МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ВОРОНЕЖСКАЯ ГОСУДАРСТВЕННАЯ ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ
И ЭКОНОМИКО-МАТЕМАТИЧЕСКИХ МЕТОДОВ

РАЗРАБОТКА WEB-СТРАНИЦ.
ЯЗЫК JavaScript.

Методические указания к самостоятельной работе
по дисциплине
«ПРОЕКТИРОВАНИЕ WEB-СТРАНИЦ»

Для студентов специальности
351400 – «Прикладная информатика (в экономике)»
дневной формы обучения

Воронеж
2003

УДК 681.3.06

Разработка WEB-страниц. Язык JavaScript: Метод. указания к самостоятельной работе по дисциплине «Проектирование Web-страниц» / Воронеж. гос. технол. акад.; Сост. А.В. Ошивалов, Д.О.Сазонов, А.С. Свиридов. Воронеж, 2003. 32с.

Методические указания разработаны в соответствии с требованиями ООП подготовки информатиков-экономистов по специальности 351400 – «Прикладная информатика (в экономике)» по дисциплине «Проектирование Web-страниц» цикла общепрофессиональных дисциплин. Самостоятельная работа предусматривает углубленное изучение теоретического материала, выполнение контрольной работы. Методические указания имеют цель научить студентов основам языка JavaScript при разработке Web-страниц. Методические указания могут быть использованы студентами заочной формы обучения.

Ил. . Библтогр.: назв.

Составители доцент А.В. ОШИВАЛОВ,
ассистент Д.О. Сазонов,
доцент А. С. Свиридов,

Научный редактор профессор М.Г. МАТВЕЕВ

Рецензент доц. кафедры ИиМПМ ВГПУ В.В. МАЛЕВ

Печатается по решению
редакционно-издательского совета
Воронежской государственной технологической академии

1. Введение

JavaScript - это новый язык программирования, используемый в составе страниц HTML для увеличения функциональности и возможностей взаимодействия с пользователями. Он был разработан фирмой Netscape в сотрудничестве с Sun Microsystems на базе языка Sun's Java .С помощью JavaScript на Web-странице можно сделать то, что невозможно сделать стандартными тэгами HTML. Скрипты выполняются в результате наступления каких-либо событий, вызванных действиями пользователя. Создание Web- документов, включающих программы на JavaScript, требует наличие текстового редактора и подходящего браузера. Некоторые просмотрщики включают в себе встроенные редакторы, поэтому необходимость во внешнем редакторе отпадает.

2. Общий синтаксис JavaScript

2.1. Типы данных, имена и литералы

JavaScript распознает следующие типы данных:

- **Числа**, типа 42 или 3.14159
- **Логические** (Булевы), значения true или false
- **Строки**, типа "Howdy!"
- **Пустой указатель**, специальное ключевое слово, обозначающее нулевое значение

2.2. Преобразование типов данных

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

var answer = 42

А позже, присвоить той же переменной следующее значение:

answer = "Ответ - рыба..."

При действиях с переменными различных типов JavaScript приводит их к одному типу, например:

```
var oneInt=1
var oneString="1"
var oneConcatenate=oneString+oneInt
```

В результате получается "11"

```
var oneAddition=oneInt+oneString
```

В результате получается 2

В первой операции сложения первый операнд является строкой. JavaScript предполагает, что производится операция с двумя строками. Когда JavaScript обнаруживает в качестве второго операнда целое число, он в соответствии со своими представлениями преобразует переменную в строку.

Поскольку JavaScript свободно типизированный язык, то это не вызовет ошибки.

Так как JavaScript не поддерживает никаких методов и свойств для определения типа текущего значения переменной, очень важно внимательно отслеживать типы переменных во избежание неожиданных результатов.

Вообще, в выражениях, включающие числовые и строковые значения, JavaScript преобразовывает числовые значения в строковые. Рассмотрим следующие утверждение:

```
x = "2+2= " + 4
y = 4 + " –это ответ."
```

Первое утверждение будет строка "2+2=4". Второе утверждение возвращает строку " 4 –это ответ".

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 указан-

ном основании системы счисления, если возможно.

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

2.3. Переменные

Переменные используются чтобы хранить значения в приложении. Эти переменные должны обладать именами, по которым возможно ссылаться на них, и существуют некоторые правила, которым имена должны соответствовать.

Идентификатор JavaScript или имя должны начаться с символа или символом подчеркивания ("_"); последовательность символов также могут быть цифры (0-9). Символы включают знаки с "A" до "Z" (верхнего регистра) и знаки с "a" до "z" (нижний регистр). JavaScript учитывает регистр.

2.4. Специальные символы

В строковых литералах JavaScript можно использовать следующие специальные символы:

- **\b** указывает возврат на один символ.
- **\f** указывает перевод страницы.
- **\n** указывает новую цифру(знак) линии.
- **\r** указывает возврат каретки.
- **\t** указывает символ табуляции.
-

2.5. Выражения

Выражение - любой имеющий силу набор литералов, переменных, операторов, и выражений, которые вычисляю простое значение. Значение может быть число, строка, или логическое значение. Существует два типа выражений: которые присваивают значение переменной, и которые вычисляют выражение без присваивания его переменной. Например, выражение **x = 7** является выражением, которое приписывает x значение 7. Это выражение вычисляет 7. Такие выражения используют операторы присвое-

ния. С другой стороны, выражение **3 + 4** просто вычисляет 7; оно не выполняет присвоения. Операторы, используемые в таких выражениях, упоминаются просто как операторы.

JavaScript имеет следующие выражения:

- **Арифметические**: вычисляет число
- **Строковые**: вычисляют строку символов, например "Привет" или "234"
- **Логические**: вычисляют true(истина) или false(ложь)

Язык JavaScript также включает в себя значение **null** для переменных, которым не присвоено никакое значение.

2.6. Условные выражения

Условное выражение может иметь одну из двух значений, основанных на условии. Синтаксис:

(Условие)? Val1: val2

Если условие истинно, то выражение имеет значение val1, иначе имеет значение val2, например:

status = (age = 16) ? "взрослый" : "маленький".

Это утверждение присваивает значение "взрослый" переменной status, если age равно 16 или больше чем 16. Иначе, приписывает значение "маленький" переменной status.

2.7. Операторы присваивания

Оператор присваивает значение левому операнду, основанному на значении правого операнда. Основной оператор присваивания равенство (=), который присваивает значение правого операнда левому операнду. То есть x = y приписывает значение y к x.

Другие операторы - стенография для стандартных арифметических действий выглядят следующим образом:

- **Сложение, конкатенация:** X + = y означает x = x + y
- **Вычитание:** x - = y означает x = x - y
- **Умножение:** X * = y означает x = x * y
- **Деление:** X / = y означает x = x / y

– **Остаток от деления**: X % = y означает x = x % y

2.8. Логические операторы

Логические операторы принимают логические (Булевы) значения как операнды. Они возвращают логическое значение **true (истина)** или **false (ложь)**.

И (&&). Использование: expr1 && Expr2

Логический "и" возвращает оператор true, если оба логических выражения и expr1 и expr2 true. Иначе, возвращается false.

Или (||). Использование: expr1 || expr2

Логический "или" возвращает оператор true, если хотя бы одно из логических выражений или expr1 или expr2 true. Если и expr1 и expr2 false, то это возвращается false.

Отрицание (!). Использование:! Expr

Логический оператор "отрицание" - унарный оператор, который отрицает выражение операнда expr. То есть если expr true, то возвращает false, и если expr false, то возвращает true.

2.9. Операторы сравнения

Оператор сравнения сравнивает его операнды и возвращает логическое значение, основанное на том, является ли сравнение true или false. Операнды могут быть численными или строковыми значениями. Когда используется на строковых значениях, то сравнения основывается на стандартном лексикографическом порядке.

Операторы:

- **Равно = = :** возвращает true, если операнды равны.
- **Не равно != :** возвращает true, если операнды не равны.
- **Больше чем > :** возвращает true, если левый операнд больше чем правый операнд. Пример: x>y возвращает true, если x больше чем y.
- **Больше или равно >= :** возвращает true, если левый операнд больше или равен правому операнду. Пример: x>=y возвращает true, если x больше или равен чем y.

- **Меньше чем < :** возвращает true, если левый операнд - меньше чем правый операнд. Пример: x<y возвращает true, если x - меньше чем y.
- **Меньше или равно < = :** возвращает true, если левый операнд - меньше или равен правому операнду. Пример: x<= y возвращает true, если x - меньше или равен y.

2.10. Операторы работы со строкой

В дополнение к операторам сравнения, которые могут использоваться на значениях строк, оператор конкатенации (+), суммирует две строки вместе, возвращая другую строку, которая является соединением двух строк операнда. Например, **"тестовая " + "строка"** возвращает строку **"тестовая строка"**.

Оператор присвоения += может также использоваться, чтобы конкатенировать строки. Например, если переменная mystring - строка, которая имеет значение "Альфа", то выражение Mystring + = " бета"

Вычисляет как "Альфа бета" и приписывает это значение mystring.

2.11. Управляющие операторы

Оператор JavaScript состоит из ключевого слова, используемого с соответствующим синтаксисом. Один оператор может занимать несколько строк. несколько операторов могут занимать одну строку, отделённые один от другого знаком "точка с запятой".

Условный оператор - if...else - выполняет набор операторов, если специфицированное условие true. Если условие false, может быть выполнен другой набор операторов.

Синтаксис:

```
if (condition)
{
  statements1
}
[else {
```

```
  statements2
}]
```

condition – условие, может быть любым выражением JavaScript, которое вычисляется в true или false. Должно быть заключено в скобки. Если true, выполняются операторы statements1.

statements1, statements2 - любые операторы JavaScript, включая вложенные if. Несколько операторов обязаны быть заключены в фигурные скобки.

Цикл с условием - do...while - выполняет специфицированные операторы, пока при тестировании условия не будет возвращено false. Операторы выполняются как минимум однократно.

Синтаксис:

```
do
  statements
while (condition);
```

statements - блок операторов который выполняется минимум один раз и выполняется повторно до тех пор, пока проверка условия возвращает true.

condition - вычисляется после каждого прохождения цикла. Если условие вычисляется в true, операторы предыдущего блока выполняются повторно. Если проверка условия возвращает false, управление передаётся оператору, следующему после do while.

Пример:

Цикл отрабатывает минимум один раз и повторяет итерации, пока i не меньше 5.

```
do
{
  i+=1;
  document.write(i);
}
while (i<5);
```

Цикл с условием – while -создаёт цикл, вычисляющий выражение, и, если оно true, выполняет блок операторов. Затем цикл повторяется, пока специфицированное условие true.

Синтаксис:

```
while (condition)
{
  statements
}
```

condition - вычисляется перед началом каждой итерации цикла. Если это условие вычисляется в true, выполняются операторы в следующем блоке. Если condition вычисляется в false, выполнение продолжается в операторах, следующих после statements.

statements - блок операторов, выполняемый, пока условие true. Хотя это и не обязательно, желательно выделять эти операторы в коде отступом относительно начала оператора.

Пример:

Следующий цикл while отрабатывает, пока n меньше трёх:

```
n = 0;
x = 0;
while (n < 3)
{
  n ++;
  x += n;
}
```

При каждой итерации цикла n увеличивается и прибавляется к x. Следовательно, x и n принимают следующие значения:

После первого прохода: n = 1 и x = 1

После второго прохода: n = 2 и x = 3

После третьего прохода: n = 3 и x = 6

После завершения третьего прохода цикла, условие n < 3 больше не true, поэтому цикл прерывается.

Цикл с параметром – for - создаёт цикл из трёх необязательных выражений, заключённых в скобки и разделённых точкой с запятой, и блока операторов , выполняемых в цикле.

Синтаксис:

```
for ([initial-expression]; [condition]; [increment-expression])
{
```

```
  statements
}
```

initial-expression - оператор или объявление переменной. Обычно используется для инициализации переменной счётчика цикла. Это выражение может объявлять новую переменную с помощью ключевого слова var. Эти переменные являются локальными относительно функции, а не цикла.

condition - условие, вычисляемое при каждой итерации цикла. Если вычисляется в true, выполняются операторы statements. Эта проверка условия не является обязательной. При её отсутствии, условие всегда вычисляется в true.

increment-expression - это выражение обычно используется для инкремента или обновления переменной счётчика цикла.

statements - блок операторов, которые выполняются, пока условие true. Это может быть один или несколько операторов. Хотя это и не требуется, хорошим стилем будет выделить эти операторы отступом относительно начала оператора for.

Пример:

Здесь оператор **for** начинается с объявления переменной i и её инициализации в 0. Он проверяет, что i меньше 9, выполняет два последующих оператора и увеличивает i на 1 после каждого прохода по циклу.

```
for (var i = 0; i < 9; i++)
{
  n += i;
  myfunc(n);
}
```

for...in - обрабатывает специфицированную переменную по всем свойствам объекта. Для каждого выделенного свойства JavaScript выполняет специфицированные операторы.

Синтаксис:

```
for (variable in object) {
  statements
}
```

variable - переменная для итерации по каждому свойству, может быть объявлена ключевым словом var. Эта переменная локальна относительно функции, а не цикла.

object - объект, по свойствам которого происходит итерация.

statements - специфицирует операторы, выполняемые для каждого свойства.

Пример:

Здесь функция принимает в качестве аргументов объект и имя объекта . Затем выполняется итерация по всем свойствам объекта и возвращает строку со списком имён свойств и их значений.

```
function show_props(obj, objName) {
  var result = "";
  for (var i in obj) {
    result += objName + "." + i + " = " + obj[i] + "\n";
  }
  return result;
}
```

break - оператор используется для прерывания выполнения операторов цикла или **switch** и передаёт управление оператору, следующему после прерванного цикла.

Пример:

В этой функции имеется оператор **break**, прерывающий выполнение цикла **while**, если е равно 3, и возвращающий затем значение 3 * x.

```
function testBreak(x) {
  var i = 0;
  while (i < 6) {
    if (i == 3)
      break;
    i++;
  }
  return i*x;
}
```

return - определяет значение, возвращаемое функцией.

Синтаксис:

return expression;

expression - возвращаемое выражение.

Пример:

Следующая функция возвращает квадрат своего аргумента x, где x это число.

```
function square(x)
{
  return x * x;
}
```

Оператор множественного выбор – switch - позволяет программе вычислять выражение и попытаться сопоставить его значение с меткой **case**.

Синтаксис:

```
switch (expression)
{
  case label :
    statements;
    break;
  case label :
    statements;
    break;
  ...
  default: statements;
}
```

expression - значение, сопоставляемое с label.

label - идентификатор, используемый при сопоставлении с expression.

statements - блок операторов, выполняемый однократно при совпадении expression с label.

Если совпадение найдено, программа выполняет ассоциированный оператор. Если несколько **case** совпадают с предоставленным значением, выбирается первый совпавший **case**, даже если все эти **case** не равны один другому.

Программа сначала ищет метку/label, совпадающую со значением expression/выражения, а затем выполняет ассоциированный оператор. Если ни одна метка не совпала, программа ищет необязательный оператор **default** и, если он найден, вы-

полняет ассоциированный оператор. Если оператор **default** не найден, программа продолжает выполняться с оператора после конца блока **switch**.

Необязательный оператор **break**, ассоциированный с каждой меткой **case**, гарантирует что прервёт выполнение блока **switch**, после того как совпадающий оператор будет выполнен, и продолжит выполнение с оператора, идущего после блока **switch**. Если **break** опущен, программа продолжает выполнение с оператора, следующего после оператора **case**.

Пример:

Если expression вычисляется до "Bananas", программа ищет совпадение этого значения с **case** "Bananas" и выполняет ассоциированный оператор. Если обнаружен **break**, программа прерывает выполнение блока **switch** и выполняет оператор, идущий после **switch**. Если **break** отсутствует, оператор для **case** "Cherries" также будет выполнен.

```
 switch (i) {
   case "Bananas" :
     document.write("Бананы.<BR>");
     break;
   case "Cherries" :
     document.write("Вишня.<BR>");
     break;
   default :
     document.write("Ничего .<BR>");
 }
 document.write("Обработка завершена<BR>");
```

3. Расширенный синтаксис JavaScript

3.1. Объектная модель браузера

При интерпретации страницы HTML браузером создаются объекты, свойства которых представляют значения параметров тэгов языка HTML. Все объекты хранятся в виде иерархической (древовидной) структуры, отражая структуру документа. Ниже показана структура объектов браузера Netscape Navigator.

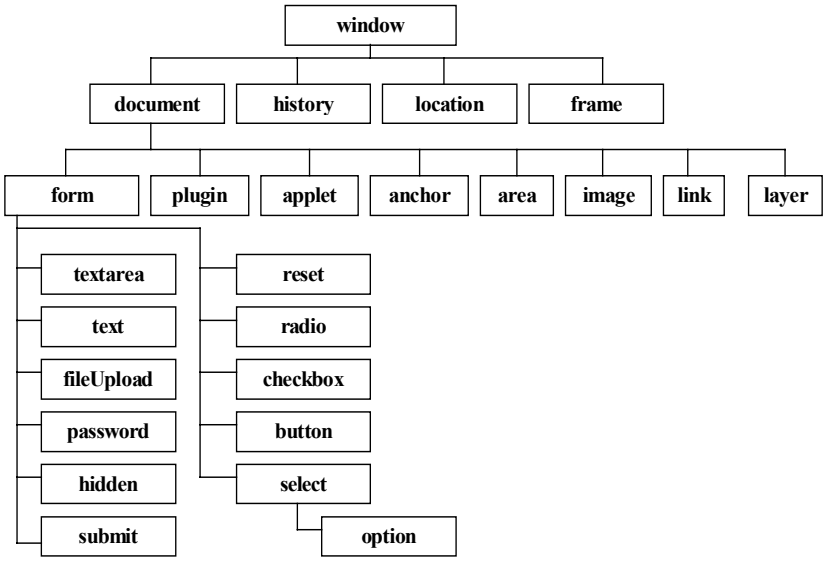

Рис. 1 Структура объектов браузера Netscape Navigator

Объектная модель браузера Internet Explorer отличается от модели браузера Netscape.

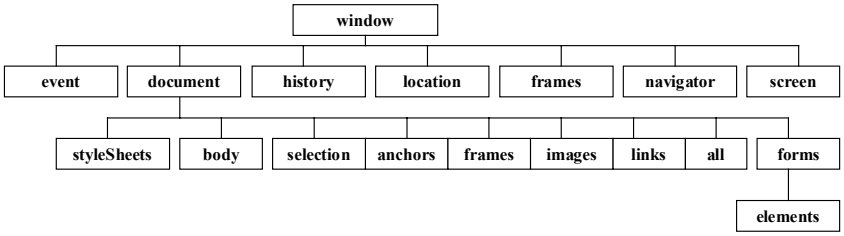

Рис. 2 Объектная модель браузера Internet Explorer

Все операции, которые можно исполнять в программе на JavaScript, описывают действия над хорошо известными и понятными объектами. Кроме этих объектов пользователь может соз-

давать и свои собственные. Но обычно большинство программ используют стандартную объектную модель и не создают новых объектов.

3.2. Объекты и свойства

Объект JavaScript имеет свойства ассоциированные с ним. Обращение к свойствам объекта осуществляется следующей простой системой обозначений:

Объект.свойство

Имя объекта и имя свойства чувствительны к регистру. Свойства могут быть определены, заданием им значений. Например, пусть существует объект, с именем myCar, тогда можно определить свойства, именованные make, model, year следующим образом:

```
myCar.make = "VOLGA"
myCar.model = "GAZ3110"
myCar.year = 2000;
```

3.3. Функции и методы

Функция – это JavaScript процедура - набор инструкций, которые выполняют определенную задачу, возвращая результат.

Определение функции состоит из ключевого слова **function**, сопровождаемого именем функции, списком аргументов в круглых скобках разделяемых запятыми, JavaScript инструкциями в фигурных скобках {...} определяющими функцию.

Использовать можно только функции, определенные на текущей странице. Лучше всего определять все функции в начале страницы. Когда пользователь загружает страницу, сначала загружаются функции.

Определение функции не выполняет ее. Для этого необходимо вызвать функцию, чтобы выполнить ее. Например, можно вызывать функцию pretty_print следующим образом:

pretty_print("Здесь находится текст")

Функция может быть рекурсивной, то есть она может вы-

зывать 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
  }
}
```

Можно показать факториалы от одного до пяти следующим образом:

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

Результаты будут следующие:

0 факториал – 1
1 факториал единицы- 1
2 факториал двойки - 2
3 факториал тройки - 6
4 факториал четверки - 24
5 факториала пятерки - 120

3.4. Определение методов

Метод - функция, связанная с объектом (см. объектную модель броузера). Метод определяется таким же образом, так как и определение стандартной функции. Чтобы связать функцию с существующим объектом необходимо использовать следующий синтаксис:

Объект.имя_метода = имя_функции

Где **объект** - существующий объект, **имя_метода** - имя, которое вы присваиваете методу, и **имя_функции** - имя функции.

Вызов метода в контексте объекта осуществляется следующим образом:

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

JavaScript имеет специальное ключевое слово **this** которое может быть использовано для обращения к текущему объекту.

3.5. Создание новых объектов

И клиент и сервер JavaScript имеют предопределенные объекты. Кроме того, возможно создание собственных объектов. Создание собственного объекта требует двух шагов:

Определить тип объекта, написанной функции.

Создать образец объекта используя оператор **new**.

Чтобы задать тип объекта, необходимо создать функцию для типа объекта, которая определяет его имя, свойства и методы.

Например, пусть необходимо создать тип объекта для автомобилей который будет назван car и будет иметь свойства make, model, year, и color. Чтобы сделать это, необходимо написать следующую функцию:

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

this используется чтобы присвоить значения свойствам объекта, основанные на значениях функции.

Теперь возможно создать объект, с именем mycar следующим образом:

```
mycar = new car("Волга", "2410", 1993);
```

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

Может быть создано любое число объектов car используя **new**. Например,

```
kenscar = new car("Жигули", "2110", 2001)
```

Объект может иметь свойство, которое является самостоятельным другим объектом. Например, пусть был определен объект с именем person:

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

И затем создано два новых объекта person следующим образом:

```
one = new person("Иванов И.И.", 33, "M")
two = new person("Петров П.П.", 39, "M")
```

Затем можно переписать определение car, чтобы включить свойство владельца, которое представляет собой объект person, следующим образом:

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