МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Озерский технологический институт — филиал НИЯУ МИФИ

Вл. Пономарев

Введение в JavaScript для программистов

Учебно-методическое пособие по дисциплине
«Современные технологии программирования»

Озерск, 2015

УДК 681.3.06
П56

Пономарев В.В. Введение в JavaScript для программистов. Учебно-методическое пособие. Озерск: ОТИ НИЯУ МИФИ, 2015. — 48 с., ил.
Редакция 2015-09-09 (Электронный документ).

В пособии описывается язык JavaScript, его применение при создании web-приложений. Рассматривается объектная модель обозревателя BOM и объектная модель документа DOM. Кратко освещается технология AJAX.

Пособие предназначено для студентов-программистов, обучающихся по специальности 09.05.01 — «Применение и эксплуатация автоматизированных систем специального назначения», и направлению подготовки 09.03.01 — «Информатика и вычислительная техника», в ходе изучения предмета «Современные технологии программирования».

Предполагается, что читатель знаком с языком программирования Си и основами объектно-ориентированного программирования.

Рецензенты:
1) Зам. начальника ИВЦ «ПО «Маяк» В.Е. Синяков.
2) Ст. преподаватель кафедры ПМ ОТИ НИЯУ МИФИ А.Ф. Зубаиров.

УТВЕРЖДЕНО
Редакционно-издательским
Советом ОТИ НИЯУ МИФИ

ОТИ НИЯУ МИФИ, 2015

Содержание

Язык JavaScript. Историческая справка

Язык JavaScript, хотя и содержит в своем названии слово Java, имеет к последнему лишь косвенное отношение. После появления языка Java фирма Netscape заинтересовалась им и, объединив свои усилия с компанией Sun (создавшей язык Java), в 1995 г. создала язык JavaScript, который работал на стороне клиента, первоначально в распространенном ранее обозревателе Netscape Navigator.

Между языками Java и JavaScript сходство только в названии и синтаксисе. Синтаксис, впрочем, происходит от языков C и C++. В остальном эти языки значительно различаются.

Язык Java является компилируемым (с последующей интерпретацией кода специальным программным обеспечением под названием «виртуальная Java-машина»), а язык JavaScript — интерпретируемым. Интерпретацию кода JavaScript, который встраивается непосредственно в текст страницы HTML, берет на себя обозреватель. С помощью языка JavaScript страница HTML может вести своеобразный «диалог» с пользователем, оперативно изменяя свой вид без перезагрузки, а также осуществлять обработку данных, вводимых пользователем в поля форм. Поэтому такие страницы HTML принято называть «динамическими» (DHTML).

Что касается объектно-ориентированных языковых свойств, то здесь различие огромное. Язык JavaScript не является объектно-ориентированным в обычном смысле. Вместо этого язык может использовать объекты, созданные обозревателем (в виде «объектной модели обозревателя»), а также объекты, созданные самим языком (точнее, — сценарием на этом языке). В JavaScript можно создать объект, но нельзя создать класс со всеми вытекающими последствиями (например, понятие абстрактного класса или интерфейса для JavaScript являются неправильными). Поэтому правильнее называть этот язык объектно-направленным.

Аналогом языка JavaScript является язык VBScript, в основе синтаксических конструкций которого лежит популярный язык Visual Basic фирмы Microsoft. Заметим попутно, что существует также язык Visual Basic Scripting Edition, который также является скриптовым, однако он не предназначен для работы в обозревателе (его цель — расширение набора команд операционной системы).

Поскольку JavaScript непосредственно взаимодействует с объектами HTML-страницы, при его использовании знание HTML и DHTML существенно необходимо. Если HTML описывает статическое размещение элементов страницы, то DHTML описывает параметры (свойства, методы и события) этих элементов. Большая часть этих сведений носит справочный характер.

Клиент-сервер

Прежде, чем начать изучение применения JavaScript, нужно уяснить его точное положение в системе «клиент-сервер», действующей в рамках технологий в сети Интернет.

Процесс получения HTML-страницы клиентом (обозревателем) по протоколу HTTP условно изображен на рисунке 1.

Рисунок 1 — Клиент и сервер

На стороне клиента пользователь в обозревателе вводит в строку адреса так называемый URL интересующей его страницы HTML. URL — Universal Resource Locator, *универсальный указатель местоположения ресурса*. Через телекоммуникационные сети запрос поступает на некоторый сервер сети Интернет, который содержит требуемый ресурс (вопросы маршрутизации или использования стека протоколов TCP/IP для установления соединения и передачи данных, а также протокол CGI здесь для простоты не рассматриваются).

На рисунке 1 в качестве ресурса выступает сценарий. Будем понимать под сценарием программу, выполняющуюся на сервере и формирующую ответ обозревателю в виде HTML-страницы. Заметим, что, в принципе, ресурсом может выступать страница HTML в чистом виде (не требующая предварительной интерпретации). Неправильно будет говорить о таком ресурсе, как о сценарии, потому что в этом случае страница просто отправляется обозревателю.

После получения ответа обозреватель размещает HTML-страницу в своем окне, пользователь ее наблюдает. Часть страницы HTML может занимать сценарий (например, на языке JavaScript). На рисунке 1 этот сценарий отмечен на странице HTML также словом «сценарий».

Разница между левым (на рисунке) и правым сценариями заключается в моменте и месте их выполнения. Правый сценарий работает на сервере (например, с помощью соответствующего интерпретатора) в момент его вызова, левый сценарий работает непосредственно в обозревателе в момент, когда это предписано самим сценарием.

Основы HTML

Здесь излагаются только начальные сведения об HTML.

HTML расшифровывается как Hypertext Markup Language, — *язык разметки гипертекста*. Это чрезвычайно лояльный в отношении ошибок язык, с помощью которого задается размещение элементов страницы, таких, как текст или рисунок, в окне произвольного размера. Особенностью являются *ссылки*, с помощью которых можно переходить с текущей страницы на другую страницу в этом же или другом окне обозревателя, или на текущую страницу в другом месте. При этом обычно производится выполнение нового запроса по описанной выше схеме.

Другой особенностью являются так называемые «формы», с помощью которых пользователь может вводить информацию, которая затем опять же отправляется серверу в виде нового запроса (с получением новой HTML-страницы).

В основе HTML лежат теги. Тег (*tag*) — это ключевое слово, заключенное в треугольные скобки. Например, <HTML> — тег, начинающий страницу HTML. Некоторые теги требуют указания окончания области действия тега. Для этой цели используются завершающие теги вида </...>. Например, тег <HTML> требует завершающего тега </HTML>. Все, что расположено в тексте между этими двумя тегами, есть содержание страницы HTML.

Некоторые теги не требуют завершения, например,
 (мягкий перенос строки) или <IMG> (рисунок). Иногда такие теги для совместимости с XML «закрывают» следующим образом:
. Некоторые завершающие теги без каких-либо последствий могут быть пропущены, например, тег </P>, завершающий параграф.

Рассмотрим пример простейшего HTML-документа.

Листинг 1

```
<HTML>
  <HEAD>
    <TITLE>Заголовок страницы</TITLE>
  </HEAD>
  <BODY>
    Содержание страницы
  </BODY>
</HTML>
```

Весь документ заключен в тег <HTML>. Документ поделен на две части тегами <HEAD> и <BODY>. В части HEAD (заголовок) размещается дополнительная информация о документе (например, используемая кодировка, мета-информация, используемые стили и т.п.). В примере в заголовке размещен тег <TITLE>, который определяет заголовок страницы, отображаемый в заголовке обозревателя.

В части BODY (тело) размещаются элементы страницы. Можно сказать, что именно здесь разворачиваются основные действия JavaScript. Тело документа содержит множество различных тегов, управляющих отображением, а также ссылками и формами.

При размещении элементов страницы обычно используется *поточный* принцип: элементы размещаются слева направо и сверху вниз по мере прочтения HTML текста. Однако некоторые элементы могут быть размещены с отступлением от этого правила. Например, элемент может быть размещен по некоторым абсолютным (относительно некоторого контейнера) или относительным (относительного поточного размещения) координатам.

Часть тегов являются контейнерами, например, <DIV>, <TABLE>, <TD>, <FORM>. Контейнеры могут содержать внутри себя другие теги. Некоторые теги не могут служить контейнерами, например,
, <HR>, <IMG>.

Важной частью тегов являются параметры. Параметры записываются внутри открывающего (или единственного) тега в следующем виде:

```
имя-параметра="значение"
```

при этом окружающие значение кавычки не являются обязательными, но рекомендуемыми. Вот пример параметра тега <BODY>, задающего стиль, который определяет цвет фона документа:

```
<BODY style="background-color:red">
```

Параметры тегов определяют характеристики элементов (то есть их свойства). JavaScript часто имеет дело с параметрами элементов, поэтому допустимые параметры тегов (объектов) нужно знать.

Важнейшими параметрами являются id (идентификатор элемента) и name (название элемента). С их помощью можно получить элемент в виде объекта, а затем выполнить обращение к его свойствам и методам.

Правила образования идентификатора допускают наличие знака "-".

Вот пример тега, описывающего элемент данных таблицы, и имеющего идентификатор "main-data":

```
<TD id="main-data">
```

Рассмотрим применение основных тегов.

Тег <TABLE> позволяет разместить данные по ячейкам таблицы. Часто таблицы используют для формирования областей документа, которые затем с помощью внутренних таблиц делят на еще более мелкие части.

Вот пример кода, создающего таблицу из четырех ячеек (два ряда по две ячейки). Строку задает тег <TR>, ячейку — тег <TD>:

Листинг 2

```
<TABLE>
  <TR>
    <TD>1-1</TD>
    <TD>1-2</TD>
  </TR>
  <TR>
    <TD>2-1</TD>
    <TD>2-2</TD>
  </TR>
</TABLE>
```

Заметим, что не рекомендуется оставлять ячейку таблицы пустой. В этом случае в нее лучше записать неразрывный пробел в виде " ".

Другой важный тег <A> — якорь (*anchor*). С его помощью формируют ссылки на другие страницы гипертекста. Вот пример этого тега:

```
<A href="http://domen.domen/page" target=_blank>Ссылка</A>
```

Здесь тег <A> имеет два параметра. Параметр href задает URL страницы, на которую будет выполнен переход, если пользователь щелкнет на слово «Ссылка», заключенное в тег. Параметр target задает окно, в которое будет выведена запрошенная страница. Два распространенных значения этого параметра: _blank — в новое окно обозревателя (или новую вкладку), _self — в это же окно (вкладку), вместо текущей страницы.

Важное значение имеют формы. Формы принимают параметры от пользователя и отправляют их в виде запроса на некоторый URL.

Рассмотрим пример формы, которая принимает логин и пароль.

Листинг 3

```
<FORM action="http://domen.domen/page" method="POST">
  <TABLE>
    <TR><TD>Логин:</TD></TR>
    <TR><TD><INPUT type="text" name="login"></TD></TR>
    <TR><TD>Пароль:</TD></TR>
    <TR><TD><INPUT type="password" name="pass"></TD></TR>
    <TR><TD><INPUT type="submit" value="Отправить"></TD></TR>
  </TABLE>
</FORM>
```

Для размещения элементов здесь использована таблица.

Тег <INPUT> задает элемент ввода, параметр type указывает, какой именно. Значение "text" задает текстовое поле, "password" — поле для ввода пароля, "submit" — кнопку для отправки формы с надписью «Отправить». Параметр формы method задает способ отправки данных формы. Значение GET отправляет параметры в строке запроса, а значение POST — в теле запроса. Параметр name элемента формы служит для идентификации элемента.

Основы CSS

Другой важной частью HTML-страниц являются CSS (*cascading style sheets* — каскадные таблицы стилей). Они позволяют задавать оформление элементов страницы отдельно от описания самих элементов.

Вообще есть три способа задать стиль элемента.

1) Стиль задается непосредственно в теге элемента как параметр.

2) Стиль задается внутри заголовка документа в контейнере <STYLE> для всех элементов одного типа или класса.

3) Стиль задается отдельным файлом типа .css также для всех элементов одного типа или класса.

При этом действуют следующие правила:

- стили, определенные в заголовке внутри тега <STYLE>, преобладают над стилями, определенными во внешнем файле стилей;

- стили, определенные внутри тега элемента, преобладают над всеми другими определениями.

Рассмотрим конкретный пример.

Листинг 4

```
<HTML><HEAD>
<LINK rel="stylesheet" type="text/css" href="page.css">
<STYLE type="text/css">
  TD {background-color:blue}
</STYLE>
</HEAD><BODY>
  <TABLE><TR>
      <TD>Ячейка 1</TD>
      <TD style="background-color:red">Ячейка 2</TD>
  </TR></TABLE>
</BODY></HTML>
```

Тег <LINK> задает внешний файл стилей page.css, который имеет следующее содержание:

```
TD {background-color:green}
```

Таким образом, внешний файл задает зеленый цвет ячеек таблиц.

Внутренняя таблица стилей, определенная в заголовке документа внутри тега <STYLE>, задает синий цвет ячеек таблиц. В теле документа определена таблица с двумя ячейками в одной строке. Первая ячейка будет иметь синий цвет, потому что внутренняя таблица преобладает над внешней. Вторая ячейка будет иметь красный цвет, потому что для этой ячейки стиль определен непосредственно для данного элемента с помощью параметра style.

Знание правил CSS важно при программировании на JavaScript, потому что одной из распространенных задач сценариев является управление оформлением элементов, которое производится через стили.

JavaScript

Задачи JavaScript

Можно выделить следующие основные задачи, выполняемые при помощи JavaScript.

1) Определение параметров обозревателя, таких, как тип, версия, поддержка сценариев или куки. Эти параметры помогают выбрать правильную модель поведения сценария, так как разные обозреватели в одинаковых ситуациях зачастую ведут себя по-разному.

2) Управление документами. Оно включает в себя, например, определение адреса текущего документа, изменение адреса (переадресацию), создание документов «на лету», создание всплывающих окон, управление размером окон, положением линеек прокрутки, чтение и запись куки и т.п.

3) Управление параметрами элементов страницы. Это позволяет придать странице HTML более привлекательный вид за счет ее реагирования на различные действия пользователя.

4) Работа с данными отправляемых форм. Перед отправкой данных формы на сервер JavaScript позволяет проверить их допустимость, а при необходимости запросить недостающие или неверные данные, или попросить подтвердить действия пользователя.

5) Взаимодействие с сервером посредством посылки запросов на него непосредственно во время отображения страницы без ее перезагрузки. Используемые при этом средства часто называют ajax.

Место сценария в документе

Код JavaScript может быть расположен во внешнем файле и непосредственно в документе HTML. Во внешнем файле обычно размещают пакеты функций. Пример включения внешнего файла в документ:

```
<SCRIPT type="text/javascript" src="name.js"></SCRIPT>
```

Используемый здесь тег <SCRIPT> должен быть размещен в заголовке HTML-страницы. Расположение внешнего файла задает параметр src.

Непосредственно в документе код JavaScript размещается внутри тега <SCRIPT>, который может находиться в произвольном месте тела документа или в его заголовочной части. При этом следует учитывать порядок загрузки документа, так как выполнение кода может начаться раньше, чем завершится загрузка элементов, к которым сценарий обращается. Обычное место размещения сценариев — заголовочная часть страницы HTML.

В следующем примере показано, как оформляется часть документа, содержащую сценарий, в заголовочной части документа.

Листинг 5

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript" src="name.js"></SCRIPT>
<SCRIPT>
<!--
  // код JavaScript
//-->
</SCRIPT>
</HEAD>
<BODY>
  текст документа
</BODY>
</HTML>
```

Отметим, что код заключен в комментарий HTML. Это сделано с целью исключения отображения кода как текста обозревателями, которые не «понимают» тега <SCRIPT>. При этом перед завершением комментария HTML поставлен комментарий JavaScript с целью предотвратить интерпретацию строки "-->" как кода JavaScript. Современные обозреватели направлены на использование сценариев JavaScript, однако указанное оформление в виде комментариев опускать не рекомендуется.

Синтаксис JavaScript

Комментарии

Комментарии JavaScript полностью аналогичны комментариям Си.

Так, два знака слеш "//" начинают строчный комментарий, а блочный комментарий заключается в пары знаков "/*" и "*/".

Переменные и значения

При объявлении *переменной* тип данных не указывается, а определяется интерпретатором во время выполнения по значению. При этом по мере выполнения сценария переменная может принимать любой тип.

JavaScript распознает следующие типы значений.

Integer — целые числа в десятичном, восьмеричном и шестнадцатеричном форматах.

Float — действительные числа в десятичном формате.

String — символы, заключенные в кавычки (двойные и одиночные).

Boolean — логические значения true и false.

null — пустой указатель.

Пример объявления (глобальных) переменных:

```
var a, b = 0, c = null, d = [0, 1, 2, 3, 4], e = "abc", f = 'ijk';
```

Вследствие слабой типизации JavaScript лояльно относится к смешению типов. Результат вычислений при этом не всегда ожидаемый. В качестве примера рассмотрим сложение двух переменных, одна из которых содержит числовое значение, а другая — строковое.

Следующий сценарий выводит сообщение "a + b = 12":

```
var a = 1, b = "2", c = a + b;
alert("a + b = " + c);
```

Результат, таким образом, имеет строковый тип. Чтобы получить числовой результат, строковое значение нужно привести к числовому при помощи функции eval, parseInt или parseFloat.

Следующий сценарий выводит сообщение "a + b = 3":

```
var a = 1, b = "2", c = a + parseInt(b);
alert("a + b = " + c);
```

Массивы задаются значениями элементов через запятую, заключенными в квадратные скобки. Следующий сценарий показывает описание массива размерности три. Сценарий выводит сообщение "undefined", так как указанный в методе alert элемент массива не задан:

```
var a = [[1, 2, 3], [4], [7, 8]];
alert(a[2][2]);
```

Массивы могут быть созданы с использованием генератора new, поскольку JavaScript определяет внутренний класс Array:

```
a = new Array(5);
for(i = 0; i < 5; i++) a[i] = i + 1;
alert(a);
```

Сценарий выводит сообщение "1, 2, 3, 4, 5".

Элементы массива могут иметь разный тип:

```
a = [1, "b", false];
alert(a);
```

Сценарий выводит сообщение "1, b, false".

Дополнительно о массивах см. класс Array.

Для формирования строковых значений, как и в Си, используются некоторые специальные символы. Они предваряются знаком "\":

"**\b**" — забой (*backspace*);
"**\f**" — перевод страницы;
"**\n**" — новая строка (перевод строки);
"**\r**" — возврат каретки;
"**\t**" — табуляция;
"****" — обратный слеш.

Операции

Операции JavaScript в общем соответствуют операциям языка Си.

Операции присваивания: "=", "+=", "–=", "*=", "/=", "%=".

Арифметические операции: "+", "–", "*", "/", "%", "++", "––".

Операции отношений:

"==", "===", ">", ">=", "<", "<=", "&&", "||", "!", "!=", "!==".

Заметим, что здесь есть две новые операции: "===" и "!==".

Операция «строго равно» ("===") проверяет точное равенство. Следующий пример поясняет применение операций "===" и "==":

```
var a = 1, b = "1"; alert(a==b); alert(a===b);
```

Здесь сравниваются переменные разных типов, но с одинаковым числовым значением. Сценарий выводит сначала сообщение "true", а затем сообщение "false". Таким образом, строгое равенство означает совпадения значений и типов переменных (выражений).

Операция «строго не равно» ("!==") точно также проверяет строгое соответствие типов и неравенство значений. Следующий сценарий сначала выводит сообщение "false", а затем сообщение "true".

```
var a = 1, b = "1"; alert(a!=b); alert(a!==b);
```

Побитовые операции: "&", "|", "^", "~", "<<", ">>", ">>>".

Здесь новая операция ">>>". Она выполняет сдвиг вправо без знака. Следующий сценарий выводит значения "–64" и "2147483584":

```
var a = -128; alert(a>>1); alert(a>>>1);
```

Как и в Си, в JavaScript есть условная операция "()?:". Следующий сценарий выводит значение "true":

```
alert((1==1)?true:false);
```

Приоритеты операций в JavaScript такие же, как и в Си.

Самый высокий приоритет имеют операции "!", "~", "–" (унарный минус), "++", "––", самый низкий — операция "," (запятая).

Операция typeof принимает в качестве параметра название переменной или выражение, а возвращает строку, описывающую тип параметра.

Следующий сценарий показывает возможные значения:

```
var a = 1;      alert(typeof(a));       // number
var b = "1";    alert(typeof(b));       // string
var c = null    alert(typeof(c));       // object
var d = true;   alert(typeof(d));       // boolean
var e;          alert(typeof(e));       // undefined
var f = [];     alert(typeof(f));       // object
                alert(typeof(a + b));   // string
```

Операторы

Пустой оператор:

```
;
```

Условный оператор:

```
if (expression) operator [else operator]
if (expression) {
[} else if (expression) {]
[} else {]
}
```

Оператор множественного выбора:

```
switch (variable-or-expression){
[case const-value: operator*[break;]]
default: operator+
}
```

Параметрический цикл:

```
for (i = expr[,j = expr];i<expr[,j>expr];i++[,j--]) operator
for (i = expr[,j = expr];i<expr[,j>expr];i++[,j--]) {operator*}
```

Цикл с предусловием:

```
while (expression) operator
while (expression) {operator*}
```

Цикл с постусловием:

```
do {operator*} while (expression);
```

Операторы перехода:

```
break [label];
continue [label];
```

В следующем примере используются оба оператора в форме с метками. Оператор "continue two;" переводит выполнение к следующей итерации цикла, помеченного меткой "two". Оператор "break one;" переводит выполнение за цикл, помеченный меткой "one".

```
var a = 0;
one: while (1) {
  while (1) {
    two: while (a < 2) {
      a++;
      while (1) {
        continue two;
      }
    }
    break one;
  }
}
alert(a);
```

Результирующее значение a равно двум.

Дополнительно см. «Объекты».

Обработка исключений

Исключения обычно возникают при обращении к объектной модели документа в случае, если отсутствует искомый её элемент. Исключения возникают также при вызове функции eval в случае, если аргумент не может быть правильным образом интерпретирован.

Обработка исключений производится при помощи оператора try:

```
try {
  /* код, предположительно ведущий к исключению */
} catch(e) {
  /* обработка исключительной ситуации */
}
```

Переменная блока catch содержит краткое описание исключения.

Следующий пример показывает обработку исключения.

```
var a = 1, b = "#2", c;
try {
  c = a + eval(b);
  alert(c);
} catch(e) {
  alert("Ошибка: " + e);
}
```

Данный сценарий выводит сообщение об ошибке (разное для разных обозревателей). Заметим, что функции преобразования parseInt и parseFloat исключения не вызывают:

```
var a = 1, b = "#", c;
c = a + parseInt(b);
alert(c);
```

Сценарий выводит сообщение "NaN". Здесь NaN является обозначением «не числа» (Not a Number).

Функции

Функции — основные элементы сценария, выполняющие логически законченные последовательности действий (операторов).

Синтаксис описания функции:

```
function имя-функции(список-параметров) {/* тело функции */}
```

Тело функции содержит объявления переменных и операторы в произвольном порядке. Для преждевременного выхода из функции используется оператор return [expression]. Функция возвращает значение, если оператор return содержит необязательное выражение expression.

Пример сценария с рекурсивной функцией, вычисляющей факториал:

```
function fact(a){
  if (a < 2) return 1;
  return a * fact(a - 1);
}
alert(fact(5));
```

Сценарий выводит сообщение "120".

Функция может быть назначена переменной:

```
function fact(a){
  if (a < 2) return 1;
  return a * fact(a - 1);
}
var a = fact;
alert(typeof(fact));
alert(typeof(a));
alert(a.valueOf());
```

Этот сценарий два раза выводит сообщение "function" и полный текст функции fact. Назначить переменной функцию можно и без указания имени функции (анонимно):

```
var a = function(a) {
  return(a * a)
};
alert(a(5));
```

Данный сценарий выводит сообщение "25".

Переменные, указывающие на функции, могут быть присвоены другим переменным, которые становятся синонимами. Кроме того, такие переменные, а также имена функций могут быть аргументами функций.

В следующем примере функция f получает в качестве аргумента ссылку на функцию и аргумент, который далее используется как аргумент функции-аргумента:

```
function fact(a){
  if (a < 2) return 1;
  return a * fact(a - 1);
}
function f(a, b){
  return a(b);
}
alert(f(fact, 5));
```

Сценарий выводит сообщение "120".

Функция как объект обладает определенными свойствами и методами. Метод valueOf был рассмотрен выше. Он возвращает значение объекта (если обозреватель предоставляет таковое). Кроме этого, строковое представление объекта возвращает также метод toString (опять же если обозреватель имеет такое представление).

Свойство arguments[] возвращает коллекцию фактических аргументов функции. Рассмотрим использование коллекции аргументов для задания произвольного их числа:

```
function sort(){
  a = [];
  for (i = 0; i < arguments.length; i++) {
    a[i] = arguments[i];
  }
  return a.sort();
}
alert(sort(5, 3, 4, 2, 1));
```

Сценарий выводит сообщение "1,2,3,4,5".

Свойство caller возвращает имя функции, вызвавшей данную:

```
function f(){
  return f.caller;
}
function g(){
  return f();
}
alert(f());
alert(g());
```

Сценарий выводит сообщения "null" и "function g(){return f();}".

Встроенные свойства и функции JavaScript

Свойства

Infinity

Обозначает значение «плюс бесконечность».

NaN

Обозначает значение NaN.

undefined

Обозначает значение undefined. Оно означает, что элемент или отсутствует, или его значение не определено.

Функции

escape(строковое-выражение)

Кодирует строку-аргумент в формат URL. Символы, числовое значение которых больше 127, переводятся в форму вида "%uXXXX", где XXXX — шестнадцатеричный код символа в *Unicode*. Например, русская буква "А" будет переведена в "%u0410".

unescape(строковое-выражение)

Декодирует строку, закодированную с помощью escape.

encodeURI(строковое-выражение)

Кодирует строку в формат URI для протокола CGI. Эта функция может использоваться вместо функции escape.

decodeURI(строковое-выражение)

Декодирует строку, закодированную encodeURI.

eval(строковое-выражение)

Вычисляет выражение, записанное в строке, как если бы оно находилось в коде программы. Например, следующий сценарий выводит сообщение "15":

```
alert(eval("1 + 2 + 3 * 4"));
```

Как отмечалось, эта функция может выбросить исключение.

parseFloat(строковое-выражение)

Преобразует строку в число с плавающей запятой. Если это сделать невозможно, возвращает NaN.

parseInt(строковое-выражение[,основание])

Преобразует строку в целое число, используя указанное основание или основание 10, если оно не задано. Если преобразование выполнить невозможно, возвращает NaN.

isFinite(выражение)

Проверяет, возвращает ли выражение конечное число.

isNaN(выражение)

Возвращает true, если выражение не число.

Объекты

Объект — составной тип данных, включающий в себя множество переменных (свойств) и функций (методов и событий). Как свойства, так и методы объекта JavaScript — это именованные поля.

Рассматривая объекты, доступные JavaScript, можно выделить следующие их группы:

- объекты встроенных в JavaScript классов, таких, как Array, Date, Math, Number и т.п. Использование этих объектов определяется методами, предоставляемыми указанными классами;

- объекты объектной модели документа. Примерами таких объектов являются window, document, navigator и т.п. От встроенных объектов они отличаются тем, что их созданием занимается обозреватель. Именно эти объекты чаще всего являются целью сценария. Объектная модель рассматривается далее в соответствующих разделах;

- пользовательские объекты. Эти объекты создаются программистом для обеспечения необходимой гибкости программы сценария.

Пользовательские объекты, в свою очередь, можно поделить на две разновидности: объекты без конструктора и объекты с конструктором.

Первые нельзя создать с помощью генератора new, вторые, наоборот, создаются только с помощью new.

Объект без конструктора описывается при помощи блока {} (как и структура в Си). Следующий пример показывает создание и использование такого объекта:

```
ob = {
  id: 0,
  ar: [1, 2, 3],
  getid: function() { return this.id; },
  setid: function(a) { this.id = a; },
  getar: function(a) { return this.ar[a]; }
};
alert(ob.getid());
ob.setid(1);
alert(ob.getid());
ob.id = 2;
alert(ob.getid());
alert(ob.getar(2));
```

Объект состоит из свойств "id" и "ar" и методов "getid", "setid" и "getar". Данный сценарий выводит сообщения "0", "1", "2" и "3".

Создавать объект можно постепенно. Следующий пример показывает создание того же объекта по мере необходимости в тех или иных свойствах и методах:

```
o = {};
o.id = 1;
o.ar = [1, 2, 3];
o.getid = function() { return this.id; };
o.setid = function(a) { this.id = a; };
o.getar = function(a) { return this.ar[a]; };
```

Функция является объектом, и она может служить в качестве основы для создания пользовательского типа данных (являясь конструктором):

```
function rect(a,b,c,d){
  this.a = a;
  this.b = b;
  this.c = d;
  this.d = d;
  this.area = function(){
    return Math.abs((this.a - this.c) * (this.b - this.d))
  }
}
r=new rect(1, 1, 5, 5);
alert(r.area());
```

Данный сценарий выводит сообщение "16".

С помощью генератора new здесь можно создавать отдельные экземпляры «класса» rect. Заметим, что добавлять свойства и методы к этим экземплярам по мере необходимости можно точно так же, как и для объектов без конструктора. Это добавление ведет к изменению только конкретного объекта, но не «класса» в целом. Следующий пример (в продолжение предыдущего) демонстрирует это.

```
r1 = new rect(1, 1, 5, 5);
r2 = new rect(1, 1, 6, 6);
r1.name = "r1";
alert(r1.name);
alert(r2.name);
```

Этот сценарий выводит сообщения "r1" и "undefined".

Изменить «класс» после его определения можно при помощи свойства prototype, которое является синонимом конструктора. Следующий сценарий (в продолжение предыдущего) демонстрирует это.

```
rect.prototype.name = "rect";
alert(r2.name);
```

Сценарий выводит сообщение "rect". Здесь в «класс» добавляется новое свойство "name" с начальным значением "rect", которое и выводится методом alert.

С помощью данного приема в «класс» можно добавить не только новое свойство, но и новый метод. Следующий сценарий (в продолжение предыдущего) демонстрирует это.

```
rect.prototype.param = function(a) { return eval("this." + a); };
alert(r2.param("c"));
```

Сценарий выводит сообщение "6" (значение свойства "c").

Заметим, что данный прием позволяет изменять также некоторые встроенные классы JavaScript, однако возможности при этом значительно ограничены обозревателями.

При необходимости с помощью JavaScript можно определить иерархические структуры данных с наследованием свойств и методов.

Следующий пример демонстрирует создание иерархии типов.

```
function figure(a){
  this.name = a||"";
  this.area = 0;
}
function square(a, b){
  this.inherit = figure;
  this.inherit(a); // вызов конструктора базового "класса"
  this.area = b * b;
}
s1 = new square("s1", 1);
alert(s1.name); // выводит "s1"
```

Здесь определяется тип "figure". Тип square наследует тип figure с помощью свойства inherit (название этого свойства совершенно произвольное). Конструктор типа square вызывает конструктор типа figure.

Следует обратить внимание на оператор

```
this.name = a||"";
```

Здесь значение свойства name определяется либо аргументом функции-конструктора figure, если аргумент задан, либо принимается равным пустой строке.

Для работы с объектами используются следующие операторы.

Определить тип объекта позволяет операция instanceof. Следующий сценарий (в продолжение предыдущего) выводит "false" и "true".

```
alert(s1 instanceof figure);
alert(s1 instanceof square);
```

Определить наличие свойства или метода объекта можно с помощью операции in. Следующий сценарий (в продолжение предыдущего) выводит сообщения "true" и "false".

```
alert("name" in s1);
alert("size" in s1);
```

Перечислить все элементы объекта (свойства, методы и события) можно с помощью оператора for-in. Следующий сценарий (в продолжение предыдущего) выводит сообщения "inherit", "name" и "area".

```
for(name in s1){
  alert(name);
}
```

Заметим, что указанные операторы работают с любыми объектами.

Это дает возможность исследовать объектные модели разных обозревателей и использовать их отличительные свойства, методы и события для достижения определенной цели. Попробуйте, например, выполнить следующий сценарий в разных обозревателях:

```
for (n in document){
  document.write("document."+n+"="+eval("document."+n)+"<BR>");
}
```

Следует отметить, что все объекты JavaScript происходят от одного наиболее общего типа Object, который передает им следующие методы:

toString

Возвращает строковое представление объекта.

valueOf

Возвращает значение объекта.

Встроенные классы

Язык JavaScript имеет несколько встроенных классов. Как правило, эти классы определяют статические методы для выполнения операций с объектами общего назначения, такими, как массив, строка или дата.

Класс Array

Класс Array определяет следующие полезные методы.

concat(массив[, массив ...])

Объединяет несколько массивов в один.

Пример:

```
var a = [0, 1], b = [2, 3], c = [4], d = a.concat(b,c,[5,6]);
alert(d);
```

Сценарий выводит сообщение "0, 1, 2, 3, 4, 5, 6".

join(разделитель)

Возвращает строку — результат объединения элементов массива. По умолчанию разделителем является запятая.

Пример:

```
var a = [1, "b", false];
alert(a.join("-"));
alert(a.join());
```

Сценарий выводит сообщения "1-b-false" и "1, b, false".

reverse

Возвращает массив в обратном порядке.

Пример:

```
var a = [1, "b", false];
alert(a.reverse());
```

Сценарий выводит сообщение "false,b,1".

pop

Удаляет последний элемент массива и возвращает его.

Пример:

```
var a = [1, 2, 3];
alert(a.pop());
alert(a);
```

Сценарий выводит сообщения "3" и "1, 2".

push(список-элементов)

Добавляет элементы в массив и возвращает его длину.

Пример:

```
var a = [1, 2, 3];
alert(a.push(4, 5));
alert(a);
```

Сценарий выводит сообщения "5" и "1,2,3,4,5".

shift

Удаляет первый элемент массива и возвращает его.

Пример:

```
var a = [1, 2, 3];
alert(a.shift());
alert(a);
```

Сценарий выводит сообщения "1" и "2,3".

sort

Сортирует массив.

Пример:

```
var a = [4, 1, 5, 3, 2];
alert(a.sort());
```

Сценарий выводит сообщение "1,2,3,4,5".

slice(start, end)

Возвращает часть массива от элемента с индексом start (включительно) до элемента с индексом end (исключительно).

Пример:

```
var a = [1, 2, 3, 4, 5];
alert(a.slice(2, 4));
```

Сценарий выводит сообщение "3,4".

splice(start, number[, список-элементов])

Удаляет из массива количество элементов, равное number, начиная с элемента с индексом start. Если определен список элементов, они вставляются вместо удаленных. Возвращается список удаленных элементов. Пример:

```
var a = [1, 2, 3, 4, 5];
alert(a.splice(2, 2, "b", "c"));
alert(a);
```

Сценарий выводит сообщения "3,4" и "1,2,b,c,5".

unshift

Добавляет элемент или элементы в начало массива и возвращает новую длину массива. Метод поддерживается не всеми обозревателями.

Пример:

```
var a = [2, 3];
a.unshift(0, 1);
alert(a);
```

Сценарий выводит сообщение "4".

Класс String

Класс String предназначен для операций со строками. Важнейшие методы класса следующие:

charAt(индекс)

Возвращает символ в указанной позиции. Если задан индекс символа, превышающий длину строки, выводится «пусто».

Пример:

```
s = "АЛЬФА";
alert(s.charAt(3));
```

Сценарий выводит значение "Ф" (сценарий в кодировке Unicode).

charCodeAt(индекс)

Возвращает код символа в указанной позиции в кодировке Unicode.

Если задан индекс, превышающий длину строки, выводится «NaN».

Пример:

```
s = "АЛЬФА";
alert(s.charCodeAt(3));
```

Сценарий выводит значение "212".

fromCharCode(код[, код ...])

Возвращает символы, код которых задан в кодировке Unicode.

Пример:

```
alert(String.fromCharCode(215, 946));
```

Сценарий выводит строку "×β".

indexOf(подстрока[, старт])
lastIndexOf(подстрока[, старт])

Возвращают позицию подстроки, начиная с позиции старт, или –1. Методы чувствительны к регистру.

Пример:

```
s = "012345";
alert(s.indexOf("23"));
```

Сценарий выводит значение "2".

split(разделитель[, количество])

Преобразует строку в массив, используя указанный разделитель.

Второй параметр, если задан, определяет максимальное количество элементов массива.

Пример:

```
s = "0 1 2 3 4 5";
alert(s.split(" "));
```

Сценарий выводит "0,1,2,3,4,5".

substr(старт, количество)

Возвращает подстроку длиной "количество", начиная от символа с индексом старт.

Пример:

```
s = "012345";
alert(s.substr(2,3));
```

Сценарий выводит сообщение "234".

substring(старт[, стоп])

Возвращает подстроку, начиная от символа с индексом старт и заканчивая символом с индексом стоп (исключительно).

Пример:

```
s = "012345";
alert(s.substring(2));
alert(s.substring(2,5));
```

Сценарий выводит сообщения "2345" и "234".

toLowerCase, toUpperCase

Переводят символы в нижний и верхний регистр соответственно.

Класс Math

Класс Math реализует статические методы для вычисления математических функций, функции округления, генерирования случайного числа.

Класс Math определяет следующие константы.

E	число Эйлера (2,71828…).
LN2	натуральный логарифм числа 2 (0,693147…).
LN10	натуральный логарифм числа 10 (2,30258…).
LOG2E	логарифм E по основанию 2 (1,44269…).
LOG10E	логарифм E по основанию 10 (0,434294…).
PI	число Пи (3,14159…).
SQRT1_2	квадратный корень из числа 1/2 (0,707106…).
SQRT2	квадратный корень из числа 2 (1,41421…).

Класс Math определяет следующие математические функции:

```
abs(x)          // абсолютное значение x
acos(x)         // арккосинус x
asin(x)         // арксинус x
atan(x)         // арктангенс x
cos(x)          // косинус x
exp(x)          // E в степени x
log(x)          // логарифм x по основанию E
pow(x,y)        // x в степени y
sin(x)          // синус x
sqrt(x)         // квадратный корень из x
tan(x)          // тангенс x
```

Следующий сценарий выводит синус 30 градусов.

```
alert(Math.sin(30 * Math.PI / 180));
```

Класс Math определяет следующие функции округления.

```
ceil(x)         // округляет x до первого большего целого числа
floor(x)        // округляет x до первого меньшего целого числа
round(x)        // округляет x до ближайшего целого числа
```

Следующий сценарий выводит случайное число в диапазоне от нуля (включительно) до единицы (исключительно):

```
alert(Math.random());
```

Класс Math определяет также функции min и max с произвольным количеством аргументов.

Класс Date

Класс Date позволяет работать с датами и временем.

Время исчисляется в миллисекундах от даты 1 января 1970.

Следующие методы возвращают элементы дат.

`getDate` возвращает день месяца (1÷31) заданной даты.

Следующий сценарий показывает день текущей даты:

```
var d = new Date();
alert(d.getDate());
```

Следующий сценарий показывает день даты 31 января 2015 года:

```
var d = new Date(2015, 0, 31);
alert(d.getDate());
```

`getDay` возвращает день недели заданной даты. При этом значение 0 соответствует воскресенью, 6 — субботе.

`getFullYear` возвращает 4 цифры года заданной даты.

`getHours` возвращает час даты в диапазоне 0÷23.

`getMiliseconds` возвращает миллисекунду даты в диапазоне 0÷999.

`getMinutes` возвращает минуту даты в диапазоне 0÷59.

`getMonth` возвращает месяц даты в диапазоне 0÷11.

`getSeconds` возвращает секунду даты в диапазоне 0÷59.

`getTime` возвращает число миллисекунд даты от 1 января 1970 г.

Следующие методы устанавливают элементы дат.

```
setDate(день)                   // устанавливает день месяца
setFullYear(год)                // устанавливает год
setHours(час)                   // устанавливает час
setMilliseconds(миллисекунда)   // устанавливает миллисекунду
setMinutes(минута)              // устанавливает минуту
setMonth(месяц)                 // устанавливает месяц
setSeconds(секунда)             // устанавливает секунду
setTime(миллисекунды)  // устанавливает дату количеством миллисекунд
```

Класс определяет также аналогичные методы для всемирного времени UTC. Эти методы содержат аббревиатуру UTC в своем названии.

Модель событий обозревателя

Событие (*event*) — это уведомление о действиях пользователя или изменении состояния документа. Когда они происходят, обозреватель может выполнять действия, заданные с помощью функций или непосредственного кода JavaScript. Эти действия назначаются событиям программистом явным образом. Каждому событию соответствует некоторое обозначение (определенное слово). Для назначения действия используется это слово с префиксом "*on*", — "*on*событие".

Так нажатию и отпусканию кнопки мыши при ее неизменном положении соответствует слово "click". Действие назначается этому событию конструкцией вида "onclick = выражение_javascript". Выражение JavaScript в данном контексте — любое вычислимое выражение.

Назначение действий событиям

Назначить действие событию можно разными способами.

Первый способ — назначить действие в параметрах тега (листинг 6).

Листинг 6

```
<HTML><HEAD>
<SCRIPT>
function setredcolor(){
  document.fgColor = "red";
}
</SCRIPT>
</HEAD>
<BODY onclick="setredcolor()">
  Пример события
</BODY>
</HTML>
```

В этом примере событию назначена функция setredcolor в параметрах тега <BODY>. Функция устанавливает красный цвет текста документа.

Второй способ — назначить действие свойству "*on*событие" некоторого элемента (объекта) HTML (листинг 7).

Листинг 7

```
<HTML><HEAD>
<SCRIPT>
document.onclick = function(){ document.fgColor = "red"; };
</SCRIPT>
</HEAD>
<BODY>
  Пример события
</BODY>
</HTML>
```

В этом примере событию назначается анонимная функция. Фактически оба способа являются одинаковыми (установка свойства).

Удаление действия, назначенного свойству "*on*событие", возможно путем очистки значения этого свойства.

Недостаток описанных способов назначения действий заключается в том, что нельзя на одно событие назначить более одного действия.

Альтернативный способ назначения действий заключается в добавлении слушателя события с помощью метода addEventListener. В следующем примере событию click объекта document назначено три действия (три обработчика), которые выполняются в порядке добавления (листинг 8).

Листинг 8

```
<HTML><HEAD>
<SCRIPT>
document.onclick = function(){alert("Hello!");};
document.addEventListener("click", function(){alert("Thanks!")});
document.addEventListener("click", function(){alert("More Thanks!")};
</SCRIPT>
</HEAD>
<BODY>
  Пример события
</BODY>
</HTML>
```

Обработчик события, добавленный методом addEventListener, может быть удален методом removeEventListener. Это возможно только в случае, когда назначенная функция не является анонимной (листинг 9).

Листинг 9

```
<HTML><HEAD>
<SCRIPT>
document.onclick = function(){ alert("Hello!"); };
document.addEventListener("click", sayThanks);
function sayThanks() {
  alert("Thanks!");
  document.removeEventListener("click", sayThanks);
}
</SCRIPT>
</HEAD>
<BODY>
  Пример события
</BODY>
</HTML>
```

Здесь событию click назначено действие с помощью свойства onclick, и добавлен обработчик этого же события методом addEventListener. Обработчиком является функция sayThanks. Она удаляет обработчик, установленный методом removeEventListener, поэтому функция sayThanks срабатывает при последовательных щелчках только один раз.

В заключение заметим, что назначенное событию действие может быть любым допустимым выражением JavaScript, но современная практика применения JavaScript предполагает назначение именно *функций*.

События элементов HTML

В следующей таблице приведены события элементов HTML. Следует учитывать, что не все события происходят во всех элементах.

Таблица 1 — События элементов HTML

Событие	*Описание*	
blur	потеря фокуса элементом	1)
change	изменение значения элемента	1)
focus	получение фокуса элементом	1)
select	выделение	1)
События мыши		
click	щелчок левой кнопкой мыши	
dblclick	двойной щелчок левой кнопкой	
contextmenu	щелчок правой кнопкой мыши	
mousedown	опускание левой кнопки мыши	
mouseup	отпускание левой кнопки мыши	
mousemove	движение указателя мыши над объектом	
mouseover	появление указателя мыши над объектом	
mouseout	уход указателя мыши с объекта	
mouseenter	появление указателя мыши над объектом	1)
mouseleave	уход указателя мыши с объекта	1)
События клавиатуры		
keydown	опускание клавиши клавиатуры	
keyup	отпускание клавиши клавиатуры	
keypress	А/Ц клавиша клавиатуры нажата и отпущена	
События форм		
submit	отправление формы	1)
reset	очистка формы	1)
События загрузки (выгрузки)		
readystatechange	изменение состояния (загрузки)	1)
load	окончание загрузки	1)
unload	закрытие окна (документа)	1)
События drag and drop		
dragstart	начало перетаскивания выделения	
dragover	выделение перетаскивается над объектом	
drop	выделение отпускается над объектом	
drag	перетаскивание над источником	
dragenter	перетаскивание над элементом назначения	
dragleave	выход из элемента-назначения	
dragend	завершение операции перетаскивания	

1) Событие не всплывает.

Объект event

Объект event позволяет сценарию получить сведения о событии. Этот объект является свойством объекта window, поэтому доступен в любом месте сценария. Заметим, что этот объект определяется не для всех событий. Следующий пример показывает использование объекта event.

Листинг 10

```
<HTML><HEAD>
<SCRIPT>
function showevent(event){
  alert("type: " + event.type);
  alert("source: " + event.srcElement);
  alert("x: " + event.x);
  alert("button: " + event.button);
  . . .
}
</SCRIPT>
</HEAD>
<BODY onclick="showevent(event)">
</BODY>
</HTML>
```

Основные свойства объекта event приведены в следующей таблице.

Таблица 2 — Свойства объекта event

Свойство (*метод*)	*Описание*
type	Тип события (без префикса "on")
target srcElement	Объект источника события
currentTarget	Объект события
srcElement.tagName target.tagName	Наименование тега
x, y	Координаты события (устарело)
clientX, clientY	Клиентские координаты события
screenX, screenY	Экранные координаты события
offsetX, offsetY	Координаты события относительно контейнера
button	Нажатая кнопка мыши. Разные обозреватели возвращают разные значения. *IE* возвращает 1 для левой кнопки и 2 для правой, а *Opera* — 0 для левой кнопки, 2 для правой
keyCode	Код нажатой клавиши
altKey, ctrlKey, shiftKey	Логическое значение, признак нажатой клавиши *Alt*, *Ctrl* или *Shift* соответственно
cancelBubble (IE) stopPropagation()	Логическое значение или метод,, запрещающий (разрешающий) всплывание события
returnValue (IE) preventDefault()	Логическое значение или метод, запрещающий (разрешающий) выполнение действия по умолчанию

Всплывание событий

HTML-документ представляет собой структурированную иерархию элементов, и каждое действие, происходящее в элементе, происходит также во всех родительских элементах вплоть до верхнего уровня иерархии. Такая модель поведения называется «*всплыванием событий*».

Следующий пример показывает всплывание событий.

Листинг 11

```
<HTML><HEAD>
<SCRIPT>
function bg(event) {
  x = e.clientX;
  y = e.clientY;
  document.elementFromPoint(x,y).style.background = "red";
}
</SCRIPT>
</HEAD>
<BODY onclick="bg(event)">
  <TABLE cellspacing="8" border="1"><TBODY>
    <TR><TD>Ячейка 1</TD></TR>
    <TR><TD>Ячейка 2</TD></TR>
  </TBODY></TABLE>
</BODY></HTML>
```

Здесь функция bg закрашивает объект цветом, заданным первым параметром. Второй параметр функции — это объект event, который несет в себе информацию о событии. Метод document.elementFromPoint определяет объект, в котором событие произошло, по координатам.

Тело документа содержит таблицу из двух строк и двух ячеек, а в теге <BODY> указана реакция на событие click (щелчок), которая приводит к вызову функции bg.

Всплывание события click происходит следующим образом.

При щелчке в ячейку событие передается ее родительскому элементу. Это происходит потому, что в ячейке для события click обработка не предусмотрена. Поскольку ни для одного родительского элемента ячейки в таблице не предписано никаких действий, событие в конечном итоге передается самому верхнему объекту в данной иерархии, то есть телу документа (объекту body).

При щелчке мышью в различные части документа происходит закрашивание тех или иных элементов: ячейки, таблицы или документа в целом.

Кроме того, многие события имеют *действия по умолчанию*. Например, щелчок на гиперссылку (тег <A>) вызывает автоматический переход, если сценарий не предписывает иное действие.

Всплывание событий и действие по умолчанию являются двумя независимыми механизмами.

В следующем примере используется всплывание события click от картинки к тегу <A>. Поскольку для тега <A> предписана обработка события click, действие по умолчанию может быть разрешено или запрещено. В примере действие по умолчанию отменяется либо при помощи свойства returnValue, либо при помощи метода preventDefault.

Листинг 12

```
<HTML><HEAD>
<SCRIPT>
function cancel(e){
  try {
    e.returnValue = false;
    alert("returnValue");
  } catch(a){}
  try {
    e.preventDefault();
    alert("preventDefault");
  } catch(a){}
}
</SCRIPT>
</HEAD>
<BODY>
  <A href="a.html" onclick="cancel(event)">
    <IMG src="ok.gif"/>
  </A>
</BODY></HTML>
```

Остановить всплывание события в элементе можно при помощи свойства cancelBubble или метода stopPropagation. В следующем примере всплывание события останавливается в элементе <IMG>.

Листинг 13

```
<HTML><HEAD>
<SCRIPT>
function stop(e){
  try {
    e.cancelBubble = true;
    alert("cancelBubble");
  } catch(a){}
  try {
    e.stopPropagation();
    alert("stopPropagation");
  } catch(a){}
}
</SCRIPT>
</HEAD>
<BODY>
  <A href="x.html" onclick="alert('click')">
    <IMG src="ok.gif" onclick="stop(event)"/>
  </A>
</BODY></HTML>
```

Тестирование показывает, что в этом случае переход выполняется, хотя событие click не возникает в элементе <A>.

Функцию stop можно определить также следующим способом:

```
function stop(e){
  if (e.cancelBubble) {
    e.cancelBubble = true;
    alert("cancelBubble");
  } else {
    e.stopPropagation();
    alert("stopPropagation");
  }
}
```

Использование this

Ссылка this указывает на объект, в котором происходит событие.

Часто this передается функции обработчика как параметр. В следующем примере надпись «Переход» меняет свой цвет при движении мыши:

Листинг 14

```
<HTML><HEAD>
<SCRIPT>
function setpassive(e){ e.style.color = "blue"; }
function setactive(e){ e.style.color = "red"; }
</SCRIPT>
</HEAD>
<BODY>
<TABLE>
  <TR>
    <TD onmouseout="setpassive(this)"
        onmouseover="setactive(this)">Цвет текста
    </TD>
  </TR>
</TABLE>
</BODY>
</HTML>
```

Здесь использованы события onmouseout (мышь вышла за пределы объекта) и onmouseover (мышь над объектом). Объект передается функции, которая использует ссылку для изменения стиля.

Использовать ссылку this на объект можно также непосредственно в коде тега, например:

Листинг 15

```
<HTML><BODY>
<TABLE>
  <TR>
    <TD onmouseout="this.style.color='blue'"
        onmouseover="this.style.color='red'">Цвет текста
    </TD>
  </TR>
</TABLE>
</BODY>
</HTML>
```

Вообще, ссылка this передается функции обработчика автоматически.

В следующем примере показывается, как это использовать.

Листинг 16

```
<HTML><HEAD>
<SCRIPT>
function init() {
  var td = document.getElementById("cell");
  td.onmouseover = function(){ this.style.color="red"; }
  td.onmouseout = function(){ this.style.color="blue"; }
}
</SCRIPT>
</HEAD>
<BODY onload="init()">
<TABLE><TR><TD id="cell">Цвет текста</TD></TR></TABLE>
</BODY>
</HTML>
```

Здесь функциям обработчиков неявно передается ссылка this. Обработчики событий записываются как свойства элемента td, ссылку на который возвращает метод getElementById по окончании загрузки страницы.

Кроме того, получить ссылку на объект можно с помощью свойств события event. Следующий пример показывает это.

Листинг 17

```
<HTML><HEAD>
<SCRIPT>
function setpassive(event){ event.target.style.color="blue"; }
function setactive(event){ event.target.style.color="red"; }
</SCRIPT>
</HEAD>
<BODY>
<TABLE>
  <TR>
    <TD onmouseout="setpassive(event)"
        onmouseover="setactive(event)">Цвет текста
    </TD>
  </TR>
</TABLE>
</BODY>
</HTML>
```

Здесь используется свойство target объекта event. В обозревателе IE вместо этого свойства, вероятно, нужно использовать свойство srcElement.

Существует также свойство currentTarget объекта event. Это свойство указывает на объект, до которого событие «всплыло» (и где оно было обработано), в то время как свойство target указывает на объект, в котором событие произошло.

Объектная модель обозревателя

Язык JavaScript управляет объектами страницы HTML с помощью объектов объектной модели обозревателя (Browser Object Model, BOM).

На рисунке 2 приведена объектная модель обозревателя.

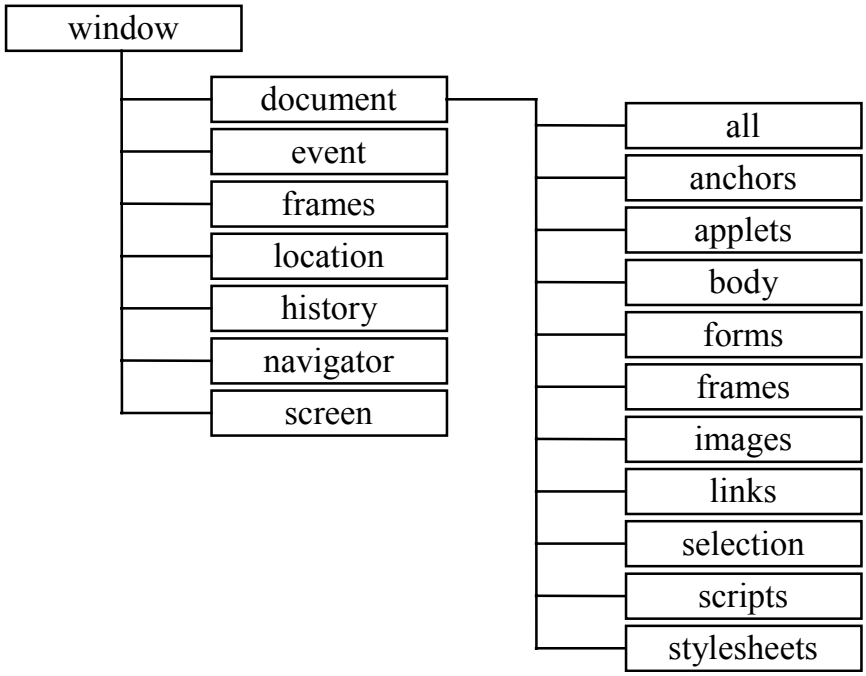

Рисунок 2 — Объектная модель обозревателя

Объект window

Это объект высшего уровня в иерархии. Для доступа к свойствам и методам текущего окна не требуется ссылка. Он предоставляет объекты, свойства и методы, часть из которых была рассмотрена ранее.

Все глобальные и необъявленные переменные являются *свойствами*, а глобальные функции являются *методами* объекта window.

Рассмотрим следующий пример (листинг 18).

Листинг 18

```
<HTML><HEAD><SCRIPT>
function A() {
  count = 0;
  B();
}
function B() {
  count++;
  alert(count);
}
</SCRIPT></HEAD>
<BODY onload="A()" onclick="B()">
</BODY></HTML>
```

Во время загрузки страницы HTML вызывается функция A, которая устанавливает значение переменной count. Переменная нигде не объявляется, поэтому она становится свойством объекта window.

После загрузки щелчок в страницу вызывает функцию B, которая увеличивает значение переменной count и выводит его во всплывающее окно сообщения. Последовательные щелчки в окно страницы будут вызывать постепенное увеличение значения переменной count.

Если вызвать функцию B до вызова функции A, произойдет исключение, — изменить значение необъявленного свойства нельзя. В этом случае возможны варианты задания начального значения свойства count.

Один из вариантов — отловить исключение с помощью блока try и задать начальное значение (листинг 19):

Листинг 19

```
<HTML><HEAD>
<SCRIPT>
function B() {
  try {
    count++;
  } catch(e) {
    count = 0;
  }
  alert(count);
}
</SCRIPT>
</HEAD>
<BODY onclick="B()">
</BODY>
</HTML>
```

Другой вариант — определить наличие (отсутствие) свойства count у объекта window и задать начальное значение (листинг 20).

Листинг 20

```
<HTML><HEAD>
<SCRIPT>
function B() {
  if (!("count" in window)) count = 0;
  count++;
  alert(count);
}
</SCRIPT>
</HEAD>
<BODY onclick="B()">
</BODY>
</HTML>
```

Следует помнить, что указанная техника применима только внутри объектной модели обозревателя. Попытка ее использования в других контекстах может привести к ошибке (язык JavaScript можно использовать не только на странице HTML).

Для взаимодействия с пользователем window предоставляет методы:

alert(строка)

Выводит сообщение строка.

prompt(подсказка, значение-по-умолчанию)

Выводит окно для ввода некоторого значения; подсказка — это поясняющий текст окна, а значение-по-умолчанию — начальное значение.

confirm(строка)

Выводит диалоговое окно для подтверждения с двумя кнопками. При нажатии на кнопку «OK» метод возвращает true, при нажатии на кнопку «Cancel» возвращается false. Пример:

```
alert(confirm("Press any button."));
```

Свойства объекта позволяют установить и прочитать строку состояния и наименование окна, определить количество фреймов в окне, положение окна на экране, управлять линейками прокрутки, размером окна и т.п.

Метод open создает новое немодальное окно (или вкладку):

open(url, name, опции, true|false)

Параметры: url — URL страницы, name — наименование окна, опции задают отображение окна (см. ниже). Четвертый параметр определяет, заменит ли новый URL текущий адрес (true) в списке или будет добавлен в его конец (false). Все параметры являются необязательными. Опции задаются в виде строки, в которой наименованию присваивается значение, опции разделяются запятыми. Возможные опции:

width, height — ширина и высота окна, left, top — положение на экране, location, menubar, toolbar, statusbar, scroolbars — управляют отображением строки адреса, меню, панелей инструментов, строки состояния и линеек прокрутки, resizeable — определяет, можно ли изменить размеры окна. Пример:

```
open("b.html", "B", "location=0,left=0,top=0,height=300,width=300");
```

Метод open возвращает ссылку на созданное окно, которая представляет объект window. Эта ссылка должна использоваться при обращении к свойствам и методам нового окна.

Метод close закрывает окно (или вкладку). При этом в *IE6* возникает диалоговое окно для подтверждения.

Объект document

Определяет структуру, содержание и стиль HTML-документа. С помощью этого объекта осуществляется доступ к элементам страницы с целью динамического изменения содержимого.

Объект обладает свойствами, задающими главные цвета страницы:

bgColor — цвет фона;
fgColor — цвет текста;
alinkColor — цвет активной ссылки;
vlinkColor — цвет посещенной ссылки;
linkColor — цвет не посещенной ссылки;

Свойство lastModified позволяют узнать дату изменения документа. Возвращаемое значение имеет разный формат в разных обозревателях.

Событие onreadstatechange возникает при изменении состояния документа или внедренного объекта. При этом свойство readyState может принимать строковые значения "uninitialized", "loading", "interactive" и "complete". Пример:

```
document.onreadystatechange = getstate;
function getstate(){ alert(document.readyState); }
```

Методы write(строка) и writeln(строка) записывают в документ новое содержимое. Используются для создания документов «на лету».

Методы open и close позволяют создавать новые немодальные окна (или вкладки) и закрывать их. При этом URL не обязательно должен содержать адрес страницы HTML. В примере открывается рисунок:

Листинг 21

```
<HTML><HEAD><SCRIPT>
function openw(){
  d = document.open("ok.gif", "", "height=40,width=40");
}
function closew(){ d.close(); }
</SCRIPT>
</HEAD>
<BODY>
<TABLE>
  <TR><TD onclick="openw()">Открыть рисунок</TD></TR>
  <TR><TD onclick="closew()">Закрыть рисунок</TD></TR>
</TABLE>
</BODY>
</HTML>
```

Получение ссылки на элемент HTML

Получить ссылку на элемент HTML можно при помощи следующих методов объекта document:

```
document.getElementById(id-value)
document.getElementsByTagName(tag-name)[index]
```

Первая функция возвращает ссылку на элемент, параметр id которого равен id-value. Вторая функция возвращает ссылку на элемент, описываемый тегом tag-name и имеющий порядковый номер index (от нуля).

В следующем примере функция init использует оба метода.

Листинг 22

```
<HTML><HEAD>
<SCRIPT>
function init() {
  var td1 = document.getElementById("cell");
  td1.onmouseover = function() { this.style.color = "red"; };
  var td2 = document.getElementsByTagName("TD")[0];
  td1.onmouseout = function() { this.style.color = "blue"; };
}
</SCRIPT>
</HEAD>
<BODY onload="init()">
<TABLE>
  <TR><TD id="cell">Цвет фона</TD></TR>
</TABLE>
</BODY>
</HTML>
```

Здесь при движении мыши меняется фон ячейки.

Следующий метод возвращает объект, задаваемый координатами:

```
document.elementFromPoint(x, y)
```

Этот метод применен в листинге 11.

Коллекции объекта document

Объектная модель DHTML представляет документ в виде набора семейств или коллекций. В следующей таблице поясняется их назначение.

Таблица 3 — Коллекции объекта document

Коллекция	*Элементы*
all	все элементы документа
anchors	<A name="...">
applets	<APPLET>, <OBJECT>
body	<BODY>
forms	<FORM>
frames	<IFRAME>
images	<IMG>
links	<A href="...">
scripts	<SCRIPT>
stylesheets	<STYLE> <LINK>

Все коллекции обладают свойством length, которое возвращает число элементов данного типа. Чтобы получить элемент коллекции, используется одна из форм (предпочтительной является последняя):

```
document.all[i]
document.all[id]
document.all(id)
document.all.id
```

Здесь i — порядковый номер элемента коллекции (его индекс), а id — параметр id или name элемента (идентификатор).

Разные обозреватели могут выстраивать разные объектные модели для одних и тех же документов, поэтому индекс можно использовать только для перечисления (перебора) элементов коллекции. Разные обозреватели по-разному воспринимают обращение к элементу через идентификатор. Например, *IE6* не воспринимает идентификатор, заданный параметром name.

Следующий пример поясняет использование свойства length и получение объекта по индексу и по идентификатору.

Листинг 23

```
<HTML><HEAD>
<SCRIPT>
function enumerate(){
  for(i = 0; i < document.all.length; i++){
    alert(document.all[i].tagName);
  }
  alert(document.all("img").tagName);
  alert(document.all.img.tagName);
}
</SCRIPT>
</HEAD>
<BODY onload="enumerate()">
<TABLE cellspacing="8" border="0">
  <TR><TD><A href="index.html">Переход</A></TD></TR>
  <TR><TD><IMG src="ok.gif" id="img"></TD></TR>
</TABLE>
</BODY>
</HTML>
```

Сценарий последовательно выводит названия всех тегов документа, после чего два раза выводит название тега IMG.

Объект location

Этот объект содержит информацию о местонахождении документа.

Основные свойства:

host, hostname — имя компьютера в сети Интернет;
href — полный URL документа;
pathname — путь и имя файла;
protocol — протокол; по умолчанию — "http:";
search — строка параметров.

Методы:

reload — перезагружает документ;
replace(url) — загружает новый документ.

Загрузить новый документ можно также, изменив значение свойства href или просто присвоив URL объекту location.

Объект navigator

Этот объект предоставляет доступ к обозревателю. Основные свойства этого объекта:

appCodeName — кодовое имя обозревателя;

appName — имя обозревателя;

appVersion — версия обозревателя и (или) операционной системы;

cookieEnabled — поддержка куки;

javaEnabled — поддержка JavaScript;

onLine — состояние подключения к сети Интернет;

platform — платформа;

userAgent — кодовая строка обозревателя, содержащая кодовое имя, имя, версию и другую информацию об обозревателе.

Заметим, что для *IE6* свойство javaEnabled является методом.

В следующей таблице приведены значения некоторых свойств, возвращаемые разными обозревателями.

Таблица 4 — Свойства объекта navigator

IE6	
appCodeName	Mozilla
appName	Microsoft Internet Explorer
appVersion	4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.04506.648; .NET CLR 3.5.21022)
userAgent	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.04506.648; .NET CLR 3.5.21022)
Opera 32	
appCodeName	Mozilla
appName	Netscape
appVersion	5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.93 Safari/537.36 OPR/32.0.1948.69
userAgent	Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.93 Safari/537.36 OPR/32.0.1948.69
Firefox 41	
appCodeName	Mozilla
appName	Netscape
appVersion	5.0 (Windows)
userAgent	Mozilla/5.0 (Windows NT 5.1; rv:41.0) Gecko/20100101 Firefox/41.0
Google Chrome 45	
appCodeName	Mozilla
appName	Netscape
appVersion	5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36
userAgent	Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36

Очевидно, что определение параметров обозревателя и среды их выполнения является не такой уж простой задачей.

Использование таймера

Объект window определяет четыре метода для управления таймерами.

```
setTimeout(выражение, интервал)
```

Устанавливает таймер, срабатывающий один раз. Возвращает идентификатор (число). Выражение — это допустимое выражение JavaScript в виде строки. Интервал задается в миллисекундах. Пример:

```
tr = setTimeout("alert('Alarm!');", 1000);
```

После обновления страницы сценарий через одну секунду выводит сообщение. В выражении могут участвовать переменные и операции, но чаще выражение задает функцию, которая вызывается по окончании установленного интервала. Если таймер требуется остановить до истечения интервала, используется следующий метод.

```
clearTimeout(tr)
```

Здесь параметр — это число, возвращаемое методом setTimeout.

Если требуется таймер, срабатывающий периодически через равные интервалы времени, то следует использовать пару методов setInterval и clearInterval, полностью аналогичные рассмотренным.

Пример установки интервального таймера:

```
tr = setInterval("somefunc('someargument');", 100);
```

При установке времени срабатывания, равным нулю, интервал будет примерно 4 мс, и это минимальный интервал.

Взаимодействие с сервером

Для интерактивного взаимодействия с сервером современные обозреватели предоставляют возможность выполнения запросов во время выполнения HTML-страниц. Для этой цели используется *объект запроса* XMLHttpRequest (называемый иногда просто XHR), который для разных обозревателей можно получить разными способами. Данная технология называется AJAX (*Asynchronous Javascript And Xml*).

Следующая функция показывает примерную схему получения этого объекта.

```
function getRO(){ var a = null;
  try { a = new XMLHttpRequest();                    } catch(e) {
  try { a = new ActiveXObject("MSXML2.XMLHTTP.3.0"); } catch(e) {
  try { a = new ActiveXObject("MSXML2.XMLHTTP");     } catch(e) {
  try { a = new ActiveXObject("Microsoft.XMLHTTP");  } catch(e) {
    a = null;
  }}}} return a;
}
```

Здесь последовательно проверяются возможные варианты получения объекта до тех пор, пока один из них не сработает.

Так, первый вариант работает во всех проверенных обозревателях (таблица 4), за исключением *IE6*, который возвращает объект в любом другом варианте.

Функция, формирующая и выполняющая запрос, может иметь разный вид в зависимости от способа выполнения запроса, — синхронный или асинхронный.

Синхронный запрос останавливает выполнение страницы и ожидает ответа сервера, после чего эта функция обрабатывает результат.

Асинхронный запрос не останавливает выполнение страницы, которая остается интерактивной. В этом случае по окончании запроса результат возвращается в обратно-вызываемую (*callback*) функцию асинхронно с выполнением страницы.

На способ выполнения запроса указывает третий параметр метода open. Значение false соответствует синхронному выполнению, а значение true — асинхронному.

Синхронное выполнение запроса

При синхронном выполнении функция запроса может иметь примерно следующий вид.

```
var inproc = false;
function request(){ var s;
  // запрос уже выполняется?
  if (inproc) return;
  // получаем объект запроса
  var ro = getRO();
  if (ro == null) return;
  // признак входа в запрос
  inproc = true;
  // открываем соединение
  ro.open("GET", "test.php?a=b", false);
  // отсылаем запрос
  ro.send();
  // ждем ответ
  if (ro.status == 200){
    s = ro.responseText;
    // операции, связанные с ответом s
  } else {
    // операции, связанные с ошибкой запроса ro.status
  }
  inproc = false;
}
```

Сначала проверяется признак выполнения запроса inproc, и если он равен лжи, выполнение функции продолжается, а признак inproc получает значение истины. Затем формируется объект запроса ro и проверяется, что он создан. Далее устанавливается соединение методом open, и запрос отправляется методом send.

Если параметры запроса указываются в строке запроса во втором параметре метода open (в примере это строка a=b в строке запроса), то дополнительных действий не требуется.

Далее запрос выполняется, а функция запроса ждет ответ (завершение выполнения метода send).

Если запрос выполнился удачно, свойство status имеет значение 200, и можно получить ответ, в примере с помощью свойства responseText.

Если значение свойства status после возврата из метода send не равно 200, запрос не был выполнен по каким-то причинам (например, отсутствует запрашиваемый сценарий), и нужно выполнить действия, связанные с ошибкой, на которую указывает значение свойства status.

В завершение признак выполнения запроса очищается.

Асинхронное выполнение запроса

Гораздо чаще используется асинхронный запрос. Асинхронные запросы можно выполнять параллельно с выполнением страницы и выполнением других запросов. В этом случае выполнение запроса в целом разбивается на две фазы, — отправление запроса и ожидание ответа, — соответствующие двум функциям.

Функция асинхронного запроса может иметь следующий вид.

```
var ro = null, tr = 0;
function request(){
  // запрос уже выполняется?
  if (tr != 0) return;
  // нет - получаем объект запроса
  ro = getRO();
  if (ro == null) return;
  // устанавливаем таймер таймаута
  tr = setTimeout("clear();", 10000);
  // открываем соединение
  ro.open("GET", "test.php?a=b", true);
  // устанавливаем функцию ответа
  ro.onreadystatechange = answer;
  // отсылаем запрос
  ro.send();
}
```

Функция сначала проверяет, выполняется запрос или нет. Для этой цели можно использовать признак выполнения запроса inproc, как в примере синхронного запроса. Здесь для этой цели использована переменная tr, указывающая на установку таймера таймаута.

Если запрос не выполняется, функция устанавливает этот таймер на время примерно 10 секунд, по истечении которого вызывается функция очистки, условно названная clear.

Запрос открывается, но перед отправкой запроса сначала устанавливается функция ответа answer, в которую вернется результат.

Функция ответа может иметь следующий вид.

```
function answer(){ var s;
  if (tr == 0) {
    // операции, связанные с таймаутом
    return;
  }
  // проверяем стадию выполнения запроса
  if (ro.readyState != 4) return;
  // проверяем результат выполнения запроса
  if (ro.status == 200){
    s = ro.responseText;
    // операции, связанные с ответом s
  } else {
    // операции, связанные с ошибкой запроса ro.status
  }
  clear();
}
```

Функция ответа проверяет истечение таймаута с помощью переменной tr. Если время ожидания истекло, до выполнения функции answer была выполнена функция clear, и переменная tr равна нулю.

Далее проверяется стадия выполнения запроса при помощи свойства readyState. По ходу выполнения запроса функция ответа может вызываться несколько раз с разными значениями этого свойства:

1 — установлено подключение к серверу;

2 — запрос получен;

3 — обработка запроса, можно получить часть данных;

4 — окончание выполнение запроса.

Далее проверяется результат запроса и выполняются операции, связанные с ответом, примерно так же, как в синхронном запросе. В конце вызывается функция очистки clear.

Вспомогательная функция clear имеет примерно следующий вид.

```
function clear(){
  // останавливаем таймер в случае нормального выполнения
  clearTimeout(tr);
  // очищаем признак выполнения запроса
  tr = 0;
  // объект запроса должен существовать
  if (ro == null) return;
  // останавливаем запрос в случае таймаута
  ro.abort();
  // удаляем объект запроса
  delete ro;
  // очищаем объект запроса
  ro = null;
}
```

Она выключает таймер, очищает переменную tr, снимает запрос с помощью метода abort и удаляет объект запроса.

Использовать таймер для обнаружения таймаута не обязательно. В этом случае код, связанный с таймером, можно удалить, а для указания на выполнение запроса использовать признак inproc.

Передача параметров

Параметры запроса передаются методами GET или POST.

При передаче методом GET параметры указываются в строке запроса в форме "имя=значение", отдельные пары разделяются знаком "&".

При передаче параметром методом POST параметры в таком же формате передаются методу send.

Если требуется передать POST-параметры, то в методе open нужно указать метод POST, сформировать заголовок запроса, например

```
w = "application/x-www-form-urlencoded; charset=utf-8";
ro.setRequestHeader("Content-Type",w);
```

и передать POST-параметры методу send, например:

```
ro.send("param1=value1&param1=value2");
```

При передаче параметров методом GET следует учитывать, что передаваемые данные должны содержать символов первой половины кодовой страницы ASCII, поэтому передаваемые значения нужно кодировать с помощью метода encodeURIComponent, например:

```
ro.open("GET", "test.php?name=" + encodeURIComponent('Петя'), true);
```

Получение ответа

Получить ответ сервера можно с помощью одного из двух методов:

responseText — возвращает ответ в виде текста;

responseXML — возвращает ответ в виде объекта XML документа.

Первым методом можно получить данные, передаваемые в формате HTML, JSON и XML (в виде текста).

Данные в формате HTML можно использовать для замены части HTML текста страницы. При этом нужно учитывать, что передаваемый текст должен быть закодирован на стороне сервера с помощью функции urlEncode, и декодирован на стороне клиента с помощью метода decodeURIComponent.

Данные в формате JSON (*JavaScript Object Notation*) удобны для передачи данных, таких, как объекты и массивы, в том числе массивы объектов. Пример записи в формате JSON:

```
{"name":"Петя","age":25}
```

Здесь описан простой объект, состоящий из полей "name" и "age".

На стороне сервера запись в формате JSON также должна быть закодирована. Пример файла PHP, формирующего запись JSON:

```
<?php # ответ в формате json
echo urlEncode('{"name":"Петя","age":20}');
?>
```

На стороне клиента ответ можно обработать следующим образом.

```
  if (ro.status == 200) {
    s = decodeURIComponent(ro.responseText);
    o = eval('('+ s +')');
    s = o.name + "<BR>" + o.age;
    document.all.rjson.innerHTML = s;
  }
```

Сначала ответ декодируется методом decodeURIComponent, а затем преобразуется в объект JavaScript методом eval. Ответ должен быть заключен в скобки, поэтому конструкция кажется сложной. После преобразования можно использовать свойства объекта для формирования элементов страницы. Приведенный вариант обработки объекта JSON не является единственно возможным, но он достаточно простой.

Чтобы получить данные в виде объекта XML, нужно либо запрашивать на сервере XML-документ, либо формировать его с помощью, например, PHP, при этом должен посылаться заголовок, указывающий на содержание ответа в формате XML. Пример PHP файла, формирующего документ XML (кодирование в формат URL не требуется):

```
<?php # ответ в формате xml
header("Content-type: application/xml");
echo "<?xml version='1.0' encoding='utf-8'?>
<object>
  <name>Петя</name>
  <age>20</age>
</object>";
?>
```

Важно, чтобы первой строкой ответа был строка "<?xml ... ?>".

Обработка на стороне клиента может быть следующей.

```
  if (ro.status == 200) {
    s = ro.responseXML;
    r = s.getElementsByTagName ("name")[0].textContent;
    r += "<BR>" + s.getElementsByTagName ("age")[0].textContent;
    document.all.rxml.innerHTML = r;
  }
```

Данные здесь извлекаются из объектной модели XML-документа.

Приведенные варианты описывают один и тот же объект и формируют один и тот же результат.

Рекомендуемая литература

1. Глушаков С. В. Программирование web-страниц / С. В. Глушаков, И. А. Жакин, Т. С. Хачиров; Худож.-оформ. А. С. Юхтман. — М.: ООО «Издательство АСТ»; Харьков: «Фолио», 2003. —387 с.

2. Будилов В. А. Основы программирования для Интернета. — СПб.: БХВ-Петербург, 2003. —736 с.: ил.

3. Х.М. Дейтел, П.Дж. Дейтел, Т.Р. Нието, Т.М. Лин, П. Садху. Как программировать на XML. Пер. с англ. — М.: ЗАО «Издательство БИНОМ», 2001 г. — 944 с.: ил.

4. Котеров Д.В., Костарев А.Ф PHP 5. — СПб.: БХВ-Петербург, 2005. — 1120 с.: ил.

5. (http://javascript.ru/).
Проверено 01.01.2015.

6) (http://ruseller.com/lessons.php?rub=28&id=1212).
Проверено 01.01.2015.