1 Объект document

В предыдущем параграфе мы рассматривали центральный объект на клиентской стороне – объект `window`. Уровнем ниже `window` располагается объект `document` (рис.1), который представляет Web-документ. То, что видит пользователь в окне документа: текст, ссылки, рисунки, кнопки и проч., – все это входит в состав объекта `document`. Эти составляющие документа представляются в объектной модели как свойства `document`.

Свойства объекта `document` возвращают данные о документе, его URL, дате последнего изменения, URL связанных документов, цветах текста и прочее. Мощные методы, которыми располагает этот объект, дают возможность программировать элементы документа. Связывая с элементами документа определенные события и записывая соответствующие сценарии, можно создать динамический документ, вид которого будет зависеть от действий пользователя.

Иерархическая структура и ссылки на элементы документа

Браузер ориентируется на определенную модель документа. Структуру объекта `document` можно изобразить в виде схемы (рис.1). Из схемы видно, что `document` содержит ряд семейств, свойств и методов, – они выделены в отдельные столбцы.

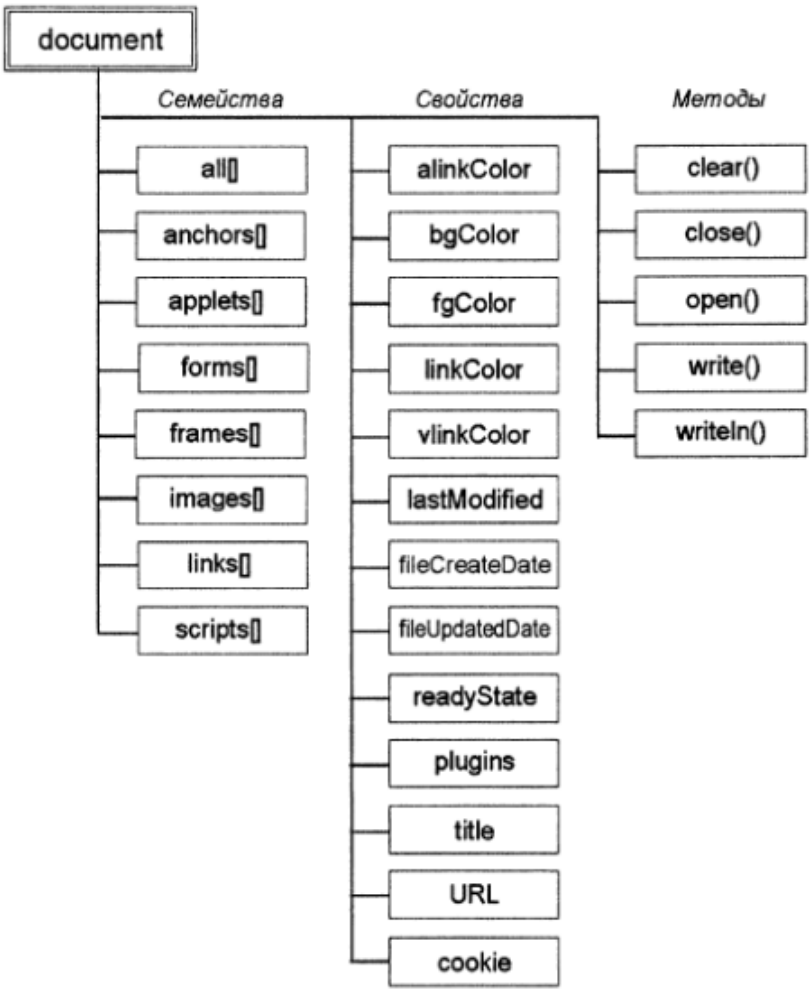

Рис.1. Семейства объекта `document`

На приведенном рисунке показаны только два верхних уровня иерархии, в то время как структура продолжается и на более низкие уровни. Так, в семействе `images[]` может содержаться несколько элементов-рисунков, а семейство `forms[]` включает в себя дочерние объекты `elements[].` Манипулирование различными компонентами Web-страниц – это фактически манипулирование элементами разных семейств объекта `document`. Поэтому очень важно знать, как получить доступ к нужному элементу в иерархической структуре.

Каждый объект в иерархической структуре относится к тому или иному семейству. Внутри семейства объекту присваивается порядковый номер (индекс) и может быть присвоено имя (идентификатор). Порядковые номера записываются так же, как номера элементов массива, например, `images[0]` (первый рисунок в документе) или `frames[1]` (второй фрейм). Нумерация элементов семейств начинается с 0.

Чтобы получить доступ к объекту, нужно указать объект верхнего уровня, т.е. `document`, и затем через точку записать путь по иерархической структуре к самому объекту, например,

```
document.images[1]
document.forms[0].elements[2]
```

Здесь `elements[]` обозначает массив элементов формы `forms[0]`. Понятно, что массив `elements[]` может иметь и другое имя, выбранное разработчиком. Более того, вместо обращения через порядковый номер элемента в массиве можно получить доступ, используя идентификаторы элементов и массивов, например,

```
document.forms[0].originalButton
```

или

```
document.myForm.originalButton
```

Использование идентификаторов удобно, когда вы имеете дело с большим документом. В этом случае ссылки на элементы с помощью индексов массивов могут запутать разработчика.

Запись ссылок на объект document

В предыдущем пункте мы разобрали, как осуществляется доступ к объектам, входящим в структуру `document`. А как обратиться к самому объекту `document`? Поскольку этот объект подчинен `window`, ссылки на него выглядят просто как

```
window.document.title
```

– это ссылка на заголовок. Вместо `window` в этих обращениях может использоваться имя окна.

Семейства, свойства и методы

Семейства объекта `document` напрямую связаны с элементами документа. Так, семейство `all` включает в себя все элементы, находящиеся в файле HTML, семейство `anchors` содержит все закладки, `applets` – апплеты на языке Java, `forms` – HTML-формы,

имеющиеся в документе, и т.д. Семейства, отображенные на рис.1, например, `frames` или `forms`, присутствуют в объекте `document` всегда, независимо от того, есть ли фреймы или формы в данном документе.

Семейства являются массивами с конечным числом элементов, причем количество элементов в семействе определяется свойством `length`. Чтобы определить полное количество элементов в HTML-документе, можно ввести в сценарий следующую команду:

```
alert(document.all.length);
```

Кроме семейств, объект `document` содержит свойства. Одно из них мы уже привели выше – это свойство `title`, которое содержит строку-заголовок документа. Перечислим остальные свойства `document`:

- `location` – возвращает унифицированный указатель ресурса;
- `lastModified` – строка, которая определяет дату и время последнего изменения, внесенного в документ; дату и время последнего изменения возвращает также свойство `fileModifiedDate`;
- `fileCreatedDate` – возвращает дату создания документа;
- `fileUpdatedDate` – возвращает дату последнего обновления файла сервером;
- `cookie` – возвращает список, содержащий пары имя-значение. `cookie` – это небольшие массивы данных, сохраняемых браузером и связанные с определенной Web-страницей или сервером. Свойство `cookie` позволяет считывать, создавать, модифицировать и уничтожать указанные данные;
- `referrer` – возвращает URL ресурса, с которого был выполнен переход по гиперссылке к данному документу;
- `bgColor` – свойство для чтения и записи, которое возвращает цвет фона документа; этому свойству отвечает одноименный атрибут в теге `<BODY>`;
- `fgColor` – свойство, отвечающее цвету текста документа и представленное атрибутом `text` в теге `<BODY>`;
- `linkColor` – свойство для чтения и записи, которое отвечает цвету гиперссылки; в теге `<BODY>` этому свойству отвечает атрибут`link`, который по умолчанию полагается равным `blue (#0000FF)`;
- `alinkColor` – свойство, возвращающее цвет активной гиперссылки и имеющее эквивалент `alink` в теге `<BODY>`; по умолчанию свойство `alinkColor` возвращает значение `red (#FF0000)`;
- `vlinkColor` – свойство, отвечающее цвету просмотренной гиперссылки и атрибуту `vlink` в теге `<BODY>`; по умолчанию `vlinkColor` возвращает значение `purple (#800080)`.

Каждое из приведенных свойств принимает строковое значение. Помимо свойств, объект `document` включает в себя ряд методов, позволяющих, например, динамически выводить текст документа, открывать и закрывать документ.

Вывод свойств объекта document

Предположим, вам необходимо просмотреть строковые значения свойств текущего документа. Запишем код страницы, содержащей простой сценарий, который выведет на экран сообщение с перечислением свойств документа:

```
HTML>
 <HEAD>
   <TITLE>Свойства</TITLE>
     <SCRIPT language="javascript">
    alert(document.title+'\n'+document.location+'\n'+
document.bgColor+'\n'+document.fgColor+'\n'+
document.linkColor+'\n'+document.alinkColor+'\n'+
document.vlinkColor+'\n'+document.lastModified);
     </SCRIPT>
 </HEAD>
 <BODY>
   <!-- Содержание домента -->
 </BODY>
</HTML>
```

На рис.2 показан пример выпадающего сообщения. Свойство `document.location` записано как URL файла, размещенного на жестком диске, а в качестве свойств, отвечающих цветам документа, указаны значения, принятые по умолчанию.

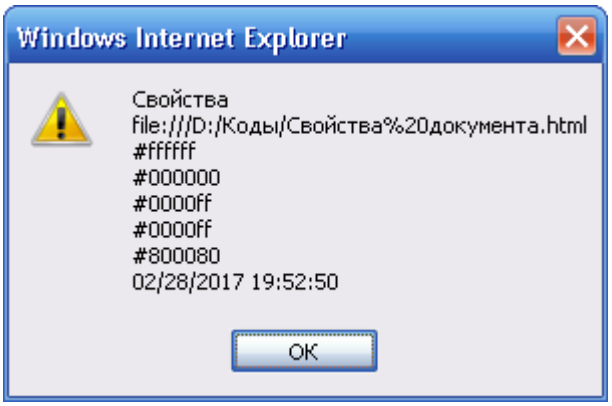

Рис.2. Пример значений свойств документа

2 Работа с документом

Как уже было отмечено, объект `document` включает в себя ряд методов, позволяющих динамически изменять отображение документа на экране. К таким методам относятся методы `write`, `writeln`, `open` и `close`.

Методы write и writeln

Метод `write` позволяет динамически изменять содержание Web-страницы. Ранее мы уже неоднократно пользовались методом `write` для вывода в окно браузера нового содержания, указанного в аргументах метода.

Метод `write` чаще всего используется для вывода текста в окно браузера, поскольку аргументы этого метода конвертируются в строки перед их выводом.

Часто считают, что метод `write` может работать только с одним аргументом. Поэтому при выводе нескольких аргументов применяют конкатенацию, например,

```
document.write("X равно:",+x);
```

Однако этому методу можно передавать более чем один аргумент, при этом аргументы конвертируются в строки. Благодаря этому возможны записи типа:

```
document.write("Сегодня: ",datetime.toLocaleString(),"<br>");
```

Метод `writeln` аналогичен методу `write`, однако он присоединяет к концу выводимой строки символ конца строки. Но, поскольку анализатор HTML-кода игнорирует разрывы строк, при составлении сценария для HTML-документа можно пользоваться как одним, так и другим методом, получая идентичные результаты.

Изменение содержания документа (методы open, close, write)

С помощью комбинации методов `write` и `open` можно создавать новые документы в других окнах или фреймах. Запишем простой код, который демонстрирует открытие потока HTML в новом окне (метод `open`), вывод информации в новом потоке (метод `write`) и закрытие потока (метод `close`):

```
<HTML>
 <HEAD>
   <TITLE>Поток в новом окне</TITLE>
 </HEAD>
 <BODY>
   <SCRIPT language="javascript">
     var NewWin=window.open("","Novost");
     //Открытие потока в новом окне NewWin
     NewWin.document.open();
     //Вывод информации в новое окно
     NewWin.document.write("Информация в новом окне");
     //Закрытие потока в новом окне
     NewWin.document.close();
    </SCRIPT>
 </BODY>
</HTML>
```

После загрузки данного документа будет открыто новое окно, в котором будет выведен текст «Информация в новом окне».

Задание цвета фона и текста (свойства bgColor и fgColor)

Как известно, фоновый цвет Web-страницы можно установить с помощью атрибута `bgColor` тега `<BODY>`. Однако цвет фона можно задавать динамически, как отклик на некоторое событие. Для этого используется одноименное свойство `bgColor` объекта

`document`. С помощью другого свойства, `fgColor`, возможно задание цвета текста в HTML-документе.

Составим сценарий, который позволит изменять цвет фона и текста. Новые значения цвета будем вводить с помощью формы. Код такого HTML-документа может выглядеть следующим образом:

```
<HTML>
 <HEAD>
   <TITLE>Определение цвета</TITLE>
   <SCRIPT language="javascript">
     function selColor(){
       document.bgColor=document.forms[0].bcol.value;
       document.fgColor=document.forms[0].tcol.value;
     }
   </SCRIPT>
 </HEAD>
 <BODY>
    <CENTER>
    <H2>Задайте цвета</H2>
    <FORM>
     Цвет фона<BR>
     <INPUT type="text" name="bcol" size=10 value="">
     <BR><BR>
     Цвет текста<BR>
     <INPUT type="text" name="tcol" size=10 value="">
     <BR><BR>
     <INPUT type="button" value="OK" onclick="selColor()">
    </FORM>
    </CENTER>
 </BODY>
</HTML>
```

В два текстовых поля вводятся значения цвета, и после нажатия на кнопку **ОК** заданные цвета будут присвоены фону и тексту документа. В результате Web-страница может приобрести вид, показанный на рис.3, на котором фону присвоен красный цвет, а шрифту – белый. Значение цвета можно вводить как в виде шестнадцатеричных чисел, так и словами.

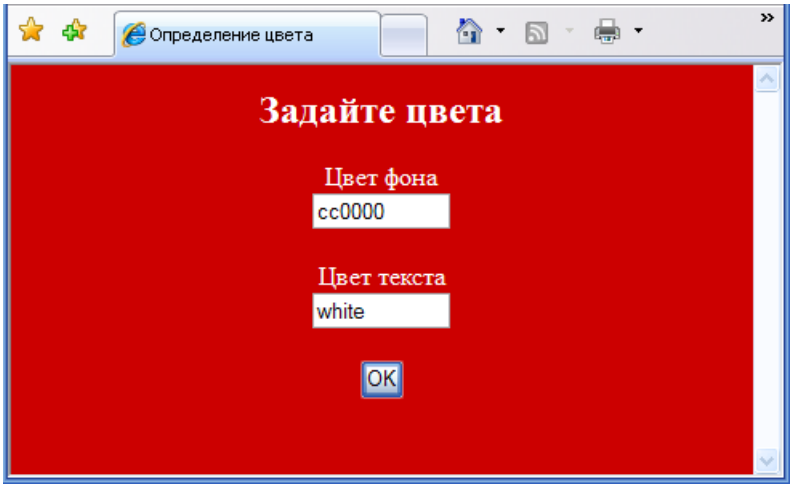

Рис.3. Задание значений атрибутов `bgColor` и `fgColor`

Задание цвета гиперссылок (свойства blinkColor, alinkColor и vlinkColor)

Как известно, атрибуты `link`, `alink` и `vlink` внутри тега `<BODY>` задают цвет гиперссылок. В JavaScript этим атрибутам соответствуют свойства `linkColor`, `alinkColor` и `vlinkColor` объекта `document`. Обращение к этим свойствам выполняется аналогично тому, как это делается для свойств `bgColor` и `fgColor`. Например, для управления цветом ссылок можно в сценарии задать функцию:

```
function setcolor(linkcol, alinkcol, vlinkcol){
  document.linkColor=linkcol;
  document.alinkColor=alinkcol;
  document.vlinkColor=vlinkcol;
}
```

С помощью такой функции можно динамически изменять цветовую схему гиперссылок. Например, оператор `setcolor("green","","olive")` задает исходный цвет ссылок и цвет просмотренных ссылок.

Изменение заголовка документа (свойство title)

В строке заголовка окна HTML-документа обычно отображается содержимое тега `<TITLE>`. Вы можете изменить содержимое строки заголовка, если воспользуетесь свойством `document.title.`, которое хранит заголовок текущего документа. Присвоив новое строковое значение свойству `document.title.`, вы измените заголовок.

Допустим, мы хотим, чтобы заголовок имел вид «`Добро пожаловать, ...`», где вместо многоточия должно подставляться имя посетителя. Для этого нужно ввести в сценарий JavaScript соответствующую функцию `deftitle()` и записать код документа в следующем виде:

```
<HTML>
 <HEAD>
   <TITLE>Изменение заголовка</TITLE>
   <SCRIPT language="javascript">
     function deftitle(){
       var gn=document.forms[0].guestname.value;
       document.title="Добро пожаловать,"+gn;
     }
   </SCRIPT>
 </HEAD>
 <BODY>
    <CENTER>
    <H2>Измените заголовок</H2>
    <FORM>
     Введите имя<BR>
     <INPUT type="text" name="guestname" size=10 value="">
     <BR><BR>
     <INPUT type="button" value="OK" onclick="deftitle()">
    </FORM>
    </CENTER>
 </BODY>
</HTML>
```

В текстовое поле нужно ввести имя посетителя и затем нажать кнопку **ОК**. После этого в строке заголовка документа появится нужная надпись (рис.4). Имя посетителя хранится в свойстве `guestname.value` элемента формы и затем после события `onclick` с помощью функции `deftitle()` имя выводится в строку заголовка.

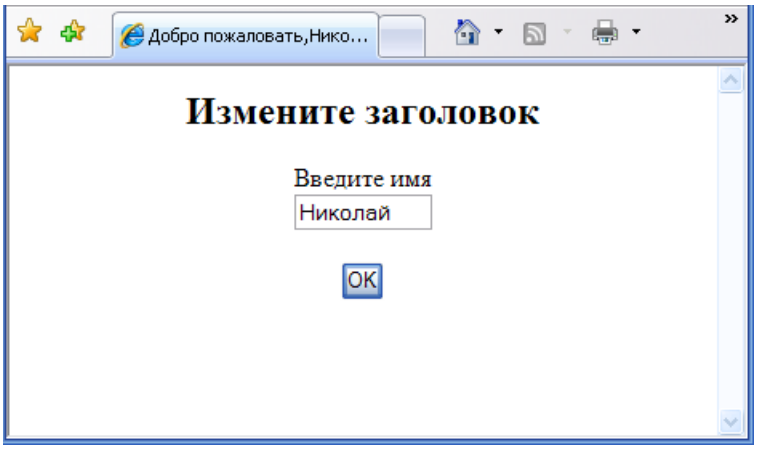

Рис.4. Пример задания заголовка документа

Даты

Иногда посетителю сайта бывает полезно знать, когда вносились последние изменения в текущую Web-страницу. Исходя из этого, он может судить, насколько актуальна представленная на странице информация.

Свойство `document.lastModified` дает информацию о том, когда документ модифицировался последний раз. Запишем простой сценарий:

```
<SCRIPT language="javascript">
   window.defaultStatus="дата последнего изменения
документа:"+document.lastModified;
</SCRIPT>
```

Сразу после загрузки документа в строке состояния появится сообщение с указанием даты последнего изменения документа.

3 Семейства элементов документов

Структура документа в объектной модели представляется набором семейств, показанных на рис.1. С помощью семейств `anchors`, `applets`, `frames`, `forms` и других обеспечивается доступ ко всем элементам HTML, содержащимся внутри документа. Для программирования Web-страниц нужно правильно представлять взаимосвязь различных элементов документа и способы доступа к ним. Эти вопросы рассматриваются в настоящем разделе.

Семейство all

Семейства, представленные различными HTML-элементами, являются, в свою очередь, подмножествами семейства `all`. Это семейство включает все элементы данного документа HTML. Универсальность семейства `all` заключается в том, что набор свойств и методов этого семейства поддерживается всеми другими семействами документа.

Количество элементов в документе (свойство length)

Самым простым и в то же время фундаментальным свойством семейства `all` является его размер. Для семейства `all` это свойство записывается как `document.all.length`. Включим в HTML-документ сценарий, который выведет на экран сообщение с указанием количества элементов в документе:

```
<HTML>
 <HEAD>
   <TITLE>Документ</TITLE>
 </HEAD>
 <BODY>
   <!--Содержание документа-->
   <SCRIPT language="javascript">
     alert(document.all.length);
   </SCRIPT>
  </BODY>
</HTML>
```

Для данного документа, содержащего в контейнере `BODY` только комментарий, будет выведено сообщение с указанием числа 6. Нетрудно видеть, что это число складывается из элементов: `HTML`, `HEAD`, `TITLE`, `BODY`, `SCRIPT` и комментария.

Для подсчета полного числа элементов сценарий необходимо размещать перед закрывающимся тегом `</BODY>`.

Доступ к элементам документа (метод item(), свойство tagName)

Проще всего доступ к элементам документа получить с помощью метода `item()`. В качестве аргумента этого метода указывается порядковый номер элемента или идентификатор строки с атрибутом `name` либо с `id` элемента.

Чтобы просмотреть множество `all`, можно составить простой сценарий с оператором цикла, который будет перебирать все элементы текущего документа, которые размещены до тега `<SCRIPT>`:

```
<SCRIPT language="javascript">
   for(var i=0; i<document.all.length; i++)
   alert(document.all.item(i).tagName);
</SCRIPT>
```

Для каждого `i` в операторе цикла выполняется сравнение `i<document.all.length`. Поскольку номера элементов семейства `all` начинаются с нуля, порядковый номер последнего элемента на единицу меньше величины `document.all.length`. Чтобы получить перечисление всех элементов, входящих в документ, необходимо сценарий поставить в конце тела `BODY`.

Массив `all[]` состоит из именованных элементов, поэтому вместо обращения к элементам через метод `item()` можно обратить к ним напрямую:

```
document.all[i].tagName
```

Итак, мы рассмотрели ссылки на элемент по его порядковому номеру в массиве `all[]`. Возможны также ссылки, в которых используются атрибуты `name` или `id`. Вернемся для

примера к разделу «Изменение заголовка документа». Применим для доступа к полю ввода guesname метод item, в котором в качестве аргумента записано имя поля:

```
document.all.item("guestname")
```

При обращении напрямую к элементу массива all[] можно записать:

```
document.all["guestname"]
```

Возможно также обращениме к элементу массива через свойство семейства:

```
document.all.guestname
```

Подытожим сказанное в простом примере. Введем в HTML-документ раздела «Изменение заголовка документа» следующий сценарий:

```
<SCRIPT language="javascript">
   alert(document.all.item("guestname").tagName);
   alert(document.all["guestname"].tagName);
   alert(document.all.guestname.tagName);
</SCRIPT>
```

После загрузки документа на экран последовательно будет выведено три одинаковых сообщения: «INPUT» – именно этот элемент имеет атрибут name="guestname". Отметим, что приведенный выше сценарий должен размещаться после элемента INPUT.

Определение индекса элемента (свойство sourceIndex)

Итак, для каждого документа браузер автоматически создает массив all[]. Доступ к какому-либо элементу этого массива возможен как по порядковому номеру элемента, так и по его имени. Однако порядковый номер элемента не всегда легко определить, особенно, если документ велик. В этом случае удобным может оказаться свойство sourceIndex, которое возвращает значение индекса элемента в массиве all[]. Например, в следующем документе:

```
<HTML>
 <HEAD>
   <TITLE id=nazv>Свойство sourceIndex</TITLE>
 </HEAD>
 <BODY>
   <H1 id=zag1>Определение индекса элемента</H1>
   <SCRIPT language="javascript">
      alert("Индексы элементов <TITLE> и <H1>:"
+nazv.sourceIndex+" и "+zag1.sourceIndex);
   </SCRIPT>
 </BODY>
</HTML>
```

на экран будет выведено сообщение, что индексы элементов <TITLE> и <H1> равны соответственно 2 и 4.