимодействовать с содержимым документов. Объект Document и содержащиеся в нем объекты позволяют программам читать части HTML-документа и иногда взаимодействовать с ними. Сам текст прочитать невозможно, но можно, например, получить список гипертекстовых ссылок, имеющихся в данном документе. На текущий момент широкие возможности взаимодействия с содержимым документов обеспечивает объект Form и объекты, которые он может содержать: Button, Checkbox, Hidden, Password, Radio, Reset, Select, Submit, Text и Textarea.

JavaScript обеспечивает взаимодействие с пользователем. Важной особенностью этого языка является реализованная в нем возможность определять обработчики событий — произвольные порции кода, которые выполняются при наступлении конкретных событий (обычно действий пользователя). JavaScript позволяет использовать в качестве обработчиков событий любые новые предварительно заданные функции. Например, можно написать программу, которая выведет в строке состояния специальное сообщение, если пользователь установит указатель "мыши" на гипертекстовую ссылку, или выведет на экран диалоговое окно с запросом на подтверждение выполнения некоторого действия, или осуществит проверку введенного пользователем значения и в случае ошибки ввода выдаст соответствующую диагностику и заставит ввести правильное значение.

JavaScript дает возможность выполнять произвольные математические вычисления. Кроме того, этот язык имеет развитые средства работы со значениями даты и времени. JavaScript был создан в качестве альтернативы CGI-программам и языку сценариев Perl, а также в качестве дополнения м в ряде случаев альтернативы языку Java.

Ниже приведена таблица, в которой проводится сравнение Java и JavaScript:

JavaScript	**Java**
Исходный код программ встраивается непосредственно в HTML-документ либо загружается из независимых файлов.	*Исходный код программ* не распространяется с приложением -апплетом. Апплеты загружаются с сервера из независимых файлов.
Программа выкладывается на сервер в виде исходного кода в текстовой форме и в дальнейшем *интерпретируется* (без предварительной компиляции) браузером после загрузки ее с сервера.	Программа *компилируется* в машинно-независимый байтовый код Java-код, после чего выкладывается на сервер. Браузер (виртуальная Java-машина) исполняет Java-код.
Объектный. Можно программировать как без использования объектного программирования, так и используя предопределенные встроенные классы. Имеется теоретическая возможность расширения этих классов, но реально она никогда не	*Объектно-ориентированный*. Программировать без использования объектного программирования нельзя. Апплеты состоят из классов с возможностью иерархического наследования по традиционной схеме наследования. Использование

используется.	наследования и полиморфизма – основа программирования в Java.
В отличие от подавляющего большинства языков объектного программирования в JavaScript *структура объектов* не задается однозначно устройством класса, а *является динамической* и может меняться на этапе выполнения программы. Объекты могут динамически получать новые поля и методы или изменять любые параметры старых.	*Структура объектов* полностью задается на этапе компиляции их классов.
Свободная типизация: элементарные типы данных переменных не описываются, при присваивании тип левой части всегда определяется по результату присваивания (т.е. по правой части)	*Строгая типизация*: типы данных любых переменных должны быть описаны перед использованием, тип левой части должен совпадать с типом правой (за редкими исключениями, когда работает автоматическое приведение типа результата к типу левой части)
Динамическое связывание кода с объектами: ссылки на объекты проверяются во время выполнения программы.	*Статическое связывание кода с объектами*: ссылки на объекты должны существовать на момент компиляции

3. Основные типы данных

Значения переменных, функций и выражений бывают следующих типов:

1. целые численные:

   в десятичной системе единиц: 0, 29, 70, -147 и т.п.;

   в 16-ричной: 0x70 или 0x70, 0XFA7D0 и т.п.;

   в 8-ричной: 070, 0710 (***Внимание!!!*** Ведущий ноль воспринимается как символ 8-ричного числа) и т.п.

2. вещественные численные:

   0.0, -2.9, 0.7E1, 14.7e-2, 1e+308 (максимальное вещественное число), 1.001e-305 (минимальное по модулю вещественное число, отличное от нуля) и т.п.;

3. логические (булевские): true и false;
4. строковые: "Привет, все!", "ОК", 'Слово "Привет!" с кавычками внутри строки', "Другой вариант 'Привет' с кавычками внутри строки" и т.п. (допускаются оба типа кавычек и многократное использование таких пар внутри друг друга). Специальные символы обозначаются комбинацией символа \ и буквы (или последовательности цифр), например: \b — "забой", \n — перевод на новую строку, \" — "кавычка".
5. null — специальное значение для обозначения “пустого множества” значений.

1. Переменные. Приведение типов

Глобальные переменные можно вводить в любом месте текста программы путем простого присваивания значения. Но необходимо, чтобы переменная была определена до того момента, когда вызывается при исполнении:

var myVariable=0.1

var B=false

и т.п. При этом тип переменной приводится к типу присваиваемого значения, причем в последующем этой же переменной можно присвоить значение другого типа:

myVariable="Теперь это текстовая переменная"

При задании переменной использование зарезервированного слова var не обязательно, но желательно, т.к. помогает при использовании отладчика фирмы Microsoft и делает текст программы более структурированным. На деле вместо переменной в текущем объекте window создается новое поле с таким именем. В функциях при задании локальных переменных использование var обязательно (иначе будет создана глобальная переменная).

При наличии численных и строковых значений в одном выражении идет приведение к строковому типу. Значением переменной

a=7+"раз отмерь,"+1+"раз присвой"

будет строка "7 раз отмерь, 1 раз присвой".

Стоит отметить, что существуют также функции parseFloat и parseInt, которые осуществляют преобразование из строкового значения в численное.

Идентификатором переменной может быть последовательность символов из набора букв от "A" до "Z", от "a" до "z", цифр от "0" до "9", а также символ подчеркивания "_". При этом первым символом имени не может быть цифра, а заглавные и строчные буквы отличаются (т. е. имена MyVariable и myvariable относятся к разным переменным).

Кроме глобальных переменных в функции или другом блоке кода можно определить локальные, для них областью видимости будет только функция (без кода), в которой они определены:

var myLocalVariable=0.

2. SCRIPT-вставки в HTML-документе

Для встраивания программы на JavaScript в HTML — файл используются теги <script> и </script>. При этом результат работы можно увидеть сразу и при необходимости внести изменения.

```
<html>

<head>

<script language="JavaScript">

document.write("Hello,world!<p>")

</script>

</head>

<body>

It was dynamically created text.

</body>

</html>
```

Будет сформирован текст:

Hello, world!

It was dynamically crested text.

Заметим, что внутри тегов <script> и </script> может содержаться любое число конструкций языка JavaScript.

Некоторые старые браузеры не поддерживают язык JavaScript и поэтому, чтобы скрыть от них вставки JavaScript, в программу добавляют следующий комментарий:

<!-- скрываемый текст -->

Пример:

```
<html>

<head>

<script language="JavaScript">

<!-- hidden text

document.write("Hello from JavaScript")

//end of hidden text -->

</script>

</head>

<body>

The end

</body>

</html>
```

Вставки могут быть и внутри <body>.

1. Операторы, выражения, функции

1. *Операторы: арифметических действий, присваивания, инкрементные, декрементные. Условные выражения*

Операторы арифметических действий в JavaScript те же, что и в C и Java:

Сложение "+", вычитание "-", умножение "*", деление "/", остаток от целочисленного деления "%".

Эти операторы могут встречаться в любом численном выражении. (Внимание! Они также могут встречаться в строковых выражениях в случаях, когда используется автоматическое приведения чисел в строки).

Операторы присваивания в JavaScript те же, что и в C и Java:

"=", "+=", "-=", "*=", "/=", "%="

x=y, как и в большинстве других языков, значит присвоить переменной "x" значение переменной "y".

Следующие операторы имеют синтаксис, сходный с синтаксисом соответствующих операторов языка C:

y+=x эквивалентно y=y+x

y-=x эквивалентно y=y-x

y*=x эквивалентно y=y*x

y/=x эквивалентно y=y/x

y%=x эквивалентно y=y%x – остаток от деления нацело y на x.

Условное выражение имеет вид

(условие)?значение1:значение2

Если значение условия true, значением условного выражения будет значение1, иначе — значение2. Условное выражение можно применять везде, где можно применять обычные выражения.

Пример:

a=(b<1)?0:(x-1)+c

Инкрементные и декрементные операторы также имеют синтаксис, заимствованный из языка C: "x++", "++x", "x--", "--x".

Выражения:

y=x++ эквивалентно двум присваиваниям: y=x; y=y+1,

y=++x эквивалентно двум присваиваниям: x=x+1; y=x,

y=x-- эквивалентно двум присваиваниям: y=x; x=x-1,

y=--x эквивалентно двум присваиваниям: x=x-1; y=x.

2. *Строковые операции*

Существуют ряд операторов работы со строками:

"+" - сложение (конкатенация) строк s1+s2 дает строку, состоящую из последовательности символов строки s1, за которыми следуют символы строки s2.

eval(s) - встроенная функция JavaScript. Она выполняет код, заданный аргументом— строкой s, который может содержать один или более операторов JavaScript (через точки с запятой). Данную функцию можно использовать не только для выполнения оператора, но и для вычисления выражения. Она возвращает значение последнего вычисленного выражения в заданном коде. Функция eval(s) обеспечивает возможность вычислять значения, введенные пользователем в пункты ввода, а также динамически модифицировать выполняемый код в JavaScript-программе. Более общая, чем функции parseInt и parseFloat.

parseFloat(s) – встроенная функция JavaScript. Находит содержащееся в строке s вещественное число (типа Float) от начала строки до первого символа, не входящего в число. Если число не найдено, возвращает значение NaN (“Not a Number”)

parseInt(s) – то же для целых чисел (Integer). При этом автоматически находится основание.

parseInt(s,n) – то же для целых чисел по основанию n (от 2 до 36). При n=0 – то же, что parseInt(s). При этом автоматически находится основание

3. *Побитовые операции присваивания*

Существуют ряд операторов побитового присваивания:

x<<=n эквивалентно x=(x<<n) — сдвиг на n битов влево двоичного представления целого

числа x;

x>>=n эквивалентно x=(x>>n)— сдвиг на n битов вправо двоичного представления целого

числа x с сохранением знакового бита (отрицательные

числа в дополнительном коде имеют первым битом

единицу. После сдвига на место первого бита записывается

1, и число остается отрицательным);

x>>>=n эквивалентно x=x>>>n — сдвиг на n битов вправо двоичного представления

целого числа x с ведущим нулем 0 (После сдвига на место

первого бита записывается 0, и число становится

положительным);.

Всё выражение (у нас — "x") в левой части преобразуется в 32-битное целое число, после чего производится необходимый сдвиг, а затем получившиеся число приводится к типу выражения – результата (у нас это опять"x"), и производится присваивание.

Примеры:

1) 9<<2 дает 36, т.к. сдвиг 1001 (число 9 в двоичном представлении) на 2 бита влево

дает 100100, т.е. 36. Недостающие биты заполняются 0.

9>>2 дает 2, т.к. сдвиг 1001 (число 9 в двоичном представлении) на 2 бита вправо

дает 10, т.е. 2. Недостающие биты заполняются 0.

"&" — побитовая AND — "И";

"|" — побитовое OR — "ИЛИ";

"^" — побитовое XOR — "ИСКЛЮЧАЮЩЕЕ ИЛИ".

Все операции совершаются с двоичным представлением чисел, однако, результат возвращается в обычном десятичном представлении.

Примеры:

15&9 возвращает 9, т.к. (1111) AND (1001) равно 1001;

15|9 возвращает 15, т.к. (1111) OR (1001) равно 1111;

15^9 возвращает 6, т.к. (1111) XOR (1001) равно 0110.

2.3 Логические выражения.

"&&" — логическое AND — "И";

"||" — логическое OR — "ИЛИ";

"!" — логическое NOT — "НЕ".

Пример:

(a>b)&&(b<=10)||(a>10)

В JavaScript всегда применяется так называемая сокращенная проверка логических выражений: в операнде B1&&B2 при B1=false оценки B2 не производится и возвращается значение false. Аналогично B1||B2 при B1=true оценивается как true. При этом анализ логического выражения идет слева направо, и как только всё выражение может быть оценено, возвращается результат. Поэтому, если в качестве B2 выступает функция, она не будет вызываться, и если она должна давать побочные эффекты, то это может привести к ошибке.

4. *Операторы сравнения*

"= =" -"равно";

">" -"больше";

"<" -"меньше";

">=" -"больше или равно";

"<=" -"меньше или равно";

"!=" -"не равно".

Операторы сравнения применимы не только к численным, но и к строковым выражениям. При этом строки считаются равными, если все их символы совпадают и идут в одном и том же порядке (пробел учитывается как символ). Если строки разной длины, то большей будет строка имеющая большую длину. Если их длины совпадают, больше считается та, у которой при просмотре слева направо встретится символ с большим номером

(a < b < c < .... < z < A < ... < Z).

Строки можно складывать, если S1="это ", S2="моя строка", то S1+S2 даст "это моя строка".

Приоритет операторов (начиная с младших; в одной строке старшинство одинаково):

"+=", "-=", "*=", "/=", "%=", "<<=", ">>=", ">>>=", "&=", "^=", "|=".

5. *Старшинство операций*

условное выражение: "?:";

логическое "ИЛИ": "||";

логическое "И": "&&";

побитовое "ИЛИ": "|";

побитовое "XOR": "^";

побитовое "И": "&";

сравнение на равенство "= =", "!=";

сравнение: "<", "<=", ">", ">=";

побитовый сдвиг: "<<", ">>", ">>>";

сложение, вычитание: "+", "-";

умножение, деление: "*", "/";

отрицание, инкремент, декремент: "!", "~", "++", "--";

скобки: "( )", "[ ]".

6. *Функции*

В JavaScript, как и в C или Java, процедуры и процедуры — функции называются функциями. Декларация функции состоит из:

зарезервированного слова function;

имени функции;

списка аргументов, разделенных запятыми, в круглых скобках;

тела функции в фигурных скобках.

function myFunction(arg1, arg2, ...)

{

...

последовательность операторов

...

}

где myFunction — имя функции, arg1, arg2 — список формальных параметров

Пример:

```
function Factorial(n)

{

if((n<0)||(round(n)!=n))

{

alert("функция Factorial не определена при аргументе "+n);

return NaN;

}

else

{

result=(n*Factorial(n-1));

return result;

}

}
```

Функция может не возвращать значения с помощью зарезервированного слова return.

Пример:

```
function Greeting(s)

{document.write("Hello,"+s+"!");

}
```

Вызов функции производится с фактическими параметрами.

Пример:

```
Factorial(2+1);
```

— внутри некого выражения возвратит 6, а

```
Greeting("world");
```

—приведет к выводу на экран строки "Hello, world!".

Каждая функция, например myFunction, является объектом с именем myFunction, аргументы которого хранятся как массив, имеющий имя arguments, при этом доступ к аргументам может осуществляться так:

myFunction.arguments[i], где i — номер аргумента (нумерация начинается с 0).

Число фактических параметров должно быть равно либо больше числа формальных параметров в описании функции. При этом, при вызове функции действительное число переданных аргументов хранится в поле myFunction.arguments.length и может динамически изменяться переприсваиванием значения этого поля.

Пример:

Вывод в документе списка в формате HTML.

Первый аргумент здесь (ListType) может иметь значение "o" или "O" для упорядоченного списка и "u" или "U" для неупорядоченного. Далее идут элементы списка.

```
function myList(ListType)

{

document.write("<"+ListType+"L");

for(var i=1; i < myList.arguments.length; i=i+1)

{document.write("<LI>"+myList.arguments[i]);

}

document.write("</"+ListType+"L>");

}
```

Вызов в тексте HTML документа этой функции:

```
<script>

myList("0", "один", 2, "3")

</script>
```

приведет к выводу текста:

один

2

3

7. ***Условный оператор if***

первый вариант синтаксиса оператора if:

```
if(условие)
```

{

утверждение

}

(Утверждением называется оператор или последовательность операторов.)

b) второй вариант синтаксиса оператора if:

if(условие)

{

утверждение1

}

else

{

утверждение2

}

Пример использования условного оператора:

```
function checkData()
{
if (document.form1.threeChar.value.length==3)
{return true;
}
else
{alert('Введите ровно 3 символа');
return false;
}
}
```

8. ***Цикл for***

for(секция инициализации; секция условия; секция изменения счетчиков)

{

утверждение

}

Каждая из секций может быть пустой. В секциях инициализации и изменения счетчиков можно писать последовательности выражений, разделяя их запятыми. Выполнение цикла происходит следующим образом. Первой выполняется секция инициализации. Затем проверяется условие. Если условие равно true, то выполняется тело цикла, а затем секция изменения счетчиков. Если условие равно false, то выполнение цикла прекращается.

Пример :

```
function HowMany(SelectObject)

{

var numberSelected=0

for (i=0; i< SelectObject.options.leght; i++)

{

if (SelectObject.options[i].selected==true) numberSelected++;

}

return numberSelected;

}
```

Оператор for может использоваться для перебора всех полей объекта (см. далее раздел про объектную модель)

Синтаксис:

```
for(переменная in объект)

{

выражение

}
```

При этом производится перебор всех возможных значений указанной переменной в объекте, и для каждого из них выполняется утверждение.

Пример:

```
function student(name, age, group)

{

this.name=name;

this.age=age;

this.group=group;

}
```

```
function for_test(myObject)
{
for(var i in myObject)
{
document.write("i="+i+" => "+myObject[i]+"/n");
}
};
Helen=new student("Helen K.", 21, 409);
for_test(Helen);
```

Вывод на экран:

i=0 => Helen K.

i=1 => 21

i=2 => 409

9. Цикл while

```
while(условие)
{
выражение
}
```

Выполнение цикла while начинается с проверки условия. Если оно равно true, то выполняется тело цикла, иначе управление передается оператору, следующему за циклом.

Пример использования оператора while:

```
n1=10
n=0
x=0
while(n<n1)
{
n=n+1;
x=x+n;
}
```

2.8 Операторы прерывания выполнения циклов.

Для прекращения выполнения текущего цикла операторов for или while служит оператор break.

Пример использования оператора break:

```
function test(x)

{

var j=0;

var sum=0;

while(n<n1)

{

if(n==x)

{ sum=x;

break;

}

sum=sum=n;

n=n+1;

}

return sum;

}
```

Оператор continue прерывает выполнение текущей итерации внутри for или while и вызывает переход к началу следующей итерации.

Пример использования оператора continue:

```
function test1(x)

{

var j=0;

while(n<n1)

{ if(n==x)

{ sum=x;

continue;

}
```

```
sum=sum=n;

n=n+1;

}

return sum;

}
```

2. Объектная модель

1. *Классы, объекты, поля данных, методы*

Подробная теоретическая информация по этому разделу содержится в курсе “Объектно - Ориентированное Программирование”. Отметим только, что класс – это тип данных, в котором описывается, как должны быть устроены переменные данного типа – объекты.

JavaScript существует большое число предопределенных классов. Также можно определять собственные классы. Однако возможность создания в JavaScript собственных классов носит итсключительно рекламно-декоративный характер: трудно представить ситуацию, когда она реально может понадобиться. До сих пор ни на одной WWW-странице нам такая ситуация не встретилась.

2. *Работа с полями данных и методами уже существующих объектов*

В JavaScript доступ к полям данных и методам может быть осуществлен обычным путем через квалификацию имени: если Data1 — поле данных, а method1 — метод объекта obj1, то obj1.Data1 означает доступ к полю Data1 этого объекта, а obj1. method1 — вызов его метода method1 (напомним, что Data1 и data1 — разные идентификаторы, как и method1 и Method1, и т.п.). Для определенных в программе объектов доступ может

быть как по чтению, так и по записи. Ряд полей предопределенных объектов доступны только по чтению (например, Math.PI- значение числа "пи"). Если до присваивания поля данных

с используемым именем его в объекте не было, оно динамически создается в момент присваивания.

Пример:

```
student.name="Алексей Иванов";

student.age=26;

student.group=409;

a=student.name
```

Возможен второй путь доступа к полям данных — ассоциативный, через массив индексов:

```
student["name"]="Алексей Иванов";

student["age"]=26;

student["group"]=409;
```

b=student["group"]

Возможно определить поле данных объекта только через индекс, без имени. В этом случае доступ к полю будет возможен только по номеру:

A[0]=1; A[1]=2; A[2]=4;

B[0]="a"; B[1]="b" ;

и т.д. Подобного рода поля — это массив, но в отличие от массивов в других языках программирования его элементы могут иметь произвольный тип.

Задание метода myMethod объекта myObject может быть произведено либо

"изнутри"(в определенный класс- смотри далее), либо "снаружи"-простым присваиванием:

myObject.myMethod=myFunction;

где myFunction — имя уже определенной функции. Допустим, это myGreeting. Тогда myObject.myMethod=myGreeting;

Вызов методов осуществляется аналогично вызову обычной функции JS. Метод может использоваться как процедура:

myObject.myMethod("method");

даст "Hello, method!"

Либо же можно использовать метод как собственно функцию, возвращающую значение.

Для создания нового класса объектов (например, student) надо определить функцию с таким именем. Вместо имени объекта для доступа к полям внутри этой функции надо использовать зарезервированное слово "this".

3. *Задание нового класса объектов. Квалификатор this*

Процедура определения класса student:

function student(name, age, group)

{

this.name=name;

this.age=age;

this.group=group;

}

В соответствии с правилами JavaScript мы фактически ввели глобальные переменные

student.name, student.age, student.group.

Теперь можно определить объекты типа student с помощью предопределенного слова new:

Paul=new student("Павел", 23, 509);

Alex=new student("Алексей", 26, 609);

Объекты- это экземпляры класса. Может быть сколько угодно экземпляров данного класса и

При необходимости можно менять параметры существующих экземпляров и создавать новые.

Следует отметить, что в зависимости от фактического списка параметров поля объектов "student" могут иметь разные типы. Более того, даже число полей, как мы знаем, может быть разным при соответствующем определении функции.

4. *Операторы for и with для работы с объектами*

вставить for..in…

Для получения доступа к полям объекта очень удобен оператор with. Его синтаксис таков:

with(объект)

{

утверждение

}

Внутри тела оператора with (т.е. в утверждении) методы и переменные, по умолчанию квалифицируются именем объекта (без явного указания в качестве префикса имени объекта

С последующуй точкой), если нет локальных методов или переменных с использованными именами. Если имеются локальные методы или переменные объекта с тем же именем, доступ к полям, как мы уже знаем, должен осуществляться "извне" через полное имя (с явной квалификацией) , а "изнутри" объекта - с помощью квалификатора this.

Пример:

var x,y;

with(Math)

{

y=PI/2;

x=sin(y/7)+pow(y,2);

}

Здесь вместо PI, sin и pow используются поле и методы объекта Math:

Math.PI

Math.sin

Math.pow

5. *Правила работы с объектами*

Объекты можно передавать в функции в качестве параметров. Например, если мы имеем следующее определение функции IndividualPlan:

```
function IndividualPlan(student, yearCount)

{

this.person=student;

this.year=yearCount;

}
```

То

```
PaulPlan=new IndividualPlan(Paul,5);
```

создаст объект с именем PaulPlan, поле PaulPlan.person которого является объектом типа student. При этом доступ к имени студента будет производиться следующим образом:

```
PaulPlan.person.name.
```

Отдельному объекту можно динамически добавлять поля:

```
Paul.award="Грант Шведского института"
```

При этом у остальных объектов типа student этого поля не будет. Аналогично добавляются методы, так так можно динамически переприсваивать методы объекта. Например, если функцию IndividualPlan определить сначала как

```
function IndividualPlan(student, yearCount)

{

this.person=student;

this.year=yearCount;

this.hello=myGreeting;

}
```

то вызов

```
PaulPlan.hello("University");
```

приведет к выводу на экран следующей строки:

```
"Hello, University!"
```

В любой момент в программе можно переопределить функцию IndividualPlan.hello.

Например, введем функцию ShowPerson, не имеющую параметров:

function showPerson()

{

document.write("Владелец индивидуального плана "+this.person.name);

}

после чего проведем присваивание

IndividualPlan.hello=showPerson;

Оно изменит не только тело функции, но и число ее аргументов. Теперь вызов

PaulPlan.hello();

приведет к выводу на экран

Владелец индивидуального плана: Павел.

В JS все элементы кроме операндов и операторов являются объектами. Поэтому для них можно вызывать соответствующие методы. Имена предопределенных объектов начинаются с большой буквы. Так при вызове методов для строковой переменной или строкового выражения вызываются методы встроенного объекта Stung.

6. Динамическое формирование документа

1)Пример программы:

```
<html>

<body>

<script language = "JavaScript">

document.write("<h2>Table of Factorials</h2>")

for (i=1,factorial=1; i<10; i++, factorial*=i)

{document.write(i+"!="+factorial)

document.write("<br>")

}

</Script>

</body>

</html>
```

3. Классы и объекты языка JavaScript

1. ***Класс Global (задан неявно)***

Класс Global является частью спецификации ECMAScript, и его задача – объединение в один объект ряда глобальных методов и свойств. При обращении к методам сам объект не указывается, так как он не имеет конструктора. К свойствам и методам данного объекта относятся:

Свойство Описание

Nan Содержит значение NaN (Not A Number).

Infinity Содержит значение Number.POSITIVE_INFINITY.

Метод Описание

escape Преобразует строку таким образом, что она может читаться на всех платформах.

eval Выполняет переданную ей строку так, как если бы это было обычное выражение JavaScript.

isFinite Позволяет определить, является ли аргумент конечным числом.

isNan Позволяет определить, является ли аргумент числом или нет.

parseFloat Преобразует строку в число с плавающей точкой.

parseInt Преобразует строку в целое число.

unescape Преобразует результат функции escape.

Функции taint() и untaint() – при включенном защитном искажении данных возвращает искаженную копию значения объекта или ссылки либо, наоборот, неискаженную.

Функции escape и unescape – возвращают копии строки s в закодированном и , соответственно, раскодированном виде.

Кодировка – замена всех пробелов, знаков препинания и так далее в форму %xx, где xx – две 16-ричные цифры, соответствующие коду символа в ISO-8859-1(Latin-1)

Функция eval(s) – интерпретирует строку s как последовательность операторов JavaScript. (См. Также объект Object)

Функция getClass(Jobj) – возвращает объекты JavaClass для аргумента типа JavaObject

Пример: var myJavaRClass=new java.awtRectangle()

Var myJavaRClass=getClass(myJavaRect)

Не путать с Java-методом getClass():

Var myJavaRCObject=myJavaRect.getClass()

Это - Java-представление класса java.awt.Rectangle

Функция isNaN(x) – проверка, является ли x “Not a Number”, то есть не числом.

Функции parseFloat(s) – находит s число типа Float(от начала строки до первого символа,

Не входящего в число). Если число не найдено, возвращает

Значение NaN (“Not a Number”).

parseInt(s) – то же для Integer

parseInt(s,n) – то же для целых чисел по основанию n (от 2 до 36). При n=0 – то же, что parseInt(s). при этом автоматически находится основание

Функция eval(s)

Функция eval(s) — встроенная функция JavaScript. Она выполняет код, заданный аргументом— строкой s, который может содержать один или более операторов JavaScript (через точки с запятой). Данную функцию можно использовать не только для выполнения оператора, но и для вычисления выражения. Она возвращает значение последнего вычисленного выражения в заданном коде. Функция eval(s) обеспечивает возможность динамически модифицировать выполняемый код в JavaScript-программе.

Функция isNaN(x)

Эта функция проверяет, является ли аргумент x "не-числом". Определяет, не является ли он зарезервированным значением NaN, которое представляет недопустимое число (например, результат деления нуля на нуль). Эта функция обязательна, потому что в JavaScript задать значение NaN как литерал невозможно. Она используется для проверки результатов выполнения parseFloat(s) и parseInt(s) (являются ли они допустимыми числами) и для проверки на предмет наличия арифметических ошибок, например, деления на нуль.

Функция **parseFloat**(s)

Выполняет синтаксический анализ и возвращает первое число, обнаруженное в строке *s* (т.е. преобразует строку в число). Анализ прекращается и значение возвращается, когда parseFloat(s) встречает в s символ, не являющийся допустимым элементом числа (т.е. знаком, цифрой, десятичной запятой, показателем степени и т.д.). Если с не начинается с числа, которое может распознать parseFloat(s), эта функция возвращает NaN.

parseFloat(*s*) (*s* — строка, подлежащая синтаксическому анализу и

преобразованию в число)

Функция **parseInt**(s)

parseInt(s)

parseInt(*s*, основание)

(*s* — строка, подлежащая синтаксическому анализу основание — целое основание подлежащего анализу числа).

Данная функция преобразует строку в целое число. Синтаксический анализ прекращается и возвращается значение, когда parseInt(s) встречает в строковом выражении *с* символ, не являющийся допустимой цифрой для указанного основания системы счисления. Аналогично parseFloat, функция parseInt возвращает значение NaN, когда s не начинается с числа, которое она может распознать.

Если основание задано как 10, то parseInt(s) анализирует строку на наличие десятичного числа. Значение 8 соответствует анализу на наличие восьмеричного числа (в этой системе используются

цифры от 0 до 7). Значение 16 соответствует анализу на наличие шестнадцатеричного числа (используются цифры от 0 до 9 и буквы от A до F). Основание может быть любым числом от 2 до 36. Если основание равно нулю или опущено, то

parseInt(s)пытается определить основание числа по содержимому самой строки. Если строка начинается с 0x, то функция анализирует остальную часть строки как шестнадцатеричное число. Если строка начинается с 0, то функция анализирует строку как восьмеричное число.

2. ***Класс Math***

Math — класс, содержащий константы (поля, доступные толькопо чтению) и методы . Они вызываются обычным для объектов образом:

Math.константа

Math.функция(i..)

Константы класса Math

E — число e (основание натуральных логарифмов)

LN10 — натуральный логарифм 10 (число ln10)

LN2 — натуральный логарифм 2 (число ln2)

LOG10E — логарифм e по основанию 10 (число log10e)

LOG2E — логарифм e по основанию 2 (число log2e)

PI — константа π (число "пи")

SQRT1_2 — число, обратное корню квадратному из 2 (число 1/√2)

SQRT2 — корень квадратный из 2 (√2)

Методы (функции) класса Math

abs(x)(x — любое число или выражение) — вычисляет абсолютную величину

acos(x) (x — число или выражение в интервале от -1.0 до 1.0 рад) — вычисляет аркко синус. Возвращаемое значение находится в интервале от 0 до π радиан

asin(x) (x — число или выражение в интервале от -1.0 до 1.0 рад) — вычисляет арксинус, возвращает значение, находящееся в интервале от -π/2 до π/2 радиан

atan(x) (x — число или выражение) — вычисляет арктангенс в радианах, возвращает значение, находящееся в интервале от -π/2 до π/2 радиан

atan2(x,y)(x,y — координаты точки в прямоугольной системе координат) —вычисляет угол точки (x,y) в полярных координатах. Возвращаемое значение лежит в интервале от

0 до 2π радиан

ceil(x) (x — любое число или числовое выражение) — округляет число в большую сторону до ближайшего целого (т.е. вычисляет наименьшее целое число ≥x); отрицательные числа округляются в большую сторону (в направлении к нулю)

cos(x) (x — угол в радианах) — вычисляет косинус; возвращаемое значение находится в интервале от -1.0 до 1.0 рад.

Exp(x) (x — число или числовое выражение) — вычисляет экспоненту e

Floor(x) (x —число или числовое выражение) — округляет число в меньшую сторону до ближайшего целого (т.е. вычисляет наибольшее целое число ≤x); например,floor(-1,1)равно (-2);floor(1,1)равно 1.

Log(x) (x — положительное число или выражение) — вычисляет натуральный логарифм

max (a,b) (a,b —числа или выражения)— возвращает большее из двух значений

min(a,b) (a,b —числа или выражения) — возвращает меньшее из двух значений

pow(x,y) — вычисляет x (возводит первый аргумент в степень)

random — вычисляет псевдослучайное число в интервале от 0 до 1

round — округляет до ближайшего целого (round(15.5) дает 16,

round(-15.5) дает -15)

Math.round(x) (x — любое число или выражение)

sin — вычисляет синус

Math.sin(x) (x — угол в радианах)

sqrt — вычисляет квадратный корень

Math.sqrt(x) (x — любое число или выражение, которое больше или

равно 0)

tan — вычисляется тангенс

Math.tan(x) (x — угол в радианах)

Класс Date

При отсутствии аргументов метод Date() создает объект Date, значением которого являются текущая дата и время. Аргументами метода Date() задается дата и (при необходимости) время для нового объекта. Объект Date встроен в JavaScritp и в HTML аналога не имеет. Большинство методов объекта Date вызываются через экземпляр этого объекта. Например:

d=new Date() //получить сегодняшнюю дату и время

system.write("Today is: "+d.toLocalString());

//и отобразить эту информацию

Приведенный выше синтаксис создания объектов Date предполагает, что значения даты и времени задаются по местному времени. Если программа должна работать независимо от часового пояса, в котором находится пользователь, необходимо указать все даты по Гринвичу (GMT) или универсальному скоординированному времени (UTC). Чаще всего объект Date используется для вычитания значения текущего времени в миллисекундном формате из другого значения времени с целью определения разницы.

Для того чтобы создать объект Date, можно воспользоваться одним из следующих пяти вариантов синтаксиса. В вариантах с третьего по пятый указанное время интерпретируется как местное (а не гринвичское).

new Date();

new Date(миллисекунды) (число миллисекунд между датой

и полуднем 1.01.1970 г.)

new Date(строка_даты) (строка_даты = имя_месяца дд,

гг [чч:мм[:сс]])

new Date(год, месяц, число) (год минус 1900;

месяц 0-11; число 1-31)

new Date(год, месяц, день , часы, минуты, секунды)

(по 24-часовой системе)

Методы класса Date.

getDate() (возвращает число объекта Date в диапазоне от 1 до 31)

getDay() (возвращает день недели объекта Date в диапазоне

от 0 [воскресенье] до 6 [суббота])

getHours() (возвращает значение поля часов объекта Date в диапазоне

от 0 [полночь] до 23)

getMinutes() (возвращает значение поля минут объекта Date в диапазоне от 0 до 59)

getMonth() (возвращает значение поля месяца объекта Date в диапазоне

от 0 [январь] до 11 [декабрь])

getSeconds() (возвращает значение поля секунд объекта Date в диапазоне от 0 до 59)

getTime() (возвращает внутреннее (в миллисекундах) представление объекта Date)

getYear() (возвращает значение поля года объекта Date. Возвращаемое значение

равно году минус 1900 (например, 1997-й выдается как 97)

parse() (выполняет синтаксический анализ строкового представления даты и

возвращает ее значение в миллисекундном формате)

setDate() (устанавливает значение поля числа объекта Date)

дата.setDate(число_месяца) //число_месяца равно 1-31

и т.д.

toLocalString() (преобразует Date в строку, используя местный часовой пояс)

UTC() (преобразует числовую спецификацию даты и времени в

миллисекундный формат)

3. *Класс Window*

Этот объект представляет окно или кадр Web-браузера. Так как код JavaScript интерпретируется в контексте объекта Window, в котором он выполняется, этот объект должен содержать ссылки на все прочие необходимые объекты JavaScript (т.е. он является корнем иерархии объектов JavaScript). Многие свойства объекта Window являются ссылками на другие важные объекты. В большинстве своем эти свойства относятся к объекту, соответствующему данному окну. Например, свойство location относится к объекту Location данного окна. Другие свойства Window (например, navigator — ссылка на объект Netscape, который относится к этому и всем остальным окнам) относятся к глобальным объектам, а два-три — только к самому окну.

В пользовательском варианте JavaScript для обращения к объекту Window, никакой специальный синтаксис не нужен, и к свойствам этого объекта можно обращаться как к переменным (т.е. можно указать не *окно*.document, а просто document). Аналогичным образом методы объекта Window, который соответствует текущему окну, можно использовать как функции (например, alert() вместо *окно*.alert()).

self текущее окно

window текущее окно

Обратиться к кадру в окне можно следующим образом:

frame[*i*] (или self.frame[*i*])

*окно.*frame[*i*]

Для обращения к родительскому окну данного кадра используются следующие варианты:

parent (или self.parent, window.parent)

окно.parent (родительское окно данного кадра)

Если же мы хотим обратиться к окну браузера верхнего уровня из любого содержащегося в нем кадра, то используется:

top (или self.top, window.top)

Новые окна верхнего уровня создаются методом Window.open(). Когда пользователь вызывает этот метод, необходимо сохранить результат вызова open()в переменной и использовать эту переменную для обращения к новому окну.

Открытие окна и динамическое формирование текста.

Метод open отображает существующее окно или открывает новое.

окно.open([URL, [*имя*, [*характеристики*]]])

Если аргументом *имя* задается имя существующего окна, то возвращается ссылка на это окно. В окне, на которое указывает возвращенная ссылка, отображается документ, полученный по указанному URL, но характеристики игнорируются. Если URL — пустая строка, то открывается пустое окно.

Если *имя* не относится к существующему окну, то этот аргумент задает имя нового окна. Используя *имя* в качестве значения атрибута target тега <a> или <form>, можно вывести документы в это окно.

Характеристики — это список характеристик (через запятые), которые должны отображаться в окне. Если этот аргумент пуст или не задан, то в окне будут отображаться все характеристики. Эта строка не должна содержать пробелов.

myWindow=window.open("myFile.html")

Перечень характеристик:

open("URL","windowName","toolbar=no,scrollbars=yes,resizable=yes")

Логические параметры могут быть просто упомянуты, что эквивалентно установке yes

или 1, либо установлены в no:

toolbar=yes или no (либо1, либо0) - панель инструментов браузера, клавиши: "Back",

"Forward" и т.д.

location — поле ввода URL;

directories — линейка клавиш "What's New", "What's Cool" и т.д.

status — строка состояния снизу

menubar — линейка меню сверху

scrollbars — полоса прокрутки документа

resizable — может ли меняться размер окна

Другие параметры:

width= ширина окна в пикселях

height= высота окна в пикселях

Пример:

```
function windowOpener() {

MyWindow=window.open("","MyWindow","menubar=yes,location,

status");

MyWindow.document.bgColor="aqua";

MyWindow.document.write("<head><title>MyWindow</title></head>");
```

MyWidow.document.write(" <big>Hello,world!</big>

");

}

Закрытие окна.

window.close() или self.close() — закрытие текущего окна. Если это фрейм, то закрывается все родительское окно. Можно закрывать окно и по имени:

MyWindow.close()

4. *Коллекция фреймов (window.frames)*

Хотя на объект Frame иногда ссылаются, такого объекта, строго говоря нет. Все кадры в окне являются экземплярами объекта Window, содержат те же свойства и поддерживают те же методы и обработчики событий, что и объект Window. Однако, на практике между объектами Window, которые представляют кадры в окне верхнего уровня, и объектами Window, которые представляют кадры в окне браузера, есть несколько различий:

Если для кадра установлено defaultStatus, то указанное сообщение о состоянии появляется только тогда, когда курсор мыши находится в этом кадре.

Свойства top и parent окна браузера верхнего уровня относятся к самому окну

верхнего уровня. Эти свойства по-настоящему полезны только для кадров.

Метод close() для объектов Window, которые являются кадрами, ценности не

представляют.

окно.frames[*i*]

окно.frames.length

frames[*i*]

frames.length

Пример:

<frameset

rows="*высота1,высота2,...*"

cols="*ширина1,ширина2,...*"

>

<frame src="*адрес_файла_для_фрейма*" name="*имя_фрейма*">

<frame src="*адрес_файла_для_фрейма2*" name="*имя_фрейма2*">

.....

</frameset>

src — это строка с возможностью чтения и записи, в которой задается URL изображения, подлежащего выводу на экран.

Размер фреймов по умолчанию в пикселях. Если в конце стоит % — то в % от общей высоты или ширины главного окна.

URL фрейма не может содержать метки (#).

Документ, загружаемый в окно фрейма, сам может содержать фреймы.

5. *Класс Document (window.document)*

Экземпляр класса Document в поле document объекта window.

Вызов: имя окна.document либо просто document (для текущего окна)

Поля: alinkColor - см. соответствующий параметр HTML

vlinkColor - то же;

anchors[i] - массив объектов класса Anchor, по одному на контейнер <a…>…

anchors.length - длина этого массива (RO)

applets[i] - массив апплетов (объектов Java) (по одному на контейнер <applet>…<1>)

applets.length - длина этого массива (RO)

bgColor - строка с заданием цвета фона

fgColor - строка с заданием цвета текста документа

domain - строка с именем домена, откуда поступил документ

embeds[i] - массив Plug-In приложений (соответствующих контейнерам)

<embed>…</embed>, по одному на контейнер)

plugins[i] - то же (синоним)

embeds.length - длина этого массива (RO)

plugins.length - то же

forms[i] - массив объектов Form(по одному на контейнер <form>…</form>)

referrer - строки с URL документа , откуда пользователь дошел до текущего документа (RO)

title - отражает содержимое контейнера <title>…</title> (RO)

URL - строка с URL документа (RO)

Методы:

document.clear() - очистка окна или фрейма с именем w

document.close() - выводит на экран остатки информации из буфера и закрывает выходной поток в документ.

document.open () - открывает входной поток в документ, чтобы операторами write(…) можно было дописывать на экран (в документ) текст.

document.write(значение1, значение2,…) - дописывание значений в документ

document.writeln(…) - аналогично write(…), но со вставлением символа перехода на новую строку. Значимо только для форматирования плоского текста . Например, обработчиков

событий в теле <body…

onLoad="…"

onUnload="…"

>

Текущий экземпляр класса Document называется document. Это просто тот HTML-документ, который выведен на экран в данный момент. Объект document хранится в поле document объекта window. При обращении к объекту document текущего окна (т.е. окна, в котором выполняется программа JavaScript) ссылку *окно* можно опускать и использовать просто document.

Как уже было упомянуто ранее, с помощью объекта document можно создать документ

"с нуля".

Надо учитывать, что при наличии фреймов, window относится к текущему окну внутри фрейма. Вызов фреймов может идти через имена фреймов

События и обработчики событий:

onBlur — потеря фокуса ввода с клавиатуры

onClick — щелчок по компоненте или гиперссылке

onChange — изменение значения после ввода

onFocus — получение фокуса ввода с клавиатуры

onLoad — загрузка документа

onMouseOver — мышь над гиперсвязью или меткой

onMouseOut — мышь выходит за область гиперсвязи или метки

onSelect — выделение (целиком или частично) текста в поле ввода

onSubmit — ″получение″ формы

onUnload — пользователь выходит из документа

Пример:

s="значение URL"

.....

<a href="..."

onClick="this.href=s"

onMouseOver="window.status=′Самый лучший сетевой адрес′; return true"

Самый лучший сетевой адрес>

Сравнение с HTML:

<body

background="Bgimg.gif"

bgcolor="aqua"

text="black"

link="blue"

alink="red"

vlink="green"

onLoad="обработчик события"

onUnLoad="обработчик события">

</body>

Параметр background — изображение, служащее фоном. Оно может быть по любому

сетевому адресу;

bgcolor — цвет фона окна;

text — цвет текста по умолчанию в данном документе;

link — цвет гиперссылок, по которым еще не было переходов;

alink — цвет активированной гиперссылки, по которой идет загрузка после ее

выбора;

vlink — цвет гиперссылок, по которым уже были переходы;

обработчик — имя функции с фактическими параметрами либо выражение

JavaScript;

onLoad — код JavaScript, выполняемый загрузке документа;

onUnload — код, выполняемый при попытке выхода из документа.

Пример:

< body

onLoad="document.location="http://phys.runnet.ru">

.....

</body>

Для всех атрибутов задающих цвет, значение может быть либо одним из стандартных имен цветов, распознаваемых языком JavaScript, например, red для красного цвета, green для зеленого и так далее, либо в формате RGB, состоящим из шести шестнадцатеричных цифр (RRGGBB). В этом случае красный цвет представляется как FF0000, зеленый — 00FF00 и так далее. Использование RGB — представления предпочтительнее в тех случаях, когда неизвестно название цвета, например blanchedalmond — это комбинация FFEBCD.

Цвета в формате RGB:

Черный 000000

Коричневый 808000

Светло-красный FF0000

Малиновый FF00FF

Светло-зеленый 00FF00

Индиго 800080

Светло-синий 0000FF

Ярко-голубой 00FFFF

Темно-красный 800000

Бирюзовый 008080

Темно-зеленый 008000

Белый FFFFFF

Темно-синий 000080

Серый 808080

Ярко-желтый FFFF00

6. *Класс Location (window.location)*

Это URL. Каждое из свойств объекта Location представляет собой строку с возможностью чтения и записи, которая содержит один или более фрагментов URL, описываемого данным объектом.

Общий формат:

Protocol://hostname:port/pathname?search#hash

Поля: см.Link

Методы: reload() – перезагрузка документа,если он был изменен

reload(B) – если булевская величина B имеет значение true, перезагрузка документа происходит даже если документ не был изменен

Позиция.replace(URL) – заменяет текущий документ новым (с URL newURL без формирования новой позиции в протоколе сеанса броузера.

Свойство location объекта window есть объект класса Location, который задает URL документа. Изменение свойств location заставляет браузер прочитать измененный URL. Для загрузки нового URL свойству location присваивается строковое значение либо устанавливается любое из свойств объекта Location. Широко используется свойство href. Если просто установить свойство hash объекта window. location, то браузер перепрыгнет на новый якорь.

Если пользователь установил свойство location или location.href для уже посещенного URL, то браузер либо загрузит этот URL из кэша, либо сверится с сервером на предмет того, изменялся ли этот документ, и в случае необходимости перезагрузит его.

Поля объекта Location относятся к различным частям URL, который имеет следующий формат:

protocol://hostname:port/pathname?search#hash

hash Хеш-часть URL (включая начальный знак #). Эта часть задает имя якоря в

одном HTML-файле.

host Комбинация имени хоста и порта в URL.

hostname Имя хоста в URL.

href Полный URL.

pathname Полное имя в URL.

port Порт в URL.

protocol Протокол в URL (включая конечное двоеточие).

search Элемент поиска или запроса в URL (включая начальный знак вопроса).

7. *Класс Link (document.link)*

подкласс объекта Location, представляет гиперссылку на экране или фрагмент чувствительной области изображения на стороне клиента в HTML-документе. Это подкласс объекта Location, но отличается тем, что не обеспечивает автоматическую загрузку нового URL (т.е. изменяет URL, на который указывает ссылка, но документ, соответствующий этому URL, не отображается до тех пор, пока его не выберет пользователь). В JavaScript гипертекстовая ссылка представляет собой объект Link, а именованный адресат ссылки — объект Anchor.

Вызов: document.links[i]

Поля: hash – хэш-часть URL,включая знак #

Host – комбинация имени хоста и номера порта в URL

Hostname – имя хоста в URL

Href – полный URL

Pathname – полный путь в URL

Part – порт в URL

Protocol – протокол в URL (включая конечный знак ”:”)

Search – элемент запроса в URL (включая начальный знак “?”)

Target – атрибут target для гиперссылки

Обработчики событий:

Onclick()

Onmouseout() – при уходе указателя “мыши” с гиперссылки

Onmauseover() – при его постановке на гииперссылку

Пример: <a onMouseOut=”…”>…

<map…>

<arca onMouseOver=”…”

onMouseOut=”…”

…….

>

</map>

Объект Link создается стандартными тегами <a> и с добавлением onClick,

onMouseOut и onMouseOver. Атрибут href обязателен для всех объектов Link.

<a href="*URL*"

name="*тег_якоря*" *создает объект Anchor;*

target="*имя_окна*" *окно, в котором будет отображаться*

обозначенный ссылкой документ;

onClick="*обработчик*" *вызывается при щелчке на ссылке;*

onMouseOver="*обработчик*" *вызывается, когда курсор мыши на*

ссылке;

onMouseOut="*обработчик*" *вызывается, когда курсор мыши вне*

ссылки

>

текст или изображение

А также объект класса Link можно создавать тегом <area>, относящимся к чувствительной к щелчкам мыши области изображения на стороне пользователя.

<map name="*имя_области*"

<area share="*форма_области*"

coords=coordinates

href="URL"

target="*имя_окна*" *окно, в котором будет отображаться*

обозначенный ссылкой документ;

onClick="*обработчик*"

onMouseOut="*обработчик*"

>

...

</map>

8. *Класс History*

Защищенный от записи массив строк (RO), задающих URL, которые до этого момента были посещены. Содержимое этого списка эквивалентно URL, перечисленным в меню Go браузера Netscape. С помощью объекта History пользователь может реализовать в окне собственные кнопки Forward и Back, а также другие органы управления навигацией.

Вызов : окно.history

Фрейм.history

History

Массив строк (RO) с URL посещенных мест.

Поля данных:

current - URL текущего документа (RO, P - доступно только при включенном защитном

Искажении данных)

length - количество URL в списке

next - URL следующего в списке документа (RO,P)

previons - то же для предыдущего

Методы:

back() - эквивалентно нажатию клавиши Back или команде history.go(-1)

forward() - то же для Forward, эквивалентно history.go(1)

go(n) - перейти на URL с относительной позицией n в списке

go(подстрока) - перейти на URL ближайшего в предыстории URL, содержащего эту

Подстроку

toString() - возвращает строку в HTML формате со списком (P)

9. *Класс MimeType*

представляет тип данных, поддерживаемых броузером и его Plug-In

приложениями.

Вызов: navigator.mimeTypes[i]

navigator.mimeTypes[“имя”]

Число объектов mimeType в массиве: navigator.mimeTypes.length

Поля: description – (RO) краткое описание типа

EnabledPlugin – ссылка на объект Plugin, поддерживающий данный тип, либо

Null - если нет такого объекта.

Name – (RO) имя типа mime (например, “text/html” или “text/plain”)

Suffixes – RO список расширений(например, “html,htm”).

Пример: var show_movie=(navigator.mimeTypes[“video/mpeg”]!=null); - проверка на

Поддержку mPEG-файлов

10. Класс Navigator

– существует в одном экземпляре.

Вызов: окно.navigator

Поля данных:

AppCodeName – RO строка с ходовым названием броузера

AppName – RO имя броузера

AppVersion – RO строка с версией броузера

mimeTypes[i] – массив объектов MimeType с типами, распознаваемыми и поддерживаемыми броузером;

mimeTypes.length – длина массива

plugins[i] – то же, что mimeTypes[i], но только для типовPlug-In приложений

plugins.length – длина массива

userAgent – RO строка для заголовка агента HTTP-запроса.

Методы:

JavaEnabled() – true, если в броузере включена поддержка Java

TaintEnabled() – true, если включено защитное искажение данных

4. Экранные формы

1. *Класс Form (document.forms[i])*

Отражает контейнер <form>…</form> в документе.

Каждая форма - элемент document.forms[i] со своим i.

Вызов: F.action - строка с URL, по которому должна быть предана форма с именем F по команде Submit

F.elements[] - массив элементов формы

F.elements.length - длина массива

F.elements[i].type - тип элемента i

Класс	HTML	type
Button	<input type=button>	"button"
Checkbox	<input type= checkbox>	"checkbox "
FileUpload	<input type=file>	"file"
Hidden	<input type=hidden>	"hidden"
Password	<input type=password>	"password"
Radio	<input type=radio>	"radio"

```
Reset <input type=reset> "reset"

Select <select> "select-one"

<option…>

Select <select multiple> "select multiple”

<option…>

…

<option…>

</select>

Submit <input type=submit> "submit"

Text <input type= text> "text"

Textarea <textarea> "textarea"

……..

</textarea>
```

F.encoding - отражает атрибут enctype тега <form>

F.method - отражает атрибут method, имеет значения "get" или "post"

F.target - отражает атрибут target

Методы:

F.reset() - установка элементов в состояние default("по умолчанию")

F.submit() - передача формы на сервер

Обработчики событий:

В теге

<form …

onReset="…"

onSubmit="…"

>

В программе: F.onreset=”код JavaScript”;

F.onsubmit=”…”;

Объект document.forms[i] — это HTML-форма с номером *i* в документе (нумерация начинается с нуля), которая представляется элементом коллекции document.forms[i]. Именованные формы также представляются свойством имя формы соответствующего документа, где имя формы — это имя, заданное атрибутом name тега <form>.

```
document.имя_формы

document.forms[номер_формы]

document.forms.length
```

Информация об элементах формы (кнопках, полях ввода и т.д.) помещается в массив form.elements[].К именованным элементам, как и к именованным формам, можно также непосредственно обращаться по имени: имя элемента используется как имя свойства объекта form.

Объект form создается стандартным HTML-тегом <form>. В JavaScript к этому тегу добавляется необязательный атрибут обработчика событий onSubmit. Форма содержит все элементы ввода, созданные с помощью тега <input>, заключенного между тегами <form> и </form>.

```
<form

name="formName" имя формы на JavaScript;

target="windowName" имя окна для ответов;

action="serverURL" URL, по которому передается форма;

method=get|post метод передачи формы;

enctype="encodingType" тип кодирования данных формы

onSubmit="обработчик" обработчик, вызываемый при передаче

формы;

>

здесь располагается текст формы и теги <input>

</form>
```

1.2. Класс Image

Отражает тег <img>. Замечание: объекты imagene содержатся в коллекции form.elements, так как могут “жить” вне тега <FORM>, как объекты link или anchor.

Вызов:

document.images[i]

document.имя_изображения

Конструктор:

I = new Image()

I = new Image (ширина, высота)

Свойства:

border, height, hspace, name, vspace, width - отражают соответствующие атрибуты (RO)

comlete - булевская величина; при загрузке равна false, иначе – true

src – отражает атрибуты src; доступнен по чтению и записи

Обработчики событий:

В теге img

```
<img src="…"

onAbort="обработчик"

onError="обработчик"

onLoad="обработчик"

>
```

```
В теле программы:

Задание:

I.onabort="…"

I.onerror="…"

I.onload="…"

Вызовы:

I.onabort()

I.onerror()

I.onload()
```

2. ***Классы Button, Checkbox, Hidden, Password, Radio, Reset, Submit, Text, Textarea***

Отражают соответствующие элементы экранных форм в HTML. Также имеется класс FileUpload, соответствующий элементу file.

Свойства: соответствующие атрибутам тегов. Большая часть из них доступна только по чтению. Например:

name, type, defaultValue, defaultChecked

и т.д..

Доступное по чтению и записи:

value.

Кроме того:

form

доступно только по чтению, является ссылкой на форму объекта класса Form, которой принадлежат соответствующие элементы.

Обработчики событий: onclick(), onblur(), onfocus(), onchange() и т.д.

Методы (кроме Hidden):

… .blur() - потерять фокус

… .focus() - приобрести фокус.

Конструкторы в документе: соответствующие теги

1.4. Класс Button.

Соответствует кнопкам в формах HTML-документов. Классы Submit и Reset — это наследники класса Button, служащие для передачи формы и сброса введенных в нее данных ,соответственно.

Вызов:

```
форма.имя_кнопки

форма.elements[i]

форма.elements["имя_кнопки"]
```

Конструктор:

В документе: тегом <input> с добавлением атрибута type=”Button”

В программе: B=new Button();

Обработчики: onClick

```
<form>

...

<input

type="button" задает, что это кнопка;

name="Button" имя, которое впоследствии может

использоваться для обращения к

кнопке;
```

```
value="метка" текст, отображаемый на кнопке;

onClick="обработчик" операторы JavaScript, подлежащие

выполнению при щелчке на кнопке;

>

...

</form>
```

3. *Класс Checkbox*

Экземпляр класса Checkbox — прямоугольник, в котором может быть проставлена "галочка", означающая, что он отмечен. Графическая опция в HTML-форме. Текст, который отображается рядом с опцией, не является частью объекта checkbox и должен задаваться вне HTML-тега <input>, создающего этот объект. Обработчик событий onClick позволяет задавать программу на JavaScript, которая должна будет выполняться при выборе или отмене данной опции. Свойство checked позволяет изменять или проверять состояние объекта checkbox.

К объекту checkbox с уникальным именем имя_кнопки можно обращаться любым из следующих способов:

```
форма.имя_кнопки

форма.elements[i] – при соответствующем номере

форма.elements["имя_кнопки"]
```

Если форма содержит группу опций с одинаковыми именами, то они помещаются в коллекцию (аналог) массива. В этом случае:

форма.имя_кнопки.length либо форма.elements[i].length (при соответствующем i) либо форма.elements["имя_кнопки"].length дает число элементов с одинаковыми именами;

Обращаться к элементам по одному можно следующим образом:

```
форма.имя_кнопки[i]

форма.elements[i][j]

форма.elements["имя_кнопки"][j]



<form>

...

<input

type="checkbox" показывает, что это опция

name="имя" имя, которое впоследствии может

использоваться для обращения к
```

```
данной кнопке или группе кнопок

с таким именем;

value="значение" значение, возвращаемое при

выборе данной кнопки;

checked показывает, что данная кнопка по

умолчанию отмечена; задает

свойство defaultChecked;

onClick="обработчик" операторы JavaScript, подлежащие

выполнению при щелчке на данной

кнопке;

>

Текст HTML-текст, который должен

отображаться

рядом с данной опцией

...

</form>
```

4. ***Класс Radio***

Графическая селекторная кнопка, аналогичная checkbox, но круглая, с точкой внутри при выделении. Так же как и в checkbox, текст, находящийся рядом с кнопкой, не является частью объекта radio, и следовательно, задается вне тега <input>. Объект класса Radio всегда используется в группах взаимоисключающих опций с одним и тем же именем.

Только один объект radio в группе может содержать атрибут checked, которым задаются начальные значения свойств checked и defaultChecked (true для данного объекта, false для всех остальных селекторных кнопок в группе). Если ни один из объектов не имеет атрибута checked, то по умолчанию обычно выбирается (и становится изначально выделенным) первый объект группы. Однако это не гарантируется, и , к примеру, в IEY не выполняется.

Синтаксис почти такой же, как у Checkbox. Группы объектов radio создаются с помощью соответственного количества тегов <input> с типом radio, которые имеют один атрибут name, заданием которого устанавливается имя радиогруппы, к которой принадлежит элемент.Обращаться к кнопке можно либо как к элементу группы, либо как к элементу массива elements данной формы:

document.form.имя_группы[i] либо document.form.elements[j]

```
<form>

...

<input
```

```
type="radio"

name="RadioGroup1"

value="value1"checked>

текст1

<input

type="radio"

name="RadioGroup1"

value="value2"

>

текст2

<input

type="radio"

name="RadioGroup1"

value="value3"

>

текст3

</form>
```

5. ***Класс Reset***

Этот класс наследует свойства и методы от класса Button, но используется только для сброса значений формы в стандартные установки. Для большинства элементов это означает сброс в значение, заданное HTML-атрибутом value (а программно – к значению RO свойства defaultValue). Если начальное значение не задано, то щелчок на кнопке Reset очищает эти объекты от введенных данных. При отсутствии в HTML теге атрибута, value, на кнопке класса Reset будет надпись "Reset".

```
<form>

...

<input

type="reset"

name="имя"

value="текст_на_кнопке"

onClick="обработчик"

>
```

```
...

</form>
```

6. *Классы Text и Password*

Пункты ввода текста в форме. Атрибут size соответствующего тега задает ширину (в символах) поля ввода на экране, а атрибут maxlength — максимальное число символов, которое пользователю разрешается ввести (0 – нет ограничений). Для получения введенных данных можно прочитать свойство value либо присвоить ему в качестве значения произвольный текст в поле ввода. Если значение, вводимое пользователем, желательно скрыть, то вместо объекта Text необходимо использовать объект Password (для ввода нескольких строк следует использовать класс Textarea, описанный далее).

Если форма содержит только один объект text или password, то она будет передаваться этим объектом автоматически при нажатии пользователем с клавиатуры клавиши <Enter> (для формы возникает событие Submit).

```
<form>

...

<input

type="text"

name="имя"

size=20

value="стандартное" стандартное значение, передаваемое

при передаче формы; задает свойство

defaultValue(защищенная от

записи строка,в которой задается

начальное значение поля ввода)

onBlur="обработчик"

onChange="обработчик"

onFocus="обработчик"

>

...

</form>
```

7. *Класс Textarea*

Экземпляр Textarea в документе создается тегом <textarea> с добавлением необязательных атрибутов. Атрибут wrap, который задает длину обрабатываемых строк, имеет три допустимых значения: off

показывает, что строки нужно оставлять в таком виде, в каком они были введены; virtual требует, чтобы строки отображались с переносами, но передавались без них; physical говорит о том, что строки должны отображаться и передаваться со вставленными переносами. Программно работают режимы textarea.wrap=’off’ и textarea.wrap=’soft’.

```
<form>

...

<textarea

name="имя"

rows=10 высота объекта в строках

cols=20 ширина объекта в столбцах

wrap=off|virtual|physical

onBlur="обработчик"

onChange="обработчик"

onFocus="обработчик"

>

простой_текст

</textarea>

...

</form>
```

8. Классы Select и Option

Класс Select представляет отображаемый на экране список элементов, из которых можно выбирать нужный. Если в HTML-определении данного объекта имеется атрибут multiple, то из данного списка позволяется выбирать любое число опций. Если этого атрибута нет, то пользователь может выбирать только одну опцию, и опции работают как селекторные кнопки (т.е. являются взаимоисключающими).

Класс Option является вспомогательным для класса Select и описывает выбираемые элементы, отображаемые в объекте select. Если sel1 - объект класса Select, то sel1.options[i] обеспечивает доступ к массиву объектов option.

Программный конструктор: opt1 = New Option();

Свойства: Opt1.defaultSelected - доступно только по чтению (RO)

Opt1.index - индекс опции в массиве общий (RO)

Opt1.selected - булевское значение;

Opt1.text - текст опции;

Opt1.value - строка, передаваемая на сервер по команде Submit формы.

Если в элементе Select значение атрибута size больше единицы или если в нем имеется опция multiple, объекты Select отображаются в виде списка. Высота поля этого списка определяется свойством size (в числе строк). Если size меньше количества элементов, то список будет содержать полоса прокрутки (scrollbars). Если значение size равно 1, а атрибут multiple не указан, то отображается выпадающий список. В видимом поле показывается элемент option, помеченный как Selected. Если же такого элемента нет, отображается первый элемент списка.

```
<form>

...

<select

name="имя"

size=3

multiple

onChange="обработчик"

onBlur="обработчик"

onFocus="обработчик"

>

<option value="result" >первый пункт

<option value="1" selected >второй пункт будет сразу выделен

<option value="2">третий пункт

<option value="myresult">четвертый пункт

<option value="3">пятый пункт

</select>

...

</form>
```

multiple означает, что это с помощью клавиши <Slift> или <Ctrl> и “мыши” можно выбирать более чем один элемент из списка.

value — значение, передаваемое серверу при приеме формы в случае, если элемент выбран;

selected — пункт выбран по умолчанию.

5. Классы для программной обработки данных

1. *Класс Object*

– родительский класс для всех базовых объектов JS:

Array, Boolean, Function, Number, String, Window и др.

Поэтому все они наследуют его методы. Object обеспечивает общую функциональность всех объектов JavaScript. Этот объект имеет методы toString и valueOf и свойства prototype и constructor. Ниже эти свойства и методы объекта Object рассматриваются более подробно.

Конструктор:

obj=new Object – объект без свойств

obj=new Object(значение) – объект – оболочка типа Array, Boolean, Function, Number или String, соответствующий значению параметра.

Методы:

obj.assign(значение) – присваивание полю данных объекта указанного значения

obj.eval() – и интерпретирует строку, соответствующую объекту, как код JS

obj.toString() – строковое представление объекта

obj.valueOf() - этот метод, присутствующий у всех объектов, возвращает либо сам объект, либо его значение.

1. Свойство constructor

   Это свойство задает конструктор для объекта. Конструкторы служат для создания экземпляров объектов и присутствуют у всех классов, кроме Global и Math. Обычно это свойство используется для проверки способа создания того или иного объекта. Например, ниже выполняется проверка, позволяющая узнать, с помощью какого конструктора создан данный объект.

```
Str= new String("Создана на базе встроенного объекта String");

if (str.constructor= = String)

{

// str создан на базе String

}

else

{

// str создан на базе другого объекта

}
```

   С помощью рассматриваемого ниже метода toString мы можем узнать конструкторы переменных:

```
<head><title>JavaScript Demos</title></head>

<body>

<script language="JavaScript">
```

```
str = new String("This is a new string");

arr = new Array();

document.write(str.constructor.toString()+"<br>");

document.write(str.constructor.toString());

</script>

</body>

</html>
```

В результате выполнения этой программы на экране будет выведено:

```
function String() {[native code]}

function Array() {[native code]}
```

2. Свойство prototype

Это свойство обеспечивает доступ к прототипам методов того или иного объекта. Также возможно использование этого свойства для создания новых методов объектов. Однако, как уже говорилось, возможности создания собственных классов в JavaScript носят рекламно-декоративный характер: трудно представить ситуацию, когда эти возможности реально могут понадобиться.

2. *Класс Number*

Конструктор: N=new Number(значение)

Обычно не нужен, так как для примитивного числового типа при необходимости автоматически – преобразование в объектную форму. Используется для доступа к 5 специальным константам:

Number.MAX_value – максимальное значение для чисел

Number.MIN_value – минимальное значение для чисел

Number.NaN – значение "не-число". Преобразуется в строку как "NaN"

NEGANIVE_INFINITY – минус бесконечность. Преобразуется в строку как "-inf"

POSITIVE_INFINITY - плюс бесконечность. Преобразуется в строку как "inf"

Методы: toString(n) – преобразование числа в строковое представление по основанию n (от 2 до 16). Можно применять не только к объектам типа Number, но и к числам.

3. *Класс Boolean*

Обычно - переходный и используется для вызова метода toString() для преобразования булевских значений в строки.

Конструкторы:

b= new Boolean(значение); значения 0, null и "" преобразуются в false,остальные(в том числе "false") - в true.

Методы:

b.toString - возвращает "true" или "false"

b.valueOf() - возвращает true или false

4. *Класс String*

Строки при вызове методов типа String преобразуются во временный экземпляр объекта.

Конструктор:

s=new string(‘моя строка’)

Поля данных:

s.length – число символов в строке (индексы символов идут от 0 до s.length-1)

s.prototype – указатель на прототип

Методы:

s.anchor(имя_якоря) – возвращает строку s, включенную вHTML-контейнер

<a> с атрибутом name: содержимое строки s

s.link(URL) – то же – атрибутом href=URL;

s.big(), s.blink(), s.bold(), s.fixed(), s.italics(), s.small(), s.sub(), s.sup() –

возвращает строку s , заключенную в контейнеры, соответствующие тегам

<big>, <blink>, <b>, <tt>, <i>, <small>, <sub> и <sup>, соответственно. s.fontcolor(цвет) – возвращает строку s , заключенную в контейнер

с атрибутом color=цвет: содержимое s

Тут цвет – либо 16-ричное число из 6 символов с префиксом #, либо один из предопределенных цветов

s.fontsize(размер) – то же с атрибутом size=размер; тут размер – целое от1 до

7, либо строка, начинающаяся со знака + или -, за которым следует целое от 1 до7(относительный размер от значения по умолчанию, обычно – от 3).

s.charAt(n) – возвращает строку – n-й символ (отсчет идет от 0) строки s.

Если n не попадает в интервал от 0 до s.length-1, возвращается пустая строка;

s.indexOf(s1) – ищет в строке s слева направо подстроку s1 и возвращает индекс символа в s, с которого она начинается, либо число -1, если не найдена

s.indexOf(s1,n) – то же, но поиск в s начинается с индекса n, а не с 0.

s.lastIndexOf(s1),

s.lastIndexOf(s1,n) – то же, но поиск в s идет справа налево (от индекса

length-1 или n, соответственно)

s.split() – преобразует s в массив строк, в котором содержимое самой строки s

s.split(разделитель) – s становится массивом строк, причем “нарезание” прежнего содержимого s на строки идет по символу или строке-разделителю

s.substring(n1),

s.substring(n1,n2) – возвращает подстроку строки s от n1 до конца строки либо до последнего индекса перед индексом n2, соответственно. Индекс с номером n1 (от 0 до s.length-1) в подстроку входит, а с номером n2 – нет.

s.toLowerCase() – возвращает копию строки с преобразованием всех букв в нижний регистр

s.toUpperCase() – то же, но в верхний регистр

s.valueOf() – возвращает строковое значение объекта типа String

Замечание: методы можно применять в любом сочетании к любому строковому выражению, строке или строковой переменной.

Пример:

```
<script>

s1=”Пример:”.bold().big()+”создание гиперсвязи”.italics()

s2=”http://www.niif.spb.su/~monakhov/“

document.write(s1.link(s2))

</script>
```

5. *Класс Array*

Создает и инициализирует массив.

Поля: length - целое с возможностью чтения и записи; отсчет от нуля.

Конструкторы:

a1= new Array(); при этом значение a1.length равно 0

a2= new Array(n); a2.length равно n, инициализация элементов нулями

a3= new Array("один","два","три",…); - инициализация массива списком

Значений

Методы: a.join() - возвращает строку с поочередной конкатенацией строкового

представления элементов массива a.

a.join(s)- то же, но между строковыми представлениями элементов вставляется строка - разделитель s.

a.reverse()- процедура, изменяющая в массиве a порядок следования элементов.

a.sort()- процедура сортировки массива a по алфавиту (для выяснения порядка элементов берутся их строковые представления).

a.sort(функция сравнения от двух аргументов) - то же, но по критерию, заданному функцией (к примеру, f(x,y)). При этом f(x,y)<0 при x<y (т.е. элемент x должен быть в отсортированном массиве раньше, чем y), f(x,y)=0 при эквивалентных для данной сортировки x и y , f(x,y)>0 при x>y

6. *Класс Function*

Оболочка базового типа "функция". При вызове обычной функции как объекта производится ее скрытое преобразование во временной объект класса Function.

Конструктор:

F=new Function(arg1,…,argN, тело функции)

Поля данных:

F.arguments[ ] - массив аргументов;

F.arguments.length - его длина;

Другой способ доступа к этим полям: F[i] и F.length

F.caller - ссылка на объект Function , из которого вызван объект F(либо null, если вызов с верхнего уровня)

F.prototype - ссылки на объект, предназначенный для использования функциями- конструкторами. Все объекты, создаваемые конструктором, будут

наследовать свойства и методы прототипа (см. далее).

Методы:

F.toString() - возвращает строковое представление F

F.valueOf() - возвращает значение F.

7. *Класс JavaArray*

обращение к Java-массиву в формате массива JavaScript. Отличие Java-массива от JavaScript-массива:

- в Java их длина фиксирована, поэтому в JS поле length (RO)

элементы Java-массива типизированы, то есть не может быть элементов разного типа.

Поля: length для класса java.awt.Polygon

MyPolygon=new java.awt.Polygon()

В нем есть поля xpoints и ypoints – объекты JavaArray – массивы целых чисел; их можно

Инициализировать

For(int i=0; i<MyPolygon.xpoints.length;i++)

{MyPolygon.xpoints[i]=math.round(math.random()*100)};

for(int i=0; i<myPolygon.ypoints.length;i++)

{MyPolygon.ypoints[i]=math.round(math.random()*100)}

8. ***Класс JavaClass***

JS-представление J-класса.

Пример: var java_coonsole=java.lang.System.out; - чтение поля out (это J-объект)

Var version=java.lang.System.getProperty("java.version"); - вызов метода

Var java_date=new java.lang.Date(); - создание new J-объекта

9. ***Класс JavaObject***

экземпляры J-класса в JS.

Java.lang.System.out.println("Hello!") – вывод на J-консоль

Var r=new java.awt.Rectangle(0,0,4,5);

Var perimeter=2*r.width+2*r.height;

10. ***Класс JavaPasckage***

– J-пакет в JS

Доступ: java.lang.System

либо Packages.java.lang.System

11. ***Класс Screen***

Для получения информации о клиентском браузере в Microsoft Internet Explorer 4.0 и Netscape Navigator 4.0 введен специальный объект screen, содержащий ряд свойств, значениями которых можно воспользоваться для получения необходимой информации. Этот объект имеет свойства, уникальные для Netscape Navigator 4.0. Свойства объекта screen перечислены в следующей таблице.

Свойство Описание IE40 NN40 Чтение/Запись

---

availHeight Высота доступной области + Чтение

экрана в пикселах

availWidth Ширина доступной области + Чтение

экрана в пикселах

bufferDepth Глубина буфера + Чтение/Запись

colorDepth Максимальное число цветов, + + Чтение

поддерживаемых в данной

системе.

height Высота экрана в пикселах + + Чтение

pixselDepth Число бит на пиксел + Чтение

updateInterval Временной интервал обновления + Чтение/Запись

экрана

width Ширина экрана в пикселах + + Чтение

Рассмотрим свойства объекта screen более подробно.

1. ## Свойства availHeight и availWidth (Netscape Navigator)

   AvailHeight – высота, availWidth – ширина доступной области экрана.

2. ## Свойство bufferDepth (Internet explorer)

   Смещение экранного буфера. Если значение равно 0, буферизация не используется, и свойство colorDepth содержит максимальное число цветов, поддерживаемых в системе. Если значение равно –1, используется буферизация. Значения 1,4,8,15,16,24 или32 указывают на использование буферизаци с данным числом битна пиксел. Так, число 15 задает 16 бит на пиксел, из которых для представления RGB-цветов используются 15.

3. ## Свойство colorDepth

   Число бит на пиксел (глубина изображения), установленных в системе. Например, в зависимости от этого числа можно изменять цвета фона и текста .

   В Internet Explorer значение этого свойства следующим образом связано со значением свойства bufferDepth. Если значение своойства bufferDepth равно 0 или –1, значение colorDepth равно числу бит на пиксел. Если же значение bufferDepth больше 0, то значение colorDepth эквивалентно значению bufferDepth.

4. Свойства height и width

Эти свойства, доступные только для чтения, позволяют определить высоту (height) и ширину (width) экрана в пикселах. Знание этих значений может потребоваться в тех случаях, когда имеется несколько вариантов дизайна страницы – отдельно для низкого и высокого разрешения. Например, можно загружать различные графические изображения в зависимости от разрешения клиентского браузера.

5. Свойство pixelDepth (Netscape Navigator)

Свойство pixelDepth позволяет узнать число бит на пиксел, установленных всистеме. Возвращаемое значение обычно эквивалентно значению свойства colorDepth.

6. Свойство updateInterval (Internet Explorer)

Позволяет узнать или установить временной интервал обновления экрана. Значение 0 отменяет обновление экрана.

Литература

1. Т.Кенцл. Форматы файлов Internet./Пер. с англ. - СПб: ПИТЕР, 1997.-320 с.
2. А.В.Фролов, Г.В. Фролов. Сервер Web своими руками. Язык HTML, приложения CGI и ISAPI, установка серверов Web для Windows. –М., ДИАЛОГ-МИФИ,1998.-288 с.
3. Дж.Мейнджер. JavaScript: основы программирования: Пер.с англ. – Киев:Издательская группа BHV, 1997.-512 с.
4. А.В.Фролов, Г.В. Фролов. Сценарии JavaScript в активных страницах Web. –М., ДИАЛОГ-МИФИ,1998.-284 с.
5. Р.Дарнелл. JavaScript: Справочник. – СПб: ПИТЕР, 1998.-192 с.
6. С.Дунаев.INTRANET-технологии. –М., ДИАЛОГ-МИФИ,1997.-288 с.
7. А.Федоров. JavaScript для всех. – М.: КомпьютерПресс, 1998.-384 с.
8. С.Айзекс DynamicHTML:пер. с англ.-СПб.-BHV-Санкт-Петербург,1998.-496 с.