Методические указания к выполнению
лабораторной работы и практикума №2

Javascript. Добавление динамического поведения. Средства отладки.
Создание сложных HTML-страниц.

Самарев Роман Станиславович
канд. техн. наук, доцент
samarev@acm.org
каф. ИУ-6 «Компьютерные системы и сети»
МГТУ им. Н.Э. Баумана

Москва 2013

Оглавление

Цель работы

Целью работы является изучение основных принципов программирования на языке Javascript, изучение способов обхода узлов модели документа DOM, изучение принципов динамического формирования кода страницы HTM, а также получение практических навыков отладки Javascript-приложения.

Краткие сведения о Javascript

Объектно-ориентированный скриптовый язык программирования. Является диалектом стандартизованного языка ECMA_262 (версия 3) European Computer Manufacturer's Association (ECMA).
Язык с динамической типизацией, автоматическим управлением памяти. Версия 1.0 была реализована в Netscape 2.0, март 1996. Предназначен для использования в веб-приложениях. Позднее область применения была расширена. В настоящее время активно используется как для выполнения программ на стороне веб-браузера, так и на стороне веб-сервера для упрощения интеграции с «тяжелыми» веб-приложениями. Кроме того, в программных продуктах Mozilla активно используется для программирования интерфейса настольных приложений в составе технологии XUL.

Подключение Javascript в HTML

В HTML-документах Javascript при помощи одного и того же элемента <script> может быть внедрён в тело документа или подключен внешний ресурс.
В случае внедрения в тело документа Javascript-код вставляется между тегами <script> и </script>:

```
<script type="text/javascript">
//Внедрённый код.
 ...
</script>
```

Рекомендуется весь Javascript-код включать в заголовок страницы, однако это требование не является жёстким.

В случае подключения внешнего ресурса используется атрибут src, которому указывается URL Javascript-кода.

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

Обратите внимание на то, что теги <script></script> всегда парные!

Пример клиентского JavaScript [1] с внедрённым кодом:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
      <title>Факториалы</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
      <h1>Таблица факториалов</h1>
      <script>
            for(fact = i = 1; i < 10; i++) {
                  fact = fact * i;
                  document.write(i + "! = " + fact + "<br />");
            }
      </script>
</body>
</html>
```

Результат выполнения этого скрипта представлен на рисунке 1. Разметка продемонстрирована с использованием отладчика Firebug.

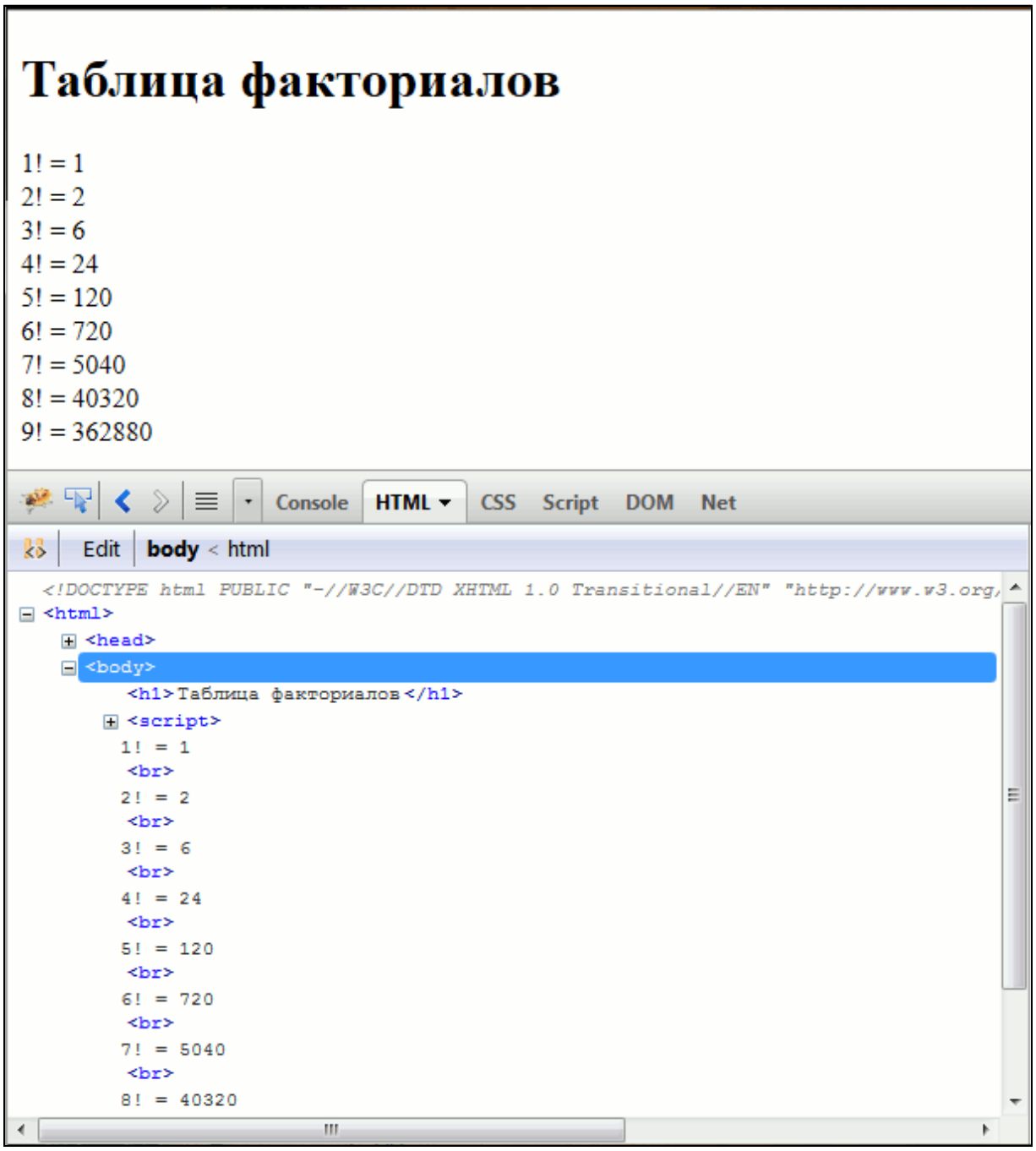

Рисунок 1 Просмотр структуры динамически сгенерированной разметки.

Синтаксис

Спецификации ECMAScript v1 и v2 предполагают использование в коде программы 7-ми битной ASCII-кодировки и использование UNICODE-символов в комментариях или строковых литералах. ECMAScript v3 позволяет использовать UNICODE в любой части программы, однако для того, чтобы исключить проблемы совместимости со старыми браузерами, этого следует избегать.

JavaScript чувствителен к регистру. Символы табуляции и перевода строк игнорируются. Однако есть одно замечание. Разделителем простых операторов является символ “;”, но не является обязательным в том случае, если операторы разделены переводом строк.

Обратите внимание на то, что следующие записи эквивалентны:

```
a=1;
b=3;
```

и

```
a=1
b=3
```

Однако запись:

```
return
true
```

является некорректной, поскольку её эквивалент:

```
return;
true;
```

но не

```
return true;
```

Комментарии

Комментарии Javascript аналогичны языкам C и C++.

```
// однострочный комментарий

/* многострочный
 комментарий*/
```

Числа

С точки зрения внутреннего представления для JavaScript все числа являются 64-х разрядными числами с плавающей точкой и описываются стандартом IEEE 754.
Все целые числа должны лежать в диапазоне от –9007199254740992 (-2^{53}) до 9007199254740992 (2^{53}), поскольку иначе произойдёт потеря младших разрядов.

Вещественные числа могут находиться в диапазоне от $\pm 1{,}7976931348623157 \times 10^{308}$ до $\pm 5 \times 10^{-324}$.

Допустимы следующие литералы:

- Десятичные целые числа (0, 234, 652345).
- Шестнадцатеричные числа (0xF1, 0xfe340b).
- Вещественные числа ([цифры][.цифры][(E|e)[(+|_)]цифры]).

При работе с числами доступны все традиционные арифметические операции. Кроме того, отметим, что существуют специальные константы Infinity, NaN, Number.MAX_VALUE и некоторые другие, которые могут быть использованы в арифметических операциях [1].

Строки

Строки являются объектами класса String.

Строковый литерал – это последовательность из нуля или более Unicode-символов, заключенная в одинарные или двойные кавычки (' или "). Строка может содержать специальные управляющие символы такие как: \n, \b, \0, \xXX, \xXXXX и некоторые другие.

При работе со строками доступны операции конкатенации:
```
str = "123" + "456"
```

Для определения длины строки можно использовать свойство length:
```
str.length
```

Извлечение подстроки производится методом substring (первая позиция, количество). Нумерация символов – с нуля.
```
substr = str.substring(2,5);
```

Метод indexOf определяет позицию первого символа в указанной строке s:
```
i = str.indexOf('a');
```

Операции, которые можно проводить со строками определяются классом String, однако их подробное рассмотрение см. в [1].

Преобразование чисел в строку

Javascript непосредственно обеспечивает преобразование чисел в строку, если они используются при конкатенации строк.

```
var n = 10
str = n + " получено в результате";
str = n.toString() + " получено в результате";
```

Если формат вывода необходимо четко определить, могут быть использованы дополнительные методы:

```
var n = 123456.789;
n.toFixed(0);     // "123457"
n.toFixed(2);     // "123456.79"
n.toExponential(1); // "1.2e+5"
n.toExponential(3); // "1.235e+5"
n.toPrecision(4); // "1.235e+5"
n.toPrecision(7); // "123456.8"
```

Преобразование строк в числа

Javascript может явно преобразовать строки в числа, если это следует из контекста.
Пример:
```
var n = "12"*"2" // результат 24
```

Однако это не сработает в случае:
```
var n = "12" + "2" // результат "122", поскольку + является конкатенацией
```

Для решения этой проблемы может быть использован класс Number(), однако требование его использования заключается в том, что число должно быть обязательно десятичным.

Функции parseInt() и parseFloat() позволяют организовать разбор чисел из строки, содержащей не только цифры. Однако, если на первом месте в строке будет находиться не цифра, то возвращенный ими результат будет NaN (не число).
Функция в форме parseInt(str, base) позволяет разобрать числа в указанной base-системе счисления.

Некоторые примеры из [1]:

```
parseInt("3 слепых мышки"); // Вернет 3
parseFloat("3.14 метров"); // Вернет 3.14
parseInt("12.34"); // Вернет 12
parseInt("0xFF"); // Вернет 255

parseInt("11", 2); // Вернет 3 (1*2 + 1)
parseInt("ff", 16); // Вернет 255 (15*16 + 15)
parseInt("zz", 36); // Вернет 1295 (35*36 + 35)
parseInt("077", 8); // Вернет 63 (7*8 + 7)
parseInt("077", 10); // Вернет 77 (7*10 + 7)
```

Логические операции

Логический тип имеет значения true или false аналогично boolean в C++.
Поддерживается почти аналогичный набор логических операций.
Пример:

```
if (a == 4)
   b = b + 1;
else
   a = a + 1;
```

Имеется дополнительный оператор тождественного сравнения = = = (троекратное =), который отличается от == (двойное =) тем, что операнды должны совпадать по типу, а неявное преобразование к ним не будет применено:

```
if (a === "4") // только строки!
   b = b + 1;

if (a === 0) // только число 0
   b = b + 1;

if (a == 0) // число 0 или пустая строка
   b = b + 1;.
```

Имеющиеся операторы сравнения представлены в таблице 1. Для определённости примем, что предварительно выполнено присвоение x=5.

Таблица 1 Операторы сравнения Javascript

Оператор	Описание	Пример (x=5)	Результат

Оператор	Описание	Пример (x=5)	Результат
==	Равенство	x==8 x==5	false true
===	Равенство с учетом типа	x==="5" x===5	false true
!=	Неравенство	x!=8	true
!==	Неравенство с учетом типа	x!=="5" x!==5	true false
>	Больше	x>8	false
<	Меньше	x<8	true
>=	Больше или равно	x>=8	false
<=	Меньше или равно	x<=8	true

Поддерживаются следующие выражения, позволяющие организовать ветвление процесса:

```
if (condition) {
  …
} else {
  …
}

variablename=(condition)?value1:value2

switch(n)
{
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
  code to be executed if n is different from case 1 and 2
}
```

Циклы

Выражения для формирования циклов в Javascript аналогичны C/C++ за исключением конструкции for (item in list){…}.

```
for (i=0; i<5; i++) {
  x=x + "The number is " + i + "<br />";
}

while (i<5) {
  x=x + "The number is " + i + "<br />";
  i++;
}

do {
  x=x + "The number is " + i + "<br />";
  i++;
} while (i<5);

var person={fname:"John",lname:"Doe",age:25};
for (x in person) {  txt=txt + person[x]; }
```

Функции

Функции в Javascript задаются в форме:

```
function sum (a,b) // декларируем функцию sum с двумя аргументами
{
   return a+b;
}
```

Обратите внимание на то, что возвращаемый результат не декларируется явно. Передаваемые параметры a, b не имеют типа, однако попытка передать значения неправильного типа приведут к ошибке при выполнении кода!

Существует возможность определить функциональный литерал в следующей форме:

```
var sum = function sum (a,b) { return a+b; };
```

В дальнейшем использование sum будет выглядеть аналогично явно определенной функции, однако реальную функцию в sum можно в любой момент переопределить.

Объекты

Объект – это коллекция именованных значений, которые обычно называют свойствами (properties) объекта. Важно то, что в Javascript свойства объекта могут быть добавлены при выполнении.

Пример:

```
var point = new Object(); // создаём пустой объект !!!
point.x = 2.3;    // присоединяем свойство x
point.y = 1.2;    // присоединяем свойство y
```

Обращение к свойствам может производиться либо в форме:

```
point.x
```

либо в форме ассоциативного массива:

```
point["x"]
```

Объектные литералы позволяют задавать значения объектов в коде. Объектные литералы могут быть простыми:

```
var point = { x:2.3, y:1.2 };
```

или вложенными:

```
var rectangle = {
    upperLeft: { x: 2, y: 2 },
    lowerRight: { x: 4, y: 4 }
};
```

Любой объект может быть преобразован в строку вызовом метода toString().

Массивы

Массивы, как и объекты, являются коллекциями значений, но важное отличие состоит в том, что каждый из элементов имеет номер. Нумерация элементов в Javascript начинается с нуля, как и в C. Объект-массив создается при помощи конструктора Array().

Пример массива, элементы которого произвольны:

```
var a = new Array();
a[0] = 1.2;
a[1] = "JavaScript";
a[2] = true;
a[3] = { x:1, y:3 };
```

Инициализация массива при создании:

```
var a = new Array(1.2, "JavaScript", true, { x:1, y:3 });
```

Создание массива на указанное количество элементов:

```
var a = new Array(10);
```

Литерал массива - это список разделенных запятыми значений, заключенных в квадратные скобки. Допустима инициализация массива с разнородными объектами:

```
var a = [1.2, "JavaScript", true, { x:1, y:3 }];
```

Инициализация двумерного массива:

```
var matrix = [[1,2,3], [4,5,6], [7,8,9]];
```

Инициализация массива динамически вычисленными значениями:

```
var base = 1024;
var table = [base, base+1, base+2, base+3];
```

Переменные и их области видимости

Переменные в Javascript не имеют типа! Тип имеет только значение переменной, поэтому переменная, которая не была инициализирована конкретным значением, не может иметь тип. Более того, одна и та же переменная может принимать значения различных типов в разные моменты времени, однако всегда можно проверить, была ли переменная вообще инициализирована.

Javascript позволяет определять глобальные и локальные переменные. Отсутствует блочная видимость в том виде, как это реализовано в языках C и C++.

Пример:

```
function test(o) {
  var i = 0; // i определена во всей функции
  if (typeof o == "object") {
    var j = 0; // j определена везде, а не только в блоке
    for(var k = 0; k < 10; k++) { // k определена везде, не только в цикле
     document.write(k);
    }
    document.write(k); // k все еще определена: печатается 10
  }
  document.write(j); // j определена, но может быть не инициализирована
}
```

Основной принцип распространения видимости переменных следующий: переменные доступны в текущей функции и во всех вложенных функциях (то есть определённых в коде этой функции). Глобальные переменные, т.е. переменные, определенные вне функций, доступны везде.

```
var x = function(){
      var i;
      mul = function(b) { return i*b; }
      for(fact = i = 1; i < 10; i++) {
            fact = mul(fact);
            document.write(i + "! = " + fact + "<br />");
      }
}
```

Локальными переменными являются аргументы функций, а также переменные, не найденные в контексте выше, либо объявленные со служебным словом var.

Пример:

```
var scope = "глобальная";
function f() {
  alert(scope); // Показывает "глобальная".
  scope = "локальная"; // Переменная глобальная.
  alert(scope); // Показывает "локальная"
}
f();
alert(scope); // Показывает "локальная"
```

Укажем внутри функции декларацию var scope.

```
var scope = "глобальная";
function f() {
  alert(scope); // Показывает "undefined", а не "глобальная".
  var scope = "локальная"; // Переменная инициализируется здесь,
                           // но определена она везде в функции.
  alert(scope); // Показывает "локальная"
}
f();
alert(scope); // Показывает "глобальная".
```

Переменные могут не декларироваться, однако для того, чтобы объявить глобальную переменную необходимо либо объявить её со служебным словом var, либо присвоить значение.

Пример, который вызовет ошибку:

```
function f1() {
  scope = "123"; // Переменная инициализируется здесь
  alert(scope);
}
function f2() {
  alert(scope);  // Переменная нигде не объявлена
}
```

Добавим переменную scope явно

```
var scope;
function f1() {
  scope = "123"; // Переменная инициализируется здесь
  alert(scope);
}
```

```
function f2() {
  alert(scope);
}

f2();// Показывает "undefined"
f1();// Показывает "123"
f2();// Показывает "123"
```

Зададим переменной scope тип строки:

```
var scope = "";
function f1() {
  scope = "123"; // Переменная инициализируется здесь
  alert(scope);
}
function f2() {
  alert(scope);
}

f2();// Показывает "" (пустая строка)
f1();// Показывает "123"
f2();// Показывает "123"
```

Основные операторы и инструкции

Язык Javascript имеет сходство с языками C++ и Java, поэтому в виду ограниченного объема изложения, подробно на них останавливаться не будем. Тем не менее, их следует изучить в главах 5 и 6 книги [1].

Объекты и классы

Язык Javascript является объектно-ориентированным языком, но не имеет классов в понимании классов C++ или Java.

Любая переменная может получить значение-объект. Если ни одна переменная не хранит ссылку на этот объект, он будет удалён. Каждый из объектов конструируется самостоятельно. При этом объекту могут быть динамически назначены свойства и методы.

Воспользуемся примерами из [1].

```
// Определяем конструктор.
// Обратите внимание, как инициализируется объект с помощью "this".
function Rectangle(w, h) {
  this.width = w;
  this.height = h;
}
// Вызываем конструкторы для создания двух объектов Rectangle.
// И проинициализируем оба новых объекта.
var rect1 = new Rectangle(2, 4); // rect1 = { width:2, height:4 };
var rect2 = new Rectangle(8.5, 11); // rect2 = { width:8.5, height:11 };
```

В роли конструктора объекта выступает функция `Rectangle`. При этом для добавления свойств объекта использован указатель this.

Добавить метод к конкретному объекту можно следующим образом:

```
r.area = function() { return this.width * this.height; }
// Теперь рассчитать площадь, вызвав метод объекта
var a = r.area();
```

Добавить метод так, чтобы объект получал его на этапе вызова конструктора можно следующим образом:

```
function Rectangle(w, h) {
  this.width = w;
  this.height = h;
  this.area = function( ) { return this.width * this.height; }
}
```

Указанный метод не является оптимальным. В данном примере созданный объект имеет три свойства (area – тоже свойство, имеющее после создания объекта функциональный тип), причём любое из них может принять любое значение. Это не соответствует концепции класса с постоянными методами. Для решения данной проблемы разработан механизм прототипов. Любой объект имеет указатель на прототип. В момент создания объекта устанавливает ссылка на конструктор прототипа, которым является функция-конструктор, создавшая объект. Ссылка на прототип объекта указывает на свойство prototype функции-конструктора. Методы и свойства, добавленные прототипу будут доступны любому объекту, созданному такой функцией-конструктором.

Пример создания класса Circle:

```
// Конструктор
function Circle(radius) {
  // r - свойство экземпляра объекта, оно определяется
  // и инициализируется конструктором.
  this.r = radius;
}
// Circle.PI - свойство класса (общее для всех объектов), свойство конструктора.
Circle.PI = 3.14159;

// Метод для экземпляра, который рассчитывает площадь круга.
// Аналог методов класса C++
Circle.prototype.area = function( ) { return Circle.PI * this.r * this.r; }

// Метод класса (аналог статических методов в C++)
// принимает два объекта Circle и возвращает объект с большим радиусом.
Circle.max = function(a,b) {
  if (a.r > b.r) return a;
  else return b;
}
// Примеры использования каждого из этих полей:
var c = new Circle(1.0); // Создание экземпляра класса Circle
c.r = 2.2; // Установка свойства экземпляра r
var a = c.area(); // Вызов метода экземпляра area()
var x = Math.exp(Circle.PI); // Обращение к свойству PI класса
var d = new Circle(1.2); // Создание другого экземпляра класса Circle
var bigger = Circle.max(c,d); // Вызов метода класса max()
```

Document Object Model (DOM) и программирование в браузере

Объектная модель документа DOM является программным интерфейсом для доступа и манипулирования документом (документ в терминологии SGML/HTML/XML). Координирующая роль в DOM принадлежит консорциуму W3C - http://www.w3.org/DOM/.

Напомним, что HTML-страница является документом со строгой разметкой. Модель DOM позволяет получить доступ, как к элементам разметки, так и свойствам, позволяющими управлять браузером (получить доступ к текущему документу, открыть окно с новым документом, получить данные из другого документа), а также установить обработчики событий на определенные действия мыши, клавиатуры, таймера и пр. Отдельно обрабатываются формы.

За прошедшие годы работы консорциума W3C выпустил следующие рекомендации группы
The Document Object Model Working Group:

- Document Object Model Level 1
- *Document Object Model Level 2 Core*
- *Document Object Model Level 2 Views*
- *Document Object Model Level 2 Events*
- *Document Object Model Level 2 Style*
- *Document Object Model Level 2 Traversal and Range*
- *Document Object Model Level 2 HTML*
- Document Object Model Level 3 Core
- Document Object Model Level 3 Load and Save
- Document Object Model Level 3 Validation.

Каждый следующий уровень (по номеру) заменяет предыдущий. При этом, браузер обязан поддерживать все предыдущие для обеспечения совместимости.

Основной принцип программирования с использованием JavaScript заключается в управлении свойствами браузера и манипулировании элементами документа, согласно модели DOM.

Базовый уровень функциональности документа обеспечивается объектами, поддерживаемыми даже самыми старыми браузерами. Эта иерархия объектов представляет объектную модель документов уровня 0 (Document Object Model level0 - DOM0), которая исторически появилась до официальных спецификаций W3C. Приведем схему из главы 13 из книги [1] см. рисунок 2.

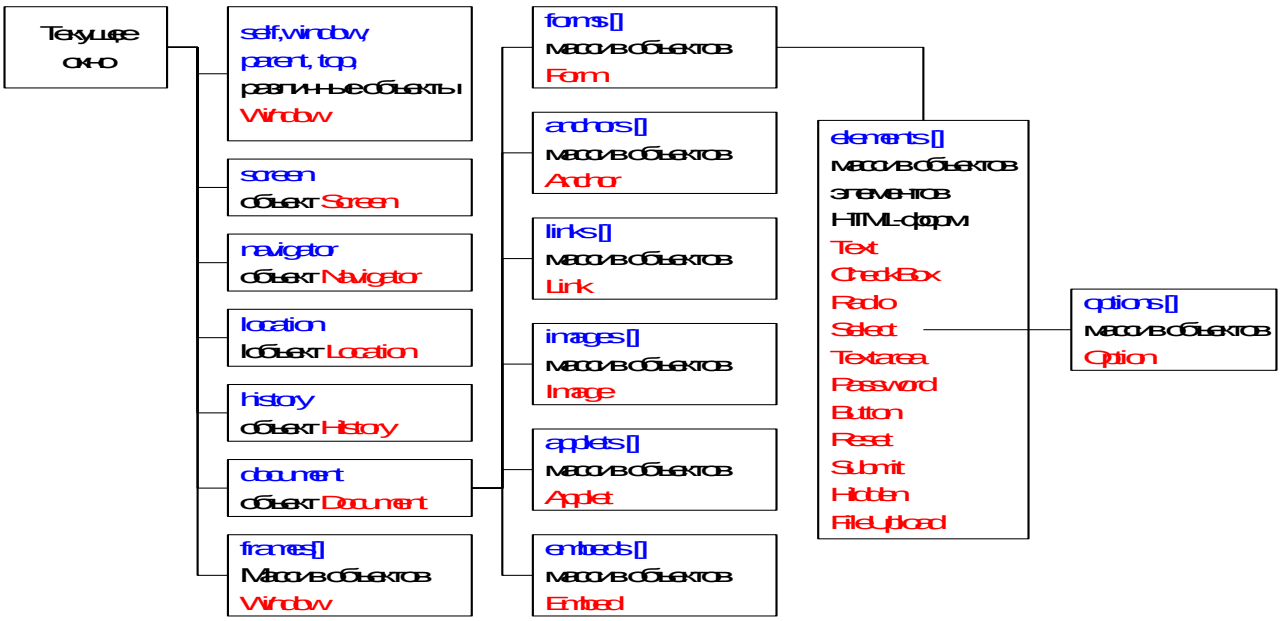

Рисунок 2 Модель DOM 0. Основные объекты.

Применительно к клиентскому Javascript, основным объектом является Window, который ссылается на текущее окно браузера. Остальные объекты[1], иерархия которых здесь представлена, являются свойствами корневого объекта Window. Почти все эти объекты имеют много полезных свойств, с ними связаны события и методы, использование которых позволяет создавать сценарии, обеспечивающие необходимую функциональность. Коротко отметим объекты верхнего уровня иерархии.

Объект Screen (свойство screen) позволяет прочитать разрешение клиентского экрана и глубину цвета. Определив разрешение экрана, можно предусмотреть разные варианты компоновки страницы, устанавливать размеры и положение новых окон, открывающихся из сценария. Методы для этого объекта не определены, но определен ряд свойств, среди которых:

- width – текущая ширина экрана в пикселах;
- height - текущая высота экрана в пикселах;
- availWidth - доступная ширина экрана в пикселах;
- availHeight - доступная высота экрана в пикселах.

Объект Navigator (свойство navigator) предоставляет информацию о версии браузера.

Объект Location (свойство location) предоставляет доступ к URL документа, отображаемого в окне браузера. Позволяет определить полный URL, а также его части: протокол, доменное имя и т.д. В отличие от двух предыдущих объектов, его свойства доступны не только для чтения, но и для изменения. То есть, в зависимости от выполнения условий, определенных в сценарии, можно

[1] См. http://vvz.nw.ru/Lessons/JavaScript/DOM0.htm

загрузить нужный документ как в текущее окно или его фрейм, так и в любое из окон, открытых из сценария. Этот объект имеет и два метода:

- reload() перезагружает указанный в качестве аргумента документ;
- replace() загружает указанный документ, который замещает текущий в списке истории просмотра.

Объект History (свойство history) имеет единственное свойство length (количество просмотренных в данном сеансе документов), и три метода, позволяющих перемещаться по истории просмотра:

- back() - на один шаг назад по истории просмотра;
- forward() - на один шаг вперед по истории просмотра;
- go(n) - на n шагов по истории просмотра (если n >0, то вперед, если n <0, то назад).

Объект Document (свойство document), его свойства и методы предоставляют наиболее богатые возможности для разработчика. Приведенная выше схема иерархии объектов включает только основные свойства этого объекта, определенные в базовой объектной модели документа (Document Object Model Level 0 - DOM0). Эти свойства также поддерживаются современными браузерами в рамках более новых моделей DOM.

Массив frames[] предоставляет доступ к документам, загруженным в фреймы.

Следует отметить, что разные браузеры, поддерживая рассматриваемую иерархию, предлагают и дополнительные свойства почти для каждого объекта. Однако использовать не рекомендуется использовать то, что не задекларировано в моделях DOM согласно спецификациям W3C, поскольку это потенциально приведет к проблемам совместимости с браузерами.

В качестве примера приведём фрагмент кода JavaScript из Wikipedia, позволяющий проверить заявленную поддержку различных расширений DOM в конкретном браузере.

```
   function domImplementationTest(){
    var featureArray = ['HTML', 'XML', 'Core', 'Views',
                        'StyleSheets', 'CSS', 'CSS2', 'Events',
                        'UIEvents', 'MouseEvents', 'HTMLEvents',
                        'MutationEvents', 'Range', 'Traversal'];
    var versionArray = ['1.0', '2.0', '3.0'];
    var i;
    var j;
    if(document.implementation && document.implementation.hasFeature){
      for(i=0; i < featureArray.length; i++){
        for(j=0; j < versionArray.length; j++){
          document.write(
            'Поддержка расширения '+ featureArray[i] + ' версии ' +
versionArray[j] + ' : ' +
            (document.implementation.hasFeature(featureArray[i],
versionArray[j]) ?
            '<font style="color:green">true</font>' : '<font
style="color:red">false</font>') + '<br/>'
```

```
            );
          }
          document.write('<br/>');
        }
      }
    }
```

Документацию по DOM-моделям и использованию в JavaScript следует смотреть по ссылкам: http://www.w3.org/standards/techs/dom#w3c_all и https://developer.mozilla.org/en/DOM_Levels .

Работа с документами

Работа с документами описывается в спецификациях DOM Core и является основой для динамического изменения отображаемых страниц.

Приведем пример из 15-й главы [1] следующей разметки:

```
<html>
<head>
  <title>Sample Document</title>
</head>
<body>
  <h1>An HTML Document</h1>
  <p>This is a <i>simple</i> document.
</body>
</html>
```

В соответствии с DOM-моделью, эта страница будет представлена деревом узлов, изображенном на рисунке 3.

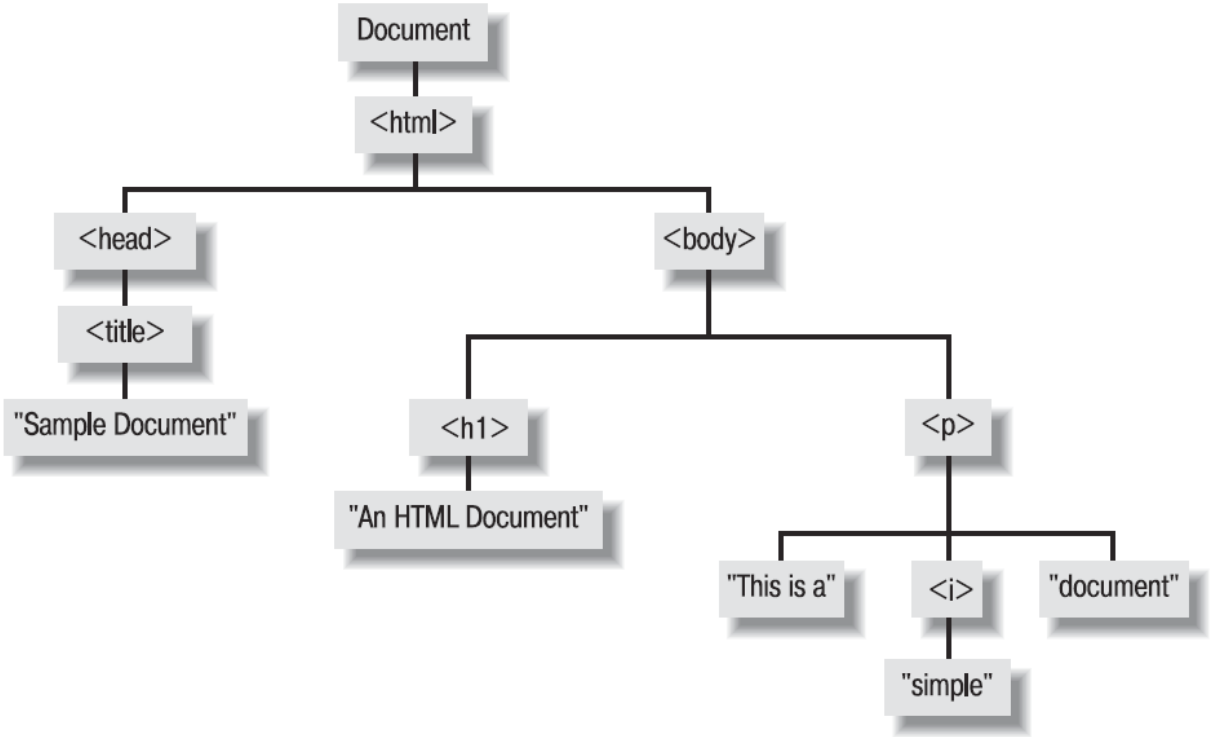

Рисунок 3 Дерево разбора HTML.

Каждый узел может иметь дочерние узлы. Для доступа к этим узлам существует интерфейс Node, который предоставляет свойства childNodes, firstChild, lastChild, nextSibling, previousSibling и parentNode, а также методы

appendChild(), removeChild(), replaceChild() и insertBefore(). Однако также возможно обратиться через свойство, имеющее имя узла. Например document.body.h1.

Каждый элемент этого дерева (т.е. объект Node) имеет определенный тип, который хранится в свойстве nodeType. Для справки некоторые из них приводятся в таблице 2.

Таблица 2 Основные типы узлов

Интерфейс	Константа nodeType	Значение nodeType
Element	Node.ELEMENT_NODE	1
Text	Node.TEXT_NODE	3
Document	Node.DOCUMENT_NODE	9
Comment	Node.COMMENT_NODE	8
DocumentFragment	Node.DOCUMENT_FRAGMENT_NODE	11
Attr	Node.ATTRIBUTE_NODE	2

Иерархия типов приводится на рисунке 4.

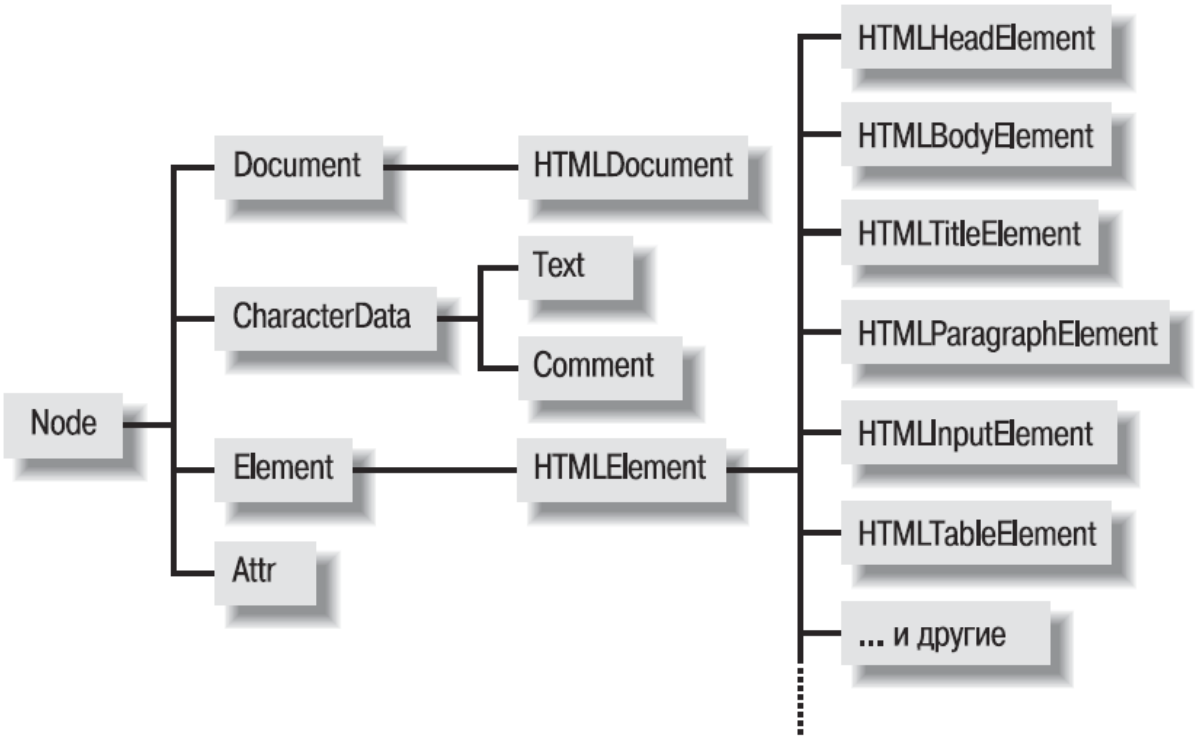

Рисунок 4 Иерархия типов узлов модели DOM.

В коде JavaScript программы предоставляется возможность поиска элементов по имени.

```
var tables = document.getElementsByTagName("table");
alert("Количество таблиц в документе: " + tables.length);
```

Если же нужен конкретный элемент, то либо необходимо знать его порядковый номер в массиве найденных элементов, либо иметь возможность идентифицировать по классу, либо идентификатору.

В следующем примере будет найден 4-й параграф

```
var myParagraph = document.getElementsByTagName("p")[3];
```

В этом примере будет найден параграф, имеющий уникальный идентификатор specialParagraph. Обратите внимание на то, что одинаковых идентификаторов в одном документе быть не может.

```
<p id="specialParagraph">
…
var myParagraph = document.getElementById("specialParagraph");
```

Более сложный пример манипулирования элементами списка:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Сортировка списка. Дэвид Флэнаган. JavaScript.</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script>
 function sortkids(e) {
      // Это элемент, потомки которого должны быть отсортированы
      if (typeof e == "string") e = document.getElementById(e);
      // Скопировать дочерние элементы (не текстовые узлы) в массив
      var kids = [];
      for(var x = e.firstChild; x != null; x = x.nextSibling)
            if (x.nodeType == 1 /* Node.ELEMENT_NODE */) kids.push(x);

      // Выполнить сортировку массива на основе текстового содержимого
      // каждого дочернего элемента. Здесь предполагается, что каждый
      // потомок имеет единственный вложенный элемент – узел Text
      kids.sort(function(n, m) { // Функция сравнения для сортировки
            var s = n.firstChild.data; // Текстовое содержимое узла n
            var t = m.firstChild.data; // Текстовое содержимое узла m
            if (s < t) return -1; // Узел n должен быть выше узла m
            else if (s > t) return 1; // Узел n должен быть ниже узла m
            else return 0; // Узлы n и m эквивалентны
      });
      // Теперь нужно перенести узлыпотомки обратно в родительский элемент
      // в отсортированном порядке. Когда в документ вставляется уже
      // существующий элемент, он автоматически удаляется из текущей позиции,
      // в результате операция добавления этих копий элементов автоматически
      // перемещает их из старой позиции. Примечательно, что все текстовые узлы,
      // которые были пропущены, останутся на месте.
      for(var i = 0; i < kids.length; i++) e.appendChild(kids[i]);
 }
</script>
</head>
<body>
      <ul id="list"> <!—Этот список будет отсортирован >
      <li>один<li>два<li>три<li>четыре<li>пять
      <!-- элементы не в алфавитном порядке -->
      </ul>
      <!-- кнопка, щелчок на которой запускает сортировку списка -->
      <button onclick="sortkids('list')">Сортировать по алфавиту</button>
</body>
</html>
```

Часто бывает полезно вставить элементы не как объекты, а как HTML текст. Для этого существует свойство `innerHTML`. Однако помните, что использование этого свойства полностью удалит все дочерние элементы, если они были у владельца `innerHTML`.

Пример заполнения заголовка в таблице у созданного элемента table:

```
var table = document.createElement("table"); // Создать элемент <table>
table.border = 1; // Установить атрибут
// Добавить в таблицу заголовок Имя|Тип|Значение
table.innerHTML = "<tr><th>Имя</th><th>Тип</th><th>Значение</th></tr>";
```

Обработка событий

Рассмотрим следующий код

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
	<title>Вывод структуры страницы.</title>
	<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
	<style>
		#result {font-style: italic;}
		div.text p { margin: 0 5px 5px 5px; background-color:yellow}
	</style>
	<script type="text/javascript">
		function print(el)
		{
			// По идентификатору находим элемент,
			// в который надо поместить результат
			var result = document.getElementById('result');
			result.innerHTML = "<hr/><p>"+Date()+"</p>" +
				"&lt;" + el.nodeName + "&gt;" + el.textContent +
				"<p>Вывод завершен!</p><hr/>";
		}
		function mouseOver(el)
		{
			el.style.backgroundColor = "green";
		}
		function mouseOut(el)
		{
			el.style.backgroundColor = "yellow";
		}
	</script>
</head>
<body>
	<div>
		<h1>События HTML</h1>
		<p>Нажмите на текст любого параграфа</p>
	</div>
	<div class="text">
		<p onclick="print(this)">1. Параграф внутри div !</p>
		<p onclick="print(this)">2. Параграф внутри div !!</p>
	</div>
	<div class="text">
		<p onclick="print(this)"
		   onMouseOut="mouseOut(this)" onMouseOver="mouseOver(this)">
		      3. Параграф внутри интересующего нас div !!!</p>
		<p onclick="print(this)"
		   onMouseOut="mouseOut(this)" onMouseOver="mouseOver(this)">
```

```
                4. Параграф внутри интересующего нас div !!!!</p>
        </div>

        <!-- Сюда поместим результат -->
        <div id="result"></div>
</body>
</html>
```

Элементы HTML имеют атрибуты, позволяющие назначить обработчики. В представленном примере для элементов <p> назначаются обработчики onclick (щелчок мышью по элементы), onMouseOver (появление указателя мыши над элементом), onMouseOut (выход указателя мыши за пределы элемента). Значение this, передаваемое обработчикам, содержит указатель на элемент, вызывающий этот обработчик.

В коде обработчиков для изменения цвета фона используются свойства, предоставленные DOM-моделью.

```
function mouseOver(el)
{
      el.style.backgroundColor = "green";
}
```

В обработчике щелчка мышью производится поиск элемента для вставки, а также извлекается содержимое инициировавшего его элемента:

```
function print(el)
{
      var result = document.getElementById('result');
      result.innerHTML = "<hr/><p>"+Date()+"</p>" +
            "&lt;" + el.nodeName + "&gt;" + el.textContent +
            "<p>Вывод завершен!</p><hr/>";
}
```

Объект, созданный вызовом Date(), содержит текущие дату и время.

В целом, в рамках модели DOM существуют обработчики, которые представлены в таблице 3. Подробнее см. [1].

Таблица 3 Модули и интерфейсы событий DOM

Имя модуля	Интерфейс Event	Типы событий
HTMLEvents	Event	abort, blur, change, error, focus, load, reset, resize, scroll, select, submit, unload
MouseEvents	MouseEvent	click, mousedown, mousemove, mouseout, mouseover, mouseup
UIEvents	UIEvent	DOMActivate, DOMFocusIn, DOMFocusOut
MutationEvents	MutationEvent	DOMAttrModified, DOMCharacterDataModified, DOMNodeInserted, DOMNodeInsertedIntoDocument, DOMNodeRemoved, DOMNodeRemovedFromDocument, DOMSubtreeModified

Средства отладки

В зависимости от того, предполагается ли отлаживать серверную или клиентскую части программ, используются различные средства отладки.

Чаще всего на JavaScript реализуют только клиентскую часть, поэтому используют средства отладки, интегрированные в браузер. Наиболее удобными средствами в данный момент являются расширение Firebug для Mozilla Firefox, а также модуль DragonFly браузера Opera, а также встроенный отладчик для Google Chrome. Для вспомогательных целей иногда используют встроенный отладчик Microsoft Internet Explorer.

В целом, доступны следующие способы отладки:

- Формирование диалоговых сообщений с текстом – вызов в коде программы функции `alert("Значение переменной...")`.
- Вывод отладочных сообщений в консоль браузера `console.log("..."),console.error("..."),console.warn("..."),console.info("...")`.
- Использование интерактивного отладчика.

Использование интерактивного отладчика в Mozilla Firefox рассмотрим более подробно. Для того чтобы воспользоваться отладчиком Firebug, необходимо иметь Mozilla Firefox и установленное расширение Firebug. Чтобы открыть отладчик на требуемой странице, нажмите клавишу F12.

В нижней части окна браузера появятся дополнительные окна. В Firebug доступны следующие закладки:

- Console – сообщения браузера об ошибках, предупреждения и отладочный вывод Javascript-программ.
- HTML – структура разметки страницы. Позволяет найти по отображенному элементу его соответствие в разметке и наоборот. Возможно изменение стилей любых элементов.
- CSS – позволяет редактировать таблицы стилей загруженной страницы.
- Script – Javascript код страниц. Позволяет просматривать код, устанавливать точки останова, выполнять пошаговую отладку и трассировку значений переменных. По-умолчанию выключено.
- DOM – просмотр и изменение значение документа по модели DOM.
- Net – просмотр данных, передаваемых между браузером и сервером. Предоставляется возможность просмотра заголовков HTTP, переданных данных, а также порядка передачи данных. По-умолчанию выключено.

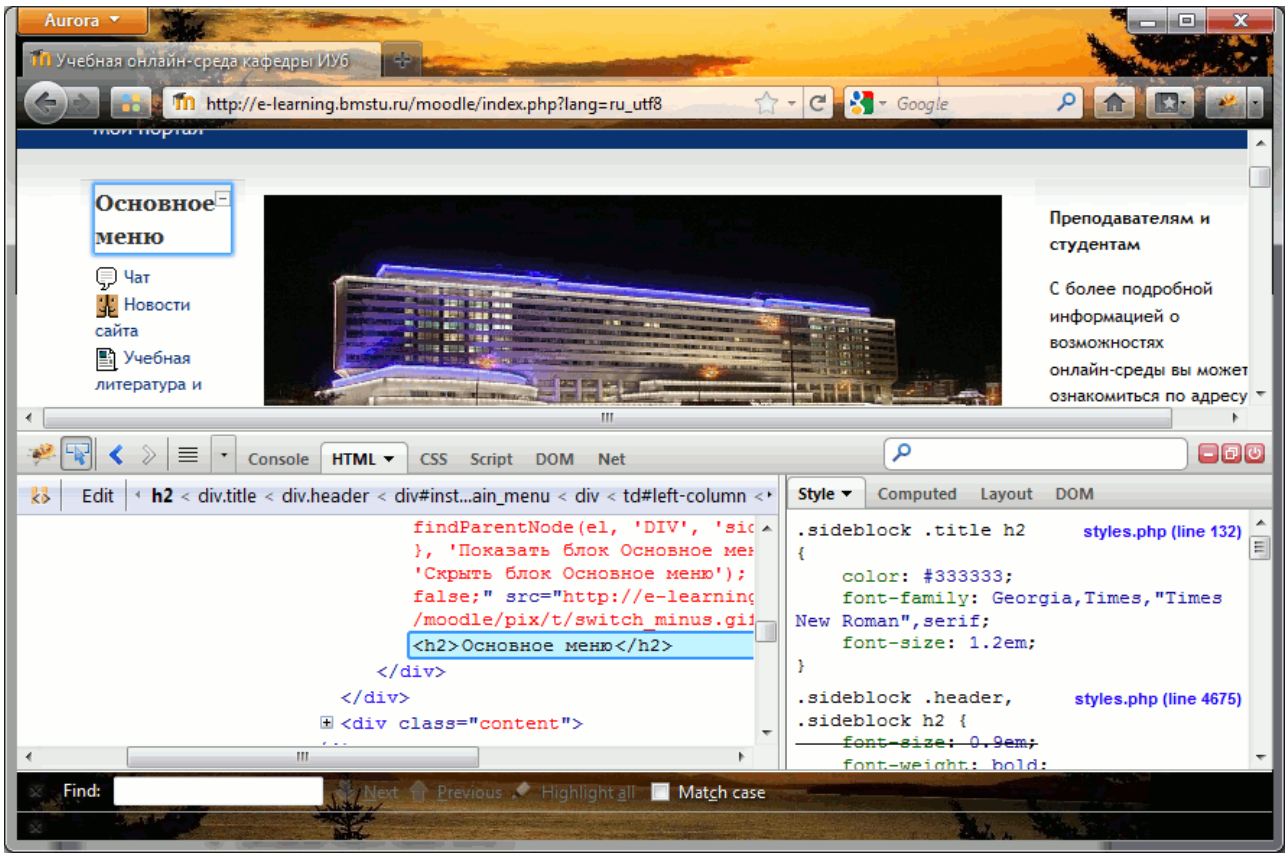

Рисунок 5 Просмотр структуры и стилей в Firebug.

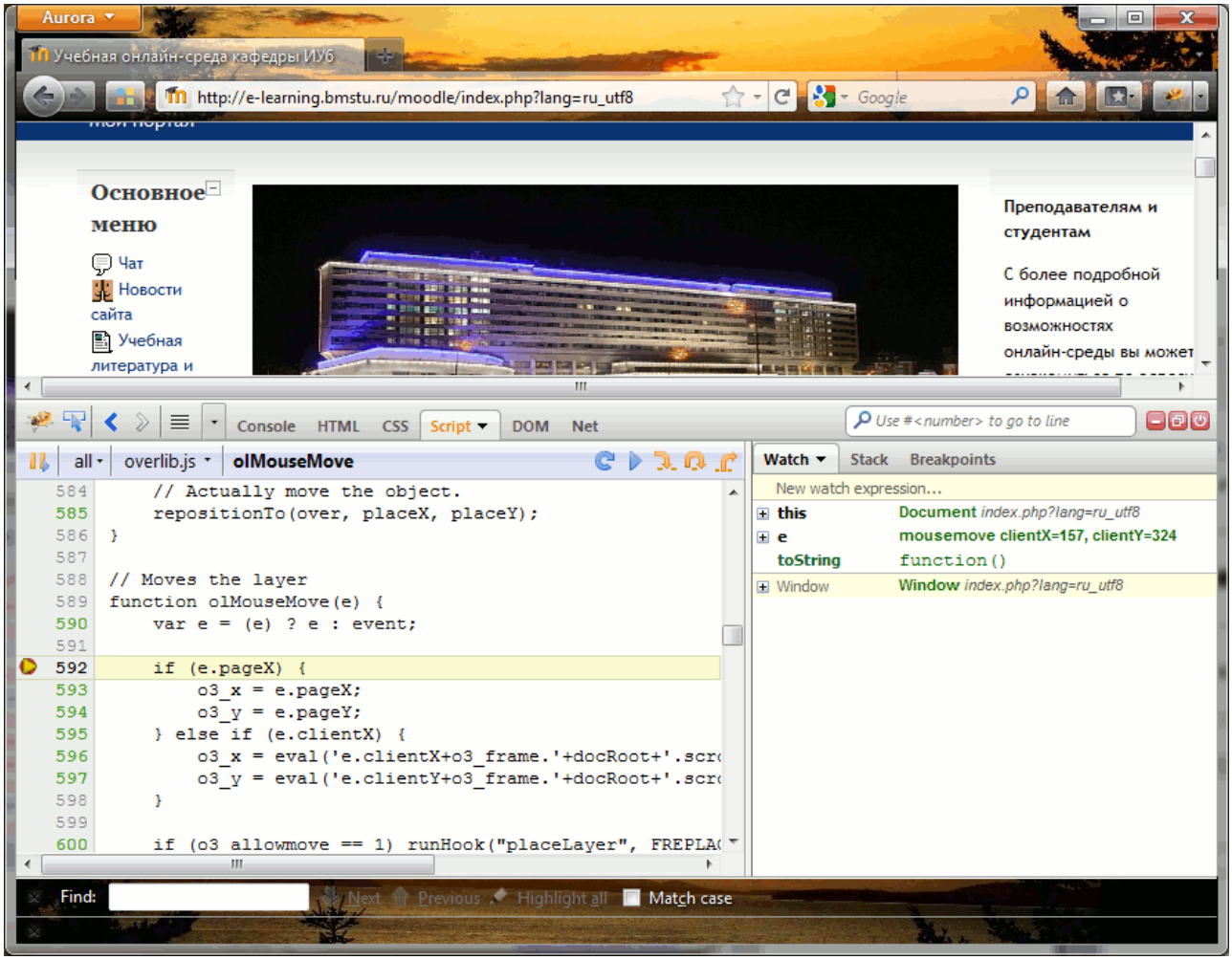

Рисунок 6 Отладчик Javascript в Firebug.

Литература

1. Флэнаган Д. JavaScript. Подробное руководство. – Пер. с англ. – СПб: СимволПлюс, 2008. – 992 с., ил.
2. Бер Бибо, Иегуда Кац. jQuery. Подробное руководство по продвинутому JavaScript. - Пер. с англ. - СПб.:Символ-Плюс, 2009. - 384 с., ил.
3. jQuery in Action. Серия: High Tech. Издательство: Символ-Плюс, 2009 г. Мягкая обложка, 384 стр.
4. Расселл М. Dojo. Подробное руководство – Пер. с англ. – СПб.: Символ-Плюс, 2009. – 560 с., ил.

5. Rawld Gill, Craig Riecke, Alex Russell. Mastering Dojo. JavaScript and Ajax Tools for Great Web Experiences. The Pragmatic Bookshelf, Raleigh, North Carolina Dallas, Texas, 2008.- 545 p.
6. Расселл М. Dojo. Подробное руководство – Пер. с англ. – СПб.: Символ-Плюс, 2009. – 560 с., ил.

Контрольные вопросы

1. Приведите основные характеристики языка Javasctipt.
2. Каким образом Javascript связан с HTML?
3. Что такое DOM?
4. Как обработать событие нажатия мыши на HTML-элемент?
5. Приведите способы позиционирования по узлам DOM средствами JavaScript.
6. Приведите способы отладки Javascript-приложений.

Задание для практикума №2

Написать Javascript-код для вывода дерева элементов страницы, с которой этот код запущен. Отступы для отображения формировать как символ

В процессе выполнения работы реализовать следующие пункты:

1. Сформировать страницу с произвольным кодом разметки, но обеспечить уровень вложенности внутри элемента <body> не менее 3.
2. добавить внутри элемента <body> секцию , предназначенную для вывода результата обхода дерева элементов страницы.
3. Выбрать способ активации рекурсивной программы обхода дерева элементов, реализовать и подключить эту программу.
4. При проходе по узлам разметки обеспечить отладочный вывод в консоль. Привести в отчете содержимое консоли.
5. Реализовать вывод на странице.

В отчете привести код страницы с программой обхода, отладочный вывод в консоль и примеры обхода дерева элементов страницы.

Задание для лабораторной работы №2

Разработать страницу для оформления заказа товаров.

1. Реализовать форму для ввода наименования и стоимости. При нажатии на кнопку «Добавить», добавить введенные значения в таблицу.
2. Сделать кнопку расчёта стоимости заказа.
3. Поставить обработчик на изменение фона ячеек таблицы со стоимостью в тот момент, когда над ними находится указатель мыши.
4. Подготовить страницу, содержащую таблицу товаров и их стоимости. На базе этой страницы обеспечить возможность выбора строк и добавления их в таблицу формирования заказа.

Примечание: в качестве усложненного варианта задания допустимо использовать библиотеки jQuery и Dojo.

В отчете привести код страницы с программой формирования таблицы, формы, отображаемые на экране и пример работы программы.

Приложение. Библиотеки jQuery. Dojo

Библиотека jQuery

jQuery – библиотека для упрощения манипулирования объектами. Позволяет существенно сократить объем кода за счет специальных объектом и методов. Основная документация доступна по адресу http://docs.jquery.com, а также в книге [2].
Библиотека не претендует на покрытие всех возможных функций, которые могут потребоваться программисту. В комплекте jQuery содержится лишь минимально необходимый набор для манипулирования с объектами и обеспечения AJAX-взаимодействия. Расширения, такие как дополнительные графические управляющие элементы, могут быть реализованы как плагины.

Основная идея в jQuery заключается в том, что выборка элементов осуществляется с помощью селекторов, аналогичных CSS с похожим, но расширенным синтаксисом. Отбор при помощи селектора осуществляется с помощью функций $() или jQuery().
Помимо доступа при помощи селекторов, предоставляется большое количество сервисных функций.

Пример. Необходимо выделить все четные строки таблицы:

```
<!DOCTYPE html>
<html>
<head>
  <style>
   table {
     background:#f3f7f5;
   }
  </style>
  <script src="http://code.jquery.com/jquery-latest.js"></script>
</head>
<body>
  <table border="1">
    <tr><td>Row with Index #0</td></tr>
    <tr><td>Row with Index #1</td></tr>
    <tr><td>Row with Index #2</td></tr>
    <tr><td>Row with Index #3</td></tr>
  </table>
  <script>$("tr:odd").css("background-color", "#bbbbff");</script>
</body>
</html>
```

Обратите внимание, что подключается версия jQuery непосредственно с сайта авторов библиотеки.
Результат представлен на рисунке 5.

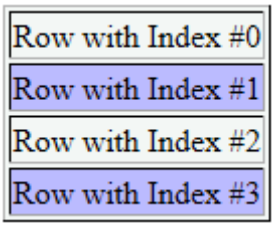

Рисунок 7 Пример таблицы с программно изменённым стилем строк.

В виду того, что концепция jQuery заключается в простоте замены свойств элементов существующей разметки, разработано большое количество графических плагинов. С некоторыми примерами можно ознакомиться по ссылке http://jqueryui.com/demos/ .

Следующий пример (http://jqueryui.com/demos/sortable/) демонстрирует возможности переопределения стандартного списка, обеспечивая возможность перемещения элементов мышью.

```
<!DOCTYPE html>
<html lang="en">
<head>
	<meta charset="utf-8">
	<title>jQuery UI Sortable - Default functionality</title>
	<script src="js/jquery.js"></script>
	<script src="js/ui/jquery-ui.js"></script>
	<link rel="stylesheet" href="js/demos.css">
	<style>
	 #sortable { list-style-type: none; margin: 0; padding: 0; width: 60%; }
	 #sortable li {
		margin: 0 5px 5px 5px; padding: 0.4em; padding-left: 1.5em;
		font-size: 1.4em; height: 18px; background-color:cyan }
	 #sortable li span { position: absolute; margin-left: -1.3em; }
	</style>
	<script>
	$(function() {
		$( "#sortable" ).sortable();
		$( "#sortable" ).disableSelection();
	});
	</script>
</head>
<body>
<div class="demo">

<ul id="sortable">
	<li class="ui-state-default">Item 1</li>
	<li class="ui-state-default">Item 2</li>
	<li class="ui-state-default">Item 3</li>
	<li class="ui-state-default">Item 4</li>
	<li class="ui-state-default">Item 5</li>
	<li class="ui-state-default">Item 6</li>
	<li class="ui-state-default">Item 7</li>
</ul>

</div><!-- End demo -->
</body>
</html>
```

Обратите внимание на то, что для того, чтобы список ul стал управляемым, необходимо лишь присвоить ему идентификатор (id="sortable"), который будет использован для переопределения элемента в специальной функции $().

```
	$(function() {
		$( "#sortable" ).sortable();
		$( "#sortable" ).disableSelection();
	});
```

Отметим, что в данном случае, функция $() получает в качестве аргумента функциональный литерал, а её назначение – гарантировать, что выполнение этого кода произойдёт после полной загрузки кода библиотеки jQuery.

Рассмотрим внедрение обработчиков. Для примера возьмем ту же задачу подсветки параграфов и реакции на нажатие на параграф, которая была рассмотрена с использованием «чистого» Javascript.

```
<!DOCTYPE html>
<html>
<head>
  <title>Вывод структуры страницы.</title>
  <meta charset="utf-8" />
  <style>
    #result {font-style: italic;}
    div.text p { margin: 0 5px 5px 5px; background-color:yellow}
  </style>
  <script src="js/jquery.js"></script>
  <script type="text/javascript">
    $(document).ready(function(){
      $("div.text p").mouseover(function(){
        this.style.backgroundColor = "green";
      }).mouseout(function(){
        this.style.backgroundColor = "yellow";
      }).click(function () {
        // По идентификатору находим элемент, в который надо поместить результат
        $("#result").html("<hr/><p>"+Date()+"</p>" +
          "&lt;" + this.nodeName + "&gt;" + this.textContent +
          "<p>Вывод завершен!</p><hr/>");
      })
    })
  </script>
</head>
<body>
  <div>
    <h1>События HTML</h1>
    <p>Нажмите на текст любого параграфа</p>
  </div>
  <div class="text">
    <p>1. Параграф внутри интересующего нас div !</p>
    <p>2. Параграф внутри интересующего нас div !!</p>
  </div>
  <div class="text">
    <p>3. Параграф внутри интересующего нас div !!!</p>
    <p>4. Параграф внутри интересующего нас div !!!!</p>
  </div>

  <!-- Сюда поместим результат -->
  <div id="result"></div>
</body>
</html>
```

Следует обратить внимание на то, что в этом случае HTML-разметка не содержит декларации обработчиков. Назначение обработчиков происходит из библиотеки jQuery при помощи соответствующих свойств DOM-модели, однако доступ к ним осуществляется через методы самой библиотеки.

Обратите внимание на то, что метод $(document).ready вызывается только после того, как браузером загружен весь код страницы и всех подключаемых js-файлов. Только после этого можно безопасно назначить обработчики. В соответствии с принципами jQuery их назначение производится через селектор `$("div.text p").mouseover(function(){…}`

Полезная статья об использовании jQuery: http://habrahabr.ru/post/149349/

Библиотека Dojo

Библиотека предназначена для полноценного программирования клиентских приложений. Включается в себя как средства манипулирования с объектами, так и средства для декларативного описания элементов страницы, а также средства для динамического создания графических управляющих элементов на странице.

Рассмотрим пример назначения обработчиков событий средствами Dojo.

```
<!DOCTYPE html>
<html>
<head>
  <title>Вывод структуры страницы.</title>
  <meta charset="utf-8" />
  <style>
    #result {font-style: italic;}
    div.text p { margin: 0 5px 5px 5px; background-color:yellow}
  </style>
  <script src="dojo/dojo.js"
          data-dojo-config='async: true, parseOnLoad: false'></script>
  <script type="text/javascript">
    require(["dojo/ready", "dojo/query"], function(ready, query){
      ready(function(){
        query("div.text p").forEach(function(node){
          node.onmouseover = function(){
            this.style.backgroundColor = "green";
          }
          node.onmouseout = function() {
            this.style.backgroundColor = "yellow";
          };
          node.onclick = function () {
          //По идентификатору находим элемент, куда следует поместить результат
            query("#result")[0].innerHTML = "<hr/><p>"+Date()+"</p>" +
              "&lt;" + this.nodeName + "&gt;" + this.textContent +
              "<p>Вывод завершен!</p><hr/>";
          }
        })
      })
    })
  </script>
</head>
<body>
  <div>
    <h1>События HTML</h1>
    <p>Нажмите на текст любого параграфа</p>
  </div>
  <div class="text">
    <p>1. Параграф внутри интересующего нас div !</p>
    <p>2. Параграф внутри интересующего нас div !!</p>
  </div>
  <div class="text">
    <p>3. Параграф внутри интересующего нас div !!!</p>
    <p>4. Параграф внутри интересующего нас div !!!!</p>
  </div>

  <!-- Сюда поместим результат -->
  <div id="result"></div>
</body>
</html>
```

Представленный пример соответствует рассмотренным ранее примерам подсветки и реакции на щелчок мышью для «чистого» Javascript и jQuery. Также как и в случае jQuery используется специальное средство **dojo/query**, гарантирующее загрузку браузером всех модулей. Также, как и в jQuery используется селектор элементов, однако назначение обработчиков производится через свойства DOM непосредственно, а не с использованием новых методов.

В части создания форм в Dojo принципиально то, что поддерживает два стиля программирования. Декларативный на основании разметки HTML, когда каркас для отображения уже задан и динамический, когда состав элементов формируется динамически в процессе выполнения Javascript-кода.
Приведём пример из учебника «Разработка HTML-виджетов с помощью Dojo. Игорь Кусаков»[2]:

Создание Dojo-виджета на основании разметки:

```
<html>
<head>
<script type="text/javascript"
      src="dojoAjax/dojo.js"></script>
<script type="text/javascript">
   dojo.require("dojo.widget.HelloWorld");
   dojo.require("dojo.widget.Button");
</script>
</head>
<body>
   <div dojoType="HelloWorld">
      <div dojoType="Button"></div>
   </div>
</body>
</html>
```

Код виджета dojoAjax/HelloWorld.js:

```
dojo.provide("dojo.widget.HelloWorld");
dojo.require("dojo.widget.HtmlWidget");

dojo.widget.defineWidget("dojo.widget.HelloWorld", dojo.widget.HtmlWidget, {
   isContainer: true,
   templateString: '<div><div dojoAttachPoint="containerNode">'
      + '</div></div>'
});
```

Виджет также модет быть создан с помощью метода `createWidget`:

```
<html>
<head>
<script type="text/javascript" src="dojoAjax/dojo.js"></script>
<script type="text/javascript">
   dojo.require("dojo.widget.HelloWorld");
   dojo.require("dojo.widget.Button");

   dojo.addOnLoad(function(){
      var button = dojo.widget.createWidget("Button",
```

[2] http://www.ibm.com/developerworks/ru/edu/wa-dojowidgets/section9.html

```
            {caption: "Submit"});
        dojo.widget.byId('someID').addChild(button);
    });

</script>
</head>
<body>
    <div id="someID" dojoType="HelloWorld">
    </div>
</body>
</html>
```

Динамический метод создания позволяет генерировать страницу на основе данных. Более подробно см. [3].

Дополнительные примеры см:

- http://demos.dojotoolkit.org/demos/
- http://docs.dojocampus.org/quickstart/index
- http://anton.shevchuk.name/wp-demo/dojo-for-beginners/
- http://www.ibm.com/developerworks/ru/edu/wa-dojowidgets/

В отличии от jQuery претендует на достаточность для решения большинства задач клиентского веб-программирования.

Отметим также, что существуют средства для графического создания форм на основе Dojo: http://www.wavemaker.com/