# Объектная модель документа

**http://anton-roman.narod.ru/glava14.html**


- Знакомство с объектной моделью документа
- Возможности объектной модели документа
- Доступ кэлементам страницы
- Модель событий
- Возможности JavaScript


Вам уже известны некоторые объекты JavaScript, например, такие, как объект **Date**. Теперь познакомимся с объектами, составляющими документ. Эта разновидность объектов браузера широко известна как *объектная модель документа* (Document Object Model - DOM).

В данной главе рассматривается:

- объектная модель документа;
- оперативное изменение текста с помощью объектной модели документа;
- модель событий.

## *Знакомство с объектной моделью документа*

Когда словосочетание «объектная модель документа» впервые стало употребляться наряду с термином «динамический HTML», в среде Web-разработчиков эта новая технология встретила активное неприятие. Как этим пользоваться? Зачем вообще это нужно? Когда-то эти вопросы были одними из самых злободневных.

В то время, когда мир завоевали Internet Explorer 4 и Netscape Navigator 4 (хотя объектная модель документа появилась раньше них), DOM была объявлена «следующим великим достижением HTML». Однако на этот раз в самом языке HTML ничего не изменилось. Объектная модель документа, не будучи новым тэгом (как, например, тэг изменения цвета <blink>) и не являясь атрибутом существующих тэгов HTML, дает возможность управлять всеми использующимися тэгами на Web-странице.

***Примечание*** *Примеры, которые будут рассматриваться в этой главе, в первую очередь предназначены для браузера Internet Explorer 5. Это не случайность, поскольку по сравнению с другими браузерами Internet Explorer имеет наиболее полную и легкодоступную DOM.*

Web Standards Organization разрабатывает спецификацию стандартов DOM. То есть через некоторое время разрыв между двумя конкурирующими браузерами начнет уменьшаться. Кстати, свежая информация о браузере Internet Explorer находится на сайте http://msdn.microsoft.cpm/workshop. а о браузере Netscape Navigator -на сайте http://developer.netscape.com.

## *Возможности объектной модели документа*

Объектная модель документа предоставляет Web-разработчикам поистине огромные возможности. Правда существует некоторое «но». Чтобы пользоваться объектной моделью документа, Web-разработчик должен уметь писать сценарии, в особенности на языке JavaScript.

Причина в том, что объектная модель документа является всего лишь отображением Web-страницы и сама по себе ничего не значит. Нечто похожее происходит с

переменными. Переменная может хранить в себе то или иное значение, но без сценария JavaScript ни на что не пригодна.

Последняя версия объектной модели документа содержит четыре ключевых нововведения:

- доступ ко всем элементам страницы;
- постоянное обновление страницы;
- полная и всеобъемлющая модель событий;
- оперативное изменение содержимого страницы.

И еще одна главная особенность - изменения на Web-странице могут происходить в любое время. До, во время и после загрузки страницы, когда пользователь нажимает клавишу, щелкает кнопкой мыши, перемещает курсор и т.д.

Давайте посмотрим, как реализуются эти возможности.

## *Доступ к элементам страницы*

В те времена, когда последними версиями браузеров были Internet Explorer 3.0 и Netscape Navigator 3.0, Web-разработчики могли обращаться к некоторым элементам Web-страницы только с помощью сценария. Этими избранными элементами были якорь <a>, апплет <applet> и форма <form>.

***Примечание*** *Оглядываясь на прошлое, можно только удивляться, какие замечательные страницы умудрялись создавать Web-разработчики несмотря на то, что поле их деятельности было так ограничено.*

Однако все имеет предел. Например, обратиться к заголовку вы не смогли бы при всем желании.

Сегодня с помощью DOM эта задача решается очень просто. Теперь доступ возможен к любому элементу Web-страницы. Документ представляет собой коллекцию всех элементов страницы, имеющую название **all**. Ее можно проиндексировать с помощью атрибутов name и **id**.

### Пример доступа к элементам страницы

Пример. Хотите проверить, содержится ли на вашей странице элемент <h1>? Возьмем простую страницу с одним элементом <h1>:

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
<!-- Маскируемся!
// Снимаем маскировку. -->
</script>
</head>
<body>
<hl id="headl">Первый заголовок</h1>
</body>
</html>
```

Вы можете определить наличие этого элемента с помощью коллекции **document.all**:

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">

<!-- Маскируемся!
function findheadl()
{
var detectElement;
detectElement = document.all("headl");
if (detectElement == headl)
{
alert("Элемент \"headl\" существует?");
}
}
// Снимаем маскировку. -->
</script>
</head>
<body onLoad="findheadl()">
<hl id="headl">Первый заголовок</h1>
</body>
</html>
```

После открытия этой страницы в браузере перед вами появляется окно предупредительных сообщений (см. рис. 14.1).

**Пример.** На основе предыдущего примера можно создать сценарий JavaScript, в котором будут перечислены все элементы страницы:

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
<!-- Маскируемся!
function findheadl()
{
var tag, tags;
tags = "Эта страница содержит следующие тэги:"
for(i =0; i < document.all.length; i++)
{
tag = document.all(i).tagName;
tags = tags + "\r" + tag;
}
alert(tags);
}
// Снимаем маскировку. -->
</script>
</head>
<body onLoad="findheadl()">
<h1>Первый заголовок</h1>
</body>
</html>
```

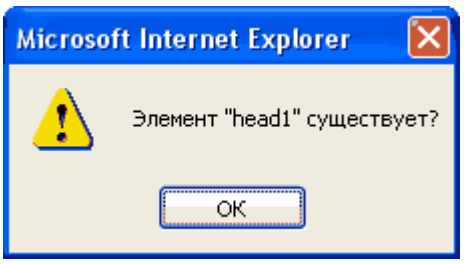


Рис. 14.1. Подтверждение наличия заголовка

Давайте поэтапно рассмотрим ход выполнения сценария.

```
var tag, tags;
```

В первой строчке сценария объявляются переменные.

```
tags = "Эта страница содержит следующие тэги:"
```

В следующей строке присваивается значение переменной **tags**. Это делается для того, чтобы впоследствии избежать неполадков в сценарии.

```
for(i =0; i < document.all.length; i++)
{
tag = document.all(i).tagName;
tags = tags + "\r" + tag;
}
```

Следующий раздел сценария - цикл **for**. В первой строке оператора цикла указывается, что нужно перебрать все тэги страницы с помощью свойства **length**, начиная с первого элемента (**i = 0**, поскольку элементы нумеруются с нуля, а не с единицы) и заканчивая последним (**i = document. all. length - 1**), причем значение **i** с каждым разом увеличивается на единицу (**i ++**).

При каждом выполнении цикла имя тэга восстанавливается с помощью метода **tagName** и сохраняется в переменной **tag**. Затем это значение передается переменной **tags** вместе с символом возврата каретки (**\r**), чтобы представить выводимую информацию в читаемом виде. Именно здесь и требуется начальное значение переменной tags. Если бы оно не было задано, то первой строчкой в окне сообщений выводилось бы undefined, что выглядит не слишком красиво.

```
alert(tags);
```

Список всех обнаруженных тэгов выводится в окне предупредительных сообщений (см. рис. 14.2).

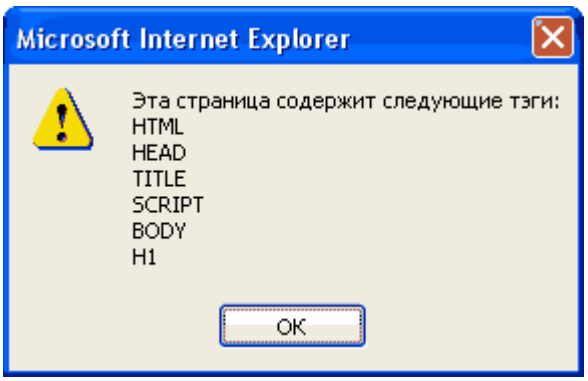


Рис. 14.2. Все тэги страницы перечислены в окне предупредительных сообщений

## Элементы страницы

Пример. Сколько бы тэгов ни содержалось на странице, с помощью коллекции all их все можно обнаружить. В данном примере некоторые тэги (<b> и <i>) размещаются внутри других тэгов (см. рис. 14.3).

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
<!-- Маскируемся!
function findheadl()
{
var tag, tags;
tags = "Эта страница содержит следующие тэги:"
for(i =0; i < document.all.length; i++)
{
tag = document.all(i).tagName;
tags = tags + "\r" + tag;
}
alert(tags);
}
// Снимаем маскировку. -->
</script>
</head>
<body onLoad="findheadl()">
<h1>Первый заголовок</h1>
<p>Немного текста</p>
<h2>Еще один заголовок</h2>
<p>Еще текст и немного <b>полужирного</b>текста</p>
<h3>Еще один заголовок</h3>
<p>Еще текст и немного <i>курсива</i></p>
</body>
</html>
```

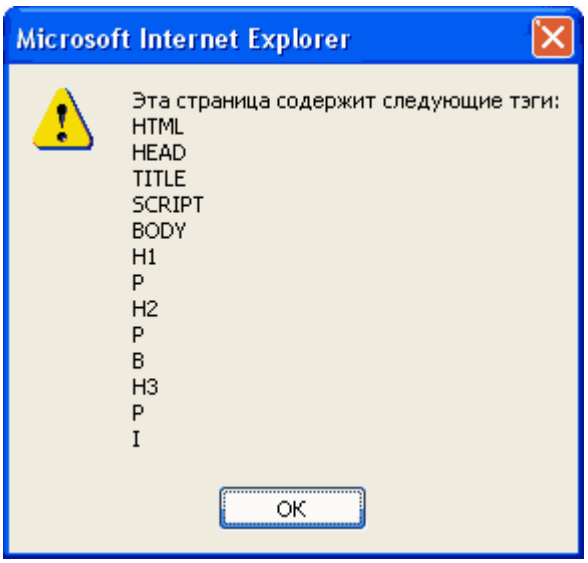


Рис. 14.3. Тэги, помещенные внутри других тэгов

### Использование srcElement

Пример. Способность отыскивать новые тэги позволяет сделать активным любой, даже самый незначительный элемент Web-страницы. В данном примере имя тэга выясняется с помощью **window.event.srcElement.tagName** и указывается в строке состояния. Объект **SrcElement** обращается к исходному элементу, то есть к элементу, генерируемому событием. Таким образом подобный элемент можно с легкостью обнаружить.

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
<!-- Маскируемся!
function tagInfo()
{
var tag;
tag = window.event.srcElement.tagName;
window.status = tag;
}
// Снимаем маскировку. -->
</script>
</head>
<body onMouseover="tagInfo()">
<h1>Первый заголовок</h1>
<p>Немного текста</p>
<h2>Еще один заголовок</h2>
<p>Еще текст и немного <b>полужирного</b> текста</p>
<h3>Еще один заголовок</h3>
<p>Еще текст и немного <i>курсива</i></p>
</body>
</html>
```

Сохранив эту страницу и открыв ее в браузере, вы увидите, что при наведении курсора на каждый из элементов имя тэга будет указываться в строке состояния

браузера (см. рис. 14.4). Это делается путем значения переменной **tag** объекту **window**, **status**.

***Примечание*** *Более подробная информация об объекте window, status изложена в главе 15.*

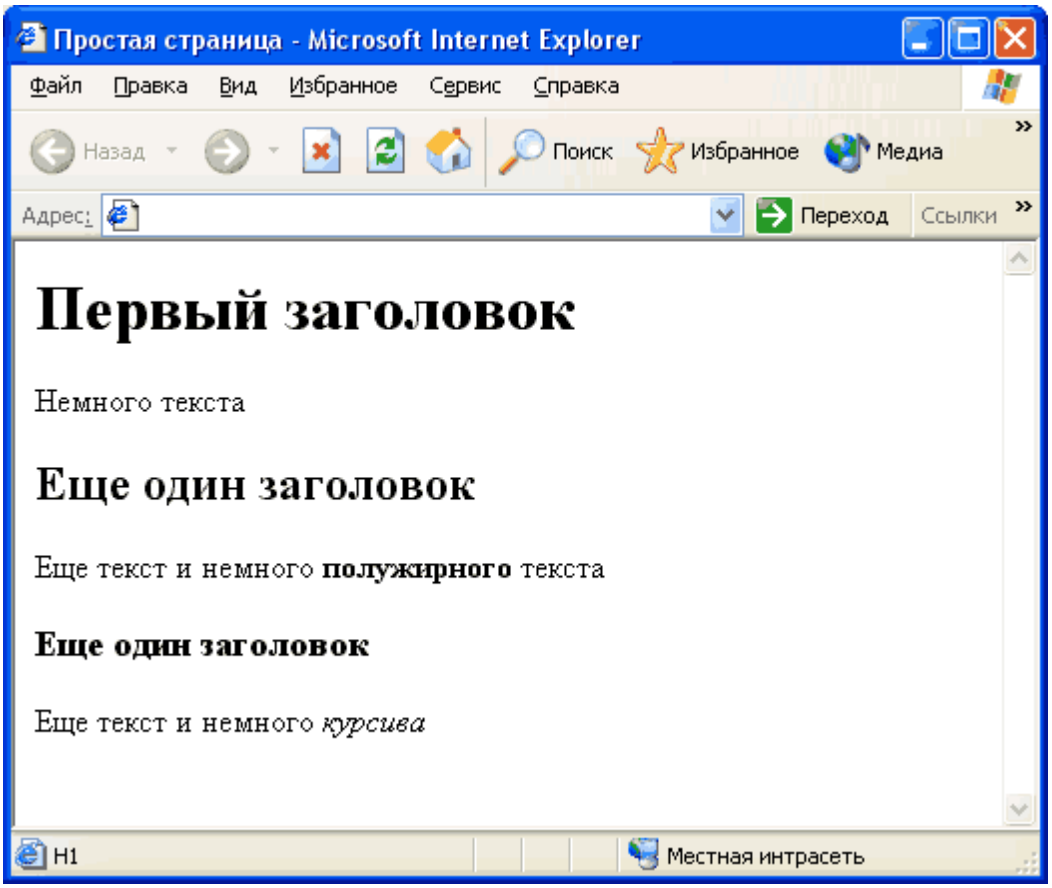


Рис. 14.4. Имя тэга в строке состояния браузера

### Добавление идентификаторов

**Пример.** Если немного изменить данный сценарий, то можно вывести на экран идентификатор требуемого тэга (см. рис. 14.5). Для этого используется свойство **window.event.srcElement.id**:

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
<!-- Маскируемся!
function tagInfoO
{
var tag;
tag = "Имя элемента: " + window.event.srcElement.tagName + " ID: " +
window.event.srcElement.id;
window.status = tag;
}
// Снимаем маскировку. -->
</script>
</head>
```

```
<body onMouseover="tagInfo()" id="bodyl">
<hl id="headl">Первый заголовок</h1>
<p id="paral">Немного текста</p>
<h2 id="head2">Еще один заголовок</h2>
<p id="para2">Еще текст и немного <b id="boldl">полужирного</b> текста</p>
<h3 id="head3">Еще один заголовок</h3>
<p id="para3">Еще текст и немного <i id="italicsl">Курсива</i></p>
</body>
</html>
```

### Родительские и дочерние элементы

Дальнейшее усовершенствование сценария - это возможность указать для определяемого элемента родительский элемент. *Родительским* (parent) называется элемент, в который вложены другие элементы, называемые *дочерними* (child).

**Пример.** Далее приводится пример сценария JavaScript, указывающего родительский элемент для любого элемента страницы (см. рис. 14.6).

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
<!-- Маскируемся!
function tagInfo()
{
var tag;
tag = "Имя элемента: " + window.event.srcElement.tagName + " ID: " +
window.event.srcElement.id + "Родительский элемент: " +
window, event. srcElement .parentElement. tagName;
window.status = tag;
}
// Снимаем маскировку. -->
</script>
</head>
<body onMouseover="tag!nfo()" id="bodyl">
<hl id="headl">Первый заголовок</h1>
<p id="paral">Немного текста</p>
<h2 id="head2">Еще один заголовок</h2>
<p id="para2">Еще текст и немного<b id="boldl">полужирного</b>
текста</p>
<h3 id="head3">Еще один заголовок</h3>
<p id="para3">Еще текст и немного <i id="italicsl">курсива</i></p>
</body>
</html>
```

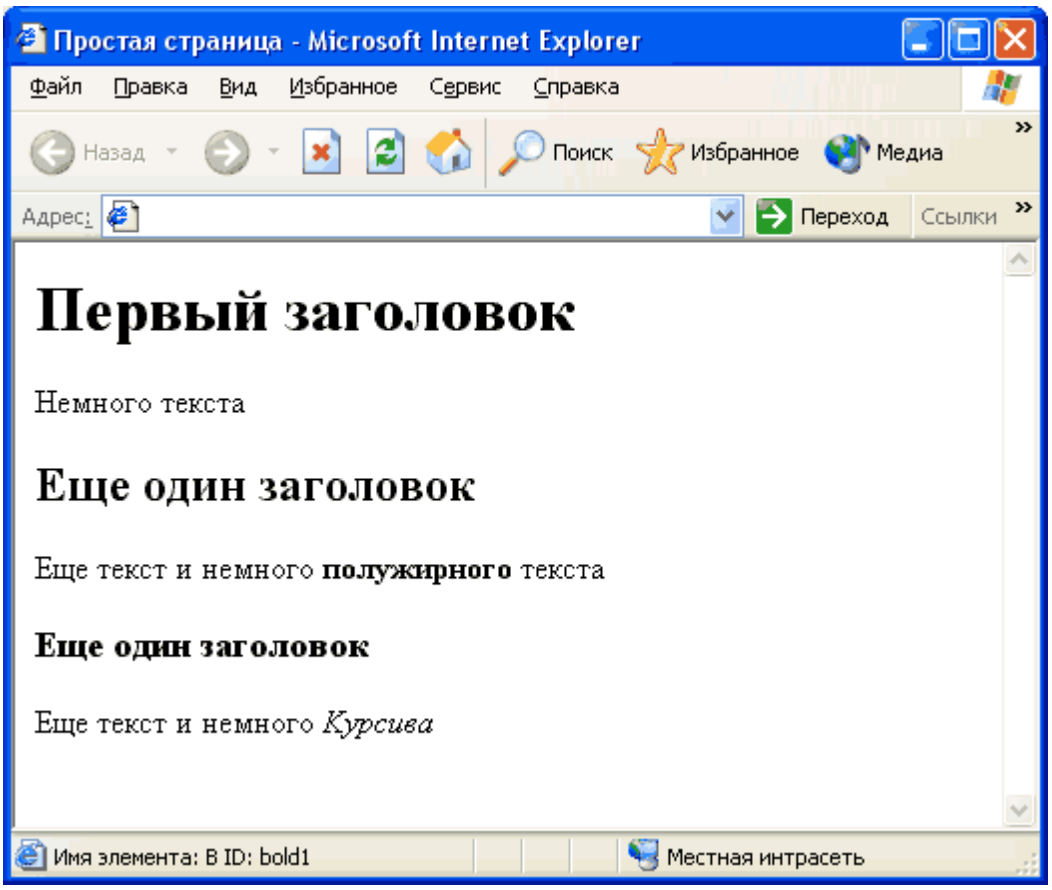


Рис. 14.5. Имя и идентификатор тэга в строке состояния браузера

**Пример.** В данном примере при наведении курсора автоматически подчеркиваются все полужирные надписи в том случае, если родительским элементом тэга <b> является элемент <p> (см. рис. 14.7). Это не касается тэга <b>, в качестве родительского элемента которого выступает тэг <h2> (см. рис. 14.8).

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
<!-- Маскируемся!
function checkBold()
{
var thistag, parentTag;
thistag = window.event.srcElement.tagName;
parentTag = window.event.srcElement.parentElement.tagName;
if (thistag == "B" && parentTag == "P")
{
window.event.srcElement.style.textDecoration = "underline";
}
}
function checkBoldOff()
{
var thistag, parentTag;
thistag = window.event.srcElement.tagName;
parentTag = window.event.srcElement.parentElement.tagName;
if (thistag == "B" && parentTag == "P")
{
```

```
window, event. srcElement. style . textDecoration = "none";
}
}
// Снимаем маскировку. -->
</script>
</head>
<body onMouseover="checkBold()" onMouseout="checkBoldOff ()" id="bodyl">
<hl id="headl">Первый заголовок</h1>
<p id="paral">Немного текста</p>
<h2 id="head2">Еще один заголовок</h2>
<p id="para2">Еще текст и немного <b id="boldl">полужирного</b> текста</p>
<h3 id= "head3">Еще один заголовок</h3>
<p id="para3">Еще текст и немного <i id="italicsl">Курсива</i></p>
<h2><b>Полужирный заголовок!</b></h2>
</body>
</html>
```

Сложность этого сценария заключается в необходимости второй функции, задача которой отменять подчеркивание, как только курсор будет убран с соответствующего тэга.

***Предупреждение*** *Имена тэгов возвращаются в виде символов верхнего регистра, поэтому при проверке не забывайте пользоваться верхним, а не нижним регистром. Использование символов нижнего регистра приведет к ошибке.*

### Пример разворачивания и сворачивания текста

Иногда целесообразно уменьшить количество текста, выводимого на экран. Вы можете убрать текстовую информацию, оставив только заголовки, при щелчке по которым откроется полный текст.

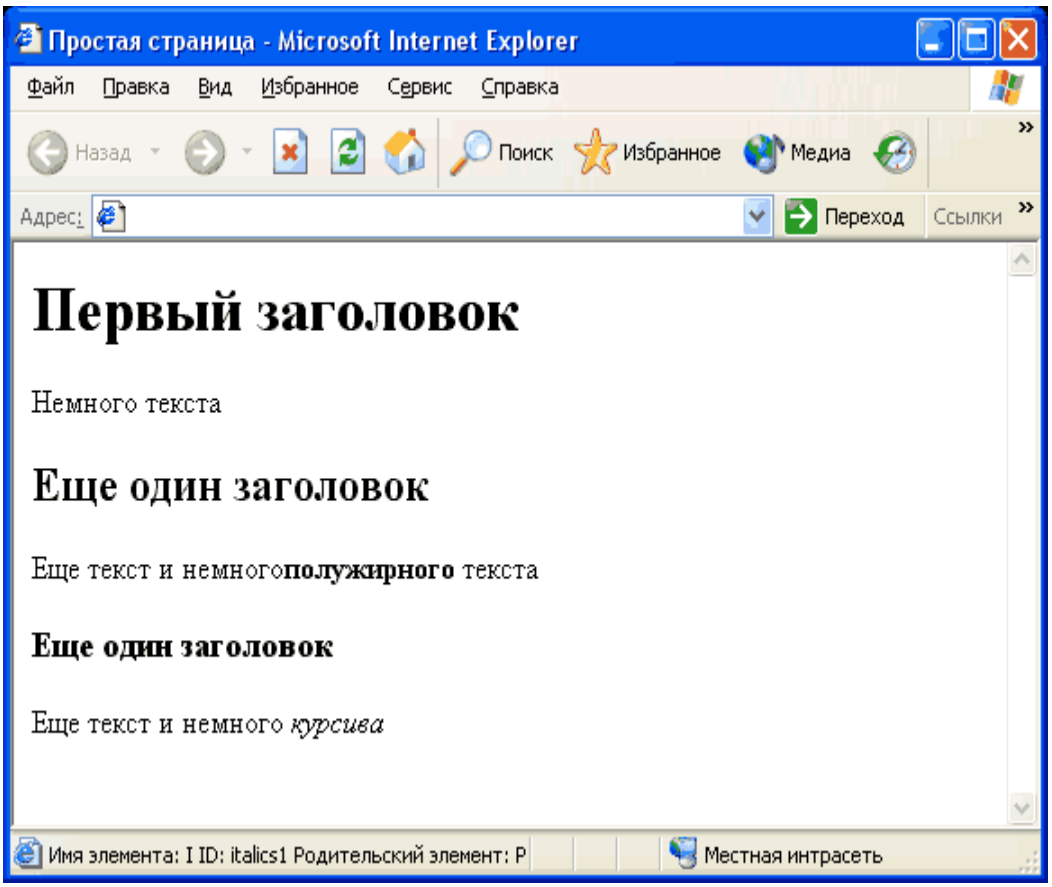


Рис. 14.6. Имя тэга, идентификатор тэга и родительский элемент в строке состояния браузера

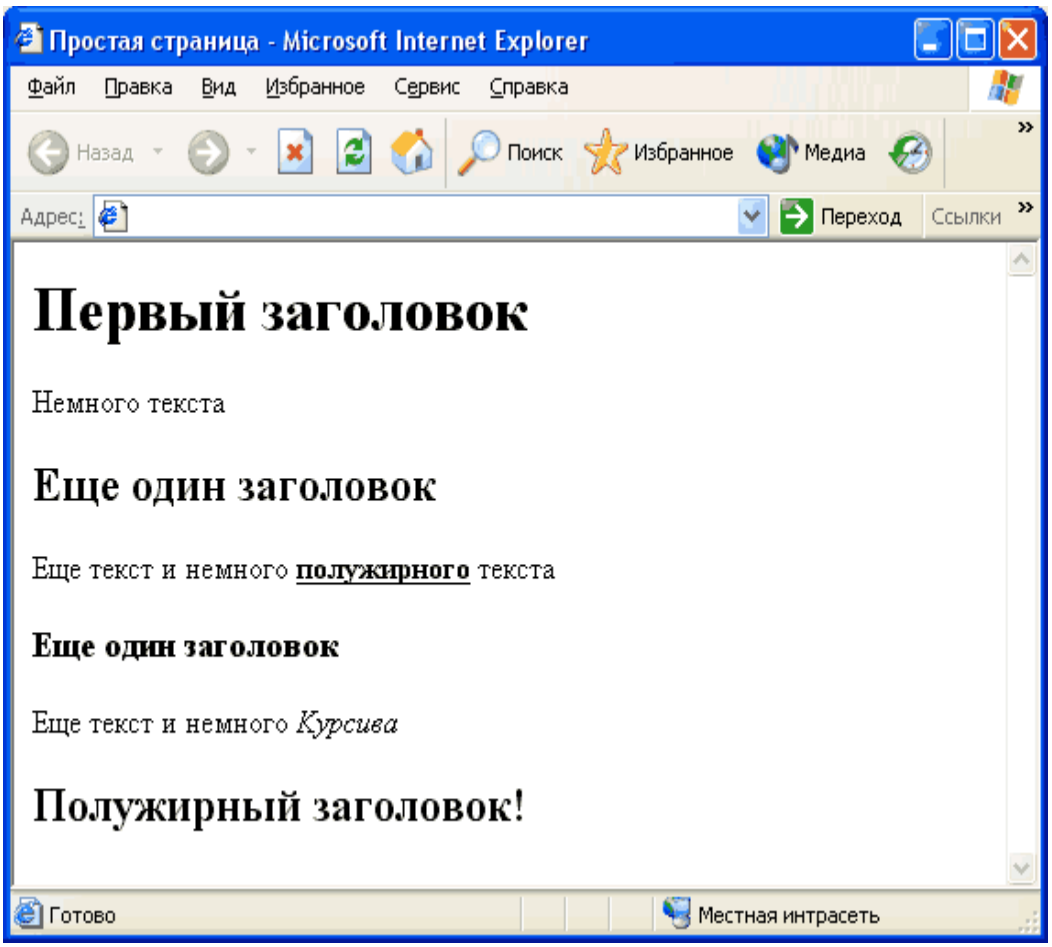


Рис. 14.7. Тэг <b>, родительским элементом которого является тэг <p>, подчеркивается

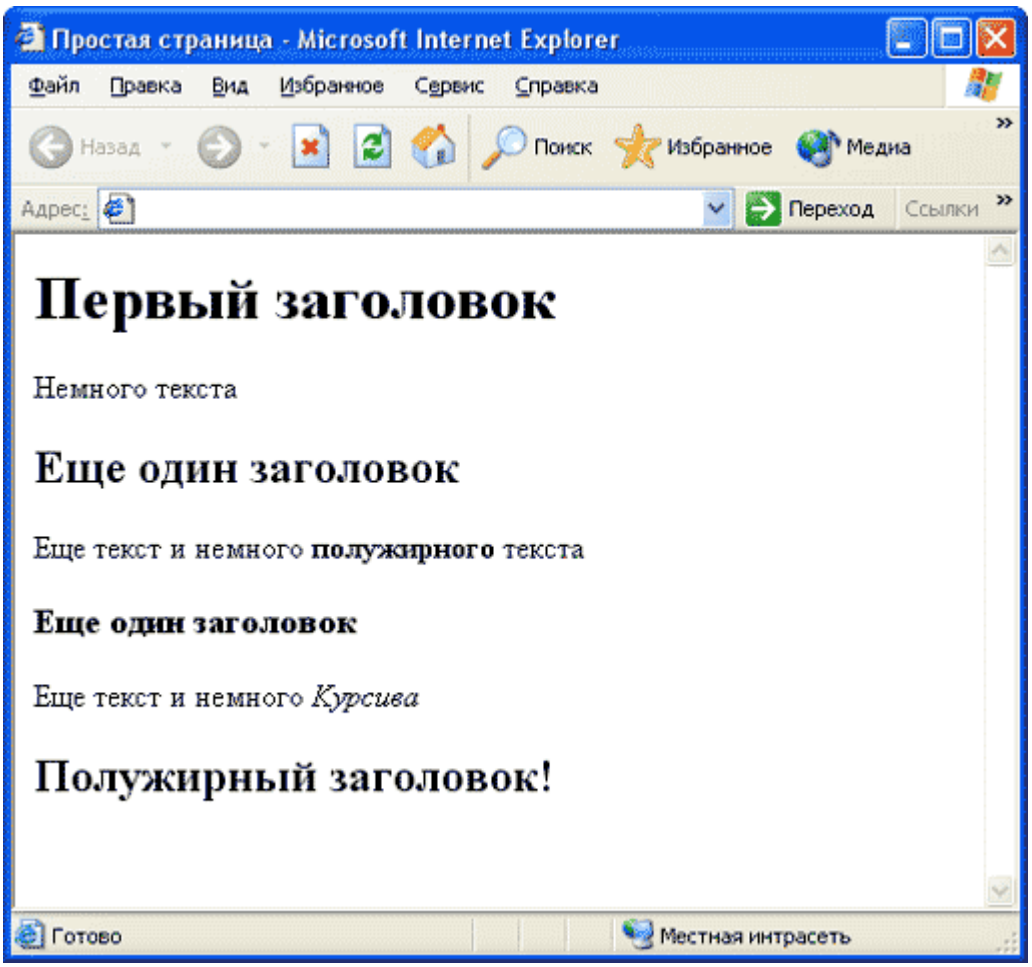


Рис. 14.8. Тэг <b>, родительским элементом которого не является тэг <p>, не подчеркивается

Вам потребуется атрибут CSS **display**. Он имеет два значения: попе и «пустота» (не символ пробела, а именно отсутствие какого-либо символа). Когда указывается последнее значение, элемент выводится на экран, а когда используется значение **none**, он становится невидимым.

**Пример.** Здесь приводится пример подобного сценария (см. рис. 14.9 и 14.10):

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
<!-- Маскируемся!
function clickEvent()
{
var thistag, parentTag;
thistag = window.event.srcElement.tagName;
parentTag = window, event. srcElement .parentElement. tagName;
if (thistag == "H1" && parentTag == "BODY")
{
if (document.all(window.event.srcElement.id + "p").style.display == "none")
{
document.all(window.event.srcElement.id + "p").style.display = "";
}
else
```

```
{
document.all(window.event.srcElement.id + "p").style.display = "none";
}
}
}
// Снимаем маскировку. -->
</script>
</head>
<body onClick="clickEvent()">
<hl id="headl">Первый заголовок</h1>
<p id="head1p" style="display:none"> Здесь много интересной
информации. Здесь много интересной информации. Здесь много
интересной информации. Здесь много интересной информации. Здесь
много интересной информации.</p>
<hl id="head2">Второй заголовок</h1>
<p id="head2p" style="display:none"> Здесь много интересной
информации. Здесь много интересной информации. Здесь много
интересной информации. Здесь много интересной информации. Здесь
много интересной информации.</p>
<hl id="head3">Третий заголовок</h1>
<p id="head3p" style="display:none"> Здесь много интересной
информации. Здесь много интересной информации. Здесь много
интересной информации. Здесь много интересной информации. Здесь
много интересной информации.</p>
</body>>
</html>
```

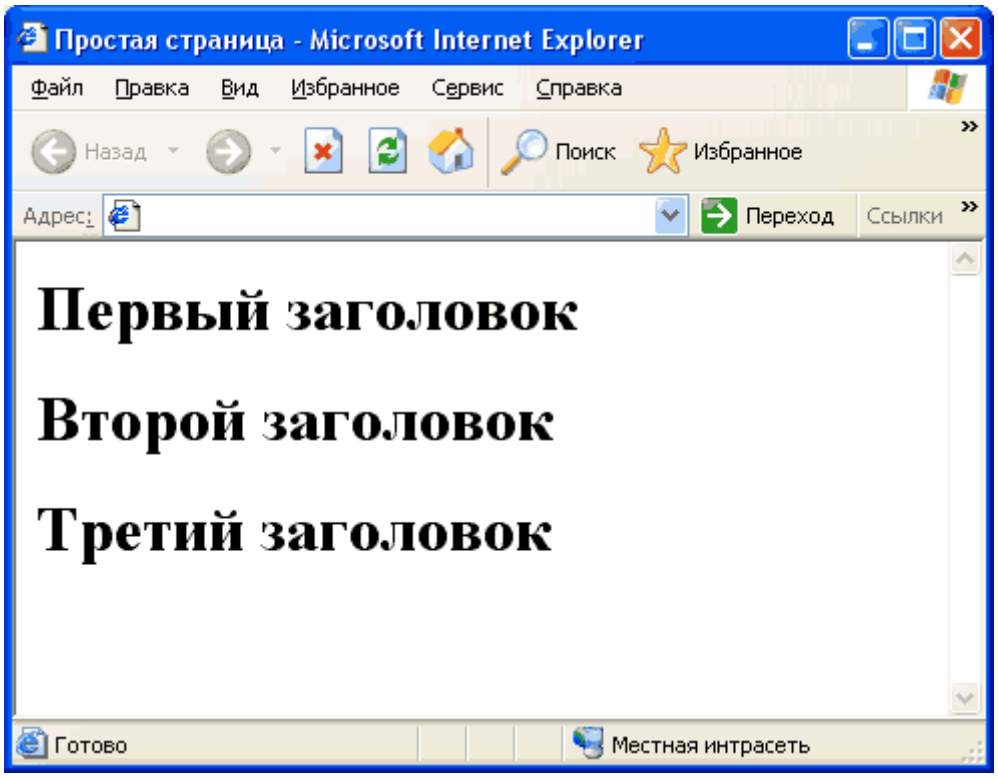


Рис. 14.9. Первоначальный вид страницы

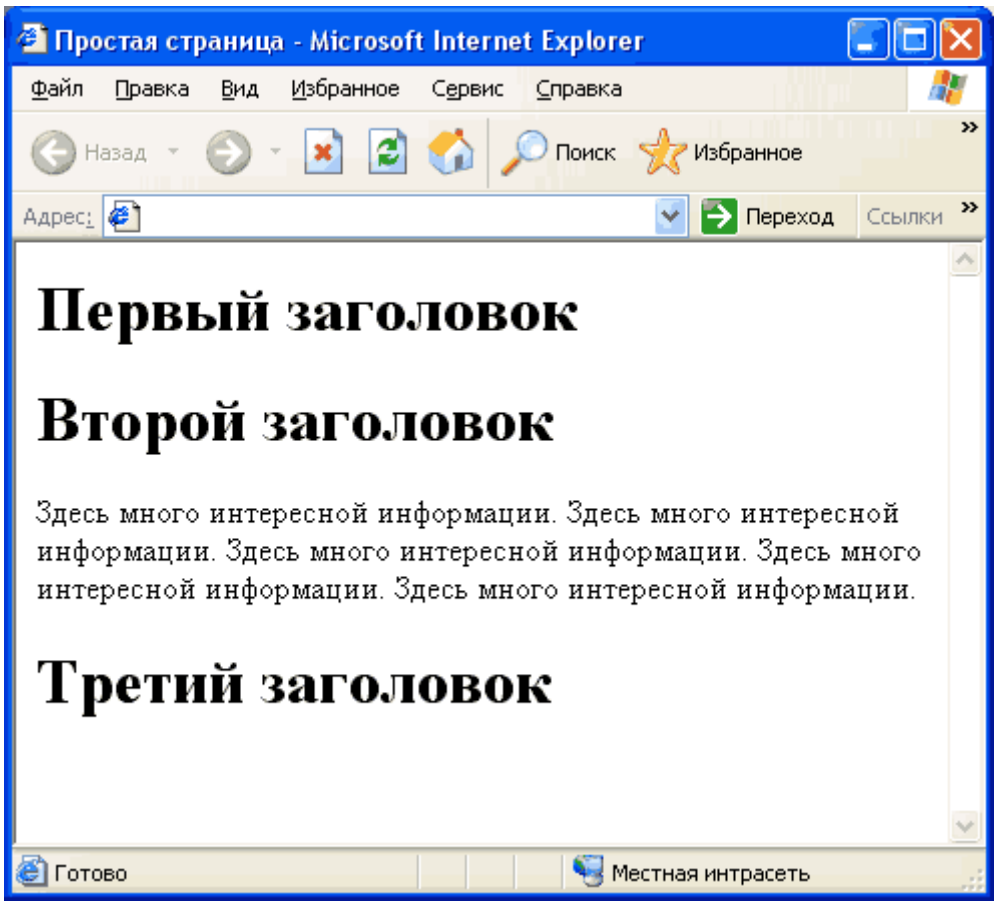


Рис. 14.10. Текст появляется после щелчка левой кнопкой мыши по заголовку

Этот сценарий может показаться сложным (здесь два оператора **if**, один внутри другого, и много фигурных скобок, за которыми трудно уследить), но весь смысл сценария сводится к трем операторам:

```
if(document.all(window.event.srcElement.id + "p").style.display == "none")
{
document.all(window.event.srcElement.id + "p").style.display = "";
}
else
{
document.all(window.event.srcElement.id+ "p").style.display = "none";
}
```

В первой строке содержится условие. Оно истинно, когда элементу с идентификатором **window.event.srcElement.id** + "**p**" соответствует значение по умолчанию none. Путем конкатенации идентификатора исходного элемента с буквой p получается идентификатор параграфа, лежащего под заголовком. (Вам придется тщательно выбирать идентификаторы при создании такого эффекта. Но, имея перед собой пример, вы можете тренироваться до тех пор, пока не обретете достаточно опыта, чтобы писать подобные сценарии с нуля.) Сначала текст страницы невидим и появляется после выполнения последующего оператора. Если условие ложно (текст не спрятан), то он пропадает с экрана. Таким образом, текст исчезает после второго щелчка по заголовку.

### Пример использования ячеек таблицы

Иногда следует предусмотреть обратную связь с пользователем. Рассмотрим тот случай, когда у вас на странице много таблиц. Ячейка таблицы будет менять цвет при наведении на нее курсора, и это может оказаться удобным при просмотре больших таблиц.

**Пример.** Здесь используется свойство **sourceIndex**, позволяющее выбрать отдельный элемент среди нескольких других - без необходимости присваивать идентификаторы всем элементам страницы. И снова сценарий JavaScript представляет собой всего лишь модификацию того, что вы делали раньше. Все эти сценарии показывают, как легко осуществляется доступ к объектной модели документа. Результат выполнения примера приводится на рис. 14.11.

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
<!-- Маскируемся!
function chgColor()
{
var thistag, parentTag;
thistag = window.event.srcElement.tagName;
if (thistag == "TD")
{
document.all(window.event.srcElement.sourceIndex).bgColor = "lemonchiffon";
}
}
function chgBack()
{
var thistag, parentTag;
thistag = window.event.srcElement.tagName;
if (thistag == "TD")
{
document.all(window.event.srcElement.sourceIndex).bgColor = "";
}
}
// Снимаем маскировку. -->
</script>
</head>
<body onMouseover ="chgColor() " onMouseout="chgBack() ">
<table border="l" width="28%">
<tr>
<td width="25%"> </td>
<td width="25%"> </td>
<td width="25%"> </td>
<td width="25%"> </td>
</tr>
<tr>
<td width="25%"> </td>
<td width="25%"> </td>
<td width="25%"> </td>
<td width="25%"> </td>
</tr>
<tr>
<td width="25%"> </td>
<td width="25%"> </td>
```

```
<td width="25%"> </td>
<td width="25%"> </td>
</tr>
</table>
</body>
</html>
```

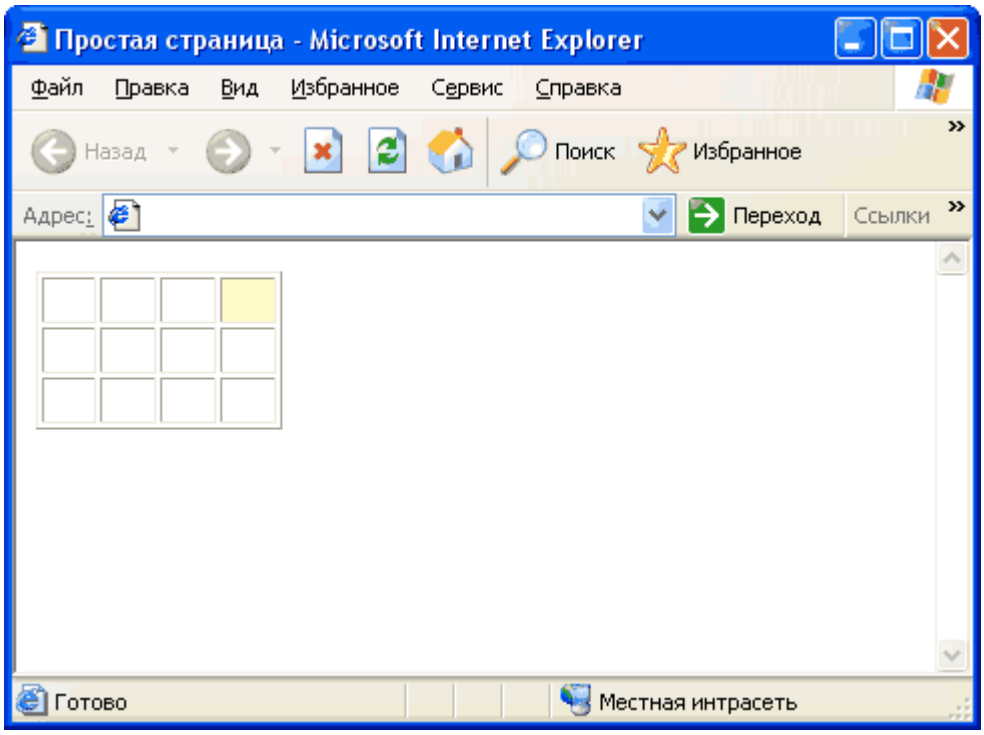


Рис. 14.11. Ячейки изменяют цвет при наведении курсора

***Примечание*** *Для осуществления этих эффектов нет необходимости соединяться с сервером. Достаточно использовать автономный режим.*

## Модель событий

Модель событий поддерживает весь диапазон событий, вызываемых пользователем с помощью мыши, клавиатуры, элементов интерфейса и др.

В следующем разделе описываются некоторые события из числа поддерживаемых браузером Internet Explorer 5.5.

### Событие onClick

Событие onClick происходит при щелчке по полю документа левой кнопкой мыши.

### Событие onContextmenu

Событие **onContextmenu** совершается, когда пользователь щелкает по полю документа правой кнопкой мыши, чтобы открыть контекстное меню. Это событие позволяет запустить сценарий до того, как меню возникнет на экране, или вовсе предотвратить появление контекстного меню (см. рис. 14.12). Последнее можно отменить, воспользовавшись свойством **event.return Value** и указав значение **false**. Тем самым отменяется событие, которое должно произойти по умолчанию. Например:

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
<!-- Маскируемся!
function menu()
{
alert("Контекстное меню отсутствует!");
event.returnValue = false;
}
// Снимаем маскировку. -->
</script>
</head>
<body onContextmenu="menu()">
</body>
</html>
```

***Примечание*** *Предыдущий пример удобно использовать для предотвращения копирования изображений с ваших Web-страниц. Это не слишком надежный способ, но многих пользователей он введет в заблуждение.*

### Событие onDblclick

Событие onDblclick происходит при двойном щелчке мышью по документу.

### Событие onHelp

Можно обеспечить посетителей вашей Web-страницы дополнительной справочной информацией. С помощью события **onHelp** следует заменить стандартные справочные файлы браузера Internet Explorer, появляющиеся на экране после нажатия клавиши **F1**. Если вы, как в предыдущем случае, хотите предотвратить появление на экране стандартного справочного файла, воспользуйтесь свойством **event.returnValue**.

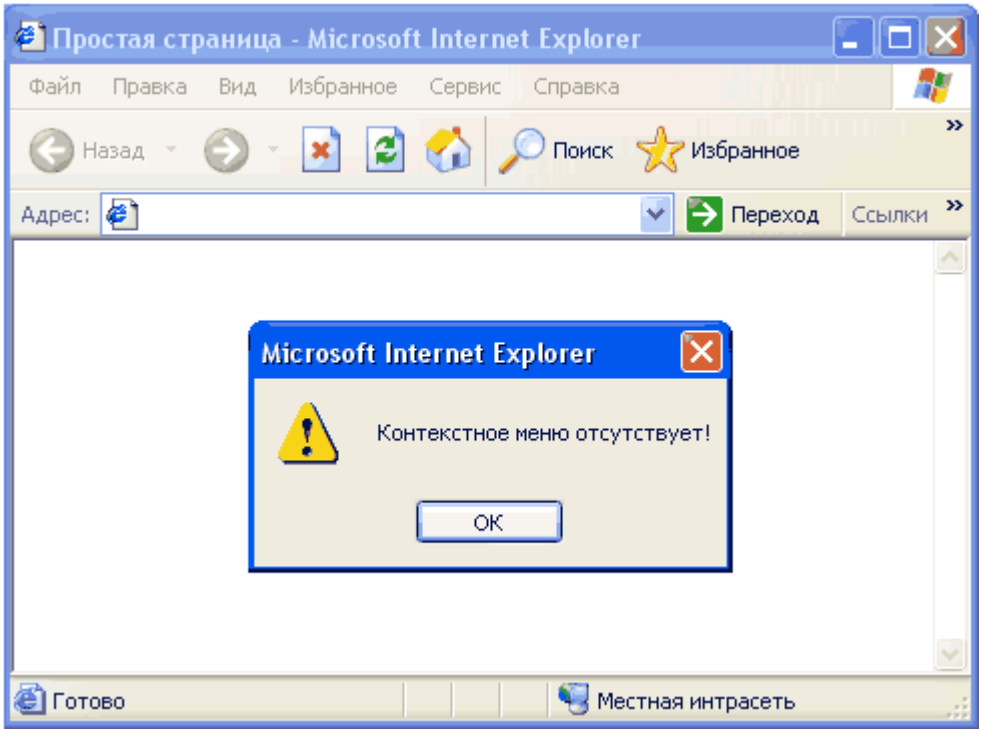

Рис. 14.12. Контекстное меню отключено

***Предупреждение*** *Было бы неблагоразумно поступать так с посетителями вашей Web-страницы, но в локальных сетях это может оказаться полезным, поскольку вы сможете дать пользователям краткие и оперативные советы.*

**Пример.** В этом примере, когда пользователь пытается открыть файл помощи, на экран выводится совершенно бесполезное окно сообщений (см. рис. 14.13).

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
<!-- Маскируемся!
function yourhelp()
{
alert("Нуууу, я даже и ке знаю, что посоветовать.");
event.returnValue = false;
}
// Снимаем маскировку. -->
</script>
</head>
<body onHelp="yourhelp()">
</body>
</html>
```

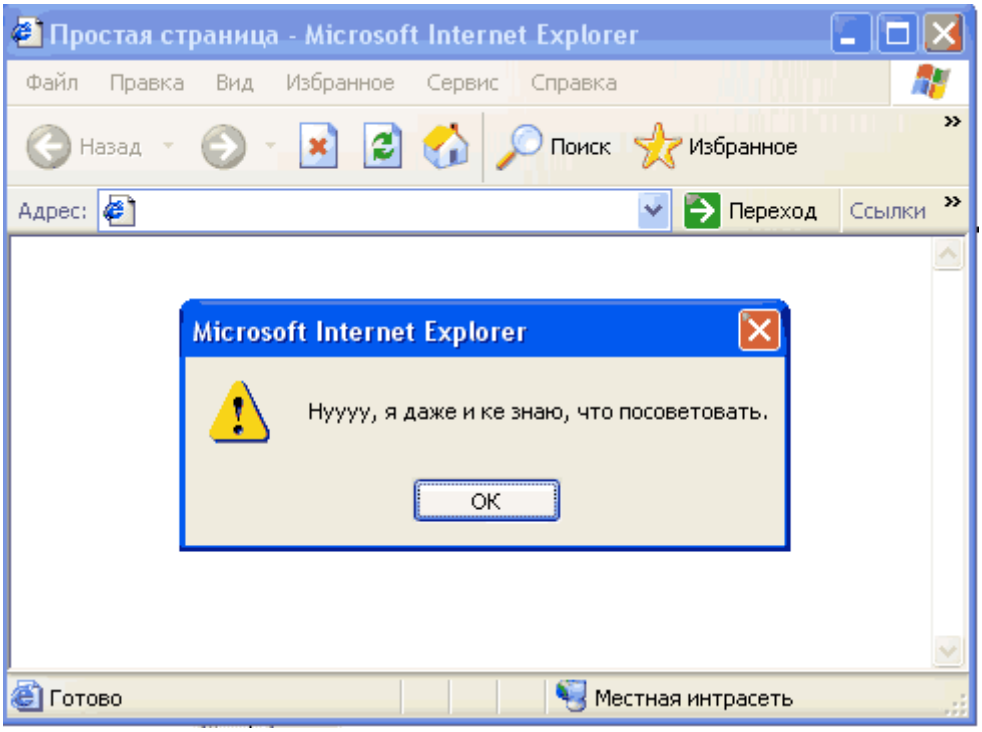


Рис. 14.13. Замена существующих справочных файлов вашим собственным файлом помощи

### Событие onKeydown

Это событие совершается, когда пользователь нажимает клавишу клавиатуры.

В браузере Internet Explorer 5.0 событие onKeydown происходит при нажатии следующих клавиш:

- редактирования - **Delete, Insert, Backspace;**
- функциональных - **F1-F12;**
- букв - **A-Z (в верхнем и нижнем регистре);**
- навигационных - **Home, End, РИСУНОК!!!!!!!!!Page Up, Page Down;**
- цифр - **0-9;**
- символов - **!, @, #, $, %, ^, &, *, (, ), _, -, +, =, <, >, [, ], {, }, ., /, ?, \, |,', ", ~ и ,;**
- системных - **Esc, пробел, Shift, Tab, Shift+Tab.**

Как и в предыдущих случаях, это событие можно отменить, присвоив атрибуту **event.returnValue** значение **false** для следующих клавиш и комбинаций клавиш:

- редактирования - **Backspace, Delete;**
- букв - **A-Z (в верхнем и нижнем регистре);**
- навигационных - **Page Up, Page Down, End, Home, РИСУНОК!!!!!!!!!!**
- цифр - **0-9;**
- символов - **!, @, #, $, %, ^, &, *, (, ), _, -, +, =, <, >, [, ], {,}, ., /, ?, \, |, ', ", ~ и ,;**
- системных - **пробел, Esc, Shift, Tab, Shift+Tab.**

**Пример.** Данное событие также вызывается нажатием клавиши. С помощью метода символ определяется и переводится из стандартного формата уникод (в котором символы представлены в цифровом виде) к действительному формату. С помощью **String.fromCharCode(event. keyCode)**. Затем он указывается в строке состояния. Результат представлен на рис. 14.14.

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
<!-- Маскируемся!
function press()
{
var char;
char = String.fromCharCode(event.keyCode);
window.status = "Вы нажали клавишу " + char;
}
// Снимаем маскировку. -->
</script>
</head>
<body onkeydown="press()">
</body>
</html>
```

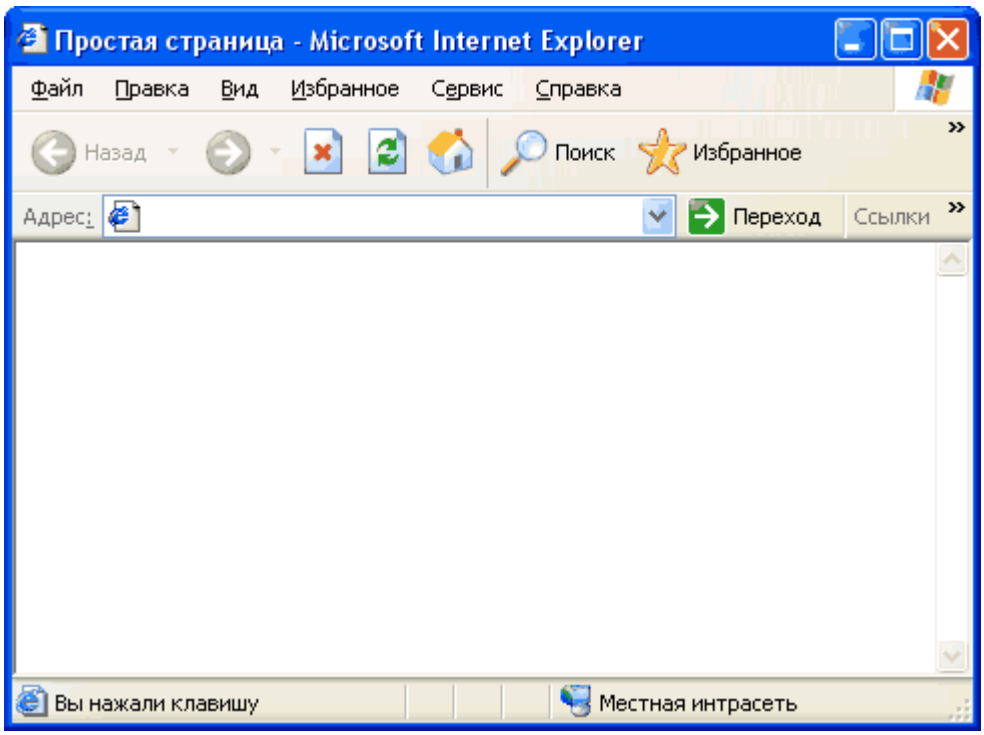


Рис. 14.14. Нажатая клавиша указывается в строке состояния

### Событие onKeypress

Событие **onKeypress** вызывается нажатием клавиши буквенно-цифрового ряда. Оно отличается от события **onKeydown**, поскольку происходит только тогда, когда клавиша будет полностью нажата (а не в процессе нажатия).

### Событие onKeyup

Событие **onKeyup** вызывается, когда пользователь отпускает клавишу. Оно происходит в тот момент, когда клавиша освобождается из положения **onKeydown**.

Событие onMousedown

Событие **onMousedown** вызывается, когда пользователь щелкает по документу любой кнопкой мыши.

**Пример.** В данном примере используются **event.x** и **event.y**, чтобы определить местоположение события на странице и вывести эту информацию в строке состояния. Результат показан на рис. 14.15.

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
<!-- Маскируемся!
function clicked()
{
window.status = "Вы щелкнули по точке с координатами: X = " + event.x + " Y = " +
event.y ;
}
// Снимаем маскировку. -->
```

```
</script>
</head>
<body onMousedown="clicked()">
</body>
</html>
```

### Событие onMousemove

Событие **onMousemove** происходит, когда пользователь перемещает курсор по документу.

**Пример.** Здесь постоянно отслеживается движение курсора и текст перемещается вслед за курсором (см. рис. 14.16).

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
<!-- Маскируемся!
function moved()
{
window.status = "Текущие координаты курсора: X = " + event.x + " Y = " + event.y;
follower.style.pixelTop = event.y
follower.style.pixelLeft = event.x
}
// Снимаем маскировку. -->
</script>
</head>
<body onMousemove="moved()">
<p id="follower" style="position:absolute">Пи-пи-пи!</p>
</body>
</html>
```

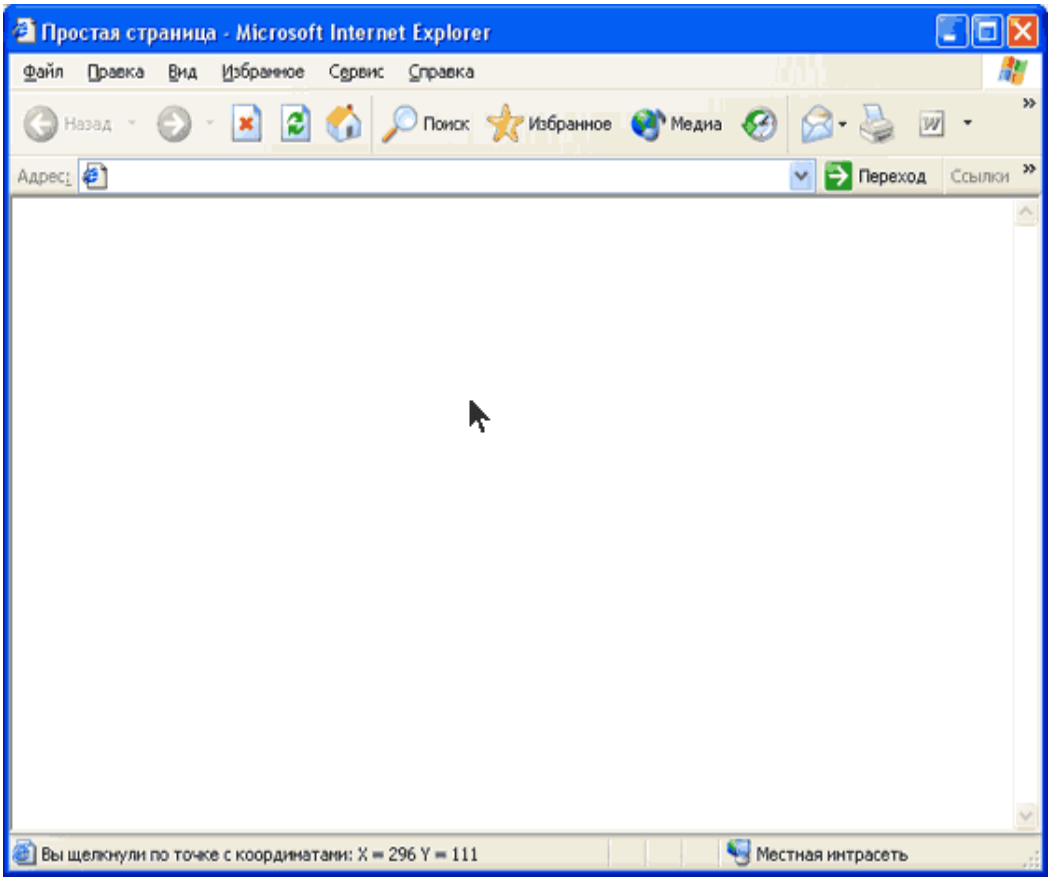


Рис. 14.15. В строке состояния указываются координаты точки щелчка

### Событие onMouseout

Событие **onMouseout** происходит, когда пользователь отводит курсор за границы документа, а также при перемещении курсора к другому элементу сценария.

**Пример.** В данном примере на экран выводится сообщение, когда курсор перемещается за пределы документа. То же самое сообщение появляется при быстром движении курсора по тексту. Причина в том, что курсор может двигаться гораздо быстрее, чем текст, и в результате он переходит к другому элементу, вызывая тем самым указанное событие.

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
<!-- Маскируемся!
function moved()
{
window.status = "Текущие координаты курсора: X = " + event.x + " Y = " + aevent.y;
follower.style.pixelTop = event.y
follower.style.pixelLeft = event.x
}
function outofbounds()
{
alert("Эй, вернись! Я за тобой не успеваю!");
}
// Снимаем маскировку. -->
```

```
</script>
</head>
<body onMousemove="moved()" onMouseout="outofbounds()">
<p id="follower" style="position:absolute">Пи-пи-пи!</p>
</body>
</html>
```

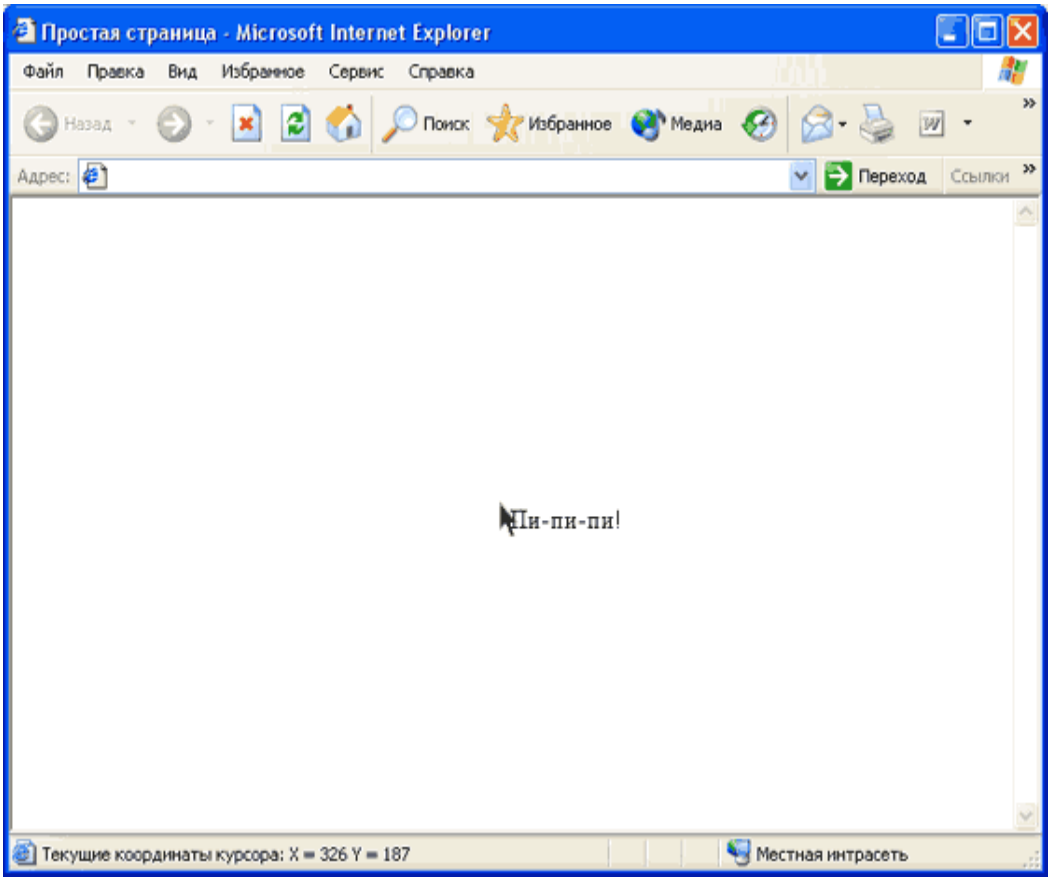


Рис. 14.16. Координаты курсора указываются в строке состояния

### Событие onMouseover

Событие onMouseover происходит, когда пользователь наводит курсор на документ.

### Событие onMouseup

Событие **onMouseup** совершается при отпускании кнопки мыши, когда курсор все еще находится в пределах документа. Оно может следовать за событием **onClick**.

### Событие onStop

Это последнее событие, которое будет рассматриваться в данном разделе. Событие **onStop** происходит, когда пользователь щелкает по кнопке **Stop** (Остановить) в браузере или переходит от одной Web-страницы к другой.

**Пример.**

```
<html>
<head>
<title>Простая страница</title>
```

```
<script language="JavaScript">
<!-- Маскируемся!
function stopped()
{
alert("Эй, не уходи!");
}
// Снимаем маскировку. -->
</script>
</head>
<body onStop="stopped()">
</body>
</html>
```

***Рекомендация*** *Возможно, такое использование события* ***onStop*** *не будет слишком раздражать пользователей. Однако не применяйте его на Web-странице, чтобы задержать пользователей на вашем сайте или выдать им целую кучу прощальных сообщений - такие уловки только действуют на нервы. Лучше использовать это событие на страницах локальных сетей или при разработке сетевой игры (чтобы предупреждать посетителей, что при выходе набранные ими очки сгорают).*

## Возможности JavaScript

Данная глава не претендует на полное описание объектной модели документа. Главная задача - с помощью примеров дать вам представление о том, что и как вы можете сделать. Примеры нетрудно изменить и пользоваться ими впоследствии на вашей Web-странице.

Как было сказано ранее, браузеры постоянно меняются, и лучшим источником сведений о них остаются сайты производителей, размещающих в сети огромное количество информации. На одном только сайте Microsoft Web Workshop содержатся сотни (если не тысячи) страниц подробных описаний производимого компанией браузера. Неизменной остается только возможность использовать имеющиеся в вашем распоряжении рабочие средства браузеров с помощью языка JavaScript. Изучите JavaScript, и вам будет легче освоиться в мире Internet. Тем более что сейчас указанные компании, и Microsoft, и Netscape, предлагают множество примеров сценариев JavaScript. Эти примеры вам безусловно пригодятся, поскольку сейчас вы уже умеете не только писать сценарии, но и читать их.