4. СОЗДАНИЕ ДИНАМИЧЕСКИХ ДОКУМЕНТОВ С ИСПОЛЬЗОВАНИЕМ DHTML

Цель: изучить основы динамического HTML, расширяющие возможности языка Javascript применительно к автоматизации вычислений.

Задачи:

- получить представление об объектной модели HTML-документа (HTML DOM);
- освоить способы взаимодействия программ-скриптов с элементами HTML-документа для ввода-вывода данных;
- получить представление об модели событий и связи событий с элементами HTML DOM;
- освоить работу с таблицами с использованием динамического HTML;
- научиться использовать библиотеки для вывода результатов в виде графиков.

В настоящем разделе рассматриваются необходимые нам понятия так называемого динамического HTML (Dynamics HTML — DHTML), используя средства которого код Javascript взаимодействует некоторым образом с элементами HTML-документа. Возможности, предоставляемые DHTML, как раз и превратили Javascript в полноценный универсальный язык программирования. Нас будут интересовать лишь те из них, которые необходимы будущему инженеру для решения своих задач вычислительного характера. Для желающих более глубоко и полно освоить DHTML можно найти много интересного в списке источников в конце раздела.

Говоря в общем, наша Javascript-программа, используя возможности DHTML, должна: прочитать значения со страницы, обработать их необходимым образом и вывести результат в нужное место страницы в текстовом, числовом или графической виде.

Содержание раздела

4.1. Объектная модель документа.

Основное новое понятие, дополнительно вводимое в DHTML к материалу, изученному в предыдущих разделах, - объектная модель документа (DOM — Document Object Model). Именно DOM является частью DHTML, связывающей HTML, CSS и Javascript.

После усвоения этого подраздела вы будете хорошо понимать: что такое DOM (вернее HTML DOM, потому что имеется, например, DOM и для XML - XML DOM); как можно использовать HTML DOM для перемещения по HTML–документу, чтобы точно найти то место, в котором необходимо получить данные или, наоборот, вывести данные, сделав изменения.

Объектная модель HTML-документа (или HTML DOM) - это представление объектов, методов, свойств и событий браузера в виде, удобном для работы с ними из кода программы или скрипта. Другими словами, HTML DOM — интерфейс, позволяющий получать, изменять, добавлять и удалять элементы HTML-документа с использованием языка программирования. Понятно, что в качестве языка программирования мы будем использовать Javascript.

Объектная модель документа реализована в виде объекта **document**. Этот объект появляется при загрузке HTML-кода в браузере и является, в свою очередь, дочерним объектом или свойством объекта **window**. Кроме объекта document объект window содержит ещё много других объектов (свойств) и методов. Перечислим некоторые из объектов, принадлежащих объекту window:

- event — позволяет получить информацию о каком-либо событии, происходящем в браузере;
- history — содержит информацию об истории текущего окна;
- location — содержит значение текущего URL-адреса текущего окна;
- screen — содержит информацию о характеристиках текущего окна.

Более подробно об объектах window можно узнать в многочисленных источниках (рекомендуется [7] и справочник [15, http://javascript.ru/window]). Мы же, перед рассмотрением объекта document, кратко рассмотрим объект location, который будем использовать в дальнейшем. Объект или свойство location, в свою очередь имеет следующие объекты (свойства):

- hash — часть URL, которая идёт после символа решётки «#», включая «#», например, #test;
- host — хост и порт, например: www.pvn.ho.ua:80;
- href — весь URL, например, при запросе программы на хосте www.pvn.ho.ua: http://pvn.ho.ua/pvn.org.ua/www/lection/example/example_5/5_5/examples_php/programm1.php?num=7&progr=1
- hostname — хост без порта, например: www.pvn.ho.ua;
- pathname - строка относительного пути (относительно хоста), например: /pvn.org.ua/www/lection/example/example_5/5_5/examples_php/programm1.php;

- `port` — номер порта, обычно: 80;
- `protocol` — протокол (`http:`);
- `searsh` — часть URL после символа «?», включая «?», например: `?num=7&progr=1`.

Мы будем использовать объект `window.location` в подразделе 5.5, а сейчас вернёмся к основному предмету изучения настоящего раздела — объекту `document`. Объектная модель (DOM) HTML-документа представляется в виде дерева узлов (DOM - древовидная модель), которыми являются элементы, атрибуты и текст.

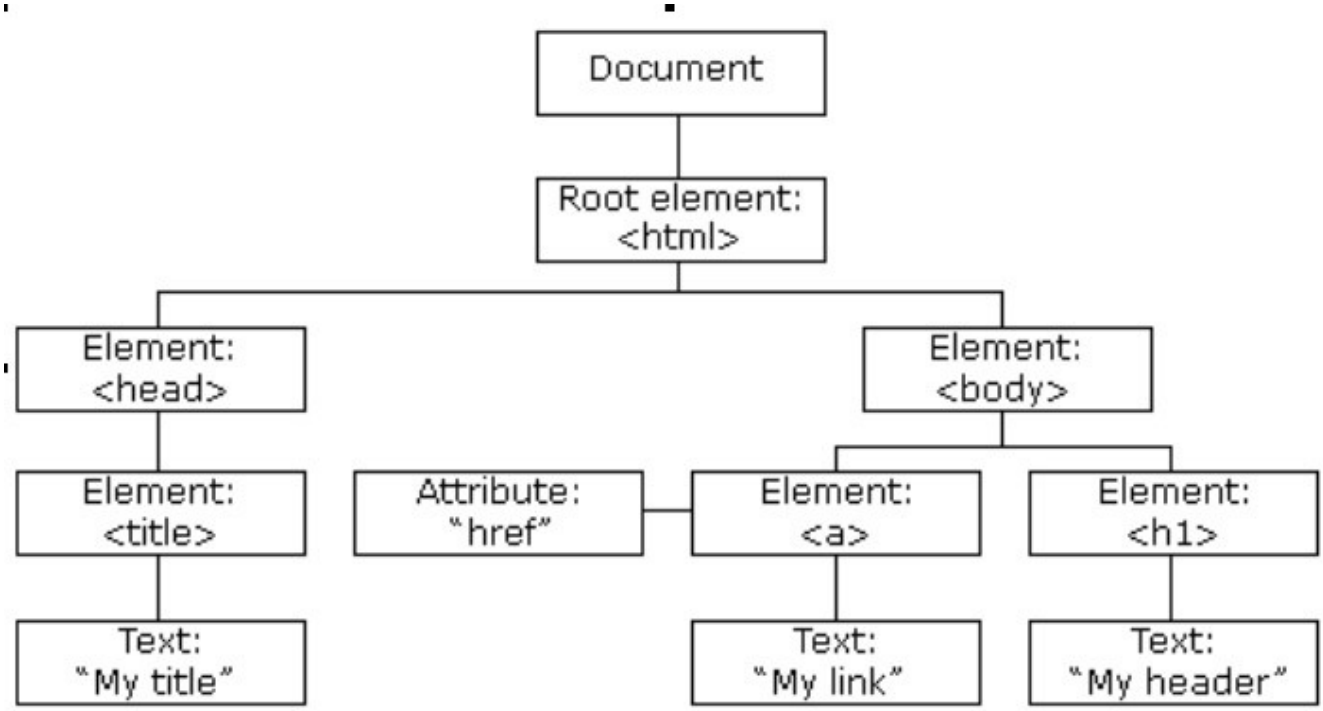


Рис. 4.1. Общий вид объектной модели простейшего HTML- документа

Это визуальное отображение `HTML`-документа в виде древовидной структуры, которая сжато представляет прямые отношения между элементами на странице, делая иерархию понятной. Можно заметить, что на схеме добавлен узел, помеченный как `document`, выше узла `html`. Этот автоматически добавляемый узел, представляющий весь документ в целом, — всегда существующая зацепка для `JavaScript` в документе. Этот узел, по существу является корнем, а не узел `HTML`, как указано на рис.4.1, хотя эта тонкость для нас является несущественной.

Объектная модель документа может иметь следующие типы узлов:

- документ — это корневой узел или элемент (root element) документа;
- каждый HTML-тэг - это узел или элемент (element) документа;
- каждый текст внутри HTML-тэгов — это текстовый (text) узел;
- каждый HTML-атрибут — это узел атрибута (attribute);
- каждый комментарий — это узел комментария.

Рассмотрим пример, приведённый в листинге 4.1.1.

Листинг 4.1.1

```
<html>
    <head>
        <title>Пример 4.1.1.</title>
        <meta charset=utf-8">
    </head>
    <body>
        <h2>Пример 4.1.1. Навыки работы с HTML DOM</h2>
        <p id='demo_p'>Текст <em>HTML</em>-документа</p>
    </body>
</html>
```

В этом примере корневым узлом является элемент `html`. У этого узла имеется два дочерних узла: `head` и `body`. Узел `head` содержит узел `title`, а узел `body` содержит узлы `h2` и `p`. Узел `title` содержит текстовый узел "Пример 4.1.1.". Узел `h2` содержит текстовый узел "Пример 4.1.1. Навыки работы с HTML DOM". Узел `p` содержит узел `em` и текстовые узлы "Текст" и "- документа". Узел `em` содержит текстовый узел "HTML".

Все узлы дерева находятся в иерархических отношениях между собой. Для описания этих отношений используются термины **родитель (или предок)**, **дочерний элемент** и **потомок.** Родительские узлы имеют дочерние узлы, а дочерние элементы одного уровня называются потомками (братьями или сестрами).

В отношении узлов дерева объектной модели соблюдаются следующие принципы:

- самый верхний узел дерева называется корневым;
- каждый узел, за исключением корневого, имеет ровно один родительский узел;
- узел может иметь любое число дочерних узлов;
- конечный узел дерева не имеет дочерних узлов;
- потомки имеют общего родителя.

Учитывая эти принципы пример листинга 4.1 можно описать так:

- `title` является потомком `head`;
- `body` имеет два потомка - элемент `h2` и элемент `p`;
- элемент `p` с `id`="demo_p" имеет своего собственного потомка - элемент `em`;
- текстовое содержимое элемента `p` также представлено в DOM как текстовые узлы. Они не имеют собственных потомков, но указывают содержащие их элементы как предков.

Для просмотра дерева объектной модели документа удобно пользоваться специальным дополнением `DOM Inspector` (инспектор объектов) к браузеру `Mozilla Firefox`, который можно найти по адресу: https://addons.mozilla.org/ru/firefox/addon/dom-inspector-6622/. Это анализатор веб-страниц, который выдаёт много информации о каждом элементе в HTML-странице и о связи каждого элемента с другими. Кроме этого, выбрав, например, любой `div`,

можно увидеть его рамку на самой странице.

Имеется подобное расширение и для браузера Google Chrome: https://chrome.google.com/webstore/detail/bmagokdooijbeehmkpknfglimnifench.

Рекомендую установить эти инструменты, так очень удобно ими пользоваться при сложной структуре HTML-элементов, в которых нужно будет разобраться. Затем можно запустить действующий пример из листинга 4.1.1, что можно сделать зайдя на сайт pvn.ho.ua (меню «Примеры» и — к примеру 4.1.1) или перейдя напрямую к примеру по ссылке: http://pvn.ho.ua/pvn.org.ua/www/lection/example/example_4/lstng_4_1_1.html. Если вы установили дополнение DOM Inspector в Firefox, то, запустив пример 4.1.1, например, в браузере Firefox, и, открыв его просмотр с помощью дополнения «DOM Inspector» («Инструменты»-«Веб-разработка»-«Инспектор DOM», вы сможете увидеть расположение всех узлов DOM для данного примера.

4.2. Доступ к элементам документа

Как уже упоминалось, `DOM` представляет собой интерфейс, включающий в себя набор стандартных **свойств** и **методов**, посредством которых обеспечивается доступ к узлам с помощью `JavaScript` или других языков программирования.

К наиболее часто используемым относятся следующие свойства `HTML DOM`:

- `x.innerHTML` - текстовое значение, содержащееся внутри элемента `x`;
- `x.nodeName` - имя элемента `x`;
- `x.nodeValue` - значение элемента `x`;
- `x.parentNode` - родительский узел элемента `x`;
- `x.childNodes` - дочерние узлы элемента `x`;
- `x.attributes` - узлы атрибутов элемента `x`;
- `x.firstChild` - первый дочерний узел элемента `x`;
- `x.lastChild` - последний дочерний узел элемента `x`;
- `x.previousSibling` - возвращает узел, предшествующий текущему узлу в списке `childNodes` узла-предка `x`;
- `x.nextSibling` - возвращает следующий узел в списке `childNodes` предка;
- `document.documentElement` - корневой узел документа;
- `document.body` - узел `body`;
- `x.style` - стили элемента `x`;
- `x.className` — имя класса стилей элемента `x`;
- `x.nodeType` — тип элемента `x`. Тип элемента выдаётся в виде кода:

Тип элемента	**Тип узла**
`Element`	1
`Attribute`	2

Text	3
Comment	8
Document	9

Поддерживаются следующие методы DOM:

- `x.getElementById(id)` - получить элемент с указанным `id`, содержащийся в элементе `x`;
- `x.getElementsByTagName(name)` - получить все элементы с указанным именем тэга `name`, содержащиеся в элементе `x`;
- `x.appendChild(node)` - вставить дочерний узел в элемент `x`;
- `x.removeChild(node)` - удалить дочерний узел из элемента `x`.

На примере листинга 4.1, для получения текста из элемента p со значением атрибута id "demo_p" в HTML-документе можно использовать следующий код:

```
text = document.getElementById("demo_p").innerHTML
```

Тот же самый результат может быть получен по-другому:

```
text=document.childNodes[0].childNodes[2].childNodes[3].innerHTML
```

Третий вариант нахождения нужного элемента и получения его содержимого - с помощью свойства `getElementsByTagName(name)`. Это свойство возвращает массив (нумерованный список) со всеми узлами с указанным наименованием тэга. Первый элемент в массиве имеет нулевой индекс. Массив возвращается даже, если имеется всего один элемент с таким именем. После этого, извлечь содержимое узла можно, обратившись к элементам полученного массива, так:

```
x = document.getElementsByTagName("p");
text = x[0].innerHTML
```

или так:

```
text = document.getElementsByTagName("p")[0].innerHTML;
```

Таким образом, в рамках DOM возможны 3 способа доступа к узлам:

1. С помощью метода `getElementById(ID)`. При этом возвращается элемент с указанным `ID`.
2. Путем перемещения по дереву с использованием отношений между узлами.
3. С помощью метода getElementsByTagName(name). При этом возвращаются все узлы с указанным именем тэга (в виде индексированного списка). Первый элемент в списке имеет нулевой индекс.

Наиболее просто использовать первый способ, но для этого надо именовать тэги с помощью атрибута `id`. Можно, конечно, использовать гибрид из перечисленных способов.

Ниже показан листинг примера поиска нужных тэгов перечисленными выше способами

и извлечения их содержимого. Кроме этого, во второй части этого листинга приведены три примера поиска узла em и извлечения его содержимого.

Листинг 4.2.1

```
<html><head>
<title>Пример 4.2.1</title>
<meta http-equiv=Content-Type content="text/html; charset=utf-8">
</head>
<body>
    <h2>Пример 4.2.1. Навыки работы с HTML DOM: поиск узлов и замена
текста</h2>
    <p id='demo_p'>Текст <em id='demo_em'>HTML</em>-документа</p>
    <hr>
<script>
// Способ 1
var text1 = document.getElementById("demo_p").innerHTML
document.write("<p>"+text1+" - получено первым способом</p>");
// Способ 2
var doc_child_nodes = document.childNodes[0].childNodes;
out1:
for (var k = 0; k < doc_child_nodes.length; k++){
    if (doc_child_nodes[k].nodeName == "BODY"){
     var body_child_nodes = doc_child_nodes[k].childNodes;
        for (var i = 0; i < body_child_nodes.length; i++){
            if (body_child_nodes[i].nodeName == "P"){
     var p_child_nodes = body_child_nodes[i].childNodes;
              for (var j = 0; j < p_child_nodes.length; j++){
                  if (p_child_nodes[j].nodeName == "EM") {
                        var text2 = body_child_nodes[i].innerHTML;
          break out1;
         }
     }
         }
     }
    }
}
document.write("<p>"+text2+" - получено вторым способом</p>");
// Способ 3
var p_nodes = document.getElementsByTagName("p");
out2:
for (var i = 0; i < p_nodes.length; i++) {
    var p_child_nodes = p_nodes[i].getElementsByTagName("em");
    for (var j = 0; j<p_child_nodes.length; j++){
        var text3 = p_nodes[i].innerHTML;
     break out2;
    }
}
document.write("<p>"+text3+" - получено третьим способом</p><hr>");
// Ещё один пример с использованием способа 1
var text1 = document.getElementById("demo_em").innerHTML
document.write("<p>"+text1+" - текст в узле em получен первым
способом</p>");
// Ещё один пример с использованием способа 2
for (var k = 0; k < doc_child_nodes.length; k++)
    if (doc_child_nodes[k].nodeName == "BODY")
        for (var i = 0; i < body_child_nodes.length; i++)
```

```
            if (body_child_nodes[i].nodeName == "P")
              for (var j = 0; j < p_child_nodes.length; j++)
                  if (p_child_nodes[j].nodeName == "EM")
                        var text2 = p_child_nodes[j].innerHTML
document.write("<p>"+text2+" - текст в узле em получен вторым
способом</p>");
// Ещё один пример с использованием способа 3
var text3 = document.getElementsByTagName("em")[0].innerHTML;
document.write("<p>"+text3+" - текст в узле em получен третьим
способом</p>");
</script>
</body></html>
```

В примере, при использовании второго способа обращения к узлам, индекс нужного нам узла определить достаточно сложно, так как в разных браузерах объектная модель может различаться и индексы узлов могут не совпасть. Такое различие получается, во-первых, из-за того, что в браузерах по разному трактуются пробелы — одни браузеры создают из пробелов текстовые узлы, другие — нет. Во-вторых, объектная модель не остаётся постоянной. В нашем примере узлы `p` появляются в модели сразу же при вставке их методом `document.write()`.

Ещё одна причина изменения объектной модели может быть вызвана тем, что обычно при использовании бесплатного хостинга хостер имеет право размещать в начале веб-страницы рекламный баннер внутри различных тэгов `<div>`, `<p>` и других тэгов, предсказать появление которых заранее мы не можем. Такое можно наблюдать на сайтах, размещаемых на бесплатном сервере ho.ua (например, на http://pvn.ho.ua) .

Поэтому обычно для поиска нужных узлов вторым способом используются различные алгоритмы. Один из таких алгоритмов показан в примере. Здесь вторым методом ищется первый узел `p`, в котором имеется узел `em`. Как только найден первый узел с таким совпадением, происходит выход из циклов с использованием оператора `break` с меткой. Меткой помечен внешний цикл, поэтому выход происходит из всех вложенных циклов.

Посмотреть пример 4.2.1 в действии можно на сайте pvn.ho.ua (меню «Примеры» и — к примеру 4.2.1) или перейдя напрямую к примеру по ссылке: http://pvn.ho.ua/pvn.org.ua/www/lection/example/example_4/lstng_4_2_1.html.

Для упрощения работы с узлами создано много специальных библиотек для `Javascript`. Наиболее часто используется библиотека `jQuery` [26, http://ru.wikipedia.org/wiki/JQuery].

Понятно, что можно не только извлекать содержимое узла, но и вносить что угодно в узел (дополнять, удалять и изменять). В следующем примере показано, как можно динамически изменять текстовое содержимое тэга `<p>`:

Листинг 4.2.2

```
<title>Пример 4.2.2</title>
```

```
<meta http-equiv=Content-Type content="text/html; charset=utf-8">
</head>
<body>
    <h2>Пример 4.2.2. Замена текстового содержимого узла</h2>
    <p id="demo_p">Текст <em>HTML</em>-документа</p>
<script>
var text_new1 = prompt("Ведите новый текст",
"<em>Новый текст для способа 1</em>");
// Замена текста при поиске узла способом 1
document.getElementById("demo_p").innerHTML = text_new1
+ " - замена способом 1";
// Замена текста при поиске узла способом 2
var text_new2=prompt("Ведите новый текст",
"<em>Новый текст для способа 2</em>")
var doc_child_nodes = document.childNodes[0].childNodes;
out1:
for (var k = 0; k < doc_child_nodes.length; k++){
    if (doc_child_nodes[k].nodeName == "BODY"){
      var body_child_nodes = doc_child_nodes[k].childNodes;
         for (var i = 0; i < body_child_nodes.length; i++){
             if (body_child_nodes[i].nodeName == "P"){
      var p_child_nodes = body_child_nodes[i].childNodes;
                  for (var j = 0; j < p_child_nodes.length; j++){
                       if (p_child_nodes[j].nodeName == "EM") {
                       body_child_nodes[i].innerHTML
                        = text_new2 + " - замена способом 2";
                        break out1;
                        }
                   }
              }
          }
    }
}
// Замена текста при поиске узла способом 3
var text_new3=prompt("Ведите новый текст",
"<em>Новый текст для способа 3</em>");
var p_nodes = document.getElementsByTagName("p");
out2:
for (var i = 0; i < p_nodes.length; i++) {
    var p_child_nodes = p_nodes[i].getElementsByTagName("em");
    for (var j = 0; j<p_child_nodes.length; j++){
        p_nodes[i].innerHTML = text_new3 + " - замена способом 3";
         break out2;
    }
}
</script>
</body></html>
```

Посмотреть пример 4.2.2 в действии можно на сайте pvn.ho.ua (меню «Примеры» и — к примеру 4.2.2) или перейдя напрямую к примеру по ссылке: http://pvn.ho.ua/pvn.org.ua/www/lection/example/example_4/lstng_4_2_2.html.

Ещё один пример, размещённый в листинге 4.2.3, демонстрирует возможность динамического изменения стилей с использованием свойства `style`. В примере при обновлении страницы изменяются фоновые цвета тэгов <h2> и <p>.

Листинг 4.2.3

```
<html><head>
<title>Пример 4.2.3.</title>
<meta http-equiv=Content-Type content="text/html; charset=utf-8">
</head>
<body>
     <h2>Пример 4.2.3. Замена стилей (фонового цвета) дочерних элементов
узла body</h2>
    <p>Текст <em>HTML</em>-документа</p>
<script>
x=document.body.childNodes;
for (var i = 0; i<x.length; i++) {
    if ((x[i].nodeName == "H2")||(x[i].nodeName == "P")) {
      r = Math.round(255.0*Math.random()).toString(16);
      g = Math.round(255.0*Math.random()).toString(16);
      b = Math.round(255.0*Math.random()).toString(16);
      x[i].style.backgroundColor ="#"+r+g+b;
    }
}
</script>
</body></html>
```

Как видно из примера, количество найденных дочерних узлов-потомков определяется с помощью свойства `length`, поскольку строка `x=document.body.childNodes` возвращает массив узлов-потомков узла `body`. Стили задаются только лишь для узлов `h2` и `p`. Выбираются эти узлы среди всех дочерних узлов с помощью проверки на равенство наименования узла, причём на этом примере можно убедиться, что наименование узла всегда задаётся в верхнем регистре.

Каждая компонента цвета задаётся с помощью датчика случайных чисел с преобразованием получаемых числовых значений в шестнадцатиричный формат и с представлением в виде строки. Результирующий rgb-цвет получается путём конкатенации полученных строк и присвоения полученного цвета правилу стиля, определяющему фоновый цвет элемента. Из примера видно, как формируются значения для правил стиля: если CSS-правило имеет дефис, то для установки `style` нужно убрать дефис и заменить на верхний регистр первый символ следующего после дефиса слова. Например, для установки свойства `background-color` в значение `#f36` нужно написать:

```
element.style.backgroundColor="#f36";
```

а свойства `z-index` в 2:

```
element.style.zIndex = 2
```

Посмотреть пример 4.2.3 в действии можно на сайте pvn.ho.ua (меню «Примеры» и — к примеру 4.2.3) или перейдя напрямую к примеру по ссылке: http://pvn.ho.ua/pvn.org.ua/www/lection/example/example_4/lstng_4_2_3.html.

4.3. Объектная модель и модель событий

Основное преимущество языка Javascript перед другими языками - его тесная интеграция с документом. Любой документ или DOM-элемент умеет инициировать различные события, а на событие можно назначить обработчик.

События и обработчики события уже рассматривались в предыдущих разделах при изучении HTML и язака Javascript.

Вспоминая полученные ранее знания, пример события с обработчиком может быть написан в следующем виде:

```
<script>
    function myFunction () {
        alert('Спасибо')
    }
</script>
<input onclick="myFunction ()" type="button" value="Ok"/>
```

В этом примере вызов обработчика осуществляется путём вставки в тэг атрибута при наборе текста HTML-документа «вручную». Таким образом можно создавать документы с относительно несложными программами.

В этом подразделе будет рассмотрена возможность создавать и обрабатывать события иначе: программным способом с использованием объектной модели и объекта event.

Event - это объект, содержащий много информации о событии и узлах-элементах объектной модели, с которыми связано это событие. Эту информацию можно передавать в программу-обработчик и создавать более насыщенные приложения, чем это мы делали ранее, когда не знали о динамическом HTML.

Существует два способа передачи этого объекта обработчику и они зависят от браузера:

- в браузерах, работающих по рекомендациям W3C, объект события всегда передается в обработчик первым параметром;
- в Internet Explorer существует глобальный объект **window.event**, который хранит в себе информацию о последнем событии и всегда доступен. А первый параметр обработчика не используется.

Код для кросс-браузерного способа получить объект события может иметь такой вид:

```
function doSomething(event) {
    event = event || window.event
    // Теперь event - объект события во всех браузерах.
}
element.onclick = doSomething
```

При описании обработчика события в HTML-разметке для получения события можно использовать переменную с названием event, причём это можно делать одинаково во всех браузерах, в чём можно убедиться, например, с помощью этого кода:

```
<input type="button" onclick="alert(event.type)" value="Ok"/>
```

Прежде, чем использовать объект **event**, приведём краткие сведения о некоторых свойствах этого объекта. При этом необходимо понимать, что свойства, содержащиеся в объекте `event`, различаются для разных событий.

При возникновении, например, событий `click` (щелчок кнопкой мыши) или `mousedown` (нажата кнопка мыши), объект `event` содержит:

- тип события в свойстве `event.type` (в данном случае — `Click` или `MouseDown`),
- координаты `x` и `y` курсора относительно окна в момент возникновения события в свойствах `event.clientX` и `event.clientY`;
- координаты `x` и `y` курсора относительно документа (с учётом прокрутки) в момент возникновения события в свойствах `event.pageX` и `event.pageY`;
- число, представляющее нажатую клавишу мыши в свойстве `event.button`: 0 - нажата левая, 1 - нажата средняя, 2 - нажата правая. Для MS IE эти числа другие: 1 - нажата левая, 4 - нажата средняя, 2 - нажата правая,
- код, клавиши-модификатора (`Control`, `Alt`, `Shift`), которая была нажата в момент события, содержащийся в свойствах `event.ctrlKey`, `event.altKey`, `event.shiftKey`.
- в каком элементе произошло событие: в свойстве `event.target` для браузеров, работающих по рекомендациям W3C, или `event.srcElement` для MS IE.

Итак, объект `event` содержит свойства, которые описывают событие и могут быть переданы в качестве аргументов обработчику события при возникновении события.

Рассмотрим установку функции-обработчика через свойство `on`*`событие`* соответствующего узла. Этот способ будет работать в любом браузере с поддержкой `JavaScript`. Для этого нужно:

1) получить нужный узел;
2) назначить обработчик свойству on+имя.

Вот так, например, можно установить обработчик события `click` на элемент с `id="myElement"`:

```
<script>
    document.getElementById('myElement').onclick = function() {
        alert('Спасибо')
    }
</script>
<input id="myElement" type="button" value="Ok"/>
```

А можно вот так:

```
<script>
    function myFunction () {
        alert('Спасибо')
    }
    document.getElementById('myElement').onclick = myFunction
</script>
<input id="myElement" type="button" value="Ok"/>
```

или так:

```
<script>
    function myFunction () {
        return alert('Спасибо')
    }
    document.getElementById('myElement').onclick = myFunction()
</script>
<input id="myElement" type="button" value="Ok"/>
```

Обратите внимание! В приведённых выше трёх примерах вызываемая обработчиком функция имеет особенности:

- не имеет имени;
- не имеет скобок;
- вызывается по имени функции со скобками (как мы обычно вызывали функцию ранее).

В первом случае определяемая функция называется анонимной функцией, потому что мы не объявляем её имя и не можем вызвать её иначе, чем через обработчик `onClick` в определённом месте.

Второй пример показывает, что объявление функции присваивается переменной, совпадающей с её именем. Поэтому с Javascript-функцией можно работать, как с обычной переменной. Это утверждение можно доказать, например, проведя следующий простой эксперимент: `alert(myFunction)`. В окно будет выведен код функции, а не результат её работы. Если во втором примере свойству `onclick` присвоить имя функции со скобками (`document.getElementById('myElement').onclick = myFunction()`), то таким образом мы запустим функцию на выполнение, а так как оператора `return` в её коде нет, то этот результат будет равен `undefined`.

В третьем примере мы вызываем функци как обычно (с именем и скобками), но при этом добавили `return` в описании функции.

Далее очень важно понять реализованную в браузерах модель поведения (механизм) события в связи с объектной моделью. Существуют два механизма: «**всплывания событий**» и «**перехват событий**». Понятие «перехват событий», сейчас редко используется, а желающие могут узнать о нём в самоучителе [15,

http://javascript.ru/tutorial/events/properties]. Ниже — о всплывании событий.

Механизм всплывания событий обеспечивает способность, при которой на одно событие может реагировать не только тот элемент, на котором произошло событие, но и элементы над ним. Это очень удобно, например, если в элементе содержатся много дочерних HTML-элементов - не обязательно ставить обработчик на каждый, достаточно указать один обработчик на родителе и в нем ловить все события. Или, можно создавать сложные реакции на события, присваивая в каждом вложенном элементе различные обработчики событию с одни и тем же именем.

Рассмотрим пример, когда имеется три элемента, вложенных друг в друга.

```
<div class="d1" onclick=alert(1)>1<!-- верхний элемент DOM-->
    <div class="d2" onclick=alert(2)>2
       <div class="d3" onclick=alert(3)>3</div><!-- нижний элемент DOM -->
    </div>
</div>
```

Возникает вопрос: если на каждом из них будет свой обработчик события, например, `onclick`, то обработчик для какого элемента будет вызван первым при клике, скажем, на `d3`?

В модели всплывания событий сначала будет выполнен обработчик на элементе 3, затем на элементе 2, и последним будет выполнен обработчик на элементе 1. Такой порядок называется "всплывающим", потому что событие поднимается с самых "глубоких" элементов в представлении DOM, к самым "верхним", как пузырек воздуха в воде.

По умолчанию "всплытие" происходит всегда. При возникновении события на элементе, сигнал будет подниматься до самого высокого элемента, выполняя имеющиеся обработчики. Однако с помощью любого обработчика можно остановить всплытие и не выпускать событие дальше вверх. Пример такого кода помещён ниже:

```
element.onclick = function(event) {
    event = event || window.event // для кросс-браузерности
    if (event.stopPropagation) {
       // Вариант стандарта W3C:
       event.stopPropagation()
    } else {
       // Вариант Internet Explorer:
       event.cancelBubble = true
    }
}
```

Полученные в этом подразделе знания о модели событий закрепим на следующем примере из листинга 4.3.1.

Листинг 4.3.1

```
<html><head>
<title>Пример 4.3.1.</title>
<meta http-equiv=Content-Type content="text/html; charset=utf-8">
<style type="text/css">
    * {
     margin: 0;
     padding: 0;
     text-align: center;
    }
  body {
     background-color: #eee;
    }
    body h2, button {
     margin: 20px;
    }
    #win4 {
     font-weight: 700;
     font-size: 400%;
     background-color: #ddd;
     position: relative;
     margin: 0 auto;
     width: 60%;
     height:300px;
    }
    #win3 {
     font-size: 50%;
     background-color: #bbb;
     position: absolute;
     width: 300px;
     height:150px;
     left: 50%;
     top: 50%;
     margin: -75px 0 0 -150px;
    }
    #win2 {
     font-size: 50%;
     background-color: #999;
     position: absolute;
     width: 150px;
     height:75px;
     left: 50%;
     top: 50%;
     margin: -37.5px 0 0 -75px;
    }
    #win1 {
     font-size: 50%;
     background-color: #777;
     position: absolute;
     width: 75px;
     height:37.5px;
     left: 50%;
     top: 50%;
```

```
      margin: -18.75px 0 0 -37.5px;
    }
</style>
</head>
<!--<body onclick="alert('Произошло событие в узле body')">-->
<body>
<h2>Пример 4.3.1. Вызов обработчиков во вложенных тэгах &lt;div&gt; при всплывании события</h2>
                <button    onclick="document.getElementById('win1').onclick = myFunction1">
      Этой кнопкой создаётся обработчик для события Click в окне №1
    </button>
<!--
<div id='win4'  onclick='myFunction4()'>Окно №4
    <div id='win3'  onclick='myFunction3()'>Окно №3
      <div id='win2' onclick='myFunction2()'>Окно №2
-->
<div id='win4'>Окно №4
    <div id='win3'>Окно №3
      <div id='win2'>Окно №2
          <div id='win1'>Окно №1
          </div>
      </div>
    </div>
</div>
<script>
    function myFunction4 (evnt) {
        var ev4 = evnt || window.event // для кросс-браузерности
      var x=ev4.clientX
      var y=ev4.clientY
      alert('Событием пройдены:\n'+this.innerHTML+'\nКоординаты клика:\nx = '+x+'\ny = '+y)
    }
    function myFunction3 () {
      alert('Событием пройдены:\n'+this.innerHTML+'\nx = '+x+'\ny = '+y)
      document.getElementById('win4').onclick = myFunction4
    }
    function myFunction2 () {
      alert('Событием пройдены:\n'+this.innerHTML+'\nКоординаты клика:\nx = '+x+'\ny = '+y)
      document.getElementById('win3').onclick = myFunction3
    }
    function myFunction1 (evn) {
        var ev1 = evn || window.event // для кросс-браузерности
      x=ev1.clientX
      y=ev1.clientY
      alert('Событием пройдены:\n'+this.innerHTML+'\nКоординаты клика:\nx = '+x+'\ny = '+y)
      document.getElementById('win2').onclick = myFunction2
    }
</script>
</body>
</html>
```

В примере листинга 4.3.1 показан код HTML-документа, состоящего из четырех тэгов `<div>`. Во внешний тэг `<div>` с `id='win4'` вложен тэг `<div>` с `id='win3'` и так далее, так что создаётся эффект вложенных окон. В каждом тэге имеется текст, соответствующий

номеру окна.

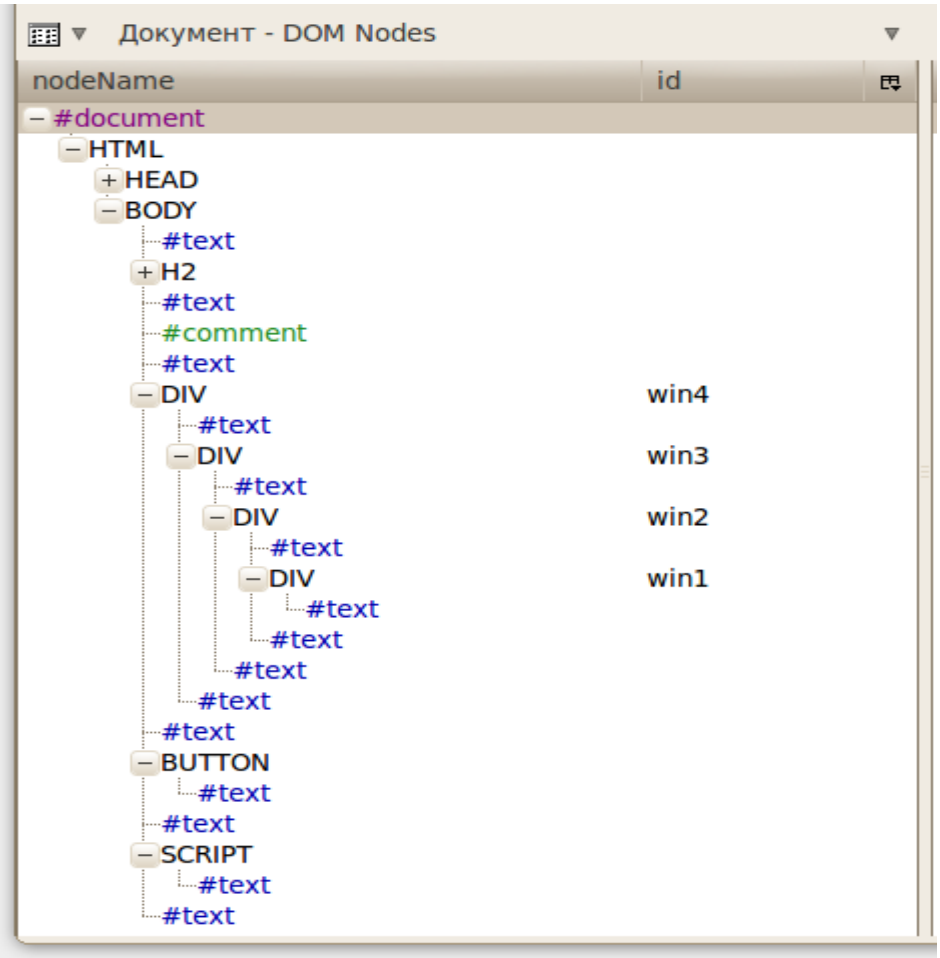
В браузере HTML-код из листинга 4.3.1 отображается примерно так, как это показано на рис. 4.2.

Рис.4.2. Вид веб-страницы, соответствующей коду из листинга 4.3.1

Как видно из рис.4.2 окна расположены по центру экрана. Такое размещение достигнуто с помощью CSS-стилей, расположенных в тэге <head >...</head> (это полезный приём центрирования и его рекомендуется освоить самостоятельно, поскольку вы уже изучили правила форматирования с помощью CSS-стилей).

Изображённый на рис. 4.2 документ после загрузки в браузере имеет объектную модель, к узлам которой обращается программа-скрипт, помещённая в коде листинга 4.3.1 между тэгами `<script ...>...</script>`. Объектную модель этого документа можно изобразить в виде древовидной схемы, показанной на рис. 4.3 (данная схема получена с помощью дополнения DOM Inspector браузера Mozilla Firefox).

Рис. 4.3. Дерево узлов объектной модели документа, изображённого на рис. 4.2 (код из листинга 4.3.1)



Запуск скрипта осуществляется пользователем с помощью события `click`. Событие запускает обработчик, указанный статически с помощью атрибута `onclick` непосредственно в тэге `<button...>...</button>`. Обработчик состоит всего лишь из одного оператора, который присваивает узлу с `id='win1'` свой обработчик события `click`: `onclick="document.getElementById('win1').onclick = myFunction1"`. Поскольку в дереве узлов выше узла `button` обработчиков события click не имеется, то на этом работа скрипта приостанавливается.

После присвоения обработчика события `click` тэгу `<div>`, имеющему `id='win1'`, пользователь имеет возможность запустить этот обработчик, кликнув мышкой на окне №1. Такой клик запустит обработчик, который в свою очередь передаст управление функции `myFunction1()`.

В функции `myFunction1()` в качестве первого параметра можно использовать переменную с любым именем и ей автоматически будет присвоено значение объекта `event` (объекта событие) со всеми свойствами узла, которому принадлежит событие. В примере после клика происходит переход к `myFunction1(evn)` и переменной `evn` автоматически неявно присваивается значение объекта `event` со свойствами узла `div` имеющему `id='win1'`. Однако так происходит лишь в браузерах, работающих по стандарту W3C. В браузере

Internet Explorer, пока ещё не поддерживающему стандарт W3C, объект event передаётся в функцию по-другому, как глобальная переменная event, к которой нужно обращаться напрямую явным образом. Поэтому, для поддержания кроссбраузерности, используется такой оператор: var ev1 = evn || window.event. При этом следует обратить внимание, что переменная ev1 является локальной, то есть существует лишь в теле функции myFunction1(), не выходя за её пределы.

Далее в функции myFunction1() используются свойства события, связанные со свойствами узла, в котором это событие произошло. В данном случае — это свойства, в которых хранятся координаты клика относительно окна-объкта windows, в котором расположен узел:

```
x=ev.clientX
y=ev.clientY
```

Следует обратить внимание, что переменные x и y являются глобальными и видны во всех функциях скрипта, поскольку не используется служебное слово var.

Затем в функции myFunction1() вызывается метод alert() для вывода окна сообщения с необходимой нам информацией. В этом операторе, по-видимому, требует пояснения свойство this.innerHTML. Служебная переменная this предусмотрена в Javascript для ссылки на текущий объект (вспомните это из подраздела 3.8, посвящённому объектам).

И, наконец, последним оператором присваивается обработчик узлу div имеющему id='win2', являющемуся внешним в дереве объектов по отношению к узлу div имеющему id='win1':

```
document.getElementById('win2').onclick = myFunction2.
```

Присвоение события второму узлу div происходит быстрее всплывания события, поэтому к моменту подхода события к этому внешнему узлу обработчик в этом узле уже имеется и теперь уже без клика вручную автоматически запускается функция myFunction2().

В функции myFunction2() имеется всего лишь два оператора. В первом из них с помощью метода alert() вначале с помощью свойства this.innerHTML выводятся фрагменты текста вместе с обрамляющими их тегами, содержащиеся в текущем узле. Это текст «Окно №2», в текущем узле div имеющему id='win2' и текст «Окно №2» во вложенном узле div имеющему id='win1'. Затем выводятся координаты клика, полученные ранее в функции myFunction1() и хранящиеся в глобальных переменных x и y.

Вторым оператором в функции myFunction2() присваивается обработчик узлу div имеющему id='win3', являющемуся внешним в дереве объектов по отношению к узлу div

имеющему `id='win2'`:

`document.getElementById('win3').onclick = myFunction3`.

Точно также, как в случае второго узла `div`, присвоение события происходит быстрее всплывания события, поэтому к моменту подхода события к внешнему узлу обработчик в этом узле уже имеется и теперь уже без клика вручную автоматически запускается функция `myFunction3()`.

Функция `myFunction3()`похожа на `myFunction2()` и её работа не требует пояснений.

Функция `myFunction4()` отличается (в учебных целях) от функции `myFunction3()`, поэтому рассмотрим её подробнее.

Первым оператором в функции `myFunction4()` заново присваивается событие своей локальной переменной `ev4`, чтобы показать, что объект события при всплывании не изменяется и равен в точности тому объекту, который был равен при запуске события. Следующие два оператора переприсваивают локальным переменным `x` и `y` значения координат клика и ещё раз подтверждают неизменность объекта `event` и всех его свойств в процессе перемещения вверх по узлам при всплывании. Если же необходимо использовать свойства текущего узла (узла, через который проплывает событие), то используется служебная переменная `this`, содержащая эти свойства. Эти рассуждения подтверждаются последним оператором в функции `myFunction4`:

```
alert('Событием пройдены:\n'+this.innerText+'\nКоординаты клика:\nx = '+x+'\ny = '+y).
```

Завершая этот подраздел перечислим основные моменты жизненного цикла событий в HTML DOM :

- Событие является сигналом от браузера, говорящим о каком-то изменении в окне страницы, которое уже произошло или должно произойти в ближайшем времени, если вы не предпримете каких-либо мер.
- Обработчик события - функция на JavaScript, которая назначается в соответствие некоторой паре из объекта и названия события. Когда соответствующее событие происходит для данного объекта, выполняются все обработчики событий, назначенные этому узлу.
- Все функции-обработчики событий выполняются последовательно, и каждое событие обрабатывается полностью (включая всплывание сквозь DOM и выполнение действия по умолчанию) перед тем, как будет выполнено следующее событие.

О тонкостях обработки событий в DHTML с использованием Javascript хорошо написано в [15].

Посмотреть пример 4.3.1 в действии можно на сайте pvn.ho.ua (меню «Примеры» и

— к примеру 4.3.1) или перейдя напрямую к примеру по ссылке: http://pvn.ho.ua/pvn.org.ua/www/lection/example/example_4/lstng_4_3_1.html.

4.4. Работа с таблицами с использованием HTML DOM

Таблицы часто используется в инженерной практике при выводе результатов вычислений и автоматизация такого вывода позволяет сэкономить много времени по сравнению, например, с формированием таблицы в электронных таблицах или средствами HTML без использования программирования. В этом подразделе подробно рассматривается один полезный пример, демонстрирующий работу с таблицами с использованием динамического HTML. На этом примере мы изучим основы автоматизации таблиц.

Прежде, чем изучать материал этого подраздела полезно вспомнить средства HTML для формирования таблиц, которые вы изучали в подразделе 2.5. Вспомним, что в HTML для создания таблицы внутри контейнера <table>...</table> можно использовать тэги-разделы:

- тэг <caption>...</caption> - наименование таблицы;
- тэг <thead>...</thead> - заголовок или «шапка» таблицы;
- тэги <tbody>...</tbody> - «тела» таблицы;
- тэг <tfoot>...</tfoot> - «подвал» таблицы.

Таблица может содержать по одному тэгу <caption>...</caption>, <thead>...</thead> и <tfoot>...</tfoot>, и много тэгов <tbody>...</tbody>.

Внутри тэгов <thead>...</thead>, <tbody>...</tbody> и <tfoot>...</tfoot> должны находиться тэги <tr>...</tr>, представляющие строки и являющиеся контейнерами для тэгов <th>...</th> и <td>...</td>.

В объектной модели HTML-таблицы имеется объект TABLE, который содержит разделы — элементы-объекты, соответствующие имеющимся HTML-тэгам:

- CAPTION — свойство;
- THEAD — свойство;
- TBODIES — семейство;
- TFOOT — свойство.

Объект TABLE содержит также семейство ROWS, соответствующее тэгам <tr>...</tr> и представляющее все строки в таблице независимо от раздела. Каждый элемент семейства ROWS представляет семейство CELLS, соответствующее тэгам <th>...</th> и <td>...</td> внутри строки.

Кроме семейства ROWS объекта TABLE, каждый раздел (THEAD, TBODIES, TFOOT) содержит своё семейство ROWS, которое содержит содержащиеся в данном разделе строки. Такое деление на разделы удобно использовать при программировании. Например, при поиске

ячейки из второй ячейки первой строки первого раздела TBODY (первого «тела») и извлечении содержащегося в ней текста можно использовать такой оператор:

```
td_val=document.getElementById("tbl").tBodies[0].rows[0].cells[1].innerHTML;
```

Последнюю ячейку той же строки можно определить , используя свойство length, и затем извлечь её содержимое. Для этого нужно написать, например, такие три оператора:

```
arr_cells =  document.getElementById("tbl").tBodies[0].rows[0].cells;
n_cells =  arr_cells.length;
td_val=arr_cells[n_cells-1].innerHTML;
```

Более подробно о работе с таблицами в рамках HTML DOM можно прочитать по адресу: http://easywebscripts.net/javascript/table.php, а мы рассмотрим полезный для нашей специальности пример работы с таблицей, исходя из тех знаний, которые у нас имеются.

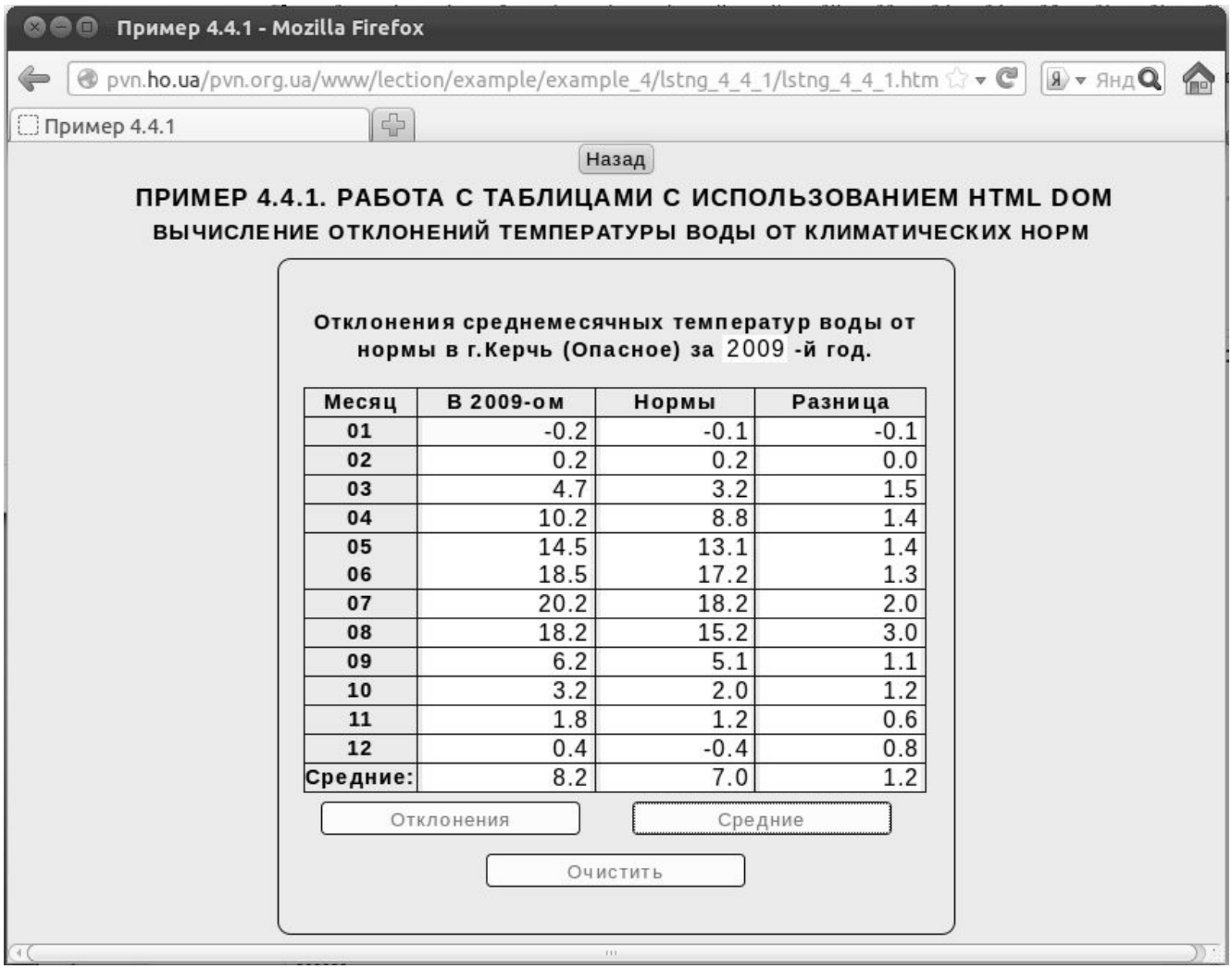
Прежде, при изучении языка Javascript в подразделе 3.6, вы уже формировали таблицы программным способом с использованием метода `document.write()`. В настоящем примере рассматривается формирование таблицы более эффективным способом с использованием свойств, методов и событий её объектной модели.

Рассмотрим задачу в следующей постановке.

1. Имеются ряды среднемесячных значений температуры воды в некоторой прибрежной точке Керченского пролива (район с.Опасное) по данным за некоторый год.
2. Известны также среднемесячные климатические значения (нормы) температуры в этой точке, полученные усреднением по данным за многолетний период.
3. Необходимо написать программу для ввода данных за текущий год, вычисления отклонений от норм и средних за 12 месяцев и вывода вводимых и вычисляемых значений в виде таблицы.
4. Все рутинные операции должны быть автоматизированы. Кроме этого, необходимо предусмотреть проверку вводимых данных, исключающую возможность ввода пустых и нецифровых значений.

Выводимая на экран таблица должна иметь вид, показанный на рис. 4.4.

Рис. 4.4. Окончательный вид таблицы.



Ниже будут приведены исходные HTML и CSS-коды, а также коды программ-скриптов на языке Javascript, реализующие поставленную задачу в виде примера 4.4.1 (листинги 4.4.1.1÷4.4.1.4). Изучение примера разобьём на два этапа, рассматривая:

1) структуру HTML-документа с HTML-таблицей и функционирование документа в браузере со стороны пользователя;
2) программирование ввода-вывода данных, проверки вводимых значений и вычисления отклонений от норм и средних за 12 месяцев.

Код HTML-документа приведён в листинге 4.4.1.1.

Листинг 4.4.1.1

```
<html>
<head><title>Пример 4.4.1</title>
<meta http-equiv=Content-Type content="text/html; charset=utf-8">
<link rel="stylesheet" href="lstng_4_6.css" type="text/css">
<script type="text/javascript" src="../lib_pvn/statistics.js"></script>
<script type="text/javascript" src="../lib_pvn/control.js"></script>
</head>
<body>
<h3>Пример 4.4.1. Работа с таблицами с использованием HTML DOM</h2>
<h4>Вычисление отклонений температуры воды от климатических норм</h3>
<div class="in">
```

```
  <table id="tbl">
  <caption>
    Отклонения среднемесячных температур воды от нормы в г.Керчь
(Опасное)<br />
    за <input class="year" type="text" id="god" value="2009"
    title="Введите настоящий год и нажмите на [tab]"
    onBlur='document.getElementById("name_col2").innerHTML=
    "В "+document.getElementById("god").value+"-ом";
    document.getElementById("ta1").focus()'/>
    </input>-й год.
  </caption>
  <thead>
  <tr>
    <th>Месяц</th>
    <th id="name_col2"></th>
    <th>Нормы</th>
    <th>Разница</th>
  </tr>
  </thead>
  <tbody>
  <tr>
    <th  scope="row">01</th>
    <td><input type="text" id="ta1" value="-0.2" onkeypress="return
no_number(event)" onBlur="t1=no_empty(event);" /></td>
    <td><input type="text" id="tn1" value="-0.1" onkeypress="return
no_number(event)" onBlur="t2=no_empty(event)" /></td>
    <td><input type="text" id="dt1" onfocus="calc_delta(event,t1,t2); delete
t1; delete t2"; readonly /></td>
  </tr>
  <tr>
    <th  scope="row">02</th>
    <td><input type="text" id="ta2" value="0.2" onkeypress="return
no_number(event)" onBlur="t1=no_empty(event)" /></td>
    <td><input type="text" id="tn2" value="0.2" onkeypress="return
no_number(event)" onBlur="t2=no_empty(event)" /></td>
    <td><input type="text" id="dt2" onfocus="calc_delta(event,t1,t2); delete
t1; delete t2" readonly /></td>
  </tr>
  <tr>
    <th  scope="row">03</th>
    <td><input type="text" id="ta3" value="4.7" onkeypress="return
no_number(event)" onBlur="t1=no_empty(event)" /></td>
    <td><input type="text" id="tn3" value="3.2" onkeypress="return
no_number(event)" onBlur="t2=no_empty(event)" /></td>
    <td><input type="text" id="dt3" onfocus="calc_delta(event,t1,t2); delete
t1; delete t2" readonly /></td>
  </tr>
  <tr>
    <th  scope="row">04</th>
    <td><input type="text" id="ta4" value="10.2" onkeypress="return
no_number(event)" onBlur="t1=no_empty(event)" /></td>
    <td><input type="text" id="tn4" value="8.8" onkeypress="return
no_number(event)" onBlur="t2=no_empty(event)" /></td>
    <td><input type="text" id="dt4" onfocus="calc_delta(event,t1,t2); delete
t1; delete t2" readonly /></td>
  </tr>
  <tr>
    <th  scope="row">05</th>
    <td><input type="text" id="ta5" value="14.5" onkeypress="return
no_number(event)" onBlur="t1=no_empty(event)" /></td>
```

```
    <td><input type="text" id="tn5" value="13.1" onkeypress="return
no_number(event)" onBlur="t2=no_empty(event)" /></td>
    <td><input type="text" id="dt5" onfocus="calc_delta(event,t1,t2); delete
t1; delete t2" readonly /></td>
  </tr>
  <tr>
    <th  scope="row">06</th>
    <td><input type="text" id="ta6" value="18.5" onkeypress="return
no_number(event)" onBlur="t1=no_empty(event)" /></td>
    <td><input type="text" id="tn6" value="17.2" onkeypress="return
no_number(event)" onBlur="t2=no_empty(event)" /></td>
    <td><input type="text" id="dt6" onfocus="calc_delta(event,t1,t2); delete
t1; delete t2" readonly /></td>
  </tr>
  <tr>
    <th  scope="row">07</th>
    <td><input type="text" id="ta7" value="20.2" onkeypress="return
no_number(event)" onBlur="t1=no_empty(event)" /></td>
    <td><input type="text" id="tn7" value="18.2" onkeypress="return
no_number(event)" onBlur="t2=no_empty(event)" /></td>
    <td><input type="text" id="dt7" onfocus="calc_delta(event,t1,t2); delete
t1; delete t2" readonly /></td>
  </tr>
  <tr>
    <th  scope="row">08</th>
    <td><input type="text" id="ta8" value="18.2" onkeypress="return
no_number(event)" onBlur="t1=no_empty(event)" /></td>
    <td><input type="text" id="tn8" value="15.2" onkeypress="return
no_number(event)" onBlur="t2=no_empty(event)" /></td>
    <td><input type="text" id="dt8" onfocus="calc_delta(event,t1,t2); delete
t1; delete t2" readonly /></td>
  </tr>
  <tr>
    <th  scope="row">09</th>
    <td><input type="text" id="ta9" value="6.2" onkeypress="return
no_number(event)" onBlur="t1=no_empty(event)" /></td>
    <td><input type="text" id="tn9" value="5.1" onkeypress="return
no_number(event)" onBlur="t2=no_empty(event)" /></td>
    <td><input type="text" id="dt9" onfocus="calc_delta(event,t1,t2); delete
t1; delete t2" readonly /></td>
  </tr>
  <tr>
    <th  scope="row">10</th>
    <td><input type="text" id="ta10" value="3.2" onkeypress="return
no_number(event)" onBlur="t1=no_empty(event)" /></td>
    <td><input type="text" id="tn10" value="2.0" onkeypress="return
no_number(event)" onBlur="t2=no_empty(event)" /></td>
    <td><input type="text" id="dt10" onfocus="calc_delta(event,t1,t2); delete
t1; delete t2" readonly /></td>
  </tr>
  <tr>
    <th  scope="row">11</th>
    <td><input type="text" id="ta11" value="1.8" onkeypress="return
no_number(event)" onBlur="t1=no_empty(event)" /></td>
    <td><input type="text" id="tn11" value="1.2" onkeypress="return
no_number(event)" onBlur="t2=no_empty(event)" /></td>
    <td><input type="text" id="dt11" onfocus="calc_delta(event,t1,t2); delete
t1; delete t2" readonly /></td>
  </tr>
  <tr>
    <th  scope="row">12</th>
```

```
    <td><input type="text" id="ta12" value="0.4" onkeypress="return
no_number(event)" onBlur="t1=no_empty(event)" /></td>
    <td><input type="text" id="tn12" value="-0.4" onkeypress="return
no_number(event)" onBlur="t2=no_empty(event)" /></td>
    <td><input type="text" id="dt12" onfocus="calc_delta(event,t1,t2); delete
t1; delete t2" readonly /></td>
  </tr>
  </tbody>
  <tbody>
  <tr>
    <th  scope="row">Средние:</th>
    <td><input type="text" readonly /></td>
    <td><input type="text" readonly /></td>
    <td><input type="text" readonly /></td>
  </tr>
  </tbody>
  <tfoot>
  <tr>
  <td colspan="2">
    <button class="btn" title="все отклонения сразу"
onclick="calc_deltas_rows()">Отклонения</button>
  </td>
  <td  colspan="2">
    <button class="btn" title="Вычислить среднегодовые"
onclick="calc_means_cols()">Средние</button>
  </td>
  </tr>
  <tr>
  <td  colspan="4">
    <button class="btn" title="Удалить введённое"
onclick="reset_in()">Очистить</button>
  </td>
  </tr>
  </tfoot>
  </table>
</div>
</body>
</html>
```

Как видно из листинга 4.6, CSS-стили и скрипты размещены отдельно в присоединяемых файлах. Код CSS-стилей приведён в листинге 4.4.1.2.

Листинг 4.4.1.2

```
* {
   font-family: Arial, Helvetica, sans-serif;
   color: #000000;
   letter-spacing: 1pt;
   margin:0;
   padding:0;
   text-align: center;
}
BODY {
  background: #f1edc8;
}
H3 {
  margin: 6px;
  text-transform: uppercase;
  width: 100%;
}
```

```
H4 {
  text-transform: uppercase;
  margin: 4px 4px 10px 4px;
  width: 100%;
}
.msg_gray {
  color: lightgrey;
}
div.in {
  position: relative;
  text-align: center;
  border-style: solid;
  border-color: #0000CD;
  border-width: 2px;
  border-radius:10px;
  -moz-border-radius:8px;
  margin: 0 auto;
  width: 520px;
  height: 520px;
}
table {
  position: absolute;
  top: 50%;
  left: 50%;
  width: 480px;
  height:380px;
  margin: -221px 0 0 -241px;
  border-collapse: collapse;
  border: none;
}
caption {
  font-weight: 700;
  height: 60px;
}
tr:hover {
  background-color: #ffc;
}
tr input:hover {
  background-color: #ff9;
}
input:hover {
  background-color: #ff9;
}
input:focus {
  background-color: #ffc;
}
td {
  border: solid  #0000CD 1px;
  height: 22px;
}
th {
  border: solid  #0000CD 1px;
  height: 22px;
}
tfoot td {
  text-align: center;
  vertical-align: middle;
  height: 40px;
```

```
  border: none;
}
input {
  width: 95%;
  border: none;
  text-align: right;
  font-size: 110%;
}
.year {
  width: 50px;
}
.btn{
  width: 200px;
  height: 25px;
  background-color: #F0FFFF;
  border-style: solid;
  border-color: #0000CD;
  border-width: 1px 2px 2px 1px;
  border-radius:4px;
  -moz-border-radius:3px;
  color: #cc6600;
  text-align: center;
}
.btn:hover {
  background-color: #ff9;
  color: #4682B4;
}
br {
  display: none;
}
```

Программы-скрипты приведены в листингах 4.4.1.3 и 4.4.1.4. В листинге 4.4.1.3 приведены программы вычислений отклонений от норм и средних годовых значений, а также программа удаления всех введённых в таблицу данных.

Листинг 4.4.1.3

```
/*********************************************/
/*  Вычисление отклонения от нормы           */
/*********************************************/
function calc_delta(e,x1,x2) {
  var evnt = window.event;
  if (!evnt) evnt = e; // для других браузеров
  if (evnt.srcElement) {
    var id_input = evnt.srcElement.id; // для IE
  }
  else {
    var id_input = evnt.target.id; // для не IE
  }
  document.getElementById(id_input).value=(Number(x1)-
Number(x2)).toFixed(1);
}
/*********************************************/
/*  Вычисление ряда отклонений от норм       */
/*********************************************/
function calc_deltas_rows() {
  var n_rows=document.getElementById("tbl").tBodies[0].rows.length;
```

```
  var
n_cols=document.getElementById("tbl").tBodies[0].rows[0].cells.length;
  for (var i = 0; i < n_rows; i++) {
      var
t_g=document.getElementById("tbl").tBodies[0].rows[i].cells[1].childNodes[0
].value;
      if (parseInt(t_g).toString() == "NaN"){
     alert("Имеются пустые поля значений температуры. Все поля обязательны
для заполнения") }
      var
t_n=document.getElementById("tbl").tBodies[0].rows[i].cells[2].childNodes[0
].value;
      if (parseInt(t_n).toString() == "NaN"){
     alert("Имеются пустые поля значений нормы. Все поля обязательны для
заполнения") }
      document.getElementById("tbl").tBodies[0].rows[i].cells[3].childNodes
[0].value = (Number(t_g)-Number(t_n)).toFixed(1);
  }
}
/*********************************************/
/*  Вычисление средних годовых значений        */
/*********************************************/
function calc_means_cols() {
  var n_rows=document.getElementById("tbl").tBodies[0].rows.length;
  var
n_cols=document.getElementById("tbl").tBodies[0].rows[0].cells.length;
  var sum = new Array();
  val_god = document.getElementById("god").value;
  if (parseInt(val_god).toString() == "NaN"){
    return alert("Поле с годом наблюдения не заполнено. Все поля
обязательны для заполнения")
  }
  for (var j = 1; j < n_cols; j++) {
    sum[j] = 0;
    for (var i = 0; i < n_rows; i++) {
     var
val_input=document.getElementById("tbl").tBodies[0].rows[i].cells[j].childN
odes[0].value;
     if (parseInt(val_input).toString() != "NaN"){
     sum[j] = sum[j] + Number(val_input)
     }
     else {
     return alert("Имеются пустые поля значений температуры. Все поля
обязательны для заполнения")
     }
    }
    document.getElementById("tbl").tBodies[1].rows[0].cells[j].childNodes[0
].value = (sum[j]/n_rows).toFixed(1);
  }
}
/*********************************************/
/*  Удаление всех введённых и вычисленных данных */
/*********************************************/
function reset_in() {
  document.getElementById("god").value="";
  var n_rows=document.getElementById("tbl").tBodies[0].rows.length;
  var
n_cols=document.getElementById("tbl").tBodies[0].rows[0].cells.length;
```

```
  for (var j = 1; j < n_cols; j+=2) {
    for (var i = 0; i < n_rows; i++) {

document.getElementById("tbl").tBodies[0].rows[i].cells[j].childNodes[0].va
lue=""
    }
    document.getElementById("tbl").tBodies[1].rows[0].cells[j].childNodes[0
].value = ""
  }
}
```

В листинге 4.4.1.4 показаны коды двух функций. Первая из них , функция no_number(), предназначена для проверки на отсутствие недопустимых символов. Вторая функция no_empty() - выдаёт вводимое данное, если его значение не пустое, и сообщение об ошибке, если пользователь пытается перейти к следующему полю, не заполнив его хотя бы одним символом.

Листинг 4.4.1.4

```
/*************************************/
/* Проверка на не цифры (и не точку)  */
/*************************************/
function no_number(e) {
  var evnt = window.event; // для IE
  if (!evnt) evnt = e; // для других браузеров
  if (evnt.charCode) var charCode = evnt.charCode;
        else if (evnt.keyCode) var charCode = evnt.keyCode;
        else if (evnt.which) var charCode = evnt.which;
        else var charCode = 0;
    if (charCode > 31 && (charCode < 48 || charCode > 57) && charCode != 46
&& charCode != 45)
    {
        alert ("Недопустимый символ!!! В это поле разрешается вводить
только лишь цифры и точку");
        //evnt.target.focus()
        return false;
    }
}

/*********************************************************/
/* Запоминаем введённое число после проверки на "не пусто" */
/*********************************************************/
function no_empty(e)
{
  var evnt = window.event; // для IE
  if (!evnt) evnt = e; // для других браузеров
  if (evnt.srcElement) {
    var id_input = evnt.srcElement.id;    //для IE
    var val_input= evnt.srcElement.value; //для IE
  }
  else {
    var id_input = evnt.target.id;    //для IE
    var val_input= evnt.target.value; //для не IE
  }
  if (parseInt(val_input).toString() == "NaN")
  {
```

```
   alert("Поле обязательно для заполнения. Повторите ввод.");
   document.getElementById(id_input).focus();
   return false;
  }
  return val_input;
}
```

Структура HTML-документа и его функционирование (взгляд со стороны пользователя). Структуру HTML-документа можно понять непосредственно из кода листинга 4.4.1.1. Для понимания того, как функционирует документ, необходимо создать соответствующие файлы и открыть файл с HTML-кодом в браузере. Полезно также открыть схему объектной модели, используя дополнение Inspector DOM браузера Mozilla Firefox или подобное расширение для Google Chrome, которые уже упоминались.

После заголовка в теле документа помещён тэг ... , который является внешним контейнером для HTML-таблицы. Это сделано для улучшения дизайна — таким образом таблицу можно разместить по центру с использованием соответствующих стилей (разберитесь самостоятельно с этим полезным способом центрирования таблицы с помощью стилей по листингу 4.4.1.2)

В HTML-коде таблицы имеется наименование таблицы (тэг <caption>...</caption>), заголовок или «шапка» таблицы (тэг <thead>...</thead>), два «тела» таблицы (тэги <tbody>...</tbody>) и «подвал» таблицы (тэг <tfoot>...</tfoot>). Как известно, таблица может содержать по одному тэгу <thead>...</thead> и <tfoot>...</tfoot>, но много тэгов <tbody>...</tbody>.

В наименовании таблицы, кроме вводимого вручную текста, предусмотрено поле для ввода четырёхзначного числа - года наблюдений, созданное с помощью тэга <input>...</input>. По умолчанию в это поле введено значение «2009» - значение года, данные за который также по умолчанию введены во второй столбец таблицы, предназначенный для ввода значений температуры воды за конкретный год. Это значение года пользователь может заменить, введя другое число.

«Шапка» таблицы состоит из четырёх полей с наименованиями столбцов, причём второе поле вначале пустое. Год в это поле не вводится вручную, а переносится из наименования таблицы после ввода года и нажатия клавиши <Tab> пользователем (к сожалению в браузере Mozilla Firefox на [Tab] нужно нажимать четыре раза, можете попытаться самостоятельно разобраться и устранить этот «хак» Firefox'a). Ввод можно осуществить также, щёлкнув мышкой на наименовании столбца с пустым значением года.

Первое «тело» (первый тэг <body>...</body>) таблицы содержит 12 строк (тэги <tr>...</tr>), каждая из которых содержит четыре ячейки (один тэг <th>...</th> и три тэга <td>...</td>). В браузере это первое «тело» отображается в виде четырёх столбцов и 12-ти строк.

В первой ячейке каждой строки записан номер месяца, при этом используется атрибут scope="row", который предписывает размещение тэга <th>...</th> в одной строке с тремя тэгами <td>...</td>, также принадлежащими этой строке. В каждый из этих трёх тэгов <td>...</td> вложен тэг <input>...</input>, который предназначен для создания поля ввода (это вам должно быть известно из подраздела 2.7, посвящённому HTML-формам). Следует заметить, что тэг <input>...</input> можно использовать без формы, тогда будет создаваться поле ввода без отсылки введённого текста серверу, как это будет происходить, если этот тэг используется внутри контейнера <form>...</form>.

Первое поле (или вторая ячейка строки) первого «тела» таблицы по умолчанию содержит среднемесячное значение за соответствующий за 2009, но вместо этого значения пользователь может вводить любое другое числовое значение. Второе поле (или третья ячейка строки) содержит выбранные из климатических таблиц значение нормы, которое также можно исправить (хотя значение климатической нормы для данной точки меняется только лишь по истечении многих лет и то на очень малую величину). В третье поле (или в четвёртую ячейку строки) помещается вычисляемое значение отклонения от нормы, причём это значение вычисляется программным способом после нажатия пользователем на клавишу [Tab]. Это поле не может быть изменено вручную пользователем, поскольку ввод в него запрещён с помощью атрибута без свойства readonly.

Таким образом, вводя значение температуры и перемещаясь с помощью клавиши <Tab> к третьему полю, пользователь может заполнить все столбцы первого «тела» таблицы, совершая минимальное количество необходимых для этого действий. За 2009-й год значения температуры заносятся по умолчанию, поэтому для вычисления отклонений за этот год достаточно перемещаться по полям с помощью клавиши [Tab].

После заполнения всех столбцов первого «тела» таблицы пользователь нажатием на имеющуюся под таблицей кнопку «Средние» может вычислить средние годовые значения температуры, нормы и отклонения от нормы. Эти значения помещаются в соответствующие ячейки второго «тела», образуемого с помощью второго тэга <tbody>...</tbody>.

Отклонения от среднего могут быть рассчитаны также не перемещением по строкам с помощью клавиши [Tab], а в целом по столбцу одним нажатием на кнопку «Отклонения», если заполнены значения в столбцах температуры и нормы за все 12 месяцев (нет пустых полей).

Под таблицей имеется ещё одна кнопки «Очистить», с помощью которой пользователь может удалить все значения из столбцов с температурой и отклонениями от норм. Удаляется также значение года из заголовка.

Упомянутые три кнопки «Средние», «Отклонения» и «Очистить» размещены в подвальной части таблицы, образуемой с помощью тэгов <tfoot>...</tfoot>.

Программирование ввода-вывода данных, проверки вводимых значений и вычисления отклонений от норм и средних за 12 месяцев.

Ввод-вывод данных, контроль вводимых значений и необходимые вычисления осуществляются в программах, показанных в листингах 4.8 и 4.9. Для понимания того, что будет изложено далее, необходимо представить себе древовидную схему объектной модели HTML-документа. Для этого, как уже упоминалось выше, воспользуйтесь кодами из листингов 4.6÷4.9 и создайте HTML-документ. Откройте созданный документ в браузере Mozilla Firefox и проверьте работоспособность, пользуясь описанием первого этапа. После этого откройте схему HTML DOM с помощью расширения Inspector DOM.

Схема элементов DOM, соответствующая HTML-коду из листинга 4.6, должна выглядеть примерно так, как это показано на рис. 4.5.

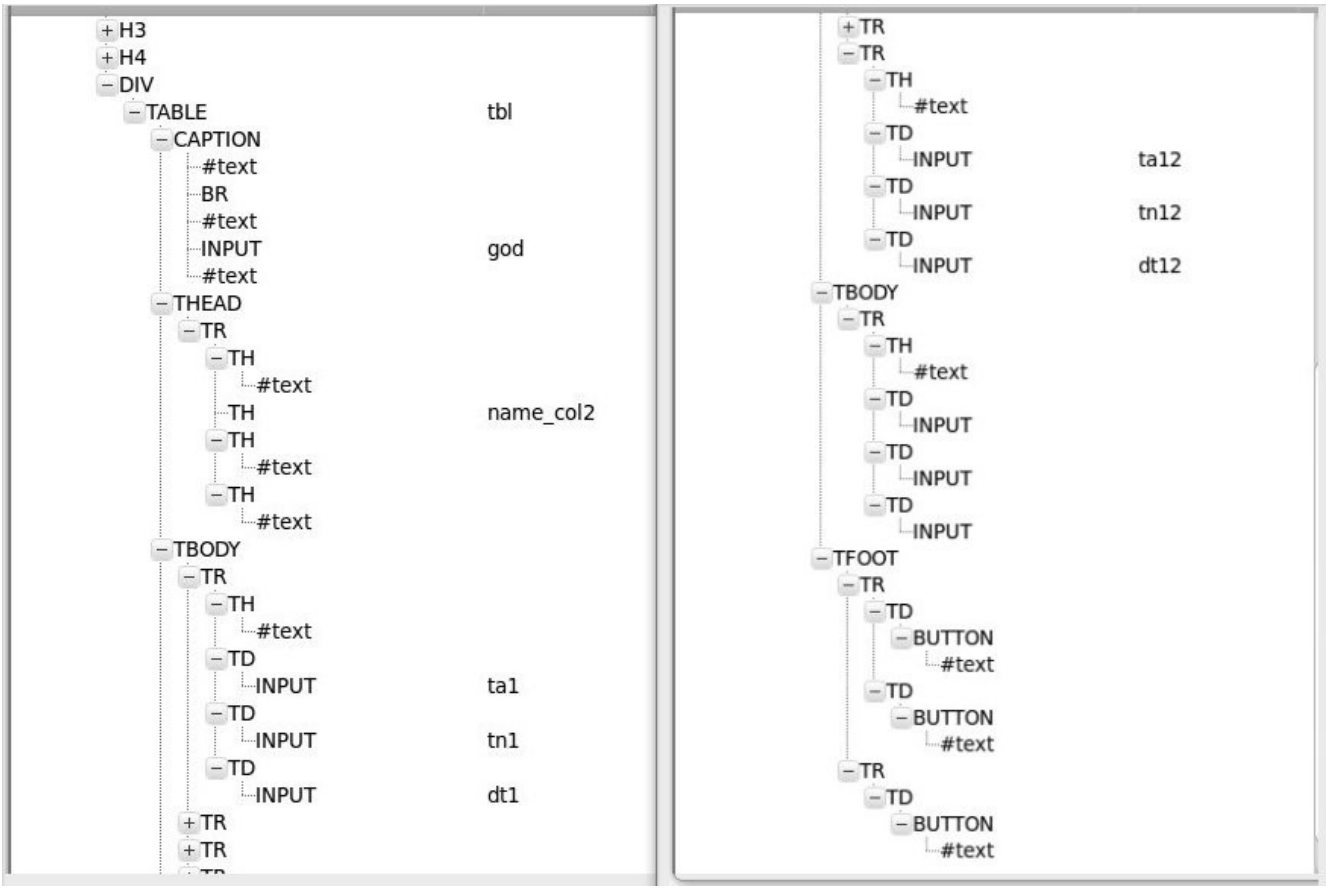


Рис. 4.5. Фрагмент древовидной схемы объектной модели документа из листинга 4.4.1.1

На рис. 4.5 с целью экономии места показан лишь фрагмент схемы, соответствующий коду документа из листинга 4.4.1.1. В левой части рисунка показана часть элементов DOM, соответствующая верхней части таблицы. Показаны лишь элементы, соответствующие первой строке первого тэга <tbody>...</tbody>. Часть схемы с элементами остальных строк первого «тела» таблицы легко представить по аналогии с элементами первой строки. В правой части показана часть схемы, соответствующая нижней части HTML-таблицы.

Вернёмся теперь к листингу 4.4.1.1 с HTML-кодом и будем последовательно просматривать все встречающиеся в нём события и их обработчики.

Первое событие предусмотрено в тэге <caption>...</caption> после ввода года (или года, устанавливаемого по умолчанию) при последующем нажатии на клавишу [Tab]:

```
<caption>
    Отклонения среднемесячных температур воды от нормы в г.Керчь
(Опасное)<br />
    за <input class="year" type="text" id="god" value="2009"
    title="Введите настоящий год и нажмите на [tab]"
    onBlur='document.getElementById("name_col2").innerHTML=
    "В "+document.getElementById("god").value+"-ом";
    document.getElementById("ta1").focus()'/>
    </input>-й год.
</caption>
```

Два Javascript-оператора обработчика события Blur (потеря фокуса — перемещение курсора из ячейки) вставлены непосредственно в тэг `<input>...</input>`. Событие происходит после ввода года по нажатию на клавишу [Tab] или клику мышкой в любой другой ячейке таблицы. Первый оператор обработчика присваивает ячейке с `id="name_col2"`, находящейся в тэге `<thead>...</thead>`, значение текущей ячейки, имеющей `id="god"`. Второй оператор устанавливает фокус (перемещает курсор) в ячейку с идентификатором ta1 (`id="ta1"`), то есть первую ячейку со значением температуры.

Следующие два события предусмотрены в ячейке, предназначенной для ввода температуры за текущий год:

```
onkeypress="return no_number(event)" onBlur="t1=no_empty(event)"
```

Эти события вызывают обработчики, в которых производится проверка вводимого пользователем числа. Обработчики реализуют **два разных, дополняющих друг друга, способа проверки:**

1) проверка в момент набора на клавиатуре символа вводимого числа;

2) проверка после введения последнего символа числа в момент попытки перехода к следующему полю.

Функции `no_number()` и `no_empty()`, в которых реализованы перечисленные выше способы проверки, рассмотрены в подразделе 5.3, посвящённом проверке полей формы.

Ввод и контроль климатической нормы осуществляется точно также, как и температуры за текущий месяц.

Значение температуры за текущий месяц и значение климатической нормы запоминаются в переменных `t1` и `t2`. Следует обратить внимание, что эти переменные имеют глобальную область видимости (при их создании не использовалось служебное слово `var`).

В третью ячейку первой строки помещается вычисленное значение отклонения от нормы. Вычисляется значение по событию `Focus`, то есть при переходе курсора в эту ячейку. Событие `Focus` может произойти, если пользователь после введения нормы нажимает либо на клавишу <Tab>, либо щелчком мышки в третьей ячейке первой строки, предназначенной для отклонения от нормы. Происходит вычисление и помещение вычисленного значения отклонения от нормы в нужную в ячейку с помощью функции-обработчика `calc_delta()`, показанной в листинге 4.4.1.3. В этой функции в качестве входных параметров, кроме события event, передаются значения температуры (`t1`) и нормы (`t2`), введённые в предыдущих ячейках этой же строки. В функции `calc_delta()` вычисленное значение отклонения от нормы округляется до десятых и присваивается значению поля в тэге `<input>...</input>`:

```
function calc_delta(e,x1,x2) {
  var evnt = window.event;
  if (!evnt) evnt = e; // для других браузеров
  if (evnt.srcElement) {
    //var id_input = evnt.srcElement.id; // для IE
    var o_input = evnt.srcElement
  }
  else {
    //var id_input = evnt.target.id; // для не IE
    var o_input = evnt.target
  }
  //document.getElementById(id_input).value=(Number(x1)-
Number(x2)).toFixed(1);
  o_input.value = (Number(x1)-Number(x2)).toFixed(1);
}
```

В обработчике третьей ячейки (см. листинг 4.4.1.1), кроме обращения к функции `calc_delta()` имеются ещё два оператора, удаляющие глобальные переменные `t1` и `t2`. Это делается для того, чтобы значения этих переменных не подставлялись в третью ячейку, если пользователю каким-то образом удастся обойти ввод значений в первые две ячейки.

Следует заметить, что в функции `calc_delta()` для поиска нужных элементов-ячеек предварительно ищется идентификатор элемента, указанный при создании HTML-документа (`id="dt1"`). Это сделано в учебных целях. Можно обойтись и без идентификатора, что упрощает создание HTML-документа:

```
function calc_delta(e,x1,x2) {
  var evnt = window.event;
  if (!evnt) evnt = e; // для других браузеров
  if (evnt.srcElement) {
    var o_input = evnt.srcElement
  }
  else {
    var o_input = evnt.target
  }
  o_input.value = (Number(x1)-Number(x2)).toFixed(1);
}
```

События и обработчики во всех 12-ти строках, в которых вводятся значения и вычисляются отклонения от норм, одинаковы. Различаются только идентификаторы в HTML-тэгах (см. листинг 4.4.1.1). Если убрать идентификаторы и переделать обработчики по типу того, как мы это сделали с функцией `calc_delta()`, то тэги во всех 12-ти строках будут также одинаковы. Таким образом, отладив работу с одной строкой, все остальные строки достаточно продублировать.

Во втором разделе `<tbody>...</tbody>` помещена всего лишь одна строка с тремя ячейками, предназначенными для вставки среднегодовых значений:

```
<tbody>
 <tr>
   <th  scope="row">Средние:</th>
   <td><input type="text" readonly /></td>
   <td><input type="text" readonly /></td>
   <td><input type="text" readonly /></td>
 </tr>
 </tbody>
```

И, наконец, в разделе `<tfoot>...</tfoot>` помещены две строки с тремя ячейками, в которых помещены кнопки для запуска функций по событию `Click`:

- `calc_deltas_rows()` - для вычисления сразу же всех отклонений во всех 12-ти строках одним нажатием мышки;
- `calc_means_cols()` - для вычисления средних годовых значений и вставки их в строку упоминавшегося выше раздела <tbody>...</tbody>;
- `reset_in()` - для очистки всех вводимых и вычисляемых значений.

Эти функции очень просты и вы их сможете освоить самостоятельно по листингу 4.4.1.3. Они могут служить примером работы с семействами `tBodies`, `raws` и `cells`.

Посмотреть в действии работу рассмотренного выше примера можно на сайте pvn.ho.ua (меню «Примеры» и — к примеру 4.4.1) или перейдя напрямую к примеру по ссылке: http://pvn.ho.ua/pvn.org.ua/www/lection/example/example_4/lstng_4_4_1/lstng_4_4_1.html .

4.5 Библиотеки для вывода результатов в виде графиков

Часто исходные данные и результаты вычислений требуется представить в виде графиков (диаграмм). Конечно, графики можно получить «вручную», с помощью электронных таблиц или другого пакета и затем снова «вручную» поместить изображение графика в документ. Однако намного эффективнее получать графики автоматически с использованием программы. При этом, при выводе графиков можно наглядно продемонстрировать динамические возможности языка `Javascript`, когда графики перестраиваются и видоизменяются сразу же после изменения данных, которые они отображают.

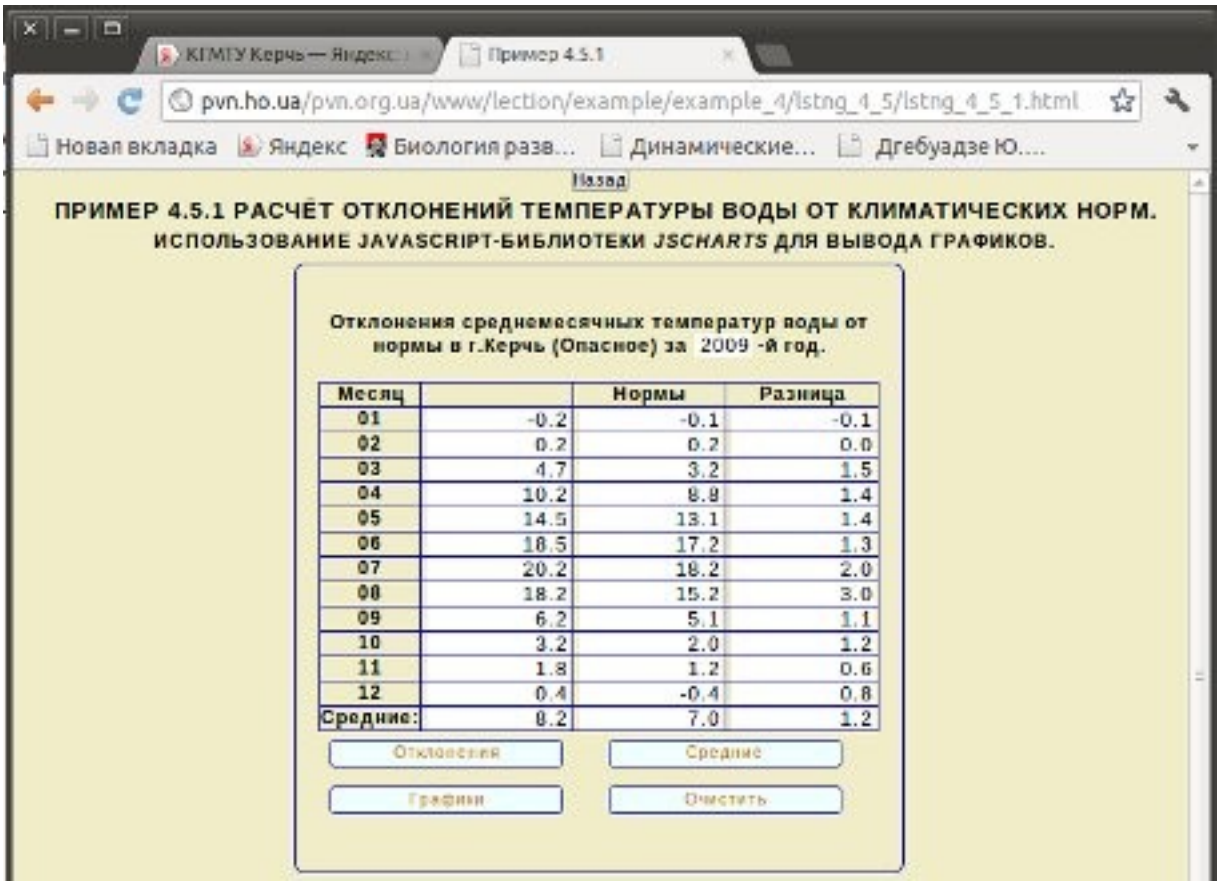
Для языка Javascript имеются расширения в виде библиотек функций или объектов, с помощью которых задача построения графиков существенно упрощается. Обращение к этим функциям и объектам не отличается от обращения к любым другим функциям и объектам, однако, как и всё в программировании, требуется некоторый навык их использования на примере типовых задач.

В этом подразделе будут описаны примеры построения графиков с помощью трёх бесплатно распространяемых библиотек: JSCharts [27], Highcharts [22]. и Rgraph [28]. Эти библиотеки разработаны на языке Javascript и для использования в программах на языке Javascript. У каждой из них есть свои преимущества и недостатки:

- для использования JSCharts не нужно никаких дополнительных библиотек, достаточно возможностей базового языка Javascript, но распространяемая бесплатно версия обладает слишком ограниченными возможностями;
- Highcharts функционирует только лишь в паре с бесплатно распространяемыми расширениями, например с библиотекой jQuery [26] и, хотя возможности её бесплатной версии намного богаче и вполне достаточны для профессионального использования, разработчику придётся познакомиться с языком расширения;
- для применения библиотеки Rgraph не нужны какие-либо дополнительные расширения типа jQuery, но её возможности слабее того, что можно сделать с Highcharts. В то же время Rgraph состоит из независимых частей, предназначенных для построения отдельных видов графиков. Разработчик подключает только лишь необходимые ему части библиотеки, что позволяет существенно уменьшить объём загружаемого кода.

Возьмём пример 4.4.1 из подраздела 4.4 и преобразуем в пример 4.5.1 с целью построения графиков с использованием JSCharts, добавив ещё одну кнопку «Графики». Пример фрагмента страницы с вычисленными значениями отклонений от норм показан на рис.4.6.

Рис.4.6. Фрагмент страницы примера 4.5.1 без графиков



После ввода и вычисления всех значений пользователь может нажать на кнопку «Графики» и в нижней части веб-страницы появятся графики среднемесячных значений, значений норм и значений отклонений от норм. Тэг, создающий кнопку имеет вид:

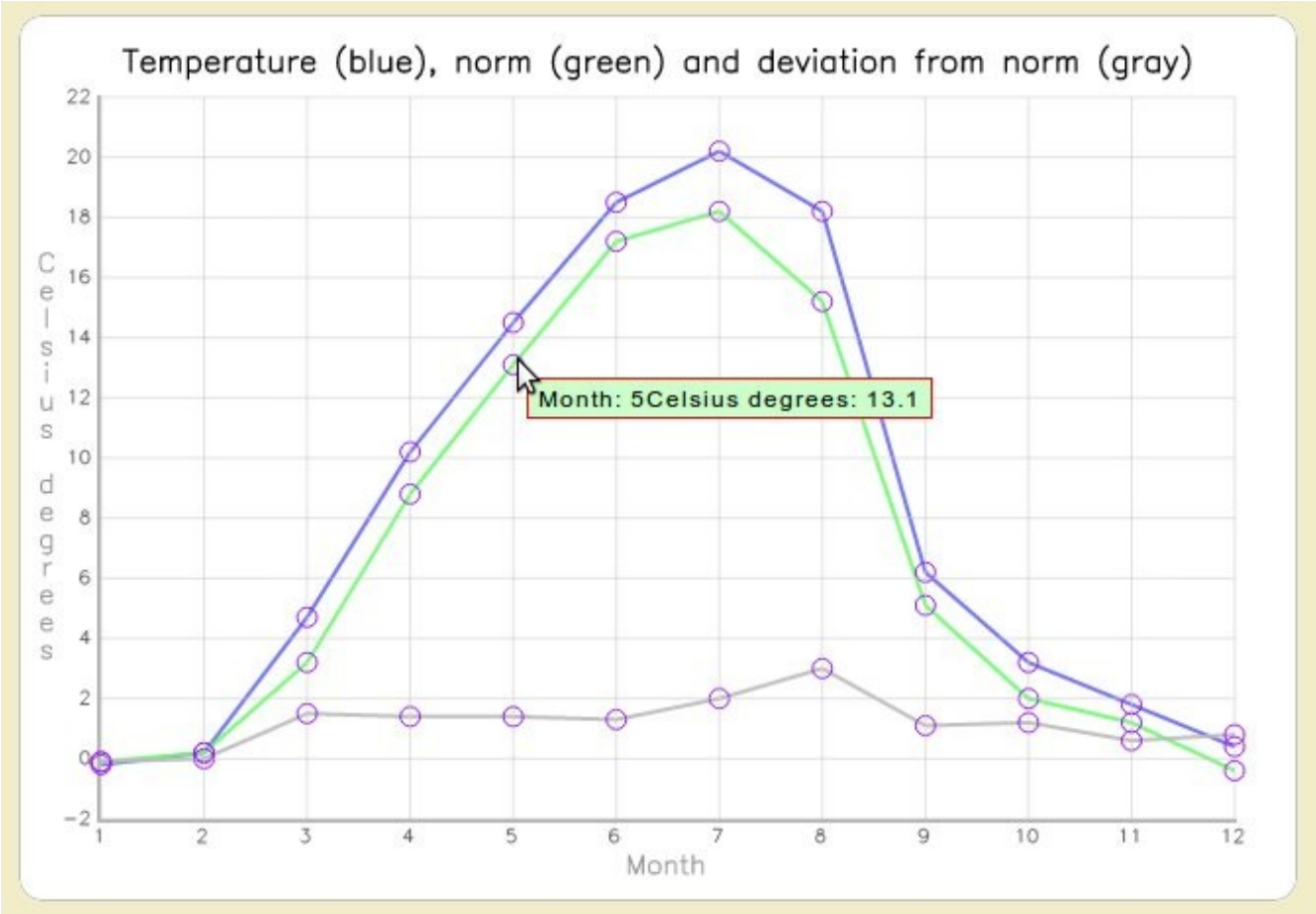
```
<button class="btn" title="Вывести графики" onclick="out_graph()">
     Графики
</button>
```

Необходимо также предусмотреть место для вывода графиков, которое создадим с помощью тэга ... :

```
<div id="out_gr" class="out_grph">
     <h2 class="msg_gray"> Здесь выводятся графики</h2>
</div>
```

Пример графиков, получаемых с использованием `JSCharts`, показан на рис.4.7.

Рис.4.7. Фрагмент страницы с графиками, получаемыми с помощью `JSCharts`



Кроме графиков на рисунке продемонстрирована также возможность вывода значений, соответствующих нанесённым точкам при подведении к этой точке указателя мышки (анимационный эффект).

В листинге 4.5.1 помещён код функции `out_graph()`, с помощью которой строится график.

Листинг 4.5.1

```
/**************************************************************************/
/* Вывод графиков годового хода температуры воды, норм и отклонений от норм */
/**************************************************************************/
function out_graph() {
      var n_rows=document.getElementById("tbl").tBodies[0].rows.length;
      var data1=new Array(n_rows);
      var data2=new Array(n_rows);
      var data3=new Array(n_rows);
      for (var i=0; i<12; i++) {
      data1[i]=new Array(2);
      data2[i]=new Array(2);
      data3[i]=new Array(2);
      }
  for (var i = 0; i < n_rows; i++) {
    var val_input1=
document.getElementById("tbl").tBodies[0].rows[i].cells[1].childNodes[0].value;
    var val_input2=
document.getElementById("tbl").tBodies[0].rows[i].cells[2].childNodes[0].value;
    var val_input3=
document.getElementById("tbl").tBodies[0].rows[i].cells[3].childNodes[0].value;
```

```
    data1[i][0]=i+1;
    data1[i][1]=Number(val_input1);
    data2[i][0]=i+1;
    data2[i][1]=Number(val_input2);
    data3[i][0]=i+1;
    data3[i][1]=Number(val_input3);
  }
  var myChart = new JSChart('out_gr', 'line');
  myChart.setTitle
      ('Temperature (blue), norm (green) and deviation from norm (gray)');
  myChart.setTitleColor('#000');
  myChart.setTitleFontSize(13);
  myChart.setDataArray(data1,'blue');
  myChart.setDataArray(data2,'green');
  myChart.setDataArray(data3,'gray');
  myChart.setLineColor('#66F', 'blue');
  myChart.setLineColor('#6F6', 'green');
  myChart.setLineColor('#BBB', 'gray');
  myChart.setAxisPaddingTop(40);
  myChart.setAxisPaddingBottom(40);
  myChart.setTextPaddingBottom(10);
  myChart.setAxisValuesNumberY(13);
  myChart.setIntervalStartY(-2);
  myChart.setIntervalEndY(22);
  myChart.setAxisValuesNumberX(12);
  myChart.setShowXValues(true);
  myChart.setAxisNameX('Month');
  myChart.setAxisNameY('Celsius degrees');
  myChart.setAxisValuesColor('#454545');
  myChart.setTooltipBackground('#CCFFCC')
  myChart.setTooltipFontColor('#000');
  myChart.setTooltipBorder('1px solid #F00');
  for (var i=1;i<=12;i++) {myChart.setTooltip([i])}
  myChart.setFlagColor('#9D16FC');
  myChart.setFlagRadius(5);
  myChart.setSize(650, 450);
  myChart.draw()
}
```

Сначала формируем три двумерных массива data1, data2, data3, в которые заносим значения ординат и абсцисс из заполненной формы.

Затем создаём объект `myCharts`, который будет содержать `html`-код будущего графика. Этот объект теперь содержит все методы библиотеки `JSCharts`. При создании указываем `id` тэга `<div>...</div>`, который был предусмотрен для размещения в нём графика и тип графика (в данном случае «`line`»). Далее, последовательно вызывая методы библиотеки, создаём нужные нам элементы графика. Последний из вызываемых методов (`myChart.draw()`) выводит график в предусмотренном нами тэге `<div>...</div>`, с `id="out_gr"`.

Все методы библиотеки и примеры их использования для построения графиков различных типов подробно описаны в документации к библиотеке на сайте http://www.jscharts.com/ [27].

На сайте www.pvn.ho.ua можно посмотреть пример 4.5.1 в действии (или прямо по ссылке: http://pvn.ho.ua/pvn.org.ua/www/lection/example/example_4/lstng_4_5/lstng_4_5_1.html).

4.6. Упражнения.

Упражнение 4.6.1. Изобразите древовидную схему узлов объектной модели документа, HTML-код которого помещён ниже. Для получения схемы узлов можно использовать дополнение к браузеру, подобное дополнению Inspector DOM, имеющемуся для браузера Mozilla FireFox.

```
<html>
    <head>
        <title>
            О бычке
        </title>
    </head>
    <body>
        Бычок для Керчи.
        <ol>
            <li>
                Бычок - важен для питания керчан
            </li>
            <li>
                ... и для экологии керченского залива
            </li>
        </ol>
    </body>
</html>
```

Ответьте на первые четыре контрольных вопроса из подраздела 4.7.

Упражнение 4.6.2. Замените в примере из листинга 4.5 свойство innerText на свойство innerHTML. Объясните, чем будет различаться при запуске в браузере работа кода из листинга 4.5 и исправленного вами кода.

Ответьте на вторую четвёрку контрольных вопросов из подраздела 4.7 (с пятого по восьмой).

Упражнение 4.6.3. Замените в примере из листинга 4.5 соответствующие тэги закомментированными тэгами , то есть тэги, в которых обработчики события click узлам div присваиваются динамически программным способом, тэгами, в которых события присваиваются статически.

Объясните, чем будет различаться при запуске в браузере работа кода из листинга 4.5 и исправленного вами кода.

Ответьте на третью четвёрку контрольных вопросов из подраздела 4.7 (с 9-го по 12-тый).

Упражнение 4.6.4. Дополните функции в примере из листинга 4.5 операторами, меняющими фоновый цвет и цвет текста в тэгах `<div>` при проплывании события `click` через эти тэги.

Ответьте на четвёртую четвёрку контрольных вопросов из подраздела 4.7 (с 13-го по 16-тый).

Упражнение 4.6.5. Повторите пример из листинга 4.5, заменив тэг `<body>` тем тэгом, который сейчас закомментирован, то есть вот этим:

```
<body onclick="alert('Произошло событие в узле body')">
```

Объясните, как изменится работа скрипта при всплывании события click и почему?

Ответьте на пятую четвёрку контрольных вопросов из подраздела 4.7 (с 17-го по 20-тый).

Упражнение 4.6.6. Получите древовидную схему объектной модели документа из листинга 4.6 с помощью дополнения `Inspector DOM` браузера `Mozilla Firefox` или подобного расширения для любого другого браузера.

Изобразите эту схему на рисунке и опишите, как изображённые на ней узлы-элементы соотносятся с тэгами HTML-документа из листинга 4.6.

Ответьте на четыре контрольных вопроса из подраздела 4.7 (с 21-го по 24-тый).

Упражнение 4.6.7. Измените пример из листингов 4.6÷4.9 так, чтобы вместо кнопки «Очистить» стояла кнопка «Восстановить» и при нажатии на эту кнопку восстанавливались все значения по умолчанию, заменяя введённые пользователем.

Ответьте на четыре контрольных вопроса из подраздела 4.7 (с 25-го по 28-ой).

Упражнение 4.6.8. Измените пример из листингов 4.6÷4.9, добавив в тэг `<input>...</input>` тэга `<caption>...</caption>` ещё одно событие и соответствующий обработчик в виде функции, с помощью которого осуществлялся бы контроль правильности ввода года наблюдений. В функции должна быть проверка на количество вводимых символов (число символов должно быть равно четырём) и является ли введённое данное числом. Если пользователь введёт данное, состоящее не из четырёх символов, или это данное не является числом, должно выводиться сообщение с соответствующей рекомендацией (вводить четыре символа или вводить только цифры), и событие должно отменяться.

Ответьте на четыре контрольных вопроса из подраздела 4.7 (с 29-го по 32-й).

4.7. Контрольные вопросы

1. Что такое динамический HTML?

2. Что такое объектная модель документа?
3. Назовите узлы, которые может иметь объектная модель документа?
4. Какие принципы соблюдаются в отношении узлов дерева объектной модели документа?
5. Покажите древовидную схему объектной модели простейшего HTML-документа, содержащего заголовок, и два абзаца с текстом и гиперссылкой в одном из абзацев?
6. Назовите свойства объектной модели документа (не менее четырёх)?
7. Назовите любое свойство HTML DOM, являющееся массивом каких-нибудь узлов?
8. Приведите пример HTML-документа с узлами, которые можно назвать потомками некоторого узла?
9. Свойство HTML DOM, предназначенное для доступа к содержимому узла?
10. Методы, с помощью которых можно перемещаться по дереву HTML DOM?
11. Три способа обнаружения нужного узла HTML-документа?
12. Свойство HTML DOM, предназначенное для доступа к стилям узла?
13. В каком регистре задаются наименования узлов при использовании свойства nodeName?
14. Какое правило существует в HTML DOM для задания свойства style, если в наименовании значения стиля имеется знак дефиса?
15. Какому атрибуту в HTML DOM соответствует свойство className?
16. Как назначается объект события программным способом?
17. Как вы понимаете термин «всплывание события»?
18. Как различается передача объекта event (событие) функции-обработчику в различных браузерах (в работающих по стандарту W3C и в MS Internet Explorer).
19. Чем различаются свойства innerHTML и innerText?
20. Как различаются свойства событий, помещённых статически в тэги с помощью атрибутов, и динамически с присвоением в программе?
21. Назовите все элементы (узлы, объекты) объектной модели таблицы, рассмотренной в подразделе 4.4 (листинг 4.6).
22. Сколько элементов CAPTION и TBODY может содержаться в элементе TABLE?
23. Сколько элементов THEAD и TFOOT может содержаться в элементе TABLE?
24. Можно ли обойтись без тэгов <caption>...</caption>, <thead>...</thead> и <tfoot>...</tfoot> при создании таблицы средствами HTML, чтобы потом можно было бы обращаться к ячейкам таблицы, как к узлам объектной модели этой таблицы?
25. Напишите оператор присвоения некоторой переменной некоторого текста четвёртой слева ячейке, расположенной в третьей строке третьего раздела TBODY в HTML-таблице, имеющей атрибут `id='tbl'` в открывающем тэге `<table>`.

26. Напишите оператор присвоения некоторого числа четвёртой слева ячейке, расположенной в третьей строке третьего раздела TBODY в HTML-таблице, не имеющей атрибутов `id` и `name`.
27. В чём различие при обращении через HTML DOM к числу, помещённому в узел TD и к числу, помещённому в узел INPUT?
28. Напишите оператор присваивания некоторой переменной текста, помещённого в некоторый узел HTML DOM, являющийся первым узлом, в котором произошло событие?
29. Опишите два разных метода проверки числа, вводимого в поле <input> ...</input>.
30. Напишите пример перемещения курсора к нужному элементу программным способом?
31. Какие свойства события event можно использовать для определения кода символа, соответствующего нажатой клавише?
32. Чем различаются события Change и Blur?

4.8. Литература и веб-источники к разделу 4

Основная

[3] Дронов В.А. Javascript в Web-дизайне / – СПб.: БХВ-Петербург, 2004. - 880 с.

[5] Кингсли-Хью Э., Кингсли-Хью К. JavaScript 1.5: учебный курс / Часть IV, главы 12, 13 и 14.

[7] Поллок Дж. Javascript. Руководство разработчика / - Спб.: Питер, 2011. - 544 с.

[8] Шмитт К. CSS. Рецепты программирования. 2-е изд.: Пер. С англ./ Кристофер Шмитт. М.: Издательство «Русская Редакция»; Спб.: «БХВ-Петербург», 2007. - 592 с.

Дополнительная

[12] Основы Javascript. Учебный курс на INTUIT.ru/ Группа разработчиков компании Opera Software под руководством Криса Милза (Chris Mills). [Электронный ресурс] - Режим доступа: http://www.intuit.ru/department/internet/jscs/

[13] Прохоренок Н.А. jQuery. Новый стиль программирования на JavaScript/ - М. : ООО “И.Д. Вильямс”, 2010. - 272 с. [Электронный ресурс] - Режим доступа: http://padabum.com/d.php?id=15156

[15.1] Справочник, самоучитель и учебник по языку Javascript. Введение. DOM в примерах/ [Электронный ресурс] - Режим доступа: http://javascript.ru/tutorial/dom/intro

[15.2] Справочник, самоучитель и учебник по языку Javascript. Введение в события./ [Электронный ресурс] - Режим доступа: http://javascript.ru/tutorial/events/intro

[15.3] Справочник, самоучитель и учебник по языку Javascript. Очерёдность событий и синхронизация в JavaScript / [Электронный ресурс] - Режим доступа: http://javascript.ru/tutorial/events/timing

[18] Храмцов П.Б. и др. Введение в Javascript. Учебный курс на INTUIT.ru. Лекция 3 «Функции и объекты»/ [Электронный ресурс] - Режим доступа: http://www.intuit.ru/department/internet/js/3/2.html

[21] Google API Loader's: позволяет загружать с помощью метода google.load() с серверов

Google наиболее популярные Javascript-библиотеки, в том числе jQuery/ [Электронный ресурс] - Режим доступа: http://code.google.com/intl/ru-RU/apis/libraries/ .

[22] HighCharts – более функциональная (чем JSChart) JavaScript-библиотека построения графиков. / [Электронный ресурс] Описание: http://troitskiy.net/2010/10/13/highcharts-javascript-biblioteka-postroeniya-grafikov/ - Режим доступа: http://www.highcharts.com/download

[23] HighSlide JS – медиа-viewer средствами JS/ [Электронный ресурс] - Режим доступа: http://highslide.com/

[24] HighStocks – Библиотека построения графиков в виде временных последовательностей/ [Электронный ресурс] http://www.highcharts.com/download

[26] jQuery – библиотека, упрощающая и расширяющая Javascript (краткое описание, справочник, позволяющие работать с jQuery/ [Электронный ресурс] - Режим доступа: http://jquery.page2page.ru/

[27] JSChart - наиболее простая в использовании JavaScript-библиотека для построения графиков и диаграмм/ [Электронный ресурс] - Режим доступа: http://www.jscharts.com/

[28] RGraph: HTML5/Javascript charts for your website（Interactive Javascript charts with the HTML5 canvas tag) – Canvas-библиотека для построения графиков и диаграмм/ [Электронный ресурс] - Режим доступа: http://www.rgraph.net/ (русс. - здесь: http://mobile-webapps.blogspot.com/2011/03/rgraph-html5.html)

[33] Wisdomweb.ru – учебники для веб-разработчиков (HTML, HTML5, CSS, Javascript, HDOM)/ [Электронный ресурс] - Режим доступа: http://www.wisdomweb.ru/