14. Программирование свойств окна браузера

Цель: обучиться JavaScript программированию, рассмотреть программирование свойств окна браузера, научиться управлять окнами с помощью JavaScript, использовать фреймы.

14.1. Теоретические сведения

Класс объектов Window — это самый старший класс в иерархии объектов JavaScript. К нему относятся объект Window и объект Frame. Объект Window ассоциируется с окном программы-браузера, а объект Frame — с окнами внутри окна браузера, которые порождаются последним при использовании автором HTML-страниц контейнеров FRAMESET и FRAME.

При программировании на JavaScript чаще всего используют следующие свойства и методы объектов типа Window:

Свойства	Методы	События
status	open()	Событий нет
location	close()	
history	focus()	
navigator		

Объект Window создается только в момент открытия окна. Все остальные объекты, которые порождаются при загрузке страницы в *окно,* есть свойства объекта Window. Таким образом, у Window могут быть разные свойства при загрузке разных страниц.

14.1.1. Поле статуса

Полем статуса (status bar) называют среднее поле нижней части окна браузера сразу под областью отображения HTML-страницы. В поле статуса отображается информация о состоянии браузера (загрузка документа, загрузка графики, завершение загрузки, запуск апплета и т.п.).

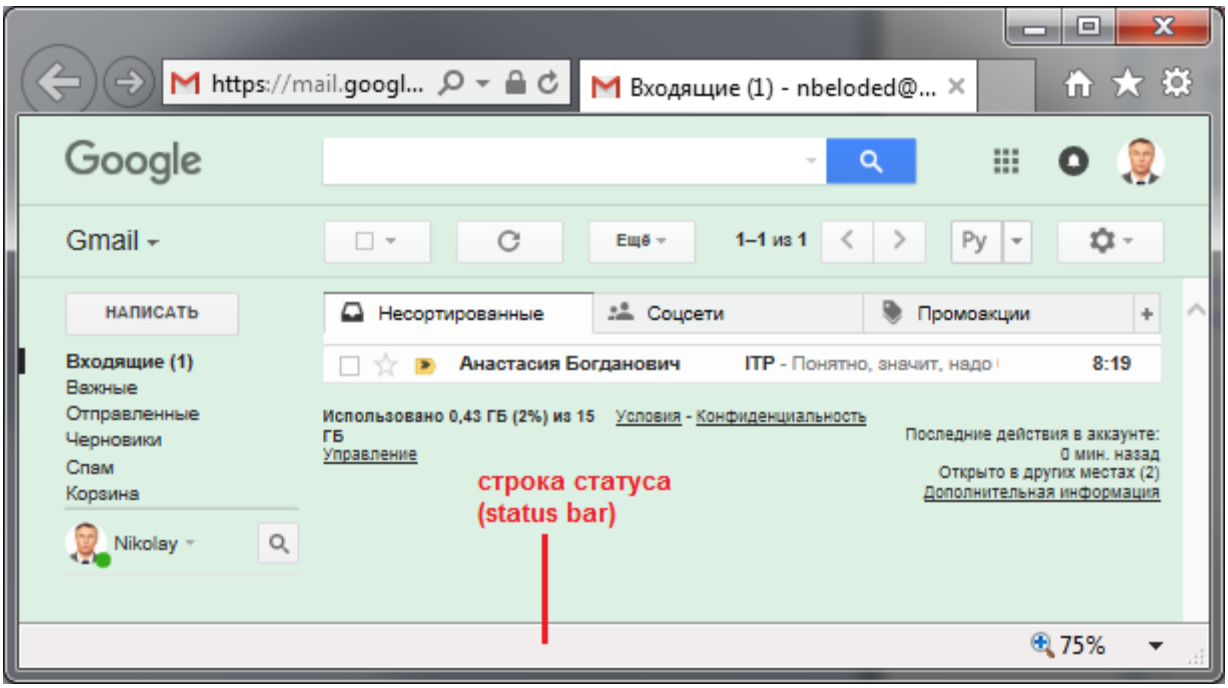


Рисунок 14.1. – Пример поля статуса

Программа на JavaScript имеет возможность работать с этим полем как с изменяемым свойством окна. При этом фактически с ним связаны два разных свойства:

- window.status;
- window.defaultStatus.

Разница между ними заключается в том, что браузер на самом деле имеет несколько состояний, связанных с некоторыми событиями. Состояние браузера отражается в сообщении в поле статуса. По большому счету, существует только два состояния: нет никаких событий (defaultStatus) и происходят какие-то события (status).

14.1.1.1. Программируем status

Свойство status связано с отображением сообщений о событиях, отличных от простой загрузки страницы. Например, когда курсор мыши проходит над гипертекстовой ссылкой, URL, указанный в атрибуте HREF, отображается в поле статуса. При попадании курсора мыши на поле, свободное от ссылок, в поле статуса восстанавливается сообщение по умолчанию (Document: Done). Эта техника реализована на данной странице при переходе на описание свойств status и defaultStatus

```
<a href=#status onmouseover="window.status='Jump to status description'; return true;"
    onmouseout="window.status='Status bar programming'; return true;">
      window.status
</a>
```

В документации по JavaScript указано, что обработчик событий mouseover и mouseout должен возвращать значение true. Это нужно для того, чтобы браузер не выполнял действий, заданных по умолчанию. Практика показывает, что Netscape Navigator 4. 0 прекрасно обходится и без возврата значения true.

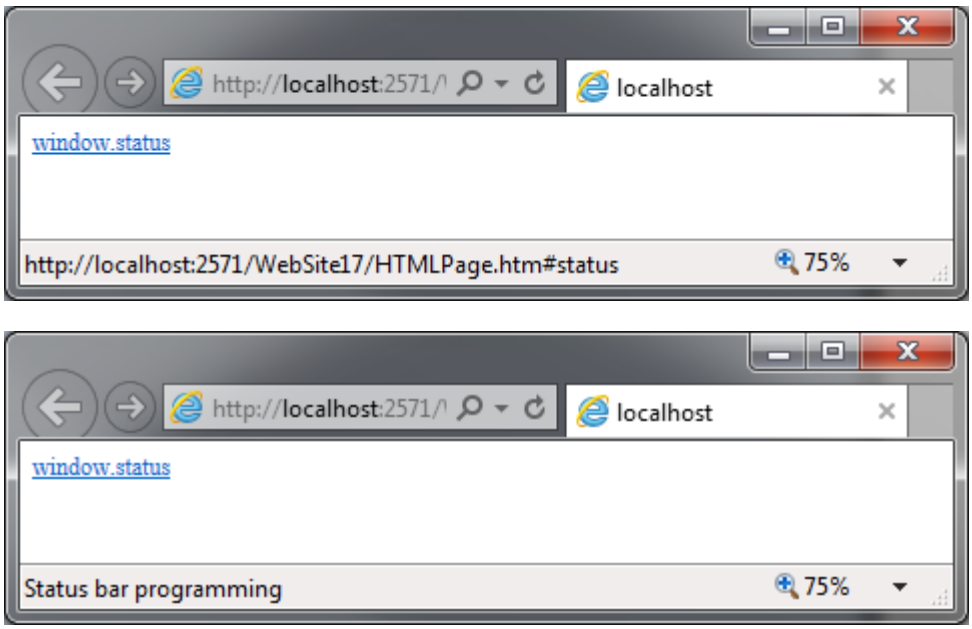


Рисунок 14.2. – Демонстрация использования свойства status

14.1.1.2. Программируем defaultStatus

Свойство defaultStatus определяет текст, отображаемый в поле статуса, когда никаких событий не происходит. В нашем документе мы определили его при загрузке документа:

```
<body onload="window.defaultStatus='Status bar programming'; ">
</body>
```

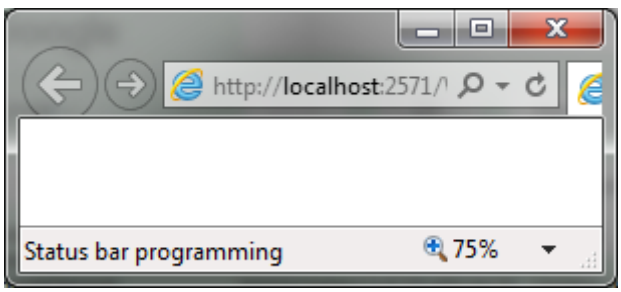


Рисунок 14.3. – Демонстрация использования свойства status

Это сообщение появляется в тот момент, когда загружены все компоненты страницы (текст, графика, апплеты и т. п.). Оно восстанавливается в строке статуса после возврата из любого события, которое может произойти при просмотре документа. Любопытно, что движение мыши по свободному от гипертекстовых ссылок полю страницы приводит к постоянному отображению defaultStatus.

14.1.2. Поле location

В поле location отображается URL загруженного документа. Если пользователь хочет вручную перейти к какой-либо странице (набрать ее URL), он делает это в поле location. Поле располагается в верхней части окна браузера ниже панели инструментов, но выше панели личных предпочтений.

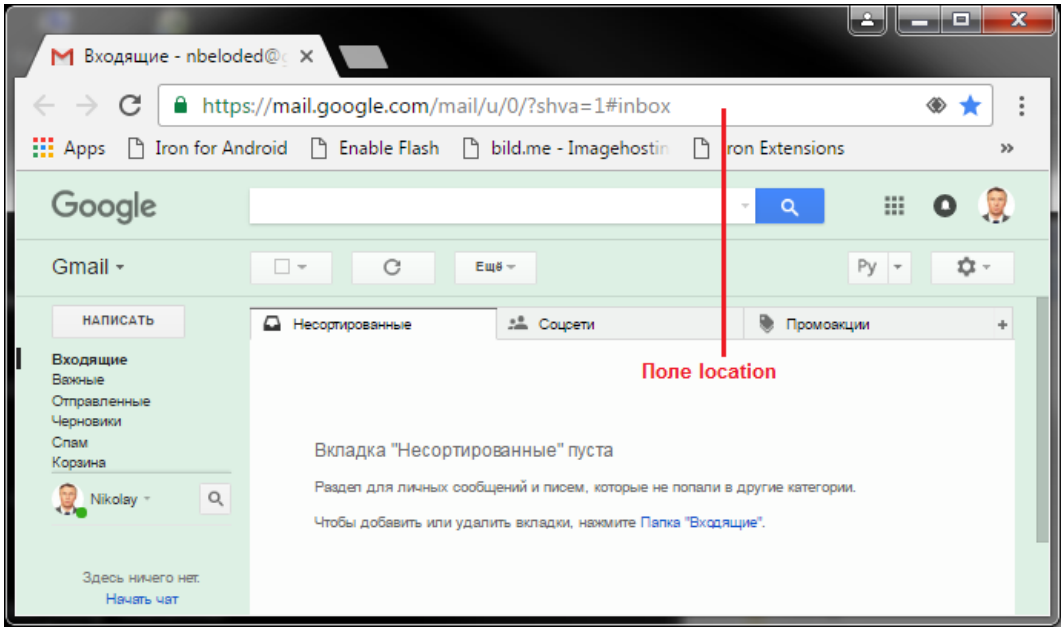


Рисунок 14.4. – Поле Location

Вообще говоря, Location — это объект. Из-за изменений в версиях JavaScript класс Location входит как подкласс и в класс Window, и в класс Document. Мы будем рассматривать Location только как window, location. Кроме того, Location — это еще и подкласс класса URL, к которому относятся также объекты классов Area и Link. Location наследует все свойства URL, что позволяет получить доступ к любой части схемы URL.

Рассмотрим характеристики и способы использования объекта Location:

- свойства;
- методы;
- событий, характеризующих Location, нет.

Как мы видим, список характеристик объекта Location неполный.

14.1.2.1. Свойства

Предположим, что браузер отображает страницу, расположенную по адресу:

http://intuit.ru:80/r/dir/page? search#mark

Тогда свойства объекта Location примут следующие значения:

```
window.location.href = http://intuit.ru:80/r/dir/page? search#mark
window. location. protocol = http;
window.location.hostname = intuit.ru;
window.location.host = intuit.ru:80;
window.location.port = 80
window.location.pathname = /r/dir/;
window.location.search = search;
window.location.hash = mark;
```

14.1.2.2. Методы

Методы Location предназначены для управления загрузкой и перезагрузкой страницы. Это управление заключается в том, что можно либо перезагрузить документ (reload), либо загрузить (replace). При этом в историю просмотра страниц (history) информация не заносится:

```
window.location.reload(true);
window.location.replace('#top');
```

Метод reload() полностью моделирует поведение браузера при нажатии на кнопку Reload в панели инструментов. Если вызывать метод без аргумента или указать его равным true, то браузер

проверит время последней модификации документа и загрузит его либо из кэша (если документ не был модифицирован), либо с сервера. Такое поведение соответствует простому нажатию на кнопку Reload. Если в качестве аргумента указать false, то браузер перезагрузит текущий документ с сервера, несмотря ни на что. Такое поведение соответствует одновременному нажатию на Reload и кнопку клавиатуры Shift (Reload+Shift).

Метод replace () позволяет заменить одну страницу другой таким образом, что это замещение не будет отражено в трассе просмотра HTML- страниц (history), и при нажатии на кнопку Back из панели инструментов пользователь всегда будет попадать на первую загруженную обычным способом (по гипертекстовой ссылке) страницу. Напомним, что при изменении свойств Location также происходит перезагрузка страниц, но в этом случае записи об их посещении в history пропадают.

14.1.3. История посещений (History)

История посещений (трасса) страниц World Wide Web позволяет пользователю вернуться к странице, которую он просматривал несколько минут (часов, дней) назад. История посещений в JavaScript трансформируется в объект класса history. Этот объект указывает на массив URL-страниц, которые пользователь посещал и которые он может получить, выбрав из меню браузера режим GO. Методы объекта history позволяют загружать страницы, используя URL из этого массива.

Чтобы не возникло проблем с безопасностью браузера, путешествовать по History можно, только используя индекс URL. При этом URL, как текстовая строка, программисту недоступен. Чаще всего этот объект используют в примерах или страницах, на которые могут быть ссылки из нескольких разных страниц, предполагая, что можно вернутся к странице, из которой пример будет загружен:

```
<form>
      <input type="button" value="Назад" onclick="history.back()" />
</form>
```

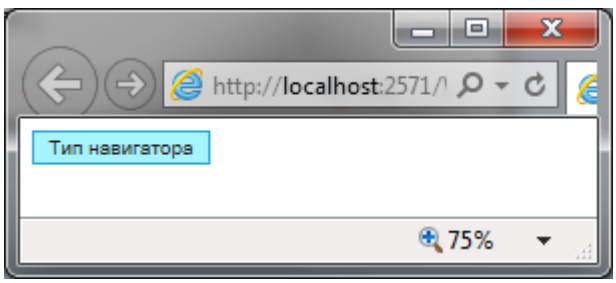
Данный код отображает кнопку «Назад», нажав на которую мы вернемся на предыдущую страницу.

14.1.4. Тип браузера (объект Navigator)

В связи с войной браузеров (которая, можно считать, уже закончилась и пользу Microsoft Internet Explorer) стала актуальной задача настройки страницы на конкретную программу просмотра. При этом возможны два варианта: определение типа браузера на стороне сервера и определение типа браузера на стороне клиента. Для последнего варианта в арсенале объектов JavaScript существует объект Navigator. Этот объект — свойство объекта Window.

Рассмотрим простой пример определения типа программы просмотра:

```
<form>
      <input type="button" value="Тип навигатора"
          onclick="window.alert(window.navigator.userAgent); ">
</form>
```



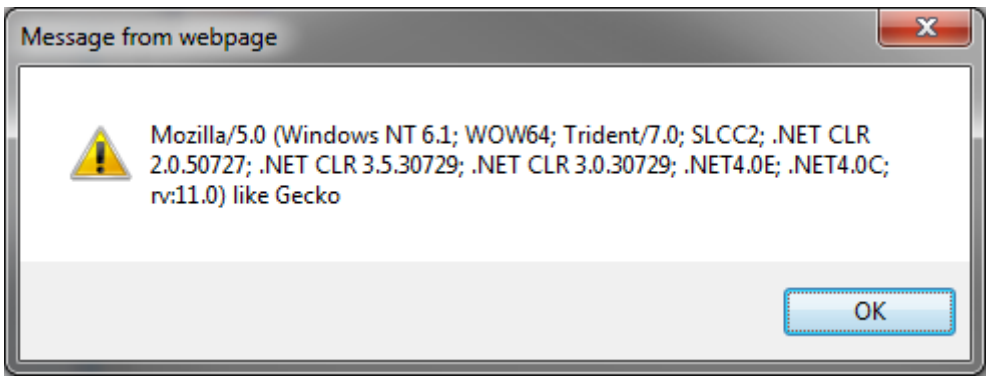


Рисунок 14.5. – Пример использования объекта Navigator

При нажатии на кнопку отображается окно предупреждения. В нем содержится строка userAgent, которую браузер помещает в соответствующий HTTP-заголовок.

Эту строку можно разобрать по компонентам, например:

Листинг 14.1.

```
navigator.appName = Microsoft Internet Explorer
navigator.appCodeName = Mozilla
navigator.appVersion =4.0 (compatible; MSIE 5.5; Windows 98)
navigator.userAgent = Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)
```

У объекта Navigator есть еще несколько интересных с точки зрения программирования применений. Например, проверка поддержки Java. Проиллюстрируем эту возможность на примере:

Листинг 14. 2.

```
<script language="JavaScript" type="text/JavaScript">
document.write("<P ID=red>"); if(navigator.javaEnabled()==true)
      document.write("Ваша программа поддерживает исполнение Java-
апплетов");
if(navigator.javaEnabled()==false)
      document.write("<FONT COLOR=red>Ваша программа не поддерживает
исполнение Java-апплетов </FONT>");
</script>
```

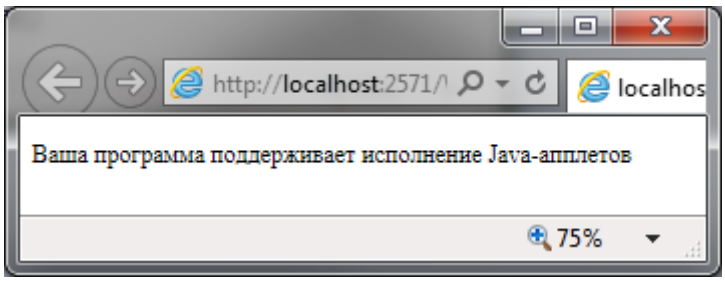


Рисунок 14.6. – Проверка поддержки Java браузером или программой

Аналогично можно проверить форматы графических файлов, которые поддерживает ваш браузер:

Листинг 14. 3.

```
<script language="JavaScript" type="text/JavaScript">
      if(navigator.mimeTypes['image/gif'] != null)
            document.write("Ваш браузер поддерживает GIF<BR>");
      if(navigator.mimeTypes['image/tif'] == null)
            document.write("Ваш браузер не поддерживает TIFF");
</script>
```

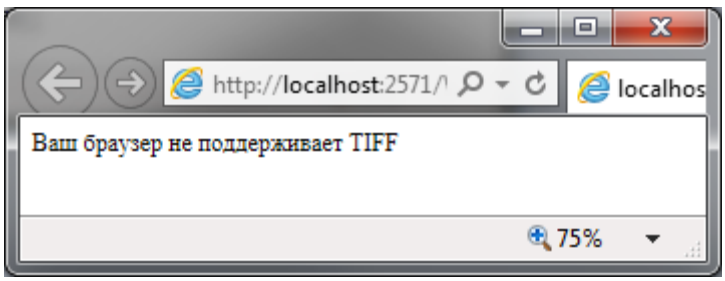


Рисунок 14.7. – Проверка поддержки форматов графических файлов браузером или программой

К сожалению, такая проверка не позволяет определить наличие автоматической подгрузки графики.

14.1.5. Управление окнами

Что можно сделать с окном? Открыть (создать), закрыть (удалить), положить его поверх всех других открытых окон (передать фокус). Кроме того, можно управлять свойствами окна и

свойствами подчиненных ему объектов. Описанию основных свойств посвящен раздел «Программируем свойства окна браузера», поэтому сосредоточимся на простых и наиболее популярных методах управления окнами:

- alert();
- confirm();
- prompt();
- open();
- close();
- focus();
- setTimeout();
- clearTimeout().

Здесь не указаны только два метода: scroll() и blur().

Первый позволяет прокрутить окно на определенную позицию. Но его очень сложно использовать, не зная координат окна. Последнее является обычным делом, если только не используется технология программирования слоев или CSS (Cascading Style Sheets).

Второй метод уводит фокус с окна. При этом совершенно непонятно, куда этот фокус будет передан. Лучше целенаправленно передать фокус, чем просто его потерять.

14.1.5.1. window. alert()

Метод alert( ) позволяет выдать *окно* предупреждения:

```
<a href="javascript:window.alert('Внимание')">
      Повторите запрос!
</a>
```

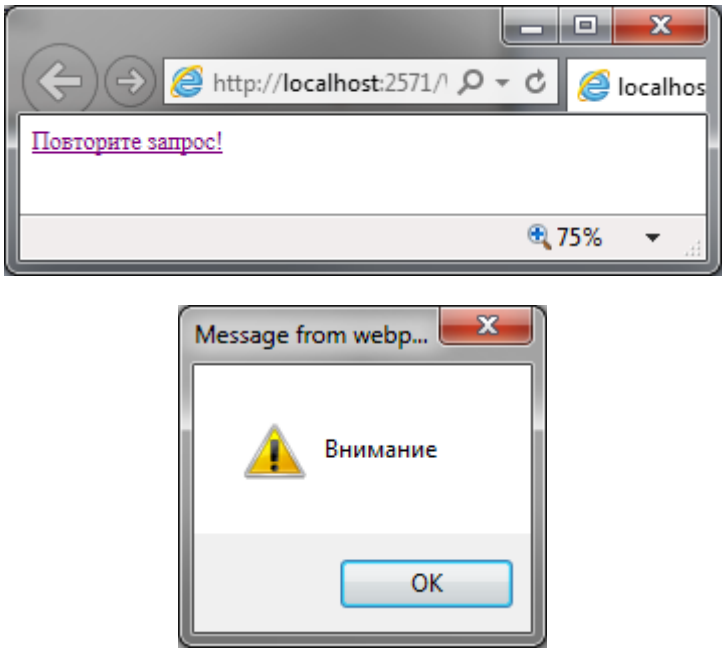


Рисунок 14.8. – Окно предупреждения

Все очень просто, но нужно иметь в виду, что сообщения выводятся системным шрифтом, следовательно, для получения предупреждений на русском языке нужна локализованная версия ОС.

14.1.5.2. window. confirm()

Метод confirm() позволяет задать пользователю вопрос, на который тот может ответить либо положительно, либо отрицательно:

```
<form>
      <input type="button" value="Вы знаете JavaScript? "
            onclick="
                if (window.confirm('Знаю все!') == true) {
                    document.forms[0].elements[0].value='Да';
                } else {
                    document.forms[0].elements[0].value='Нет';
                }
                "/>
```

```
                </br>
        </form>
```

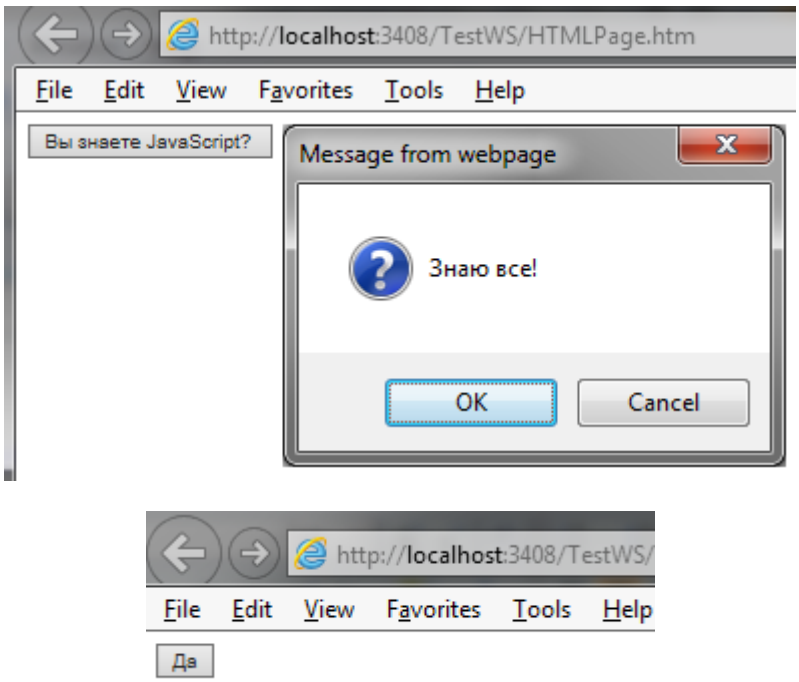


Рисунок 14.9. – Пример использования window.confirm()

Все ограничения для сообщений на русском языке, которые были описаны для метода alert( ), справедливы и для метода confirm().

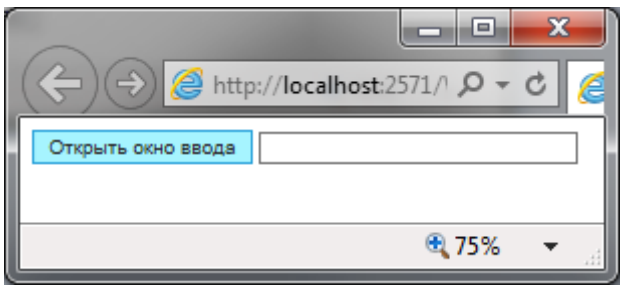
14.1.5.3. window.prompt()

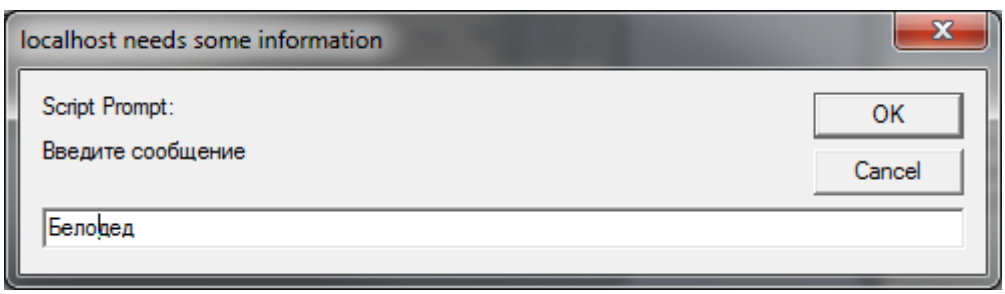
Метод prompt() позволяет принять от пользователя короткую строку текста, которая набирается в поле ввода информационного окна:

```
        <form>
                <input type="button" value="Открыть окно ввода"

        onclick="document.forms[0].elements[1].value=window.prompt('Введите
сообщение'); " >
                <input size="30" >
        </form>
```

Введенную пользователем строчку можно присвоить любой переменной и потом разбирать ее в JavaScript-программе.





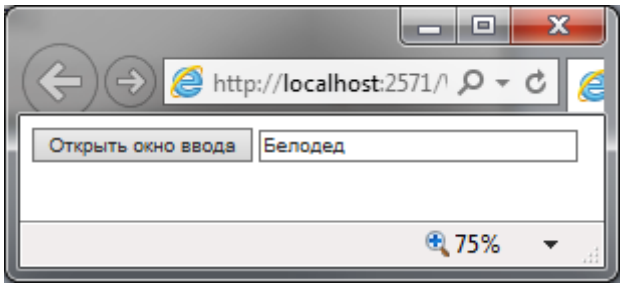


Рисунок 14.10. – Пример использования window.prompt()

14.1.5.4. window. open()

У этого метода окна атрибутов больше, чем у некоторых объектов. Метод open() предназначен для создания новых окон. В общем случае его синтаксис выглядит следующим образом:

open("URL", "window_name", "param, param,...", replace);

где: URL — страница, которая будет загружена в новое *окно*, window_name — имя окна, которое можно использовать в атрибуте TARGET в контейнерах A и FORM.

Параметры	Назначение
replace	Позволяет при открытии окна управлять записью в массив History
param	Список параметров
width	Ширина окна в пикселах
height	Высота окна в пикселах
toolbar	Создает окно с системными кнопками браузера
location	Создает окно с полем location
directories	Создает окно с меню предпочтений пользователя
status	Создает окно с полем статуса status
menubar	Создает окно с меню
scrollbars	Создает окно с полосами прокрутки
resizable	Создает окно, размер которого можно будет изменять

Приведем следующий пример:

Листинг 14. 4.

```
<form>
      <input type="button" value="Простое окно"
            onclick="window.open('about:blank', 'test1', 'directories=no,
height=200, location=no, menubar=no, resizable=no, scrollbars=no, status=no,
toolbar=no, width=200'); " />
      <input type="button" value="Сложное окно"
           onclick="window.open('about:blank', 'test2', 'directories=yes,
height=200, location=yes, menubar=yes, resizable=yes, scrollbars=yes,
status=yes, toolbar=yes, width=200'); " />
</form>
```

При нажатии кнопки «простое окно» получаем окно со следующими параметрами:

- directories=no — окно без меню
- height=200 — высота 200 px
- location=no — поле location отсутствует
- menubar=no — без меню
- resizable=no — размер изменять нельзя
- scrollbars=no — полосы прокрутки отсутствуют
- status=no — статусная строка отсутствует

- toolbar=no — системные кнопки браузера отсутствуют
- width=200 — ширина 200

При нажатии кнопки «сложное окно» получаем окно, где:

- directories=yes — окно с меню
- height=200 - высота 200 px
- location=yes — поле location есть
- menubar=yes — меню есть
- resizable=yes — размер изменять можно
- scrollbars=yes - есть полосы прокрутки
- status=yes — статусная строка есть
- toolbar=yes — системные кнопки браузера есть
- width=200 — ширина 200

14.1.5.5. window.close()

Метод close() — это оборотная сторона медали метода open(). Он позволяет закрыть *окно.* Чаще всего возникает вопрос, какое из окон, собственно, следует закрыть. Если необходимо закрыть текущее, то:

```
window.close();
self.close();
```

Если необходимо закрыть родительское окно, т. е. окно, из которого было открыто текущее, то:

```
window.opener.close();
```

Если необходимо закрыть произвольное окно, то тогда сначала нужно получить его идентификатор:

```
id = window.open();
id.close();
```

Как видно из последнего примера, закрывают окно не по имени (значение атрибута TARGET тут ни при чем), а используют указатель на объект.

14.1.5.6. window.focus()

Метод focus() применяется для передачи фокуса в *окно,* с которым он использовался. Передача фокуса полезна как при открытии окна, так и при его закрытии, не говоря уже о случаях, когда нужно выбирать окна. Рассмотрим пример.

Открываем *окно* и, не закрывая его, снова откроем окно с таким же именем, но с другим текстом. Новое *окно* не появилось поверх основного окна, так как фокус ему не был передан. Теперь повторим открытие окна, но уже с передачей фокуса:

Листинг 14.5.

```
<html>
<title></title>

<head>
<script language="JavaScript" type="text/JavaScript">
function myfocus(a)
{
      id    =    window.open("",    "example",    "scrollbars,    width=300,
height=200");
      //открываем окно и заводим переменную с указателем на него
      //если окно с таким именем существует, то новое окно не создается,
      //а открывается поток для записи в окно с этим именем
      if (a == 1) {
          id.document.open();
          //открываем поток ввода в уже созданное окно
          id.document.write("<center>Открыли окно в первый раз");
          //Пишем в этот поток
      }
      if (a == 2) {
        id.document.open();
        id.document.write("<center>Открыли окно во второй раз");
```

```
            }
            if (a == 3) {
                  id.focus();
                  //передаем фокус, затем выполняем те же действия, что и в
                  //предыдущем случае
                  id.document.open();
                  id.document.write("<center>Открыли окно в третий раз");
            }
                              id.document.write("<form><input      type='button'
onclick='window.close(); 'value='Закрыть окно'/></<center>");
            id.document.close();
      }
      </script>
      </head>
      <form>
            <input type="button" value="myfocus(1)" onclick="myfocus(1);" /><br
/>
            <input type="button" value="myfocus(2)" onclick="myfocus(2);" /><br
/>
            <input type="button" value="myfocus(3)" onclick="myfocus(3);" /><br
/>
      </form>
      </html>
```

Поскольку мы пишем содержание нового окна из окна старого (родителя), то в качестве указателя на объект используем значение переменной id.

14.1.5.7. window. setTimeout()

Метод setTimeout() используется для создания нового потока вычислений, исполнение которого откладывается на время (ms), указанное вторым аргументом:

idt = setTimeout("JavaScript_код", Time);

Типичное применение этой функции — организация автоматического изменения свойств объектов. Например, можно запустить часы в поле формы:

Листинг 14. 6.

```
<html>
<title></title>

<head>
<script language="JavaScript" type="text/JavaScript">
    var flag = 0;
    var idp = null;
    function myclock()
    {
          if(flag == 1)
                d = new Date();
          window.document.c.f.value =
          d.getHours() + ": " + d.getMinutes() + ": " + d.getSeconds();
        idp=setTimeout("myclock(); ", 500);
    }

    function flagss()
    {
          if(flag == 0) flag = 1; else flag = 0;
    }

</script>
</head>
<form name="c">
      Текущее время:
      <input name="f" size="8" />
      <input    type="button"    value="Start/Stop"    onclick="flagss();
myclock(); " />
</form>
</html>
```

Нужно иметь в виду, что поток порождается всегда, даже в том случае, когда часы стоят. Если бы он создавался только при значении переменной flag равном единице, то при значении 0 он исчез бы, тогда при нажатии на кнопку часы продолжали бы стоять.

14.1.5.8. window. clearTimeout

Метод clearTimeout() позволяет уничтожить поток, вызванный методом setTimeout(). Очевидно, что его применение позволяет более эффективно распределять ресурсы вычислительной установки. Для того чтобы использовать этот метод в примере с часами, нам нужно модифицировать функции и форму:

Листинг 14. 7.

```
<html>
<title></title>

<head>
<script language="JavaScript" type="text/JavaScript">
    var idp1 = null;
    function startX() {
        d = new Date();
        window.document.c1.f1.value =
          d.getHours() + ": " + d.getMinutes() + ": " + d.getSeconds();
        idp1 = setTimeout("startX(); ", 500);
    }

    function stop() {
        clearTimeout(idp1);
        idp1 = null;
    }

</script>
</head>
<form name="c1">
Текущее время:
      <input name="f1" size="8" />
      <input  type="button"  value="Start"  onclick="if(idp1  ==  null)
startX(); " />
      <input type="button" value="Stop" onclick="if(idp1 != null) stop();
" />
</form>

</html>
```

В данном примере для остановки часов используется метод clearTimeout() При этом, чтобы не порождалось множество потоков, проверяется значение указателя на объект потока.

Замечание.

Имя функции `start()` приводит к ошибке, а `startX()` превосходно работает.

14.1.6. Фреймы (Frames)

Фреймы — это несколько видоизмененные окна. Отличаются они от обычных окон тем, что размещаются внутри них. У *фрейма* не может быть ни панели инструментов, ни меню, как в обычном окне. В качестве поля статуса фрейм использует поле статуса окна, в котором он размещен. Существует и ряд других отличий.

Мы остановимся на:

- иерархии фреймов;
- именовании фреймов;
- передаче данных во фрейм.

Естественно, что иерархия определяет и правила именования *фреймов,* и способы передачи фокуса фрейму.

14.1.7. Иерархия фреймов

Рассмотрим сначала простой пример. Разделим экран на две вертикальные колонки:

index.html

```
<html>
<head>
```

```
</head>
      <frameset cols="50%, *">
            <frame name="left" src="left.html">
            <frame name="right" src="right.html">
      </frameset>
</html>
```

left.html

```
<html>
ЛЕВЫЙ
</html>
```

right.html

```
<html>
ПРАВЫЙ
</html>
```

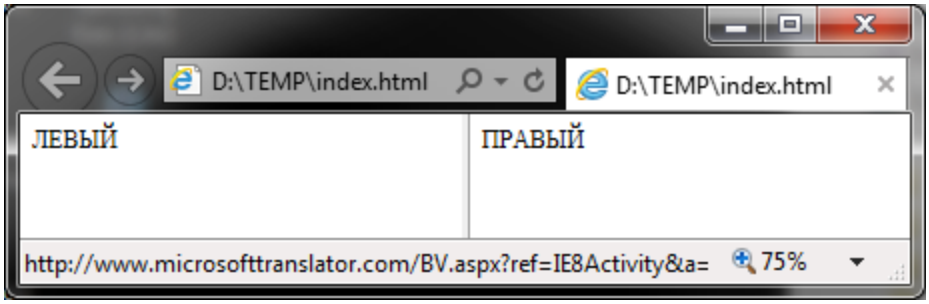


Рис. 14.11. – Фрейм с двумя вертикальными колонками

Назовем окно, в которое помещают *фреймы,* _top(_parent). Усложним пример: разобьем правый *фрейм* на два по горизонтали:

Index1.html

```
<html>
<head>
</head>
      <frameset cols="50%, *">
            <frame name ="left" src="left.html">
            <frameset rows="50%, *">
                  <frame name="top" src="top.html">
                  <frame name="bottom" src="bottom.html">
            </frameset>
      </frameset>
</html>
```

top.html

```
<html>
ВЕРХ
</html>
```

bottom.html

```
<html>
НИЗ
</html>
```

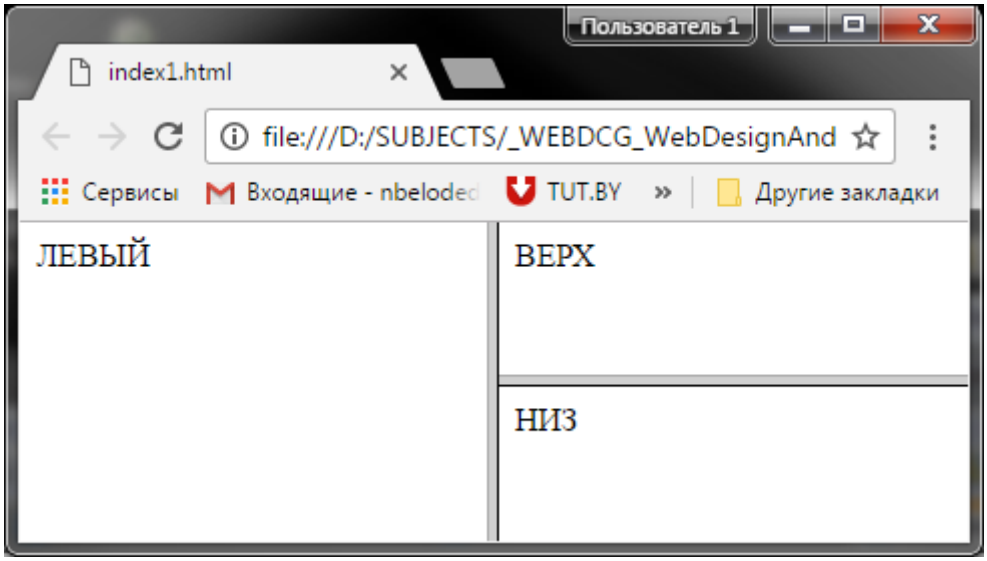


Рис. 14.12. – Правый фрейм, разбитый на два по горизонтали

Обратите внимание на два момента: во-первых, следует различать _top и top, во-вторых, исчез *фрейм* right. По поводу первого замечания: _top — это зарезервированное имя старшего окна, а top — имя *фрейма,* которое назначил ему автор страницы. По поводу второго замечания: старшим окном для всех фреймов является все окно браузера, фрейма с именем right в данном случае не существует.

Для того чтобы он появился, нужно свести оба наших примера в один. Это значит, что во *фрейм* right мы снова должны загрузить фреймовый документ.

Первый документ:

index2.html

```
<html>
<head>
</head>
      <frameset cols="50%, *">
            <frame name ="left" src="left2.html">
            <frame name ="right" src="right2.html">
      </frameset>
</html>
```

left2.html

```
<html>
ЛЕВЫЙ
</html>
```

right2.html

```
<html>
<head>
</head>
      <frameset rows="50%, *">
            <frame name="top" src="top.html">
            <frame name="bottom" src="bottom.html">
      </frameset>
</html>
```

В этом случае подчинение страниц будет выглядеть иначе, чем в примере с тремя фреймами.

Таким образом, мы получили тот же результат, что и с тремя фреймами и одним старшим окном. Однако этот вариант более гибкий: он позволяет задействовать фрейм, содержащий горизонтальную разбивку. Именно такой прием используется на домашней странице «Web-инжиниринг».

Такая интерпретация фреймовой структуры страницы находит отражение и в именовании фреймов JavaScript.

14.1.8. Именование фреймов

Обратиться к фрейму можно либо по имени, либо как к элементу массива frames[]. Рассмотрим HTML-документ:

```
<HTML>
	<HEAD>
	</HEAD>
	<FRAMESET R0WS="20%, *">
		<FRAME NAME=left SRC=framel.htm>
		<FRAME NAME=right SRC=frame2.htm>
	</FRAMESET>
</HTML>
```

Предположим, что на странице, загруженной в правый *фрейм,* есть две картинки. Для изменения свойства src второй из них можно использовать следующие записи:

```
top.frames[1].images[1].src="pic.gif";
```

или

```
top.right.images[l].src="pic.gif";
```

В связи с индексированием фреймов возникает вопрос о том, как они нумеруются в одномерном встроенном массиве фреймов объекта Window. Проиллюстрируем это на примере:

index3.html

```
<html>
<head>
</head>
	<frameset rows="50, *, 50">
		<frame name ="top" src="top3.html">
		<frameset cols="150, *, 150">
			<frame name ="left" src="left3.html">
			<frame name ="center" src="center3.html">
			<frame name ="right" src="right3.html">
		</frameset>
		<frame name ="bottom" src="bottom3.html">
	</frameset>
</html>
```

top3.html

```
<html>
Фрейм[0](Верхний)
</html>
```

center3.html

```
<html>
Фрейм[2](Центральный)
</html>
```

right3.html

```
<html>
Фрейм[3](Правый)
</html>
```

bottom3.html

```
<html>
Фрейм[4](Нижний)
</html>
```

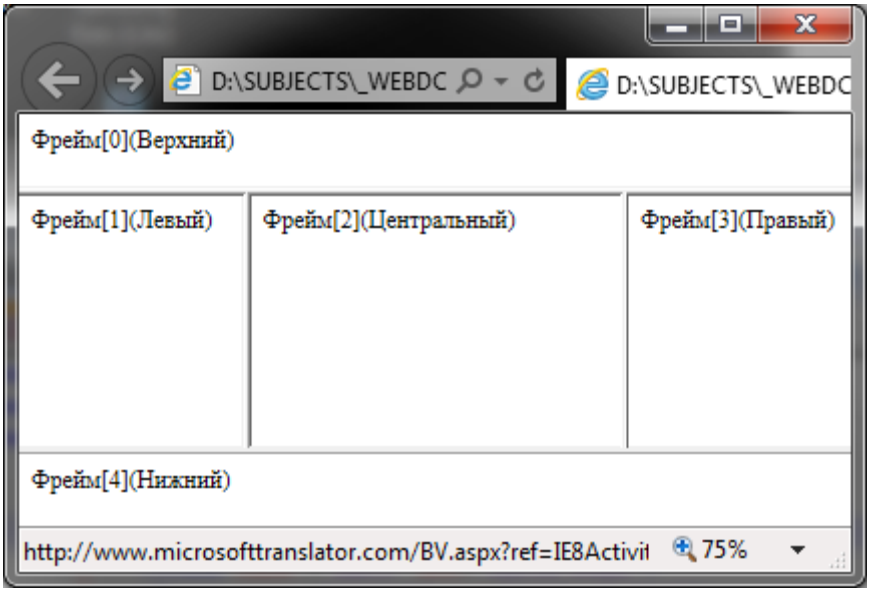


Рисунок 14.13. – Центральный фрейм, разбитый на три вертикальных

При нумеровании фреймов в одномерном массиве фреймов на странице система придерживается правила «слева направо, сверху вниз».

14.1.9. Передача фокуса во фрейм

Обычной задачей при разработке типового Web-узла является загрузка результатов исполнения CGI-скрипта во *фрейм,* отличный от фрейма, в котором вводятся данные для этого скрипта. Если путь загрузки результатов фиксированный, то можно просто использовать атрибут TARGET формы. Сложнее, если результат работы должен быть загружен в разные фреймы, в зависимости от выбранной кнопки, например.

Эту задачу можно решать по-разному: открывать ранее открытое окно или переназначать свойство target. Последнее решение, конечно, более изящное, с него и начнем:

Листинг 14. 8.

```
function load()
{
        if(self.document.f.s.options[document.f.s.selectedIndex]. text=="top")
        {
                document.f.target = "mytop";
                self.top.frames[2].document. open();
                self.top.frames[2].document. close();
        }
        else
        {
                document.f.target = "mybottom";
                self.top.frames[1].document.open();
                self.top.frames[1].document.close();
   }
   return true;
}
```

Функция load() вызывается как обработчик события submit, она является логической функцией. Возврат значения true позволяет реализовать перезагрузку документа.

Теперь рассмотрим **второй вариант**. Его идея состоит в том, что при попытке открыть *окно* с именем существующего окна новое *окно* не открывается, а используется уже открытое. *Фрейм* — это тоже окно, поэтому на него данное правило распространяется, но вот функция, которая реализует этот вариант, отличается от предыдущей:

Листинг 14. 9.

```
function load()
{
        if(self.document.f.s.options[document.f.s.selectedIndex].text=="top")
        {
                window.open("./framer.htm",      "mytop");      self.top.frames[2].document.open();
self.top.frames[2].document.close();
```

```
                }
                else
                {
                        window.open("./framer.htm",    "mybottom");    self.top.frames[1].document.open();
self.top.frames[1].document.close();
                }
                return false;
        }
```

Обратите внимание на то, что данная функция возвращает значение false. Это происходит потому, что надо маскировать событие submit. Ведь документ уже перезагружен и снова его загружать не надо.

14.2. Контрольные вопросы

1. Какие состояния браузера существуют?
2. Для чего предназначено поле location
3. Как используется история посещений?
4. Что можно сделать с окном?
5. Какой метод позволяет уничтожить поток, вызванный методом setTimeout()?
6. Каково отличие объекта Window от других объектов?
7. Что такое поле статуса?
8. За что отвечает свойство status?
9. За что отвечает свойство defaultstatus?
10. Что такое фрейм?
11. Как можно обратиться к фрейму?

14.3. Задания

1. Используйте объект Navigator для определения типа браузера; (R15001)
2. Продемонстрируйте управление окнами. Используйте при этом методы window.alert(), window.confirm(), window.prompt(), window.open(), window.focus(), window.setTimeout(), window.clearTimeout() (R15002)
3. Создайте простую страницу, разделённую двумя фреймами. Передайте фокус во фреймы обеими способами, упомянутыми в теоретической части. (R15003)
4. Создайте страницу, которая спрашивает имя и выводит его. (R15004)