МИНЕСТЕРСТВО ПО РАЗВИТИЮ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИИ И КОМУНИКАЦИИ РЕСПУБЛИКИ УЗБЕКИСТАН

КАРШИНСКИЙ ФИЛИАЛ ТАШКЕНТСКОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ФАКУЛТЕТ «КОМПЬЮТЕРНЫЙ ИНЖИНИРИНГ»

КАФЕДРА «ПРОГРАММННЫЙ ИНЖИНИРИНГ»

РЕФЕРАТ НА ТЕМУ: **ПРОГРАММИРУЕМ ГРАФИКУ**

Выполнил : **студент АКТ-12-13 группы**
Ж.Кахрамонов

Проверила: **Д.Нуржабова**

Карши – 2015

План:

1. Объект Image
2. Изменение картинки
3. Запуск и остановка мультипликации
4. Оптимизация отображения
5. Оптимизация при загрузке
6. Контрольные вопросы.

Программируем графику

Ключевые слова: Объект Image, src и lowsrc , Мультипликация , Событие onLoad(),Предварительная загрузка, Нарезка картинок, Графика и таблицы , Графика и обработка событий , Вертикальные и горизонтальные меню , Вложенные меню

В этой лекции рассказано о приемах программирования изменений графических образов на HTML-страницах. Подробно разбираются основные приемы программирования образов и приводятся примеры.

Объект Image

Наиболее зрелищные эффекты при программировании на JavaScript достигаются при работе с *графикой*. При этом в арсенале программиста не так уж много инструментов: встроенные в документ картинки, возможность генерации объекта Image, комбинирование картинок с гипертекстовыми ссылками и таблицами. Тем не менее обилие различных эффектов, которые достигаются этими нехитрыми средствами, впечатляет.

Программирование *графики* в JavaScript опирается на объект Image, который характеризуется следующими свойствами, методами и событиями:

Свойства	Методы	События
• border • complete • height • hspace • name • src • vspace • width • lowsrc	нет	• onAbort • onError • onLoad

Несмотря на такое обилие свойств, их абсолютное большинство можно только читать, но не изменять. Об этом свидетельствует, прежде всего, отсутствие методов. Но два свойства все же можно изменять: src и lowsrc. Этого оказывается достаточно для множества эффектов с картинками.

Все объекты класса Image можно разделить на встроенные и порожденные программистом. Встроенные объекты — это картинки контейнеров *IMG*. Если эти картинки поименовать, к ним можно обращаться по имени:

```
<A HREF="javascript:void(0);"
onClick="window.alert('Image name:'+
      document.images[0].name)">
<IMG NAME=tuit SRC=images.gif BORDER=0>
</A>
```

Картинка активна. Если на нее нажать, получим имя контейнера *IMG*. Обращение document.images[0].name позволяет распечатать это имя в окне предупреждения. При этом само имя указано как name=tuit в контейнере *IMG*.

К встроенному *графическому* объекту можно обратиться и по индексу:

```
document.images[0];
```

В данном случае images[0] — это первая картинка документа.

src и lowsrc

Свойства src и lowsrc определяют URL изображения, которое монтируется внутрь документа. При этом lowsrc определяет временное изображение, обычно маленькое, которое отображается, пока загружается основное изображение, чей URL указывается в атрибуте SRC контейнера *IMG*. Свойство src принимает значение атрибута SRC контейнера *IMG*. Программист может изменять значения и src, и lowsrc. Рассмотрим пример с src:

```
document.i2.src="images2.gif";
```

Как видно из этого примера, существует возможность модифицировать вмонтированную картинку за счет изменения значения свойства src встроенного объекта Image. Если вы в первый раз просматриваете данную страницу (т.е. картинки не закешированы браузером), то постепенное изменение картинки будет заметно. Как ускорить это изменение, мы рассмотрим в следующем разделе.

Изменение картинки

Изменить картинку можно, только присвоив свойству src встроенного объекта Image новое значение. На странице "Программирование графики" показано, как это делается в простейшем случае. Очевидно, что медленная перезагрузка картинки с сервера не позволяет реализовать быстрое листание. Попробуем решить эту проблему.

Собственно, решение заключается в разведении по времени подкачки картинки и ее *отображения*. Для этой цели используют конструктор объекта Image:

```
<TABLE>
<TD>
<A HREF="javascript:void(0)";
onMouseover="document.m0.src=color[0].src;
          return true;"
onMouseout="document.m0.src=mono[0].src;
         return true;">
<IMG NAME=m0 SRC=images0.gif border=0>
</A>
</TD>
...
</TABLE>
```

Фрагмент кода показывает типовой прием замещения и восстановления картинки при проходе курсора мыши. Естественно, что менять можно не одну, а сразу несколько картинок.

Главное, тем не менее, не в том, что картинки замещаются, а в том, с какой скоростью они это делают. Для достижения нужного результата в начале страницы создаются массивы картинок, в которые перед отображением перекачивается *графика* (обратите внимание на строку статуса при загрузке страницы):

```
color = new Array(32);
mono = new Array(32);
for(i=0;i<32;i++)
{ mono[i] = new Image();
color[i] = new Image();
if(i.toString().length==2)
{
mono[i].src = "images0"+i+".gif";
color[i].src = "images0"+i+".gif";
}
else
{
mono[i].src = "images0"+i+".gif";
color[i].src = "images0"+i+".gif";
}
}
```

Еще один характерный прием — применение функции отложенного исполнения JavaScript-кода (eval()):

```
function def()
{
for(i=0;i<32;i++)
{
eval("document.m"+i+".src=mono["+i+"].src");
}
for(i=0;i<5;i++)
{
eval("document.r"+i+".src=rm["+i+"].src");
}
}
```

В данном случае eval() избавляет нас от необходимости набирать операции присваивания (32 строки — это не фунт изюму).

Мультипликация

Естественным продолжением идеи замещения значения атрибута SRC в контейнере *IMG* является мультипликация, т.е. последовательное изменение значения этого атрибута во времени. Для реализации мультипликации используют метод объекта Window — setTimeout().

Собственно, существует два способа запуска мультипликации:

- onLoad();
- onClick(), onChange() ...

Наиболее популярный — setTimeout() при onLoad().

Событие onLoad()

Событие onLoad() наступает в момент окончания загрузки документа браузером. Обработчик события указывается в контейнере BODY:

```
...
<BODY onLoad="JavaScript_code">
...
```

В нашем случае при загрузке документа должен начать выполняться цикл изменения картинки:

```
function movie()
```

```
{
eval("document.images[0].src='clock"+
     i+".gif';");
i++;if(i>6) i=0;
setTimeout("movie();",500);
}
...
<BODY onLoad="movie();">
...
```

В примере используется бесконечный цикл, хотя можно реализовать и конечное число подмен:

```
function movie()
{
eval("document.images[0].src='clock"+
     i+".gif';");
i++;
if(i<7)
{
setTimeout("movie();",500);}
}
...
<BODY onLoad="movie();">
```

В обоих примерах следует обратить внимание на использование метода setTimeout(). На первый взгляд, это просто рекурсия. Но в действительности все несколько сложнее. JavaScript разрабатывался для многопоточных операционных систем, поэтому правильнее будет представлять себе исполнение скриптов следующим образом:

1. Скрипт получает управление при событии onLoad().
2. Заменяет картинку.
3. Порождает новый скрипт и откладывает его исполнение на 500 миллисекунд.
4. Текущий скрипт уничтожается JavaScript-интерпретатором.

После окончания срока задержки исполнения все повторяется. В первом примере (бесконечное повторение) функция порождает саму себя и, тем самым, поддерживает непрерывность своего выполнения. Во втором примере (конечное число итераций) после девяти повторов функция не порождается. Это приводит к завершению процесса отображения новых картинок.

Запуск и остановка мультипликации

Перманентная мультипликация может быть достигнута и другими средствами, например многокадровыми *графическими* файлами. Однако движение на странице — не всегда благо. Часто возникает желание реализовать запуск и останов движения по требованию пользователя. Удовлетворим это желание, используя предыдущие примеры (запустить или остановить мультипликацию):

```
var flag1=0;
function movie()
{
if(flag1==0)
{
eval("document.images[0].src='clock"+
     i+".gif';");
i++;if(i>6) i=0;
}
setTimeout("movie();",500);
}
...
<BODY onLoad="movie();">
...
<FORM>
<INPUT TYPE=button VALUE="Start/Stop"
onClick="if(flag1==0) flag1=1; else flag1=0;">
</FORM>
```

В данном случае мы просто обходим изменение картинки, но при этом не прекращаем порождение потока. Если мы поместим setTimeout() внутрь конструкции if(), то после нажатия на кнопку Start/Stop поток порождаться не будет, и запустить движение будет нельзя.

Существует еще один способ решения проблемы остановки и старта мультипликации. Он основан на применении метода clearTimeout(). Внешне все выглядит по-прежнему, но процесс идет совсем по-другому:

```
var flag1=0;
var id1;
function movie()
{
eval("document.images[0].src='clock"+
     i+".gif';");
i++;if(i>6) i=0;
id1 = setTimeout("movie();",500);
}
```

```
...
<BODY onLoad="movie();">
...
<FORM>
<INPUT TYPE=button VALUE="Start/Stop"
onClick="if(flag1==0)
  { id1=setTimeout('movie();',500); flag1=1;}
  else {clearTimeout(id1); flag1=0;};">
</FORM>
```

Обратите внимание на два изменения. Во-первых, объявлен и используется идентификатор потока (id1); во-вторых, применяется метод clearTimeout(), которому, собственно, идентификатор потока и передается в качестве аргумента. Чтобы остановить воспроизведение функции movie() достаточно "убить" поток.

Оптимизация отображения

При программировании *графики* следует учитывать множество факторов, которые влияют на скорость отображения страницы и скорость изменения *графических* образов. При этом обычная дилемма оптимизации программ — скорость или размер занимаемой памяти — решается только путем увеличения скорости. О размере памяти при программировании на JavaScript думать как-то не принято.

Из всех способов оптимизации отображения картинок мы остановимся только на нескольких:

- оптимизация отображения при загрузке;
- оптимизация отображения за счет предварительной загрузки;
- оптимизация отображения за счет нарезки изображения.

Если первые две позиции относятся в равной степени как к отображению статических картинок, так и к мультипликации, то третий пункт характерен главным образом для мультипликации.

Оптимизация при загрузке

Практически в любом руководстве по разработке HTML-страниц отмечается, что при использовании контейнера *IMG* в теле HTML-страницы следует указывать атрибуты WIDTH и HEIGHT. Это продиктовано порядком загрузки компонентов страницы с сервера и алгоритмом работы HTML-парсера. Первым загружается текст разметки. После этого парсер разбирает текст и начинает загрузку дополнительных компонентов, в том числе *графики*. При этом *загрузка картинок*, в зависимости от типа HTTP-протокола, может идти последовательно или параллельно.

Также параллельно с загрузкой парсер продолжает свою работу. Если для картинок заданы параметры ширины и высоты, то можно отформатировать текст и отобразить его в окне браузера. До тех пор, пока эти параметры не определены, отображения текста не происходит.

Таким образом указание высоты и ширины картинки позволит отобразить документ раньше, чем картинки будут получены с сервера. Это дает пользователю возможность читать документ или задействовать его гипертекстовые ссылки до момента полной загрузки (событие load).

С точки зрения JavaScript, указание размеров картинки задает начальные параметры окна отображения *графики* внутри документа. Это позволяет воспользоваться маленьким прозрачным образом, для того, чтобы заменить его полноценной картинкой. Идея состоит в передаче маленького объекта для замещения его по требованию большим объектом.

Предварительная загрузка

Замена одного образа другим часто бывает оправдана только в том случае, когда это происходит достаточно быстро. Если перезагрузка длится долго, то эффект теряется. Для быстрой подмены используют возможность предварительной загрузки документа в специально созданный объект класса Image.

Реальный эффект можно почувствовать только при отключении кэширования страниц на стороне клиента (браузера). Кэширование часто используют для ускорения работы со страницами Web-узла. Как правило, загрузка первой страницы — это достаточно длительный процесс. Самое главное, чтобы пользователь в этот момент был готов немного подождать. Поэтому, кроме *графики*, необходимой только на первой странице, ему можно передать и *графику*, которая на ней не отображается. Но зато при переходе к другим страницам узла она будет отображаться без задержки на передачу с сервера.

Описанный выше прием неоднозначен. Его оправдывает только то, что если пользователь нетерпелив, то он вообще отключит передачу *графики*.

Нарезка картинок

Нарезка картинок применяется довольно часто. Она позволяет достигать эффекта частичного изменения отображаемой картинки. Чаще всего он применяется при создании меню.

Кроме подобного эффекта нарезка позволяет реализовать мультипликацию на больших картинках. При этом изменяется не весь образ, а только отдельные его части.

Графика и таблицы

Одним из наиболее популярных приемов дизайна страниц Web-узла является техника нарезки картинок на составные части. Можно выделить следующие способы применения этой техники для организации навигационных компонентов страницы:

- горизонтальные и вертикальные меню;
- вложенные меню;
- навигационные *графические* блоки.

Главной проблемой при использовании нарезанной *графики* является защита ее от контекстного форматирования страницы HTML-парсером. Дело в том, что он автоматически переносит элементы разметки на новую строку, если они не помещаются в одной. Составные части нарезанной картинки должны быть расположены определенным образом, поэтому простое их перечисление в ряд не дает желаемого эффекта:

```
<IMG SRC="image1.gif"><IMG
SRC="image2.gif"><IMG
SRC="image3.gif"><IMG
SRC="image4.gif">
```

Рис. 16.1.

Элементы переносятся на новую строку, так как ширина раздела меньше общей ширины всех картинок. Проблема решается, если применить защиту от парсера — <PRE>:

```
<PRE>
<IMG SRC="image1.gif"><IMG
SRC="image2.gif"><IMG
SRC="image3.gif"><IMG
SRC="image4.gif">
</PRE>
```

Рис. 16.2.

Использование такого меню требует определения на нем гипертекстовых ссылок, что приводит к следующему эффекту:

```
<PRE>
<A HREF="javascript:void(0);"><IMG
SRC="image1.gif"></A><A
```

```
HREF="javascript:void(0);"><IMG
SRC="image2.gif"></A><A
HREF="javascript:void(0);"><IMG
SRC="image3.gif"></A><A
HREF="javascript:void(0);"><IMG
SRC="image4.gif"></A>
</PRE>
```

Рис. 16.3.

Этого можно достичь за счет применения атрибута BORDER равного 0:

```
<PRE>
<A HREF="javascript:void(0);"><IMG
SRC="image1.gif" BORDER="0"></A><A
HREF="javascript:void(0);"><IMG
SRC="image2.gif" BORDER="0"></A><A
HREF="javascript:void(0);"><IMG
SRC="image3.gif" BORDER="0"></A><A
HREF="javascript:void(0);"><IMG
SRC="image4.gif" BORDER="0"></A>
</PRE>
```

Рис. 16.4.

Теперь попробуем тем же способом реализовать многострочное меню:

Пример 16.1. (html, txt)

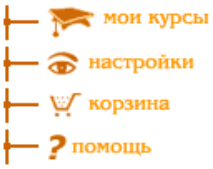

Рис. 16.5.

Сплошной картинки не получается, так как высота строки не равна высоте картинки. Подогнать эти параметры практически невозможно. Каждый пользователь настраивает браузер по своему вкусу. Решение заключается в использовании таблицы:

Пример 16.2. (html, txt)

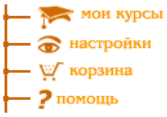

Рис. 16.6.

В данном случае все картинки удается сшить без пропусков и тем самым достичь непрерывности навигационного дерева. Пропуски устраняются путем применения атрибутов BORDER, CELLSPACING и CELLPADDING. Первый устраняет границы между ячейками, второй устанавливает расстояние между ячейками равным 0 пикселов, третий устанавливает отступ между границей ячейки и элементом, помещенным в нее, в 0 пикселов.

Графика и обработка событий

В данном разделе мы не будем рассматривать обработчики событий контейнера *IMG*. Мы остановимся на наиболее типичном способе комбинирования обработчиков событий и изменения *графических* образов. Собственно, не имело бы смысла применять нарезанную *графику*, если бы не возможность использования обработчиков событий для изменения отдельных частей изображения. Продолжая обсуждение примера с навигационным деревом, покажем его развитие с обработкой событий, вызванных наведением мыши на объект, и изменением картинок:

Пример 16.3. (html, txt)

В данном примере при проходе курсор мышки через картинки меню последние изменяются. Этот эффект достигается за счет применения двух событий: onMouseover и onMouseout. По первому событию картинка меняется с позитива на негатив, по второму событию восстанавливается первоначальный вариант. Следует заметить, что события определены в контейнере якоря (A), а не в контейнере *IMG*. Это наиболее устойчивый с точки зрения совместимости браузеров вариант.

Вертикальные и горизонтальные меню

Практически все, что изложено в разделах "Графика и таблицы" и "Графика и обработка событий" касается вопросов построения одноуровневых меню. Поэтому в данном разделе мы постараемся привести более или менее реальные примеры таких меню. Графическое меню удобно тем, что автор может всегда достаточно точно расположить его компоненты на экране. Это, в свою очередь, позволяет и другие элементы страницы точнее располагать относительно элементов меню:

Пример 16.4. (html, txt)

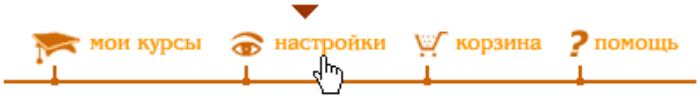

Рис. 16.7.

В данном случае стрелочка бежит точно над тем элементом, на который указывает мышь. По большому счету, применение атрибута ALT у *IMG* и его дублирование в строке статуса является гораздо более информативным, чем добавление нового *графического* элемента. Правда, отображается содержание ALT с некоторой задержкой:

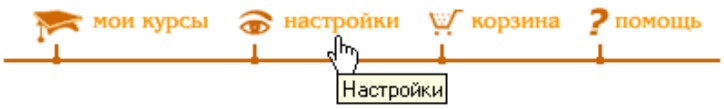

Рис. 16.8.

Посмотрим теперь на реализацию вертикального меню, построенного на основе *графических* блоков текста, как сейчас это принято делать:

Пример 16.5. (html, txt)

Рис. 16.9.

При движении мыши у соответствующего компонента, попавшего в фокус мыши, "отгибается уголок". В данном случае "уголок" — это самостоятельная картинка. Все уголки реализованы в правой колонке таблицы. Для того чтобы гипертекстовая ссылка срабатывала по обеим картинкам (тексту и "уголку"), применяются одинаковые контейнеры A, охватывающие *графические* образы. В этом решении есть один недостаток: при переходе от текста к "уголку" последний "подмигивает". Картинки можно разместить и в одной ячейке таблицы, но тогда нужно задать ее ширину, иначе при изменении размеров окна браузера картинки могут "съехать". Чтобы убрать "подмигивание", необходимо сделать полноценные картинки замены.

"Подмигивание" происходит при переходе с одного элемента разметки контейнера на другой. При этом заново определяются свойства отображения элемента.

Вложенные меню

При обсуждении программирования форм отмечено, что в HTML нет стандартного способа реализации вложенных меню. Тем не менее за счет *графики* можно создать их подобие. При этом следует понимать, что место, на которое ложится *графика*, нельзя заполнить текстом:

Пример 16.6. (html, txt)

Рис. 16.10.

В этом примере вложенное меню расположено справа от основного. Эффект вложенности достигается за счет изменения цвета. Подчиненность меню можно подчеркнуть изменением его положения относительно основного меню:

Рис. 16.11.

В этом случае для продвижения меню вниз необходимо зарезервировать место при помощи невидимых или видимых картинок. Это не обязательно должны быть иллюстративные картинки, которые не несут никакой нагрузки.

При использовании слоев можно создать настоящее выпадающее меню.

Пример 16.1.

```
<PRE>
<IMG SRC=tree.gif WIDTH=27 HEIGHT=21 BORDER=0><A
HREF="javascript:void(0);"><IMG SRC="image1.gif" WIDTH=103 HEIGHT=21
BORDER=0></A>
<IMG SRC=tree.gif WIDTH=27 HEIGHT=21 BORDER=0><A
HREF="javascript:void(0);"><IMG SRC="image2.gif" WIDTH=103 HEIGHT=21
BORDER=0></A>
<IMG SRC=tree.gif WIDTH=27 HEIGHT=21 BORDER=0><A
HREF="javascript:void(0);"><IMG SRC="image3.gif" WIDTH=103 HEIGHT=21
BORDER=0></A>
<IMG SRC=tree.gif WIDTH=27 HEIGHT=21 BORDER=0><A
HREF="javascript:void(0);"><IMG SRC="image4.gif" WIDTH=103 HEIGHT=21
BORDER=0></A>
</PRE>
```

Пример 16.2.

```
<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0 ALIGN=center>
<TR>
<TD><IMG SRC=tree.gif WIDTH=27 HEIGHT=21 BORDER=0></TD>
<TD><A HREF="javascript:void(0);"><IMG SRC="image1.gif" WIDTH=103
HEIGHT=21 BORDER=0></A></TD>
</TR>
<TR>
<TD><IMG SRC=tree.gif WIDTH=27 HEIGHT=21 BORDER=0></TD>
<TD><A HREF="javascript:void(0);"><IMG SRC="image2.gif" WIDTH=103
HEIGHT=21 BORDER=0></A></TD>
```

```
</TR>
<TR>
<TD><IMG SRC=tree.gif WIDTH=27 HEIGHT=21 BORDER=0></TD>
<TD><A HREF="javascript:void(0);"><IMG SRC="image3.gif" WIDTH=103
HEIGHT=21 BORDER=0></A></TD>
</TR>
<TR>
<TD><IMG SRC=tree.gif WIDTH=27 HEIGHT=21 BORDER=0></TD>
<TD><A HREF="javascript:void(0);"><IMG SRC="image14.gif" WIDTH=103
HEIGHT=21 BORDER=0></A></TD>
</TR>
</TABLE>
```

Пример 16.3.

```
<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0 ALIGN=center>
<TR>
<TD><IMG SRC=image.gif WIDTH=20 HEIGHT=20 BORDER=0></TD>
<TD><IMG SRC=image1.gif WIDTH=20 HEIGHT=20 BORDER=0></TD>
</TR>
<TR>
<TD><IMG SRC=image2.gif WIDTH=20 HEIGHT=20 BORDER=0></TD>
<TD><A HREF="javascript:void(0);"
onMouseover="document.manual.src='image3.gif';return true;"
onMouseout="document.manual.src='image4.gif'; return true;">
<IMG SRC=image5.gif BORDER=0 WIDTH=20 HEIGHT=20></A></TD>
</TR>
<TR>
<TD><IMG SRC=image6.gif WIDTH=20 HEIGHT=20 BORDER=0></TD>
<TD><A HREF="javascript:void(0);"
onMouseover="document.desk.src='image7.gif';return true;"
onMouseout="document.desk.src='image8.gif';return true;">
<IMG SRC=image9.gif BORDER=0 WIDTH=20 HEIGHT=20></A></TD>
</TR>
</TABLE>
```

Пример 16.4.

```
<TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0" ALIGN="center">
<TR ALIGN="center">
<TD><IMG NAME="e0" SRC="empty.gif" WIDTH="15" HEIGHT="8"
BORDER="0"></TD>
<TD><IMG NAME="e1" SRC="empty.gif" WIDTH="15" HEIGHT="8"
BORDER="0"></TD>
<TD><IMG NAME="e2" SRC="empty.gif" WIDTH="15" HEIGHT="8"
BORDER="0"></TD>
<TD><IMG NAME="e3" SRC="empty.gif" WIDTH="15" HEIGHT="8"
BORDER="0"></TD>
```

```
</TR>
<TR>
<TD><A HREF="javascript:void(0);"
onMouseover="document.e0.src='arrowdw.gif';return true;"
 onMouseout="document.e0.src='empty.gif';return true;">
 <IMG SRC="image1.gif" BORDER="0"></A></TD>
<TD><A HREF="javascript:void(0);"
onMouseover="document.e1.src='arrowdw.gif';return true;"
onMouseout="document.e1.src='empty.gif';return true;">
<IMG SRC="image2.gif" BORDER="0"></A></TD>
<TD><A HREF="javascript:void(0);"
onMouseover="document.e2.src='arrowdw.gif';return true;"
onMouseout="document.e2.src='empty.gif';return true;">
<IMG SRC="image3.gif" BORDER="0"></A></TD>
<TD><A HREF="javascript:void(0);"
onMouseover="document.e3.src='arrowdw.gif';return true;"
 onMouseout="document.e3.src='empty.gif';return true;">
 <IMG SRC="image4.gif" BORDER="0"></A></TD>
</TR>
</TABLE>
```

Пример 16.5.

```
<TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0" ALIGN="center">
<TR>
<TD><A HREF="javascript:void(0);"
onMouseover="document.evente1.src='corner.gif';"
onMouseout="document.evente1.src='clear.gif';">
<IMG SRC="image1.gif" border="0"></A></TD>
<TD><A HREF="javascript:void(0);"
onMouseover="document.evente1.src='corner.gif';"
onMouseout="document.evente1.src='clear.gif';">
<IMG NAME="evente1" SRC="clear.gif" border="0"></A></TD>
</TR>
<TR>
<TD><A HREF="javascript:void(0);"
onMouseover="document.evente2.src='corner.gif';"
onMouseout="document.evente2.src='clear.gif';">
<IMG SRC="image2.gif" border="0"></A></TD>
<TD><A HREF="javascript:void(0);"
onMouseover="document.evente2.src='corner.gif';"
onMouseout="document.evente2.src='clear.gif';">
<IMG NAME="evente2" SRC="clear.gif" border="0"></A></TD>
</TR>
<TR>
<TD><A HREF="javascript:void(0);"
onMouseover="document.evente3.src='corner.gif';"
```

```
onMouseout="document.evente3.src='clear.gif';">
<IMG SRC="image3.gif" border="0"></A></TD>
<TD><A                                                    HREF="javascript:void(0);"
onMouseover="document.evente3.src='corner.gif';"
onMouseout="document.evente3.src='clear.gif';">
<IMG NAME="evente3" SRC="clear.gif" border="0"></A></TD>
</TR>
<TR>
<TD><A                                                    HREF="javascript:void(0);"
onMouseover="document.evente4.src='corner.gif';"
 onMouseout="document.evente4.src='clear.gif';">
 <IMG SRC="image4.gif" border="0"></A></TD>
<TD>
<A HREF="javascript:void(0);" onMouseover="document.evente4.src='corner.gif';"
onMouseout="document.evente4.src='clear.gif';">
<IMG NAME="evente4" SRC="clear.gif" border="0">
</A></TD>
</TR>
</TABLE>
```

Пример 16.6.

```
<SCRIPT>
function submenu(a)
{
if(a==1)
{
document.menu00.src="image1.gif"; // 1 (активна)
document.menu10.src="image2.gif"; // 2
document.menu01.src="image3.gif"; // 1 пункт вложенного меню 1
document.menu02.src="image4.gif"; // 2 пункт вложенного меню 1
}
if(a==2)
{
document.menu00.src="image1.gif"; // 2
document.menu10.src="image2.gif"; // 1 (активна)
document.menu01.src="image3.gif"; // 1 пункт вложенного меню 2
document.menu02.src="image4.gif"; // 2 пункт вложенного меню 2
}
}
</SCRIPT>

<TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0" ALIGN="center">
<TR>
<TD><A HREF="javascript:void(0);" onMouseover="submenu(1);return true;">
<IMG NAME=menu00 SRC=image1.gif BORDER=0></a></td>
```

```
<TD><IMG NAME=menu01 SRC=image3.gif BORDER=0></TD>
</TR>
<TR>
<TD><A HREF="javascript:void(0);" onMouseover="submenu(2);return true;">
<IMG NAME=menu10 SRC=image2.gif BORDER=0></td>
<TD><IMG NAME=menu02 SRC=image4.gif BORDER=0></TD>
</TR>
</TABLE>
```

Контрольные вопросы:

1. Что такое Объект Image ?
2. Процесс изменение картинки?
3. Процесс Запуска и остановка мультипликации ?
4. Как Оптимизируется отображения ?
5. Как проходить Оптимизация при загрузке ?