17. Программирование гипертекстовых переходов

Цель: Рассмотреть вопросы построения динамических списков, гипертекстовых ссылок и программирования гипертекстовых переходов в зависимости от условий просмотра HTML-страниц и действий пользователей.

17.1. Теоретические сведения

Гипертекстовая ссылка относится к классу объектов URL.

Область применения URL на HTML-страницах:

- графика (атрибут SRC контейнера IMG);
- формы (атрибут ACTION контейнера FORM);
- ссылки (атрибут HREF контейнера A);
- «чувствительные» картинки (атрибут HREF контейнера AREA).

17.1.1. Объект URL

Объект класса URL обладает свойствами, которые определены схемой URL. В качестве примера рассмотрим ссылку на применение атрибута SRC в контейнере IMG:

http://intuit.ru/help/index.html

Значения свойств:

- href: http: //intuit.ru/help/index.html
- protocol: http:
- hostname: intuit. ru
- host: intuit.ru:80
- port: 80
- pathname: help/index.html
- search:
- hash:

Обращение к свойству объекта класса URL выглядит как:

имя_объект_класса_URl.свойство

Например, так:

document.links[0].href document.location.host document.links[2].hash

Свойства объекта URL дают программисту возможность менять только часть URL — объекта.

Здесь следует заметить, что чаще всего все-таки меняют весь URL. Это связано с тем, что такое действие более понятно с точки зрения HTML-разметки. Ведь у контейнера A нет атрибута PROTOCOL, но зато есть атрибут HREF.

17.1.2. Массивы встроенных гипертекстовых ссылок

К встроенным гипертекстовым ссылкам относятся собственно ссылки (<A HREF=...></A>) и ссылки «чувствительных» графических картинок. Они составляют встроенный массив гипертекстовых ссылок документа

(document.links[]).

К сожалению, обратиться по имени к гипертекстовой ссылке нельзя. Точнее такое обращение не рекомендуется в силу различий между браузерами. Поэтому обращаться к ним можно только как к массиву встроенных ссылок. В качестве примера напечатаем гипертекстовые ссылки некоторого документа:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
<!--
<script language="javascript"
    type="text/javascript"
    src="JScript21.js">
</script>
<link type="text/css" rel="stylesheet"
    href="StyleSheet.css">
-->
</head>
<body>
<a href="https://mail.google.com"></a>
<a href="https://www.google.by"></a>
```

```
<a href="https://www.tut.by"></a>
<script language="javascript"
    type="text/javascript">
    document.write("На страницы имеются ссылки на:" + "<br />");
    for (i = 0; i < document.links.length; i++)
        document.write(document.links[i].href + "<br />");
</script>
</body>
</html>
```

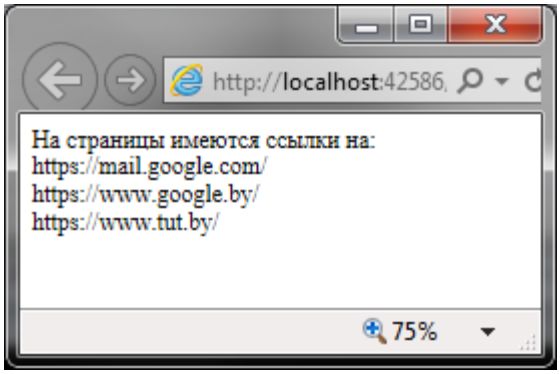

Рис. 17.1.

Вставим в документ контейнер MAP:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
<!--
<script language="javascript"
    type="text/javascript"
    src="JScript21.js">
</script>
<link type="text/css" rel="stylesheet"
    href="StyleSheet.css">
-->
</head>
<body>
<a href="https://mail.google.com"></a>
<a href="https://www.google.by"></a>
<a href="https://www.tut.by"></a>
<map name="test">
    <area shape="rect" coords="0, 0, 0, 0"
        href="javascript: window.alert('Area_Link_1'); void(0); " />
    <area shape="rect" coords="0, 0, 0, 0"
        href="javascript: window.alert('Area_Link_2'); void(0); " />
</map>
<script language="javascript"
    type="text/javascript">
    document.write("На страницы имеются ссылки на:" + "<br />");
    for (i = 0; i < document.links.length; i++)
        document.write("links[" + i + "]: " + document.links[i].href + "<br
/>");
</script>
</body>
</html>
```

И снова напечатаем массив ссылок:

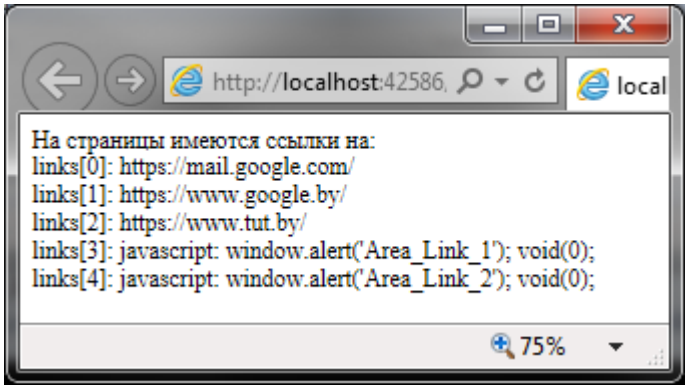

Рис. 17.2.

Две новые ссылки — это ссылки из контейнера MAP, который не отображается, но ссылки из него попадают в массив встроенных ссылок. При этом, как в нашем случае, они могут попасть между обычными гипертекстовыми ссылками, если контейнер MAP расположить внутри текста документа. На данной странице он помещен перед контейнером SCRIPT, в котором мы печатаем массив встроенных ссылок.

17.1.3. Изменение части URL

17.1.3.1. Адрес URL

Одним из основных методов SEO является работа с URL адресом. Хорошая структура помогает процессу индексации веб сайта, но основная масса сайтов использует неправильные и неоптимальные адреса URL. Здесь мы рассмотрим различные элементы URL и как можно создать оптимальную структуру, которая поможет поднять ранг сайта.

17.1.3.1.1. Элементы URL

Начнем с обзора элементов, которые составляют адрес URL. На первый взгляд URL может показаться простым, но в действительности он состоит из нескольких важных блоков информации.

Рассмотрим пример:

Рис. 17.3.

1. Протокол: в данном случае используется HTTP - самый популярный протокол в WWW. Может быть HTTPS, FTP и telnet.
2. Имя домена веб сайта.
3. Подкаталог файла.
4. Имя файла запрашиваемого ресурса.

Есть три части URL адреса, на которые мы можем влиять: имя домена, подкаталог и имя файла. Далее мы рассмотрим каждую из данных частей и способы их оптимизации.

17.1.3.1.2. Имя домена

Имя домена является адресом сайта в WWW. Это наиболее заметная часть адреса URL, ее часто можно встретить в почтовых сообщениях, на визитках, рекламных буклетах и так далее.

Хорошее доменное имя легко запоминается. Его лучше делать как можно более коротким, предпочтительно из одного или двух слов. Такой подход поможет вашим посетителям запоминать и выделять адрес вашего сайта среди прочих.

Не стоит делать домен, точно соответствующий какому-либо запросу. Такой подход может сыграть совсем не так, как хочется владельцу сайта.

17.1.3.1.3. Расширение домена

Влияет ли доменное расширение на SEO? Нет. Расширение доменного имени не позволяет определить качество сайта. Существуют другие факторы, которые влияют на качество, такие как содержание и ссылки.

Специфические для стран расширения доменов (белорусское .by, российское .ru) имеют некоторые преимущества. Они помогают получить существенный поток локального для страны трафика.

17.1.3.1.4. Глубина подкаталогов

Глубина - количество подкаталогов, в которых располагается домен. Обычно также отражается в "хлебных крошках". "Хлебные крошки" для страницы на коммерческом сайте могут выглядеть так:

Рис. 17.4.

На примере приведена страница для каталога мужских рубашек с двумя уровнями вложенности. В данном случае правильная структура URL для страницы продукта будет выглядеть так:

Рис. 17.5.

Использование ключевого слова (в нашем примере ‘tshirts’) существенно лучше, чем использование обычного имени или числа (например, ‘cat433’). Такое имя не только облегчает навигацию для пользователя, но и помогает поисковому сервису правильно индексировать ваш веб сайт.

Насколько глубокой может быть структура подкаталогов? Плоская структура сайтов работает лучше. Глубокие страницы реже обрабатываются поисковыми роботами. Обычно рекомендации сводятся к тому, что глубина структуры каталогов не должна превышать 3-х уровней.

17.1.3.1.5. Ширина подкаталогов

Ширина соответствует количеству страниц в категории. Выше приведенный пример может содержать каталог рубашек для женщин и детей, кроме мужских сорочек. В таком случае ширина категории будет равна 3 (мужчины, женщины и дети).

Ширина каталога не влияет на ранжирование. Можно использовать столько страниц сколько хочется. Так как страницы являются уникальными, то широкая категория не вызывает проблем.

17.1.3.1.6. Подкаталоги или поддомены

Что лучше использовать поддомены или подкаталоги? Вот разница между поддоменами и подкаталогами:

Рис. 17.6.

Поддомены часто рассматриваются Google как отдельные домены, что подразумевает двойную работу для SEO.

Но если у вас несколько не связанных друг с другом продуктов, использование поддоменов может оказаться хорошей идеей. Google использует такой подход для карт (maps.google.com), новостей (news.google.com) и так далее. Подобный метод может использоваться и на локальных сайтах (например, boston.website.com и lasvegas.website.com).

17.1.3.1.7. Имена страниц

Адрес страницы является самой доступной для модификаций частью адреса URL. Не всегда возможно изменить имя домена, его расширение или структуру каталогов, но изменение имени страницы выполняется существенно легче.

Когда приходит время выбирать имя для страницы, следует задуматься об удобстве использования и ключевых словах. Рассмотрим задачу на примере.

На рисунке приведены два адреса URL. Какой следует выбрать?

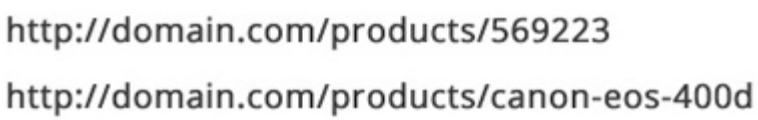

Рис. 17.7.

Вероятно, что вы выберите второй вариант. Можно четко распознать, что это адрес страницы товара - камеры Canon EOS 400d. При взгляде на первый адрес нельзя четко определиться с содержанием страницы.

Поисковые системы сталкиваются с той же проблемой. Если они рассматривают только адрес URL, они не могут определиться, о чем данная страница. Поэтому, в данном случае хорошей идеей будет

использование ключевого слова. Данное слово отображается жирным шрифтом в результатах поисковых запросов и облегчает пользователю идентификацию подходящих страниц.

Обратите внимание на использование тире в адресе URL. Рекомендуется разделять ключевые слова таким образом: http://domain.com/white-tennis-shoes, так как это существенно лучше, чем http://domain.com/whitetennisshoes. Применения других разделителей, таких как подчеркивание (_) и плюс(+), следует избегать.

17.1.3.1.8. Статический адрес URL или динамический

Динамический адрес URL легко идентифицировать по специальным символам, таким как знак вопроса и знак амперсанда &. Данные символы служат для идентификации параметров (например, язык страницы или ID пользователя). Но они означают, что одинаковое содержание может иметь различные адреса URL.

Некоторые веб-мастеры преобразуют динамические адреса URL так, чтобы они выглядели как статические. В таких работах нет нужды, так как содержание страницы разбирается поисковым роботом, и Google интерпретирует разные параметры. Алгоритм может определять, какие параметры не следует принимать во внимание.

17.1.3.2. window.location

Получает/устанавливает URL окна и его компоненты

Значением этого свойства является объект типа Location. Метод toString этого объекта возвращает URL, а различные свойства позволяют получить/установить отдельные компоненты адреса.

Для некоторых строковых операций необходимо явно преобразовать Location к строке:

```
window.location.toString().charAt(17)
```

Свойства объекта Location

Колонка "Пример" содержит их значения для URL http://www.google.com:80/search?q=javascript#test:

Свойство	Описание	Пример
hash	часть URL, которая идет после символа решетки '#', включая символ '#'	#test
host	хост и порт	www.google.com:80
href	весь URL	http://www.google.com:80/search?q=javascript#test
hostname	хост (без порта)	www.google.com
pathname	строка пути (относительно хоста)	/search
port	номер порта	80
protocol	Протокол	http:
search	часть адреса после символа ?, включая символ ?	?q=javascript

Таблица 17.1.

В Firefox есть баг: если hash-компонент адреса содержит закодированные символы, свойство hash возвращает раскодированный компонент. Например, вместо %20 будет пробел и т.п. Другие браузеры ведут себя корректно и не раскодируют hash.

Методы объекта Location:

assign(url)	загрузить документ по данному url
reload([forceget])	перезагрузить документ по текущему URL. Аргумент forceget - булево значение, если оно true, то документ перезагружается всегда с сервера, если false или не указано, то браузер может взять страницу из своего кэша.
replace(url)	заменить текущий документ на документ по указанному url. Разница, по сравнению с assign() заключается в том, что после использования replace() страница не записывается в истории посещений. В частности, это значит, что посетитель не сможет использовать для возврата кнопку браузера "Назад".
toString()	возвращает строковое представление URL для объекта Location

Таблица 17.1.

При изменении любых свойств window.location, кроме hash, документ будет перезагружен, как если бы для модифицированного url был вызван метод

```
window.location.assign().
```

Пример: перейти на документ по адресу

```
function goJs() {
   window.location = "http://javascript.ru"
}
```

Пример: вывести свойства текущего location

```
function showLoc() {
    var x = window.location;
    var t = ['Property - Typeof - Value',
        'window.location - ' + (typeof x) + ' - ' + x ];
    for (var prop in x){
        t.push(prop + ' - ' + (typeof x[prop]) + ' - ' +  (x[prop] || 'n/a'));
    }
    alert(t.join('\n'));
}
```

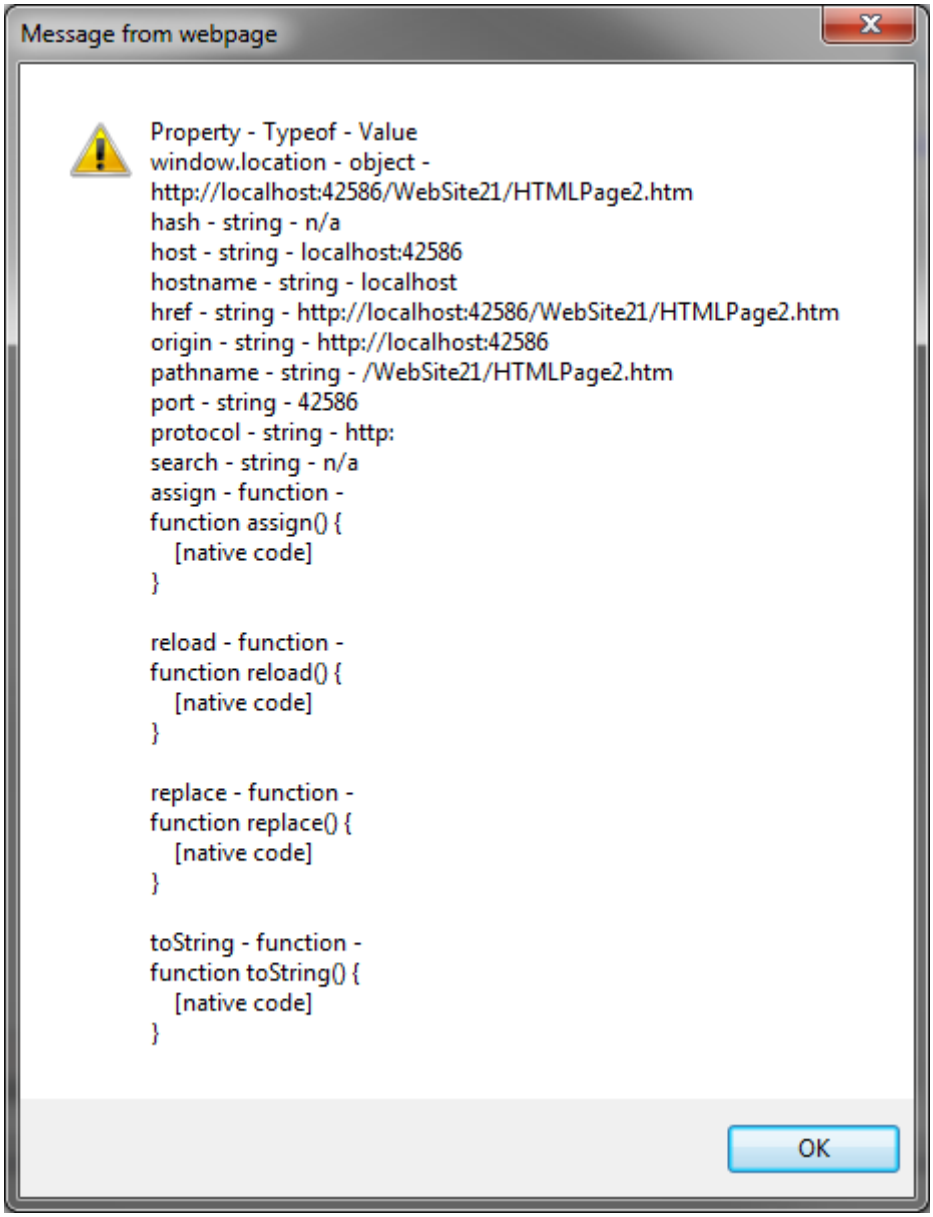

Рис. 17.8.

Пример: сделать запрос с новыми параметрами

```
function sendData(My Data)
{
    window.location.search = My Data;
}
```

При вызове sendData('My Data') на сервер отправится строка с параметрами "?My%20Data".

Гипертекстовая ссылка — это объект класса URL. У этого объекта можно изменять и другие свойства. Проиллюстрируем эту возможность при частичном изменении ссылки. Распечатаем сначала свойство, которое не зависит от протокола (в нашем случае от javascript)

```
document.all.next.pathname: href: --> http://intuit.ru/help/index.html
pathname: --> help/index.html
```

Изменим теперь pathname:

```
document.all.next.pathname"test";
document.write(window.document.all.next.pathname);
href: --> http://intuit.ru:80/test
pathname--> test
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<script language="javascript"
    type="text/javascript">
    document.write("Свойства URL</br>");
    document.write("href: <b>" + window.document.location.href + "</b></br>");
    document.write("hash: <b>" + window.document.location.hash + "</b></br>");
    document.write("host: <b>" + window.document.location.host + "</b></br>");
        document.write("pathname:  <b>"  +  window.document.location.pathname  +
"</b></br>");
    document.write("port: <b>" + window.document.location.port + "</b></br>");
        document.write("protocol:  <b>"  +  window.document.location.protocol  +
"</b></br>");
          document.write("search:  <b>"  +  window.document.location.search  +
"</b></br>");
    document.write("</br></br>");
    document.write("Изменение свойства pathname отражают окна");
    var pathname = "localhost:8080/abc/index.html";
    var lastItem = pathname.substring(pathname.lastIndexOf('/') + 1);
    alert('Last item: ' + lastItem);
    var startPart = pathname.substring(0, pathname.lastIndexOf('/') + 1);
    alert('Start part: ' + startPart);
    var newUrl = startPart + 'test.html';
    alert('New url: ' + newUrl);
</script>
<body>
</body>
</html>
```

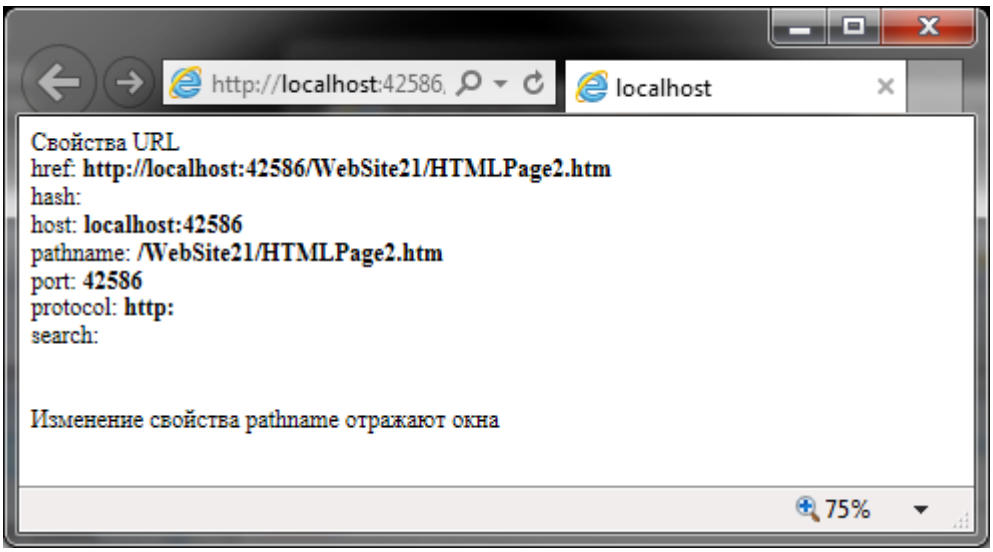

Рис. 17.9.

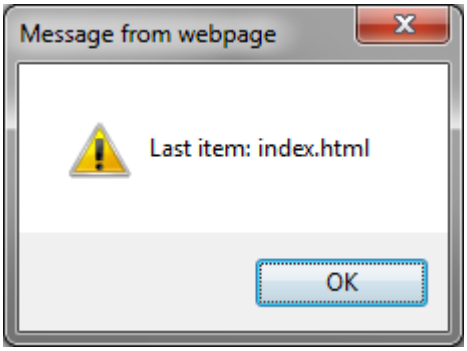

Рис. 17.10.

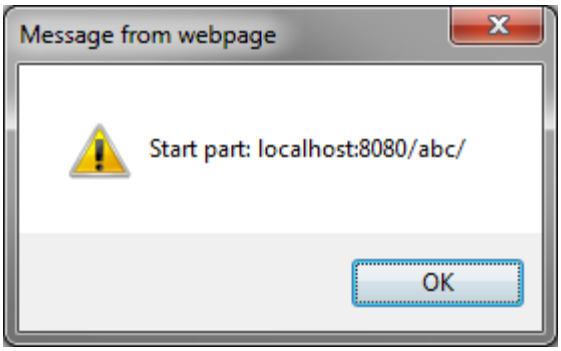

Рис. 17.11.

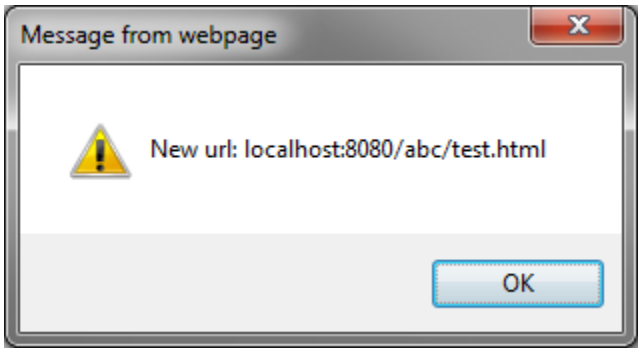

Рис. 17.12.

Обратите внимание, что Internet Explorer самостоятельно добавил в ссылку номер порта. По этой причине использовать свойства, отличные от href, в ссылках, где используется схема javascript, не рекомендуется.

17.1.4. Обработка событий Mouseover и Mouseout

Событие onMouseover срабатывает, когда курсор мыши наводится на элемент, к которому добавлен атрибут onMouseover. Это один из самых популярных атрибутов, применяемый для создания различных эффектов с изображениями и другими объектами веб-страницы. Обычно работает в связке с событием onMouseout.

```
<html>
	<head>
		<title></title>
	</head>
	<body>
		<p><img src="images/graph1.png"
		onmouseout="this.src='images/graph1.png'"
		onmouseover="this.src='images/graph2.png'"> 
		<img src="images/graph1.png"
		onmouseout="this.src='images/graph1.png'"
		onmouseover="this.src='images/graph2.png'"></p>
	</body>
</html>
```

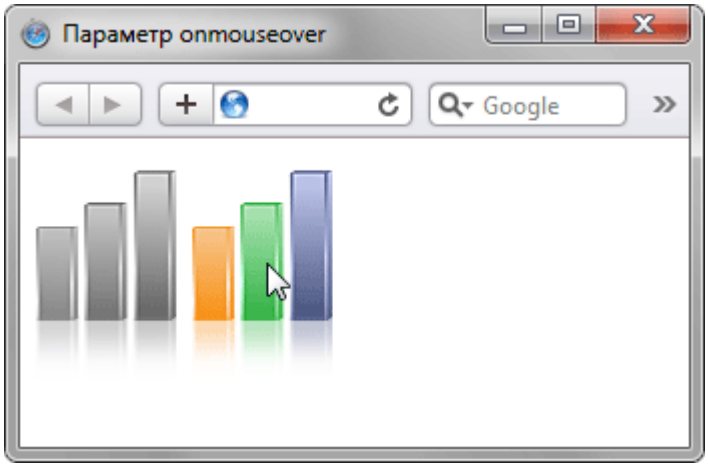

Рис. 17.13.

В данном примере при наведении курсора на изображение оно заменяется на другое, что создает эффект превращения черно-белой картинки в цветную

17.1.5. Обработка события click

Вообще говоря, обработчик события click в современном JavaScript не нужен. Можно прекрасно обойтись URL-схемой javascript, которая была специально придумана для перехвата события гипертекстового перехода. Обработчик onClick следует рассматривать как реликт, доставшийся нам в наследство от предыдущих версий языка, который поддерживается в версиях Netscape Navigator и Internet Explorer.

Основная задача обработчика данного события - перехват события гипертекстового перехода. Если функция обработки данного события возвращает значение true, то переход происходит, при значении false — не происходит:

Отменим переход в начало страницы описания события обработчика onClick:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
<body>
</head>
<script language="javascript"
    type="text/javascript">
    document.title = "Белодед Н.И.";
</script>
<a href="#click" onclick="window.alert('Нет перехода на #click'); return false;
">
    onClick
</a>
</body>
</html>
```

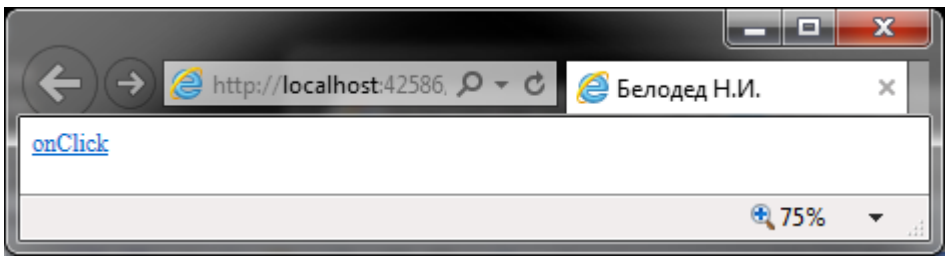

Рис. 17.14.

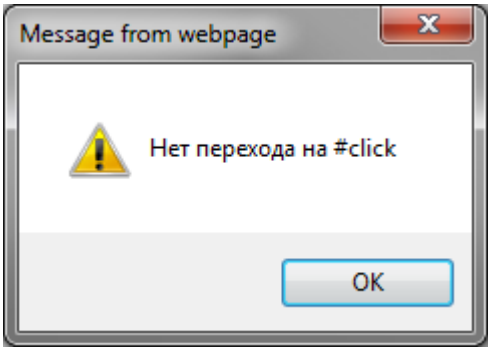

Рис. 17.15.

А теперь дадим пользователю право выбора перехода в начало страницы посредством окна подтверждения:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
<body>
</head>
<script language="javascript"
    type="text/javascript">
    document.title = "Белодед Н.И.";
</script>
<a href="#top" onclick="return window.confirm('Перейти в начало страницы? ');
">
    переход в начало страницы
</a>
</body>
</html>
```

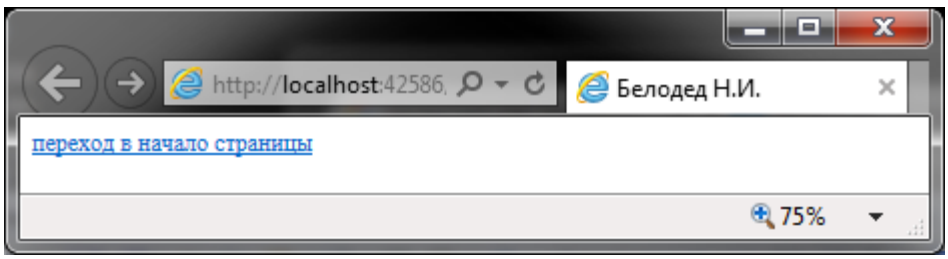

Рис. 17.16.

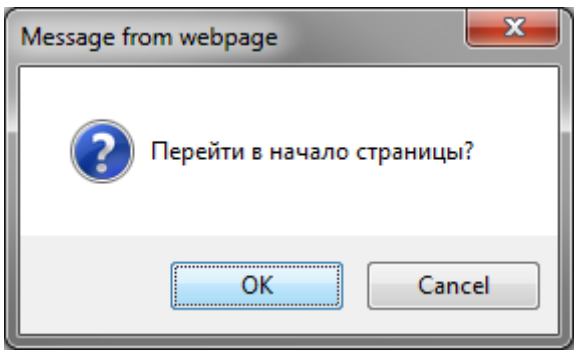

Рис. 17.17.

Обратите внимание на место применения функции window. confirme) - аргумент команды return. Логика проста: функция возвращает значение true или false, и именно оно подставляется в качестве аргумента. Если просто написать функцию без return, то ничего работать не будет.

Можно ли вообще обойтись одним обработчиком onClick без использования атрибута HREF? Видимо, нет. Первое, что необходимо браузеру - это определение типа контейнера A. Если в нем есть только атрибут NAME, то это якорь, если присутствует атрибут HREF - ссылка. Это два разных объекта. Они имеют различные составляющие, в том числе и обработчики со бытий. В контексте текущего раздела нам нужна именно ссылка, т. е. контейнер A с атрибутом HREF. Проверим наше предположение:

```
<a id="red" onclick="window.alert(""); return false; ">
```

```
    Нет атрибута HREF1
</a>
```

Текст «Нет атрибута HREF1» - это якорь. Обработчик на нем не работает, так как на него нельзя указать мышью.

```
<a href="" id="A1" onclick="window.alert('URL: ' + this.href); return false; ">
    Нет атрибута HREF2
</a>
```

Теперь мы указали пустую ссылку (см. поле статуса). Содержание окна - это база URL.
Полностью код имеет вид:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
<body>
</head>
<script language="javascript"
    type="text/javascript">
    document.title = "Белодед Н.И.";
</script>
<a id="red" onclick="window.alert(""); return false; ">
    Нет атрибута HREF1
</a>
<br /><br />
<a href="" id="A1" onclick="window.alert('URL: ' + this.href); return false; ">
    Нет атрибута HREF2
</a>
</body>
</html>
```

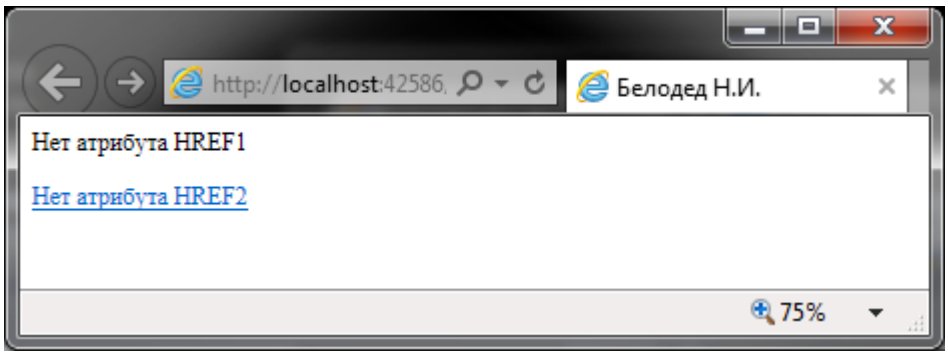

Рис. 17.18.

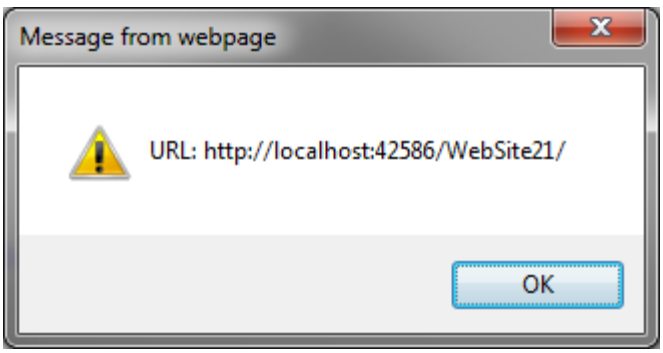

Рис. 17.19.

17.1.6. Схема URL- "javascript:..."

Для программирования гипертекстовых переходов в спецификацию универсального идентификатора ресурсов (URL) разработчики JavaScript ввели отдельную схему по аналогии со схемами http, ftp и т. п. — javascript. Эта схема URL упоминается в разделе «Размещение JavaScript-кода» в контексте передачи управления JavaScript-интерпретатору от HTML-парсера. Кроме того, о программировании гипертекстового перехода рассказано в разделе «Обработка события click». Теперь мы рассмотрим более общий случай обработки события гипертекстового перехода при выборе гипертекстовой ссылки.

Схема URL javascript в общем виде выглядит следующим образом:

```
<a href="javascript:...; ">... </a>
<form action="javascript:... ">
```

Одним словом, в любом месте, где мы используем URL, вместо любой из стандартных схем можно применить схему javascript. Единственное исключение составляет контейнер IMG. URL в нем используется в атрибуте SRC. Принять определенное значение SRC может при помощи либо назначения в IMG, либо обращения к свойству IMG. По большому счету, применение JavaScript в SRC может только проинициализировать картинку.

Рассмотрим пример простой гипертекстовой ссылки:

```
<html>
<head>
<title></title>
</head>
<body>

<a href="javascript: window.alert('window. alert() изменяет HREF' ); void(0);
">
    Заменили обычную ссылку
</a>


</body>
</html>
```

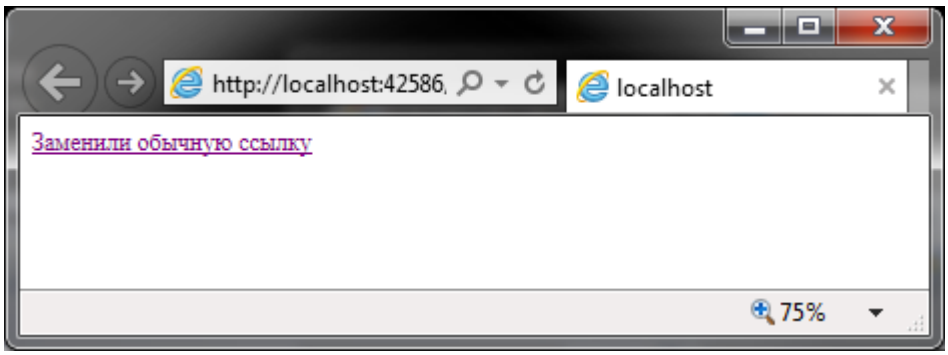

Рис. 17.20

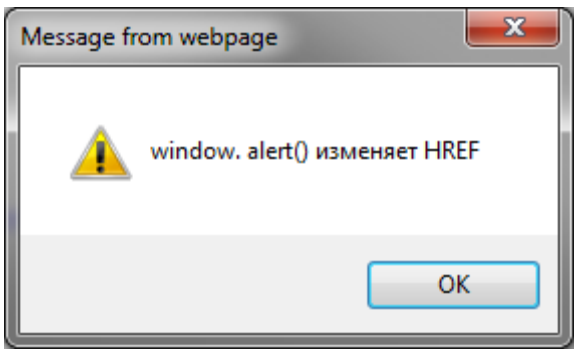

Рис. 17.21.

Можно выполнить аналогичную операцию, но над картинкой:

```
<html>
<head>
<title></title>
<script language="javascript"
    type="text/javascript">
    var flag = 0;
function ichange() {
    if (flag == 0) {
        document.i1.src = "gr101.png";
        flag = 1;
    }
    else {
```

```
        document.i1.src = "gr102.png";
        flag = 0;
    }
}
</script>
</head>
<body>

<a href="javascript: ichange(); void(0); ">
    <img name="i1" src="gr101.png" boder="0" with="200" height="200" alt="" />
</a>
</body>
</html>
```

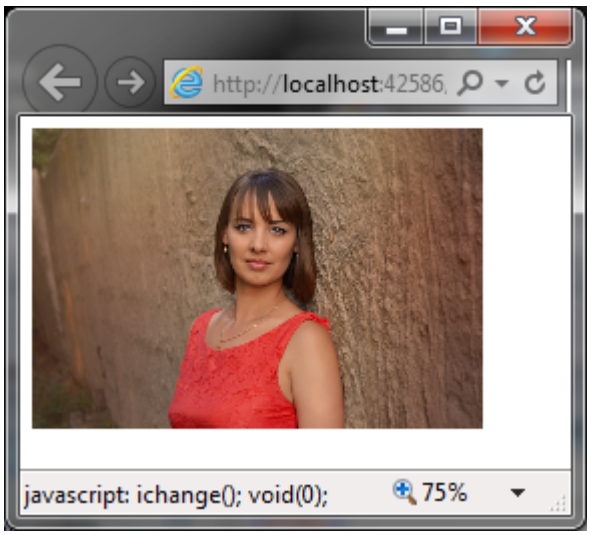

Рис. 17.22.

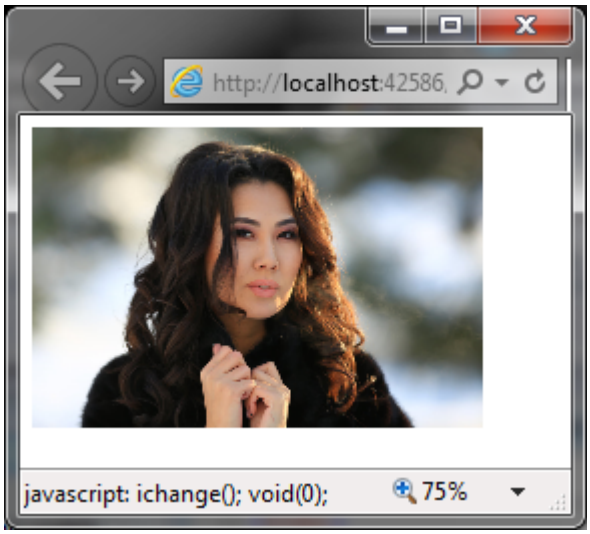

Рис. 17.23.

Попробуем теперь выполнить JavaScript-код применительно к контейнеру FORM:

```
<html>
<head>
<title></title>
<script language="javascript"
    type="text/javascript">
    var flag = 0;
function ichange() {
    if (flag == 0) {
        document.i1.src = "gr101.png";
        flag = 1;
    }
    else {
```

```
                document.i1.src = "gr102.png";
                flag = 0;
            }
        }
        </script>
        </head>
        <body>
        <form name="f" action="javascript: window.alert(document.f.fio.value); void(0);
" method="post" >
            Введите текст для отображения в окне и нажмите ввод:
        <input name="fio" size="20" maxlength="20" />
        </form>
        </body>
        </html>
```

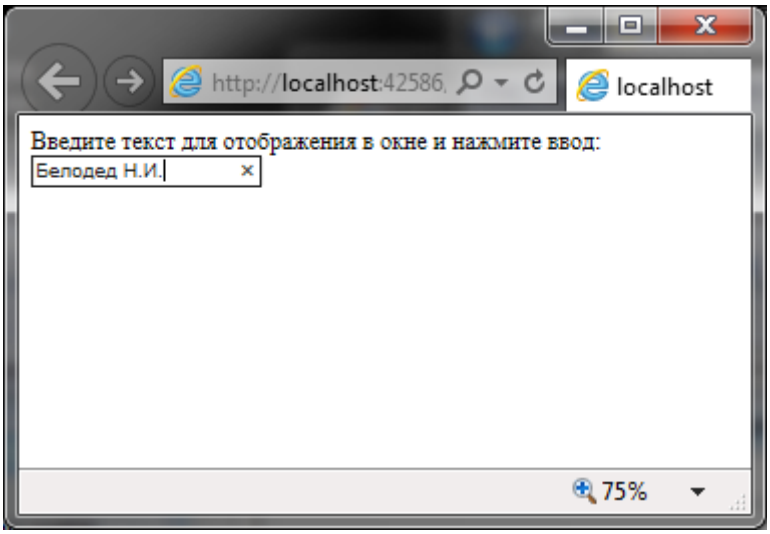

Рис. 17.24.

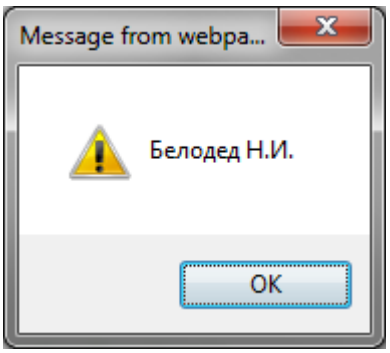

Рис. 17.25.

Следует отметить, что все-таки использование схемы javascript в этом месте HTML-разметки выглядит неудачно. Для того чтобы все работало так, как задумано автором, приходится использовать метод POST. Гораздо логичнее применить обработчик события onSubmit.

17.2. Контрольные вопросы

1. Перечислите элементы, составляющие URL.
2. Сколько уровней не должна превышать глубина структуры каталогов? Почему?
3. Какой адрес лучше использовать: http://mysite.com/catalog/45275 или http://mysite.com/catalog/computers. Почему?
4. Когда срабатывает событие onMouseover?
5. Для чего используются события Mouseover и Mouseout?
6. Какова область применения URL на HTML-страницах?
7. Как можно идентифицировать динамический URL
8. Для чего используются события Mouseover и Mouseout?
9. Какова область применения URL на HTML-страницах?
10. Как можно идентифицировать динамический URL

17.3. Задания

1. Создайте простую HTML-страницу и разместите на ней несколько изображений. Используя событие onMouseover, сделайте так, чтобы каждое изображение изменялось при наведении курсора. (R1701)

2. Создайте еще одну страницу на которой поместите ссылку на страницу из задания 1. Создайте окно подтверждения перехода по ссылке, которое появляется при нажатии на эту ссылку. (R1702)

3. На первой странице сделайте так, чтобы одно из изображений менялось на другое при нажатии на него. (R1703)