WEB-ДИЗАЙН

JavaScript

Для придания web-документу динамичности, чтобы организовать взаимодействие с пользователем, управлять браузером, изменять содержимое документа, обрабатывать данные пользователя и прочих действий, для которых средств HTML недостаточно, применяют *сценарии* (скрипты — от script).

Одним из наиболее популярных языков сценариев сейчас является JavaScript, разработанный Бренданом Айком (Brendan Eich) в 1995 году. Все современные web-браузеры поддерживают язык JavaScript и имеют его встроенные интерпретаторы (так называемый *клиентский* JavaScript, который мы и будем рассматривать далее). Программы (сценарии), написанные на JavaScript, выполняются в браузере пользователя при загрузке страницы или в ответ на определённые действия пользователя. Они позволяют автоматически заполнять формы, переформатировать страницы, изменять поведение клиентской части web-приложений, добавлять элементы управления на страницу и т. д. Таким образом, JavaScript позволяет управлять не только содержанием HTML-документов, но и их *поведением*. Вообще говоря, JavaScript — это язык программирования общего назначения, и его применение не ограничено только web-браузерами. Изначально JavaScript разрабатывался, чтобы его можно было встраивать в любые приложения и иметь возможность исполнять сценарии.

Рис. 1. Брендан Айк

Javascript — это интерпретируемый язык программирования с объектно-ориентированными возможностями. С точки зрения синтаксиса[1] JavaScript имеет сходство с языками C, C++ и Java, но его семантика[2] значительно отличается. JavaScript вполне пригоден для создания простых программ, и не сложен в изучении даже для начинающих.

Особенности синтаксиса JavaScript:

- Имена (идентификаторы) переменных, функций, объектов и т. д. могут содержать латинские буквы, арабские цифры и символ «_»; имена нельзя начинать с цифры. Идентификаторы не могут совпадать ни с одним из ключевых слов JavaScript.
- Чувствительность к регистру: Name, name, NAME — разные имена.
- Код программы можно писать в свободной форме — пробелы, табуляции и переводы строк, присутствующие между лексемами в программе игнорируются интерпретатором.
- Простые JavaScript-инструкции обычно (но не всегда обязательно) завершаются символами точки с запятой «;».
- JavaScript — это нетипизированный (untyped) язык. Это значит, что *любая* переменная может содержать значение *любого типа*.

[1] Синтаксис (от др.-греч. συνταξις — «порядок, составление») — здесь правила построения операторов, функций, типов и т. д. языка программирования.

[2] Семантика (от др.-греч. σημαντικος — «обозначающий») в программировании — дисциплина, изучающая значения операторов, функций, типов и т. д. языка программирования.

- Блоки операторов, такие как, например, тела функций, заключаются в фигурные скобки.
- Комментарии могут быть двух типов:

```
// однострочный комментарий
/* многострочный
   комментарий */
```

  Комментарии игнорируются интерпретатором JavaScript и предназначаются для человека, читающего исходный код, — для пояснений и различной служебной информации.

- JavaScript не делает различия между целыми и вещественными числами. Все числа представляются 64-разрядными вещественными значениями с плавающей точкой в интервалах от $\pm 1{,}7976931348623157 \times 10^{308}$ до $\pm 5 \times 10^{-324}$. Когда вещественное число превышает самое большое представимое значение, результату присваивается специальное значение «бесконечности», которое в JavaScript обозначается как `Infinity`. Аналогично, для слишком малых отрицательных чисел: `-Infinity`. Если математическая операция (например, деление на нуль) приводит к неопределенности или ошибке, то результатом является специальное значение `NaN` (Not-a-Number — нечисло). Величина `NaN` необычна: она не равна ни одному другому числу, и даже самой себе.

Одной из важнейших характеристик языка программирования является поддерживаемый им набор типов данных. JavaScript позволяет работать с тремя элементарными типами данных: числами, строками текста и логическими значениями. Соответственно, *литералы*, т. е. значения, которые буквально (literally) представлены в исходном коде, имеют вид:

```
−1, 2012                  // целые числа
 0xA5                     // целое в 16−тиричной форме
 33.99, 6.01E5            // вещественные числа
 "строка", 'String01'     // строки текста
 false , true             // логические значения
```

Переменные позволяют хранить данные в программе и работать с ними. Имена переменных задаются по общим правилам. Объявления переменных не обязательны. Типы переменных определяются по присвоенным им значениям. Поскольку JavaScript — слаботипизированный язык, его интерпретатор в случае необходимости легко (!) и автоматически (!!) преобразует значения из одного типа в другой:

```
var i = 10;              // объявлена переменная i и присвоено значение числа 10
i = i + 5;               // i <−− 15
i = "десять";            // переменной i присвоено значение строки "десять"
numb = "10" − 5;         // ?
numb = "10" + 5;         // ?!
```

☞ Поэтому очень важно внимательно следить за типами переменных, во избежание неожиданных результатов.

Операторы JavaScript:

- Арифметические операторы: +, −, /, ∗, %; инкремент и декремент:

```
++i;      i++;          // то же, что i = i + 1
--value;  value--;      // value = value - 1;
```

- Операторы присваивания: =, +=, −=, /=, ∗=, %= . Например, присваивание x += 5; — то же, что x = x + 5;.
- Логические операторы: && — логическое «и», || — логическое «или», !— логическое отрицание. Операторы сравнения: <, >,<=, >=, ==, !=, возвращают логические изначения.
- Строковые операторы. Конкатенация (соединение строк):

```
half   = "поло";
clever = "умный";
half   = half + clever; // или короче half += clever;  half <-- "полоумный"
```

Операторы сравнения также применимы к строкам. Сравнение строк основано на алфавитном порядке их символов.

Приоритет операторов. Оператор умножения ∗ имеет больший приоритет по сравнению с оператором сложения +, поэтому умножение выполняется раньше сложения. Оператор присваивания = имеет наименьший приоритет, поэтому присваивание выполняется после завершения всех операций в правой части. Приоритет операторов можно переопределить, как обычно, с помощью скобок.

Операторы ветвлений и циклов:

- Условный оператор

```
if ( логическое_выражение ) // скобки здесь обязательны
{
 блок_операторов_1
}
else                         // необязательная
{                            // часть,
 блок_операторов_2           // может
}                            // отсутствовать
```

Если логическое_выражение == true (истина), то работает блок_операторов_1, иначе (else), в случае когда логическое_выражение == false (ложь), будет исполнен блок_операторов_2.

- Оператор выбора

```
switch( n ) {
            case 1:  блок_кода_1        // выполняется, если n == 1
                     break;             // здесь выходим из switch
            case 2:  блок_кода_2        // выполняется, если n == 2
                     break;             // здесь выходим из switch
            . . . . . .
            default: блок_кода_D        // если все остальное не подходит
                     break;             // здесь выходим из switch
           }
```

В JavaScript, в отличие от C/C++, после case возможно *любое* выражение.

- Операторы цикла.

```
while ( логическое_выражение )   // пока логическое_выражение истинно
{
 блок_операторов                 // выполнять блок_операторов
}
```

или, другая форма:

```
do                                // выполнять
{
 блок_операторов                  // этот блок_операторов
}
while ( логическое_выражение );   // пока логическое_выражение истинно
```

Более общий оператор цикла имеет вид

```
for ( инициализация; условие; инкремент )
{
 блок_операторов
}
```

Например, сумма всех натуральных чисел, не превосходящих 10:

```
var  summa = 0;
     for ( var i = 1; i <= 10; i++ )
     {
      summa += i;
     }
document.write( "summa = " + summa );
```

- Любая инструкция в JavaScript-коде может быть помечена специальной меткой (label):

```
имя_метки: JavaScript−инструкция
```

где имя_метки — правильный идентификатор. В частности, в операторе выбора switch имеются метки case и default. Оператор break приводит к немедленному выходу из цикла, *в котором он указан*, а инструкция break имя_метки; после выхода из цикла передаст программное управление оператору, помеченному меткой имя_метки. Пример — определение простоты числа методом *простого деления:*

```
var n         = 101;                                  // тестируемое число
var sqrt_n    = Math.sqrt( n );                       // кв. корень из него
var is_prime = true;                                  // флаг простоты
      for ( var divider = 2; divider <= sqrt_n; divider++ )
      {
            if ( ( n % divider ) == 0 )               // если n делится на divider,
            {
             is_prime = false;                        // то оно не простое
             break;                                   // покидаем цикл
            }
       }
if ( is_prime ) document.write( n + " &mdash; простое."   );
else            document.write( n + " &mdash; составное." );
```

Оператор continue прерывает текущую итерацию цикла и запускает следующую за ней.

Отдельные блоки кода, выполняющие некоторые *неоднократно* повторяющиеся действия, в языках высокого уровня обычно оформляют в виде *функций*. В JavaScript объявление функции имеет следующий формат:

```
function имя_функции( формальные_параметры )
{
 тело_функции
}
```

где имя_функции — правильный идентификатор, формальные_параметры — последовательность (возможно, пустая) правильных имён аргументов функции, разделённых запятыми, тело_функции — блок операторов функции. Фигурные скобки здесь обязательны. Функции объявляются только один раз, а используются, как правило, многократно.

Пример — вычисление среднего арифметического трёх чисел:

```
function mean( a, b, c )
{
 var s = ( a + b + c )/3;
 return s;
}
document.write( mean( 10, 20, 30 ) );
```

Оператор return *возвращает* вычисленное функцией значение s. Вызов функции осуществляется указанием её имени со списком *фактических параметров*. Фактические параметры присваиваются соответствующим формальным параметрам функции, после чего выполняются операторы тела функции.

С переменными в языках высокого уровня обычно связаны т. н. *области видимости* (scope) — те сегменты кода, где переменная сохраняет своё значение (откуда переменная «видна»). В JavaScript область видимости распространяется на участки кода внутри *функций*, а не *блоков* (образуемых, например, такими операторами, как while, if, for и т. д), в отличии, скажем, от языков C/C++. Другими словами, переменные, объявленные внутри функций, являются *локальными*, их область видимости ограничена телом функции, а переменные, объявленные вне функций, — *глобальными*, они «видны» везде. Кроме того, если переменная не объявлена явно (без ключевого слова var), она становится глобальной, даже если используется только внутри функции.

Функции позволяют практически оформить расчленение сложного кода на более простые блоки и, тем самым, свести трудную задачу к более лёгким подзадачам. При использовании функций рекомендуется:

- использовать для них и параметров логичные, «говорящие» имена;
- по возможности, избегать глобальных переменных — увеличиваются программные *связи*, что чревато появлением трудноуловимых ошибок;
- тщательно отлаживать каждую функцию в отдельности и, затем, всю их совокупность во всем коде скрипта;
- соблюдать меру — слишком «мелкое» разбиение кода на функции приводит к полной его нечитаемости, а слишком крупное — к усложнению связей; функция должна делать только *одну работу* и делать её хорошо.

Особенности JavaScript-функций:

- Тип параметров функций указывать не требуется, и интерпретатор не проверяет, соответствует ли тип данных требованиям функции. Также не проверяется соответствие количества фактических параметров числу формальных аргументов. Если аргументов больше, чем требуется функции, то лишние значения просто игнорируются. Если аргументов меньше, то отсутствующим присваивается специальное значение undefined (неопределенный).
- Если в теле функции нет оператора return, функция возвращает undefined.
- JavaScript позволяет определять функции в виде так называемых *функциональных литералов:*

```
function square( x )   { return x*x; }   // объявление функции
var  f  = function( x ) { return x*x; }   // f -- функциональный литерал
```

- В JavaScript функции фактически являются *объектами* — см. далее.
- Хотя JavaScript-функция определяется с фиксированным количеством именованных аргументов, при вызове ей может быть передано любое их число. Функцию можно определить так, чтобы она работала с любым количеством аргументов. При этом получить доступ к параметрам функции можно с помощью специального свойства: имя_функции.arguments[i], где i — номер параметра по порядку, начиная с нуля (так arguments[1] совпадает с параметром b в примере выше). Общее число аргументов хранится в arguments.length.

Объект — это составной тип данных, объединяющий неупорядоченное множество именованных значений разного типа, называемых *свойствами* (properties) объекта. Свойствами могут быть переменные простых типов (числа, строки, логические величины), функции (их называют *методами* объекта), а также другие объекты. Примеры создания объектов:

```
var point  = { x      : 0,              // точка
               y      : 0,              // с координатами (x, y)
               color  : "green" };      // и цветом color
//-----------------------------------------------------
var circle = { x0     : 1,                                  // круг с центром
               y0     : 2,                                  // в точке (x0, y0)
               r      : 5,                                  // и радиуса r
               square : function()                          // метод, возвращающий
                        {                                   // площадь круга
                         return Math.PI*this.r*this.r;      // PI -- const pi
                        }
             };
//-----------------------------------------------------
var empty = new Object();   // пустой объект создан спец. оператором new
```

Доступ к свойствам объекта в JavaScript не ограничен (то есть в терминах языка C++ они по умолчанию всегда относятся к public). Обычно для доступа к значениям свойств объекта используется оператор . (точка):

```
point.color;          // значение -- строка "green"
circle.square();      // значение -- число 25*PI
circle.r = 10;        // переопределение величины радиуса круга, теперь r == 10
```

назад
закрыть

Таким образом, свойства объекта работают как обычные переменные: в них можно сохранять значения и считывать их. При обращении к несуществующему свойству ошибки не будет, просто вернется значение undefined.

В языках со строгой типизацией, типа Java, Object Pascal, C++, объект может иметь только фиксированное число свойств, и имена этих свойств должны быть определены заранее. В JavaScript допускается *в любой момент* добавить или удалить любое свойство. Новое свойство объекта можно добавить, просто присвоив этому свойству значение:

```
circle . color  = "red";                 // добавлено свойство color и задано его значение
point.move   = function( x1, y1 )  // объекту добавлен новый метод move
               {
                point.x = x1;
                point.y = y1;
               }
```

Свойства также легко можно удалить специальным оператором:

```
delete point. color ;   // точка теперь лишена своего цвета
```

Кроме доступа к свойствам объекта «через точку» есть возможность применить для этого оператор [ ]:

```
circle ["color"]    // то же, что  circle . color
```

Чтобы метод не просто вызывался из объекта, но и мог менять находящиеся в нем данные, используется ключевое слово this (этот) — см. пример выше.

В отличии от классических объектно-ориентированных языков программирования, таких как C++ и Java, JavaScript не имеет формального понятия *класса*, хотя и поддерживает тип данных, называемый *объектом*. JavaScript, являясь объектно-ориентированным языком, использует другой механизм *наследования*, на базе *прототипов* (prototype), а не на основе классов [1], [5].

Все объекты в JavaScript содержат внутреннюю ссылку на объект, называемый прототипом. Любые свойства прототипа становятся свойствами другого объекта, для которого он является прототипом. Таким образом, любой объект в JavaScript *наследует* свойства своего прототипа. Пример — функция-объект Circle наследует свойства (координаты) объекта Point:

```
function Point( _x, _y )                // функция–объект Точка
{                                       // с координатами x, y
 this.x = _x;
 this.y = _y;
}
function Circle( x0, y0, _radius )      // объект Круг
{                                       // с центром
 Point.call( this, x0, y0 );            // в Точке
 this.radius = _radius;                 // и радиусом radius
}
c = new Circle( −5, 5, 10 );            // теперь у Circle есть свойства x,y
document.write( "центр круга:  " + c.x + ", " + c.y + "</BR>радиус: " + c.radius );
```

Все объекты в JavaScript наследуют свойства и методы объекта Object. Например, Object (и, следовательно все объекты JavaScript) имеет метод toString(), возвращающий строку, каким-либо образом представляющую тип и/или значение объекта, для которого он вызывается. Интерпретатор JavaScript вызывает этот метод во всех случаях, когда требуется преобразовать объект в строку — примеры были приведены выше.

JavaScript содержит несколько *встроенных* функций и объектов, таких как:

- объект Math, имеющий свойства и методы для математических вычислений (см. пример выше), например, константы e (Math.E), π (Math.PI), функции $\sin x$ (Math.sin(x)), e^x (Math.exp(x)) и т. д;
- объект String, содержащий средства для работы со строками;
- объект Date, включающий методы для установки, получения, и управления величинами даты и времени. Пример — вывод даты и времени в соответствии с установками «локали»:

```
var d = new Date();
document.write( d.toLocaleString() );
```

Массив — это тип данных, содержащий пронумерованные значения, называемые *элементами* (компонентами) массива, а их номера именуются *индексами*. Аналог массива в математике — вектор. Но в JavaScript элемент массива может иметь любой тип, причем разные элементы одного массива могут иметь разные типы. Элементы массива могут даже содержать другие массивы, что позволяет создавать «массивы массивов». Длина массива в JavaScript не фиксирована.

Массивы задаются либо литерально, либо оператором **new**. Элементы массива и их количество могут быть изменены в любой момент. Доступ к элементам массива осуществляется оператором [ ] по индексу, который записывается внутри скобок. Индекс — любая целая неотрицательная величина ($\leqslant 2^{32}-1$), нумерация элементов массива всегда начинается с нуля. Примеры:

```
var  _5primes = [ 2, 3, 5, 7, 11 ];          // задан массив из 5 чисел
var  Orwell   = [ true, "is", false, 1984 ]; // элементы разных типов
var  trigs    = [ 19.61, Math.sin( Math.PI/2) ];
var  mass     = new Array( 8 );              // массив из 8 неопред. элементов
_5primes[ 1 ];                               // == 3
trigs [1];                                   // == 1
mass[3];                                     // == undefined
5primes[ 5 ] = 13;                           // в массив добавлен 6-й элемент
delete Orwell[ 3 ];                          // теперь Orwell[ 3 ] == undefined
var m = [ [ 11, 12 ],                       // двумерный массив,
          [ 21, 22 ]   ];                    // матрица 2x2
m [0][1];                                    // == 12
```

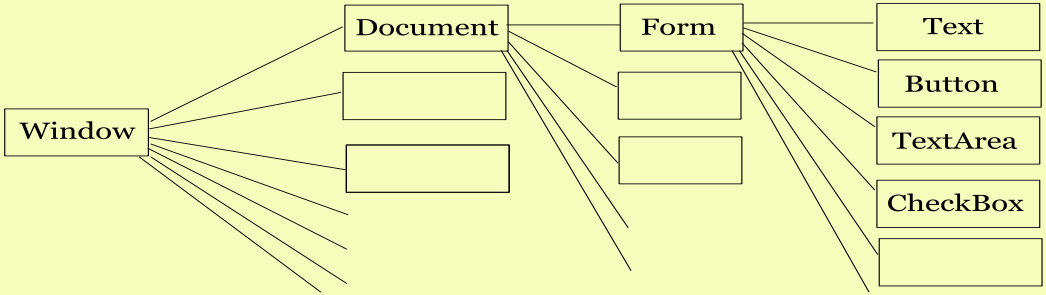
Массив является специализированным *объектом*, поэтому у него есть различные свойства, к которым можно обращаться «через точку». Например, массив, среди прочих, имеет свойство length, в котором хранится его текущее количество элементов, причем свойство length доступно как для чтения, так и для записи. Если задать length меньшим текущего, массив укорачивается до новой длины; «лишние» элементы теряются. Если сделать свойство length большим, чем его текущее значение, в конец массива добавляются новые неопределенные (undefined) элементы. Массив также обладает множеством методов, предназначенных для манипуляций с его элементами [1], [3].

Для примера рассмотрим программную реализацию решета Эратосфена

```
function sieve( n )                     // находит все простые числа, <= n
{                                       // массив A сначала содержит числа 2,..,n
 var sqrt_n = Math.sqrt( n );
 var d      = 1;
     while ( ++d <= sqrt_n )            // для всех делителей, <= кв. корня из n
     {
         for ( var i = d - 1; i < A.length; i++ )
         {
             if ( !( A[ i ] % d ) )     // все элементы, кратные d
             A.splice ( i, 1 );         // удаляем из массива A
         }
     }
 return A.join( ", " );                 // преобразуем массив в строку и возвращаем
}
```

назад
закрыть

Для получения доступа к содержимому веб-документа существует программный интерфейс, называемый DOM (Document Object Model — объектная модель документа), не зависящий от платформы и языка. Любой документ с помощью DOM может быть представлен в виде *дерева узлов*, каждый узел которого представляет собой некоторый элемент, текстовый, графический или любой другой объект. Узлы образуют многоуровневую иерархию. Консорциумом W3C был выработан стандарт DOM, поддерживаемый в настоящее время всеми современными браузерами [9].



Основная задача веб-браузера состоит в отображении *документа* в *окне*. Поэтому JavaScript может обращаться к глобальному узлу Window. В силу глобальности Window, все глобальные переменные, с ним связанные (в том числе методы и свойства Document), определяются *как свойства окна*, то есть

```
var x = 100500;                  /* то же, что */   window.x = 100500;
document.write( "Ура!" );        /* то же, что */   window.document.write( "Ура!" );
```

Javascript-сценарии могут обращаться ко всем объектам, которые представляют элементы (узлы) окна или документа, и манипулировать ими. Пример — заливка фона страницы случайным цветом:

```
<HEAD>
    <script>
     function change_color()
     {
      document.bgColor = 100000*Math.random();
     }
    </script>
</HEAD>
<BODY>
    <FORM NAME = "FORMA">
        <INPUT TYPE = "button"
          NAME       = "btn"
          VALUE      = "сменить цвет"
          ONCLICK    = "change_color()" >
    </FORM>
</BODY>
```

Важно помнить, что всякий раз, когда в окно броузера загружается новая страница, объект Window переводится в состояние по умолчанию. Каждая веб-страница начинается «с чистого листа». Сценарии *не наследуют* измененное окружение, оставшееся от предыдущего документа, все переменные и функции, определяемые сценарием, существуют только до того момента, пока текущий документ не замещен новым.

JavaScript-код может внедряться в HTML-документы тремя различными способами:

- Между парой тегов <SCRIPT> </SCRIPT> — см. примеры выше.
- В обработчик события, заданный в качестве значения HTML-атрибута, такого как onclick или onmouseover.
- Из внешнего файла, заданного атрибутом SRC тега <SCRIPT>.

```
<script  src = "путь_к_файлу/файл.js"> </script>
```

где в качестве пути_к_файлу можно указать URL любого файла в Интернете, а файл.js — обычный текстовый ASCII-файл, содержащий JavaScript-код. Закрывающий тег обязателен.

Наиболее предпочтителен последний способ. Основные преимущества размещения кода сценария в отдельном файле следующие:

- Отделение *поведения* HTML-документа от его *содержания* и *вида*.
- Если JavaScript-код, используемый несколькими HTML-файлами держать в одном файле и считывать лишь при необходимости, то уменьшается объем занимаемой дисковой памяти и облегчается поддержка программного кода, ускоряется загрузка документа браузером (за счет кэширования).
- Поскольку SRC принимает в качестве значения произвольный URL-адрес, страница с одного веб-сервера может воспользоваться кодом (например, из библиотеки подпрограмм), предоставляемыми другими веб-серверами.

Список литературы

1. Флэнаган Д. JavaScript. Подробное руководство. 5-е изд., СПб.: Символ-Плюс, 2008.
2. Вайк Аллен м др. JavaScript. Справочник. СПб.: ДиаСофтЮП, 2002.
3. Goodman D. JavaScript Bible, Gold Edition. NY: 2001.
4. Рейсиг Д. JavaScript. Профессиональные приёмы программирования. СПб.: Питер, 2008.
5. Zakas N. Professional JavaScript for Web Developers. Wiley, 2005.
6. Джамса К. и др. Эффективный самоучитель по креативному Web-дизайну. HTML, XHTML, CSS, JavaScript, PHP, ASP, ActiveX. Текст, графика, звук и анимация. М.: ООО ДиаСофтЮП, 2005.
7. HTTP://JAVASCRIPT.RU/
8. HTTP://JAVASCRIPT.CROCKFORD.COM/
9. HTTP://WWW.W3.ORG/DOM/