

Введение в JavaScript

Добавление сценария JavaScript на страницу:

Сценарии могут размещаться в четырех различных частях документа HTML:

1. В теле документа между тегами `<body>` и `</body>`. В этом случае результат сценария отображается на web-странице при её загрузке в браузере.
2. В заголовке страницы между тегами `<head>` и `</head>`. Сценарий, размещенный в заголовке, не выполняется сразу же при загрузке, а используется другими сценариями.
3. В теге HTML. Такая конструкция называется обработчиком событий и позволяет выполнять сценарий вместе с дескриптором.
4. В отдельном файле. JavaScript позволяет создавать собственные файлы с расширением `.js`, содержащие готовые сценарии. Внешний файл со сценарием JavaScript к html документу можно присоединить следующим образом:

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

где `script.js` – имя файла со сценарием, данная конструкция должна находиться внутри дескрипторов `<head>` и `</head>`.

Начало сценария JavaScript

Сценарий JavaScript обозначается парой дескрипторов `<script>` и `</script>` с указанием типа `<text/javascript>`:

```
<script type="text/javascript">
</script>
```

Выявление и устранение ошибок в сценариях

Если ваш сценарий выполняется с ошибками, то для их поиска очень удобно использовать расширение Firebug для браузера Mozilla Firefox.

Оператор вывода текста

Для вывода текста на страницу применяется оператор

```
document.write()
```

Простейший сценарий JavaScript

Создадим простейший сценарий JavaScript, который будет выводить в браузере текст «Hello, World»:

```
<html>
<head><title>простейший сценарий</title></head>
<body>

<script type="text/javascript">
document.write("Hello, World");
</script>

</body>
</html>
```

Комментарии в JavaScript

Для того чтобы вставить комментарий в сценарий JavaScript, перед ним необходимо указать `//`. Строки, начинающиеся с данной конструкции, сценарием пропускаются. Также JavaScript поддерживает C-подобные комментарии, начинающиеся с `/*` и заканчивающиеся `*/`. Эти комментарии могут занимать больше, чем одну строку:

```
<script type="text/javascript">
// это комментарий
/* а этот комментарий состоит
из нескольких строк, здесь можно писать все, что угодно */
</script>
```

Переменные в JavaScript

Существует несколько правил определения имени переменных:

- Имена переменных могут содержать все буквы алфавита, как строчные, так и прописные, а также цифры (0-9) и символ подчеркивания.
- Имя переменной должно начинаться с буквы или символа подчеркивания.
- В имени переменных различаются регистры букв. Например, переменные `totalnum`, `Totalnum` и `TotalNum` в JavaScript различаются и могут принимать различные значения.
- Не существует официального ограничения на длину имени переменной, но оно должно располагаться в одной строке кода программы.

Переменным можно присваивать различные значения – числа или текст:

```
a = 12.34;
strg = "текст";
```

Использование функций

Функции – это группа операторов JavaScript, которые выполняются как одно целое. Перед тем как использовать функцию, её необходимо определить:

```
function hello() {
    document.write("Hello, World");
}
```

Чтобы сделать функцию более гибкой, используются *параметры*, или *аргументы*. Это переменные, которые запрашиваются функцией при каждом её вызове:

```
function hello(who) {
    document.write("Hello," + who);
}
```

Для того чтобы использовать функцию, нужно её вызвать. Для вызова функции в качестве оператора необходимо указать её имя. В скобках после названия функции указывают параметры и значения:

```
hello("Fred")
```

Традиционно функции в документе HTML определяются в области заголовка. Поскольку область заголовка определяется самой первой в документе, то функция определяется до ее использования:

```
<html>
<head>
<title>Вызов функции</title>

<script type="text/javascript">
function hello(who) {
    document.write("Hello," + who + "<br>");
}
</script>

</head>
<body>

<script type="text/javascript">
hello("Fred");
hello("Dick");
</script>

</body>
</html>
```

При открытии данного документа в браузере, на экране получим:

```
Hello, Fred
Hello, Dick
```

Функции также могут возвращать в сценарий определенные значения, которые получаются в результате её вызова. Для этого нужно указать ключевое слово `return`:

```
<html>
<head>
<title>Вычисление среднего значения</title>

<script type="text/javascript">
function Average(a,b,c,d) {
    result = (a+b+c+d)/4;
    return result;
}
</script>

</head>
<body>

<script type="text/javascript">
score = Average(3,4,5,6);
document.write(score);
</script>

</body>
</html>
```

В данном примере мы передаем функции четыре числа: 3, 4, 5, 6. Функция вычисляет среднее значение, которое мы заносим в переменную `score`. Затем мы данную переменную выводим на экран.

Глобальные и локальные переменные

Существуют два типа переменных:

- *Глобальные переменные.* Используются во всем сценарии (и других сценариях одного и того же документа HTML).
- *Локальные переменные.* Используются только как аргументы одной функции. Они применяются только в той функции, в которой были созданы.

Чтобы создать глобальную переменную её следует объявить в главном сценарии, а не в функции. Для объявления переменной используется ключевое слово `var`:

```
var students = 25;
```

Если этот оператор использовать вне функции, то будет создана глобальная переменная, если же внутри – то локальная. Иногда ключевое слово `var` вводить не обязательно. Следующий пример равносителен предыдущему:

```
students = 25;
```

Чтобы более наглядно объяснить используемые в JavaScript типы данных и методы объявления переменных, рассмотрим следующий пример:

```
<html>
<head>
<title>Локальные и глобальные переменные</title>
  <script type="text/javascript">
    var name1 = "Fred";
    var name2 = "Dick";

    function hello(who) {
      document.write("Hello, " + who + "<br>");
      var name2 = "Alex";
    }
  </script>
</head>
<body>

<script type="text/javascript">
  hello(name1);
  hello(name2);
</script>

</body>
</html>
```

В данном сценарии на строках 5 и 6 определяются две глобальные переменные – `name1` и `name2`, на строке 10 определяется локальная переменная `name2`, она является локальной потому, что при её задании использовали оператор `var`, таким образом, данная переменная не пересекается с глобальной переменной `name2`, заданной на строке 6 и на экране получим:

```
Hello,Fred
Hello,Dick
```

Типы данных в JavaScript

В JavaScript используются следующие основные типы данных:

- *Числовой*. Например, 3, 25 или 3.14529.
- *Булев*, или логический. Принимает два значения: `true` (правда) или `false` (ложь)
- *Строковый*. Например, «Hello, world». Он состоит из одного или многих текстовых символов.
- *Нулевой*. Определяется ключевым словом `null`. Это значение принимает неопределенная переменная. Например, оператор `document.write(fig)` принимает это значение, если переменная `fig` раньше не определялась.

Операции над числовыми типами данных

присваивание	<code>a = 20;</code> <code>b = -1.424;</code>
сложение	<code>c = a + b;</code>
вычитание	<code>c = a - b;</code>
умножение	<code>c = a*b;</code>
деление	<code>c = a/b;</code>
увеличение на определенное значение	<code>d += 5;</code> (эквивалентно записи <code>d = d + 5;</code>)
уменьшение на определенное значение	<code>d -= 5;</code> (эквивалентно записи <code>d = d - 5;</code>)
увеличение на 1	<code>d ++;</code>
уменьшение на 1	<code>d --;</code>

Операторы `++` и `--` можно использовать и перед именем переменной. Например `++d`. Разница заключается в моменте выполнения операции приращения:

```
d = 40;
document.write(++d);
d = 40;
document.write(d++);
```

Будут выведены на экран числа 41 и 40.

Операции над строковыми типами данных

присваивание	<code>strg1 = "Hello, ";</code> <code>strg2 = "World";</code>
объединение	<code>strg = strg1 + strg2;</code>
добавление текста	<code>strg += "world";</code>

Определение длины строки

Длина строки определяется свойством `length`. Чтобы использовать это свойство в сценарии необходимо ввести название объекта, а после него `.length`:

```
strg = "Hello, World";
document.write("длина строки " + strg.length + " символов")
```

Изменение регистра текста

Для изменения регистра символов текста используется два метода:

- `toUpperCase()`. Преобразует символы текста в прописные.
- `toLowerCase()`. Преобразует символы текста в строчные.

Например, при такой записи на экране все символы будут строчными:

```
test = "Hello, World"
document.write(test.toLowerCase());
```

При этом значение переменной `test` не изменилось. Для того чтобы изменить значение этой переменной, например, перевести все символы в нижний регистр, необходимо:

```
test = test.toLowerCase();
```

Подстроковые переменные

При частом использовании строковых объектов возникает необходимость разбивки их значений на отдельные значения, сохраняемые в подстроковых переменных. Для получения части строковой переменной используется метод `substring()`, а метод `charA` применяется для возвращения отдельного её символа.

Метод `substring()`, возвращает часть значения строкового объекта, определенного двумя индексами, указанными в скобках. Для примера, приведем оператор, который используется для отображения 4-6 символов значения переменной `test`:

```
document.write(test.substring(3,6));
```

Данные числа 3 и 6 указаны соответственно следующим правилам:

- Индексирование текста начинается с 0. Поэтому четвертый символ будет иметь индекс 3.
- Второй индекс определяется исключительно. Поскольку шестой символ имеет индекс 5, то в скобках указывается индекс 6.
- Оба индекса указываются в произвольном порядке. В нашем примере сначала указан меньший индекс. Вариант (6,3) приведет к тому же результату.

Приведем еще один пример. Пусть английскому алфавиту определяется переменная `alpha`:

```
alpha = "ABCDEFGHJKLMNOPQRSTUVWXYZ"
```

Тогда следующие операторы вернут значения:

<code>alpha.substring(0,4)</code>	ABCD
<code>alpha.substring(10,12)</code>	KL
<code>alpha.substring(12,10)</code>	KL
<code>alpha.substring(6,7)</code>	G
<code>alpha.substring(24,26)</code>	YZ
<code>alpha.substring(0,26)</code>	весь алфавит
<code>alpha.substring(6,6)</code>	нулевое значение (пустая строка)

Возвращение одного символа

Метод `charA` применяется для возвращения отдельного символа строки. Индексирование значения строкового объекта начинается с 0.

<code>alpha.charA(0)</code>	A
<code>alpha.charA(12)</code>	M
<code>alpha.charA(25)</code>	Z
<code>alpha.charA(27)</code>	нулевое значение (пустая строка)

Поиск подстроковой переменной

Для отыскания строковых переменных по содержащемуся в них тексту используется метод `indexOf`. В приведенном ниже примере в значении объекта `test` ищется текст «hello»:

```
location = test.indexOf("hello");
```

Значение, которое принимает переменная `location`, – это диапазон индексов символов строки, которые составляют данное слово.

При необходимости допускается добавлять в качестве условия поиска и начальный индекс, с которого проводится поиск текста. Например, следующее выражение позволяет отыскать слово «рыба» в значении строкового объекта `temp`, начиная с 20-го символа:

```
location = test.indexOf("рыба",19);
```

Второй метод, `lastIndexOf`, выполняется подобным образом, но он возвращает индекс *последнего* найденного текстового фрагмента в значении строковой переменной:

```
location = test.lastIndexOf("hello");
```

Как и в первом случае, второй параметр используется для задания начального индекса поиска (поиск производится в направлении убывания индексов, а не увеличения).

Использование массивов

Массив – это набор элементов, содержащих значения, сохраненный под одним именем. Массивы могут состоять из чисел, строковых переменных, объектов и других типов данных.

В отличие от большинства используемых в JavaScript типов данных, массивы необходимо объявлять перед использованием:

```
scores = var Array(30);
```

В данном случае объявили массив, состоящий из 30 элементов. Индексирование элементов массива начинается с 0, поэтому элементы объявленного выше массива имеют индексы 0–29. Следующие операторы определяют значения первых четырех элементов массива:

```
scores[0] = 39;  
scores[1] = 40;  
scores[2] = 100;  
scores[3] = 49;
```

Существует более быстрый способ создания массивов, для этого все элементы массива перечисляются в одной строке:

```
var day_of_week = new Array("Понедельник", "Вторник", "Среда", "Четверг",  
                            "Пятница", "Суббота", "Воскресенье");
```

Подобно строковым переменным массивы имеют свойство `length`. Оно определяет количество элементов, из которых состоит массив:

```
document.write(scores.length);
```

Разделение строковой переменной

JavaScript содержит метод `split()`, позволяющий разделять строку на составные части. Для того чтобы правильно её использовать, укажите необходимый объект и символ, по которому производится разделение:

```
test = "Виталий";  
parts = test.split("и");
```

После выполнения этого метода массив определен следующим образом:

```
parts[0] = "В";  
parts[1] = "тал";  
parts[2] = "й";
```

Оператор `join()` выполняет обратное действие – объединяет элементы массива `parts` в одну строку:

```
fullname = parts.join("и");
```

В результате получим исходное имя «Виталий». Если символ объединения указывать нет необходимости, обязательно введите в качестве параметра двойные кавычки – `""`.

Сортировка элементов массива

JavaScript содержит метод `sort()`, используемый для сортировки элементов массива. Он возвращает упорядоченную версию исходного массива. Упорядочивание происходит как по алфавиту (для строковых значений), так и по возрастанию или убыванию (для числовых значений). Например:

```
names[0] = "Public";
names[1] = "Tillman";
names[2] = "Clinton";
names[3] = "Mouse";
sortednames = names.sort();
```

Последний оператор создает в массиве `sortednames` упорядоченные по алфавиту элементы массива `names`.

Условные операторы

Оператор *if*

Оператор `if` управляет последовательностью выполнения команд и позволяет выбрать и запустить на выполнение одну из альтернативных групп операторов.

```
if (условие) {
    оператор_1
} else {
    оператор_2
}
```

Если условие принимает значение `true`, то выполняется оператор_1, иначе выполняется оператор_2. Простой пример:

```
var a = 1;
if (a == 1) {
    document.write("переменная a равна 1");
} else {
    document.write("переменная a не равна 1");
}
```

В данном случае сравнивается переменная `a` и число `1`. Между двумя сравниваемыми операторами вводится *условный оператор*. Этот оператор задает условие, которому должны удовлетворять оба значения. Ниже приведены все используемые в JavaScript условные операторы:

<code>==</code>	равно
<code>!=</code>	не равно
<code><</code>	меньше
<code>></code>	больше
<code><=</code>	меньше или равно
<code>=></code>	больше или равно

Также можно в одном операторе `if` проверять сразу несколько условий, для этого их нужно объединять с помощью логического оператора ИЛИ (`||`) и логического оператора И (`&&`). Например:

```
If (phone == "" || email == "") {document.write("ошибка!");}
```

В данном случае проверяются переменные `phone` и `email` и если одна из них не определена или неопределены обе, то появляется сообщение об ошибке.

Использование условных выражений

В дополнение к оператору `if` в языке JavaScript введена дополнительная конструкция, так называемое условное выражение:

```
Переменная = (условие) ? если выполняется : если не выполняется;
```

Это выражение позволяет определить переменной одно из двух значений, например:

```
value = (a == 1) ? 1 : 0;
```

Также приведем его аналог с помощью оператора `if`:

```
if (a == 1) value = 1;
else value = 0;
```

Примечание

Если в условном операторе `if` используется только один оператор (в предыдущем примере `value = 1`), то фигурные скобки добавлять не нужно.

Оператор `switch`

Условный оператор `switch` позволяет задавать сразу целый блок условий:

```
switch(location) {
  case "page1" :
    document.write("это первая страница");
    break;
  case "page2" :
    document.write("это вторая страница");
    break;
  case "page3" :
    document.write("это третья страница");
    break;
  default :
    document.write("страница не известна");
}
```

В данном случае, если переменная `location` принимает значение `page1`, то появляется сообщение «это первая страница», если значение `page2` – сообщение «это вторая страница», если не один из операторов `case` не содержит правильных значений, то выполняется оператор `default` и появляется сообщение «страница не известна». Оператор `break` используется для досрочного выхода из условного оператора `switch`.

Использование циклов

Цикл `for`

Оператор цикла `for` выглядит следующим образом:

```
for (i = 1; i < 10; i++) {
  тело цикла
}
```

Здесь:

- Первый параметр (например, `i = 1`) определяет счетчик и указывает его начальное значение
- Второй параметр (`i < 10`) – это условие, которое должно быть справедливым, чтобы цикл выполнялся.
- Третий параметр (`i++`) – увеличивает счетчик на единицу

Примечание

Если в теле оператора цикла `for` стоит только один оператор, то фигурные скобки добавлять не нужно.

Цикл *while*

В отличие от циклов *for*, цикл *while* не требует использование счетчиков, они выполняются до тех пор, пока выполняется указанное условие:

```
while(total < 10) {
  n++;
  total += values[n]
}
```

Примечание

Если условие не выполняется вообще, то и цикл *while* выполняться тоже не будет.

Цикл *do...while*

Этот тип циклов сильно похож на циклы *while*, единственная разница заключается в расположении условия:

```
do {
  n++;
  total += values[n]
}
while(total < 10);
```

Таким образом, условие выполняется в конце листинга, это означает, что этот цикл будет выполнен по меньшей мере один раз.

Прерывание цикла

Выход из циклов осуществляется при выполнении определенного условия, однако, если иногда нужно завершить цикл досрочно, для этого есть специальный оператор – *break*:

```
while(total < 10) {
  n++;
  total += values[n];
  if (values[n] == 0) {break;}
}
```

В данном случае, если переменная *values[n]* принимает нулевое значение, то выполнение цикла прерывается.

Продолжение выполнения цикла

Для прерывания одной операции текущей итерации цикла и продолжения их выполнения со следующей итерации используется оператор *continue*:

```
for (I = 1; i < 10; i++) {
  if (score[i] == 0) continue;
  document.write("Номер студента ",i," оценка: ",score[i])
}
```

В данном случае, если *score[i]* принимает нулевое значение, то данная переменная на экран не выводится.

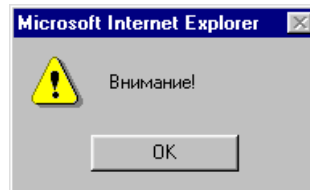
Стандартные окна сообщений

С помощью JavaScript можно выводить стандартные окна для ввода/вывода информации.

- **Alert** (Предупреждение) – служит для вывода информации. Данное окно вызывается с помощью сценария:

```
<script type="text/javascript">
  alert("Внимание!");
</script>
```

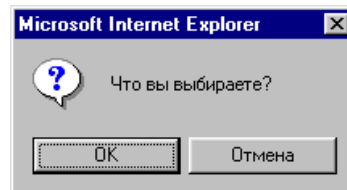
Его вид в Internet Explorer:



- **Confirm** (Подтверждение) – предназначено для вывода информации и позволяет пользователю сделать выбор в формате ответа Да/Нет на вопрос. Данное окно вызывается с помощью сценария:

```
<script type="text/javascript">
  confirm("Что вы выбираете?");
</script>
```

Его вид в Internet Explorer:



- **Prompt** (Запрос) – служит для вывода информации и позволяет пользователю ввести ответ с клавиатуры. Данное окно вызывается с помощью сценария:

```
<script type="text/javascript">
  prompt("Как вас зовут?");
</script>
```

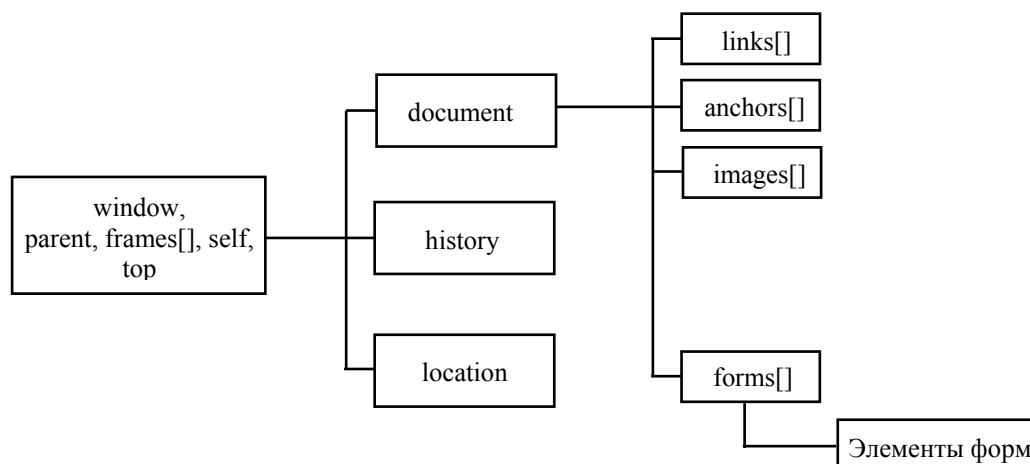
Его вид в Internet Explorer:



JavaScript: Объектная модель документа

С помощью JavaScript можно управлять внешним видом web-страницы и поведением браузера. Для этого в JavaScript включена специальная иерархическая система – объектная модель документа (DOM – Document Object Model).

Иерархическая структура объектов DOM в JavaScript



На этой блок-схеме представлены только основные объекты браузера.

Объекты window

Иерархическая структура объектов браузера начинается с объекта `window`. Он представляет окно браузера.

- Свойство `window.status` используется для изменения вида строки состояния.
- Методы `window.alert`, `window.confirm` и `window.prompt` позволяют отображать диалоговые окна с разными запросами.

Одновременно можно использовать несколько объектов `window`, каждый для открытого окна браузера. Фреймы также представляются объектами `window`.

Управление web-документами

Объект `document` представляет объект web-документа или web-страницы. Документы или страницы отображаются в окне браузера, поэтому объект `document` дочерний по отношению к объекту `window`. Изменения, проведенные с помощью объекта `document`, будут отображаться в окне браузера, а поэтому сказываться на объекте `window`.

Использование нескольких окон браузера или фреймов подразумевает существование нескольких объектов `window` с одним дочерним объектом `document`.

Получение информации о браузере

Некоторые свойства объекта `document` используются для получения сведений о браузере и текущем документе.

- Свойство URL используется для определения адреса текущей web-страницы. Адрес вводится одним словом. Изменить это свойство нельзя. Если необходимо открыть в окне браузера другую страницу, нужно использовать объект window.location.
- Свойство title содержит заголовок текущей страницы, определенный после дескриптора <title>.
- Свойство referrer определяет адрес web-страницы, просматриваемой до отображения текущего web-документа. Как правило, на ней есть ссылка на текущую страницу.
- Свойство lastModified содержит дату последнего изменения web-страницы.

Ниже приведен пример web-документа, в котором отображается дата его последнего изменения:

```
<html>
<head><title>дата последнего изменения</title></head>
<body>

Эта страница последний раз была изменена:
<script>
document.write(document.lastModified);
</script>

</body>
</html>
```

Использование ссылок

Еще один дочерний объект объекта document – это link. В одном объекте document одновременно может существовать несколько объектов link. Каждый из них содержит сведения о ссылке на другую страницу или анкер.

Объект links управляется с помощью массива links. Каждый элемент массива представляет собой объект link текущей страницы. Свойство массива document.links.length определяет количество ссылок на странице.

Следующий пример демонстрирует работу с объектом links:

```
<html>
<head><title>объект links</title></head>
<body>

<a href="old.htm">эта ссылка переопределена на "new.htm"</a><br>
<a href="page.htm">ссылка 2</a><br>

<script type="text/javascript">
document.links[0].href = "new.htm";
document.write("ссылка 2 ведет на страницу: "+document.links[1].href);
document.write("<br>на странице всего ссылок: "+document.links.length);
</script>

</body>
</html>
```

Подобно ссылкам, анкеры управляются с помощью массива anchors. Каждый элемент этого массива – отдельный объект anchor. Свойство document.anchors.length определяет количество анкеров документа HTML.

Примечание.

Анкер – это именованный элемент документа HTML, переход к которому осуществляется мгновенно. Он определяется следующим дескриптором: . Ссылка же на него задается как: .

Получение сведений о работе браузера

Объект history содержит сведения о страницах, которые отображались и отображаются в окне браузера, а также содержит методы перехода к ним.

Объект history обладает четырьмя свойствами.

- history.length. Указывает длину списка адресов посещаемых страниц, сохраняемых в объекте history. Другими словами, это количество посещенных за текущий сеанс web-страниц.
- history.current. Содержит одно значение – адрес URL текущей страницы, открытой в браузере.
- history.next. Тоже содержит одно значение – адрес URL страницы, к которой перейдет пользователь, если щелкнет на панели инструментов браузера на кнопке Forward (Вперед). Поскольку эта кнопка не активна до тех пор, пока хотя бы один раз не воспользоваться кнопкой Back (Назад), это свойство до определенного момента использовать нельзя.
- history.previous. Тоже содержит одно значение – адрес URL страницы, к которой перейдет пользователь, если щелкнет на панели инструментов браузера на кнопке Back (Назад).

Объект history можно также обрабатывать и как массив. Каждый элемент массива содержит адрес URL из списка посещаемых страниц. Текущая страница представлена элементом history[0]. Ниже представлены методы объекта history:

- history.go. Открывает страницу по указанному адресу в списке адресов посещаемых страниц. Положительные и отрицательные числа соответствуют расположению адресов в массиве history. Например, элемент history.go(-2) соответствует двойному щелчку на кнопке Back (Назад).
- history.back. Загружает страницу, которая отображалась в окне браузера перед текущей. Этот метод аналогичен щелчку на кнопке Back (Назад).
- history.forward. Загружает страницу, которая отображается в списке адресов посещаемых страниц после текущей (если она указана). Этот метод аналогичен щелчку на кнопке Forward (Вперед).

Примечание

Методы history.back и history.forward в некоторых версиях Netscape Navigator выполняются неправильно. По этой причине лучше всегда использовать метод history.go (history.go(-1) и history.go(1)).

Создание кнопок Back и Forward

Одно из самых частых применений методов back и forward – это добавление на web-страницу соответствующих кнопок, позволяющих перемещаться по списку посещаемых страниц:

```
<html>
<head><title>Использование кнопок Back и Forward</title></head>
<body>

<a href="javascript:history.go(-1);">назад</a>&nbsp;
<a href="javascript:history.go(1);">вперед</a>

</body>
</html>
```

Объект location

Объект location содержит сведения о документе HTML, который открыт в окне. Например, следующий оператор дает указание загрузить страницу в текущем окне:

```
window.location.href="http://www.site.ru/page.htm"
```

Объект location имеет следующий набор свойств:

- location.protocol. Определяет протокол или метод.
- location.hostname. Определяет имя узла.
- location.port. Определяет порт соединения.
- location.host. Комбинация двух предыдущих свойств.
- location.pathname. Каталог расположения документа на узле и имя файла.
- location.hash. Название анкера в документе, если такой определен.

- location.target. Атрибут target ссылки, которая привела к открытию текущего документа.
- location.query. Определяет строку запроса.
- location.href. Определяет полный URL.

Объект location имеет два метода:

- location.reload. Перезагружает текущий документ. Аналогичен по функции кнопке Reload (обновить), расположенной на панели инструментов браузера.
- location.replace. Замещает текущую страницу указанной. Действует аналогично указанию свойств объекта location вручную.

Объект navigator

Объект navigator не принадлежит к иерархической структуре объекта браузера. Этот объект содержит данные о версии браузера. Он используется для определения типа браузера и платформы компьютера, на которой он запущен.

Объект navigator имеет следующий набор свойств:

- appName. Кодовое имя браузера.
- appVersion. Полное имя браузера.
- appVersion. Номер версии браузера.
- userAgent. Заголовок агента пользователя, отправляемый на узел при запросе web-страницы.
- plugIns. Массив сведений обо всех установленных в браузере плагинах.
- mimeType. Массив элементов, представляющих все типы MIME.

Объект event

Объект event позволяет коду сценария получить больше информации о каком-либо событии, происходящем в браузере. Среди свойств данного объекта можно выделить следующие:

X	Горизонтальная позиция курсора мыши (в пикселях)
Y	Вертикальное положение курсора мыши (в пикселях)
clientX	Возвращает горизонтальную координату элемента, исключая отступы, рамки, линейку прокрутки и т. д.
clientY	Возвращает горизонтальную координату элемента, исключая отступы, рамки, линейку прокрутки и т. д.
offsetX	Возвращает горизонтальную координату курсора мыши относительно содержащего его контейнера, когда происходит событие.
offsetY	Возвращает вертикальную координату курсора мыши относительно содержащего его контейнера, когда происходит событие.
screenX	Возвращает горизонтальную координату курсора мыши относительно экрана, когда происходит событие.
screenY	Возвращает вертикальную координату курсора мыши относительно экрана, когда происходит событие.
button	Кнопка мыши, если она нажата для появления события.
altKey	Возвращает состояние клавиши <i>Alt</i>
ctrlKey	Возвращает состояние клавиши <i>Ctrl</i>
shiftKey	Возвращает состояние клавиши <i>Shift</i>
keyCode	Код ASCII нажатой клавиши. Может быть изменен для посылки другого символа при той же нажатой клавише
reason	Показывает либо что пересылка данных успешна, либо почему она прервана.
type	Возвращает название события в формате строки без приставки on, например click вместо onclick.
fromElement	Элемент, вышедший из-под курсора мыши
toElement	Элемент, находящийся под курсором мыши
returnValue	Определяет возвращаемое значение для события
srcElement	Самый нижний элемент в иерархии, в котором появляется событие
cancelBubble	Может быть установлено для запрета прохождения события вверх по иерархической структуре

Обработка событий

Порядок выполнения программы JavaScript может изменяться при возникновении определенных *событий*. События – это действия, происходящие в браузере – щелчки мышью, нажатие клавиш, перемещение указателя мыши, загрузка рисунка и т.п.

Сценарии, которые позволяют определять события и выполнять соответствующие им действия, называются *обработчиками событий*.

В таблице ниже приведено описание некоторых событий.

Обработчик события	Описание	Поддерживаемые HTML-элементы
onBlur	Потеря текущим элементом фокуса, т.е. переход к другому элементу. Возникает при щелчке мышью вне элемента либо нажатии клавиши табуляции	A, AREA, BUTTON, INPUT, LABEL, SELECT, TEXTAREA
onChange	Изменение значений элементов формы. Возникает после потерей элементом фокуса, т.е. после события blur	INPUT, SELECT, TEXTAREA
onClick	Одинарный щелчок (нажата и отпущена кнопка мыши)	* Практически все HTML-элементы
onDbClick	Двойной щелчок	* Практически все HTML-элементы
onFocus	Получение элементом фокуса (щелчок мышью на элементе или очередное нажатие клавиши табуляции)	A, AREA, BUTTON, INPUT, LABEL, SELECT, TEXTAREA
onKeyDown	Нажата клавиша на клавиатуре	* Практически все HTML-элементы
onKeyPress	Нажата и отпущена клавиша на клавиатуре	* Практически все HTML-элементы
onKeyUp	Отпущена клавиша на клавиатуре	* Практически все HTML-элементы
onLoad	Закончена загрузка документа	BODY, FRAMESET
onMouseDown	Нажата кнопка мыши в пределах текущего элемента	* Практически все HTML-элементы
onMouseMove	Перемещение курсора мыши в пределах текущего элемента	* Практически все HTML-элементы
onMouseOut	Курсор мыши выведен за пределы текущего элемента	* Практически все HTML-элементы
onMouseOver	Курсор мыши наведен на текущий элемент	* Практически все HTML-элементы
onMouseUp	Отпущена кнопка мыши в пределах текущего элемента	* Практически все HTML-элементы
onMove	Перемещение окна	WINDOW
onReset	Сброс данных формы (щелчок по кнопке <code><input type="reset"></code>)	FORM
onResize	Изменение размеров окна	WINDOW
onSelect	Выделение текста в текущем элементе	INPUT, TEXTAREA
onSubmit	Отправка данных формы (щелчок по кнопке <code><input type="submit"></code>)	FORM
onUnload	Попытка закрытия окна браузера и выгрузки документа	BODY, FRAMESET

* *Практически все HTML-элементы* : все, за исключением APPLET, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE

Ниже приведен пример по обработке события onMouseUp:

```
<html>
<head><title>Обработка события OnMouseUp</title>
<script type="text/javascript">
function MouseUp(){
alert("Вы отпустили кнопку мышки");
}
</script>

</head>
<body>

<p onMouseUp="MouseUp();">щелкните здесь</p>

</body>
</html>
```

Временные задержки

Метод window.setTimeout определяет временную задержку перед выполнением указанной команды. Он имеет два параметра. Первый – это оператор JavaScript (или группа операторов), заключенный в скобки. Второй параметр – это время задержки в миллисекундах.

```
ident = window.setTimeout("alert('Ваше время истекло')", 1000);
```

Переменная (в примере ident) имеет собственное имя и представляет собой идентификатор временной задержки. До истечения указанного времени можно с помощью метода clearTimeout() прервать задержку. Для этого необходимо в скобках указать индентификатор:

```
window.clearTimeout(ident);
```

Управление окнами

Новое окно браузера создается с помощью метода window.open(). Типичный оператор создания нового окна имеет вид:

```
WinObj = window.open("URL", "WindowName", "Feature List");
```

В записи этого оператора используются следующие элементы:

- Переменная WinObj сохраняет данные нового объекта window. Её имя можно использовать с методами и свойствами созданного объекта window.
- Первый параметр – это URL документа, загружаемого в окне. Если его оставить незаполненным, то окно останется пустым.
- Второй параметр определяет название окна (WindowName). Его значение приписывается свойству name нового объекта window.
- Третий параметр представляет список необязательных опций, разделенных запятой. С их помощью можно определить вид нового окна: наличие в нем панелей инструментов, строки состояния и других элементов – например, параметры width и height, определяющие размеры окна, а также toolbar, location, directories, status, menubar, принимающие одно из двух значений – да (1) или нет (0).

Ниже приведен полный список всех доступных аргументов свойств, используемых для управления методом `open` объекта `window`:

Атрибут	Значение	Описание
<code>channelmode</code>	<code>yes no 1 0</code>	Показывает элемент управления Channel
<code>directories</code>	<code>yes no 1 0</code>	Включает кнопки каталога
<code>fullscreen</code>	<code>yes no 1 0</code>	Полностью разворачивает окно
<code>height</code>	<i>число</i>	Высота окна в пикселях
<code>left</code>	<i>число</i>	Положение по горизонтали относительно левого края окна в пикселях
<code>location</code>	<code>yes no 1 0</code>	Текстовое поле Address
<code>menubar</code>	<code>yes no 1 0</code>	Стандартное меню браузера
<code>resizable</code>	<code>yes no 1 0</code>	Может ли пользователь изменять размер окна
<code>scrollbars</code>	<code>yes no 1 0</code>	Горизонтальная и вертикальная полосы прокрутки
<code>status</code>	<code>yes no 1 0</code>	Стандартная строка состояния
<code>toolbar</code>	<code>yes no 1 0</code>	Включает панели инструментов браузера
<code>Top</code>	<i>число</i>	Положение по вертикали относительно верхнего края экрана в пикселях
<code>width</code>	<i>число</i>	Ширина окна в пикселях

Для закрытия окон используется метод `window.close`. Ниже приведен листинг HTML-документа, в котором поясняются основные принципы работы с окнами.

```
<html>
<head>
<title>Создание нового окна</title></head>
<body>

<p>Для открытия и закрытия окна используйте эти кнопки</p>
<p><a href="#" onClick="NewWin=window.open('','NewWin',
'toolbar=no,status=no,width=100,height=200');">Открыть новое окно</a></p>
<p><a href="#" onClick="NewWin.close();">Закрыть новое окно</a></p>
<p><a href="#" onClick="window.close();">Закрыть главное окно</a></p>
</body>
</html>
```