Часть 5: Строка состояния и таймеры

Строка состояния

Составленные Вами программы на JavaScript могут выполнять запись в строку состояния - прямоугольник в нижней части окна Вашего браузера. Все, что Вам необходимо для этого сделать - всего лишь записать нужную строку в *window.status*. В следующем примере создаются две кнопки, которые можно использовать, чтобы записывать некий текст в строку состояния и, соответственно, затем его стирать.

Данный скрипт выглядит следующим образом:

```
<html>
<head>
<script language="JavaScript">
<!-- hide

function statbar(txt) {
   window.status = txt;
}

// -->
</script>
</head>
<body>

<form>
  <input type="button" name="look" value="Писать!"
    onClick="statbar('Привет! Это окно состо\яни\я!');">
  <input type="button" name="erase" value="Стереть!"
    onClick="statbar('');">
</form>

</body>
</html>
```

Итак, мы создаем форму с двумя кнопками. Обе эти кнопки вызывают функцию statbar(). Вызов от клавиши *Писать!* выглядит следующим образом:

```
statbar('Привет! Это окно состо\яни\я!');
```

В скобках мы написали строку: *'Привет! Это окно состо\яни\я!'*. Это как раз и будет текст, передаваемый функции statbar(). В свою очередь, можно видеть, что функция statbar() определена следующим образом:

```
function statbar(txt) {
   window.status = txt;
}
```

В заголовке функции в скобках мы поместили слово *txt*. Это означает, что строка, которую мы передали этой функции, помещается в переменную *txt*. Передача функциям переменных - прием, часто применяемый для придания функциям большей гибкости. Вы можете передать функции несколько таких аргументов - необходимо лишь отделить их друг от друга запятыми.

Строка *txt* заносится в строку состояния посредством команды `window.status = txt`.
Соответственно, удаление текста из строки состояния выполняется как запись в *window.status* пустой строки.

Механизм вывода текста в строку состояния удобно использовать при работе со ссылками. Вместо того, чтобы выводить на экран URL данной ссылки, Вы можете просто на словах объяснять, о чем будет говориться на следующей странице. Так link демонстрирует это - достаточно лишь поместить указатель вашей мыши над этой ссылкой: Исходный код этого примера выглядит следующим образом:

```
<a href="dontclck.htm"
  onMouseOver="window.status='Don\'t click me!'; return true;"
  onMouseOut="window.status='';">link</a>
```

Здесь мы пользуемся процедурами *onMouseOver* и *onMouseOut*, чтобы отслеживать моменты, когда указатель мыши проходит над данной ссылкой. Вы можете спросить, а почему в onMouseOver мы обязаны возвращать результат *true*. На самом деле это означает, что браузер не должен вслед за этим выполнять свой собственный код обработки события MouseOver. Как правило, в строке состояния браузер показывает URL соответствующей ссылки. Если же мы не возвратим значение *true*, то сразу же после того, как наш код был выполнен, браузер перепишет строку состояния на свой лад - то есть наш текст будет тут же затерт и читатель не сможет его увидеть. В общем случае, мы всегда можем отменить дальнейшую обработку события браузером, возвращая *true* в своей собственной процедуре обработки события. в JavaScript 1.0 процедура *onMouseOut* еще не была представлена. И если Вы пользуетесь Netscape Navigator 2.x, то возможно на различных платформах Вы можете получить различные результаты. Например, на платформах Unix текст исчезает даже несмостря на то, что браузер не знает о существовании процедуры *onMouseOut*. В Windows текст не исчезает. И если Вы хотите, чтобы ваш скрипт был совместим с Netscape 2.x для Windows, то можете, к примеру, написать функцию, которая записывает текст в окно состояния, а потом стирает его через некоторый промежуток времени. Программируется это с помощью таймера timeout. Подробнее работу с таймерами мы рассмотрим в одном из следующих параграфов.

В этом скрипте Вы можете видеть еще одну вещь - в некоторых случаях Вам понадобится печатать символы кавычек. Например, мы хотим напечатать текст *Don't click me* - однако поскольку мы передаем эту строку в процедуру обработки события onMouseOver, то мы используем для этого одинарные кавычки. Между тем, как слово *Don't* тоже содержит символ одинарной кавычки! И в результате если Вы просто впишете *'Don't ...'*, браузер запутается в этих символах '. Чтобы разрешить эту проблему, Вам достаточно лишь поставить обратный слэш \ перед символом кавычки - это означает, что данный

символ предназначен именно для печати. (То же самое Вы можете делать и с двойными кавычками - ").

Таймеры

С помощью функции Timeout (или таймера) Вы можете запрограммировать компьютер на выполнение некоторых команд по истечении некоторого времени. В следующем скрипте демострируется кнопка, которая открывает выпадающее окно не сразу, а по истечении 3 секунд.

Скрипт выглядит следующим образом:

```
<script language="JavaScript">
<!-- hide

function timer() {
  setTimeout("alert('Врем\я истекло!')", 3000);
}

// -->
</script>

...

<form>
  <input type="button" value="Timer" onClick="timer()">
</form>
```

Здесь *setTimeout()* - это метод объекта window. Он устанавливает интервал времени - я полагаю, Вы догадываетесь, как это происходит. Первый аргумент при вызове - это код JavaScript, который следует выполнить по истечении указанного времени. В нашем случае это вызов - *"alert('Врем\я истекло!')"*. Обратите пожалуйста внимание, что код на JavaScript должен быть заключен в кавычки.
Во втором аргументе компьютеру сообщается, когда этот код следует выполнять. При этом время Вы должны указывать в миллисекундах (3000 миллисекунд = 3 секунда).

Прокрутка

Теперь, когда Вы знаете, как делать записи в строке состояния и как работать с таймерами, мы можем перейти к управлению прокруткой. Вы уже могли видеть, как текст перемещается строке состояния. В Интернет этим приемом пользуются повсеместно. Теперь же мы рассмотрим, как можно запрограммировать прокрутку в основной линейке. Рассмотрим также и всевозможные усовершенствования этой линейки.

Создать бегущую строку довольно просто. Для начала давайте задумаемся, как вообще можно создать в строке состояния перемещающийся текст - бегущую строку. Очевидно, сперва мы должны записать в строку состояния некий текст.

Затем по истечении короткого интервала времени мы должны записать туда тот же самый текст, но при этом немного переместив его влево. Если мы это сделаем несколько раз, то у пользователя создастся впечатление, что он имеет дело с бегущей строкой.

Однако при этом мы должны помнить еще и о том, что обязаны каждый раз вычислять, какую часть текста следует показывать в строке состояния (как правило, объем текстового материала превышает размер строки состояния).

Итак, исходный код скрипта - я добавил к нему еще некоторые комментарии:

```
<html>
<head>
<script language="JavaScript">
<!-- hide


// define the text of the scroller
var scrtxt = "Это JavaScript! " +
    "Это JavaScript! " +
    "Это JavaScript!";
var len = scrtxt.length;
var width = 100;
var pos = -(width + 2);

function scroll() {

  // напечатать заданный текст справа и установить таймер

  // перейти на исходную позицию для следующего шага
  pos++;

  // вычленить видимую часть текста
  var scroller = "";
  if (pos == len) {
    pos = -(width + 2);
  }

  // если текст еще не дошел до левой границы, то мы должны
  // добавить перед ним несколько пробелов. В противном случае мы должны
  // вырезать начало текста (ту часть, что уже ушла за левую границу
  if (pos < 0) {
    for (var i = 1; i <= Math.abs(pos); i++) {
      scroller = scroller + " ";}
    scroller = scroller + scrtxt.substring(0, width - i + 1);
  }
  else {
    scroller = scroller + scrtxt.substring(pos, width + pos);
  }

  // разместить текст в строке состо\яни\я
  window.status = scroller;

  // вызвать эту функцию вновь через 100 миллисекунд
  setTimeout("scroll()", 100);
}

// -->
</script>
```

```
</head>

<body onLoad="scroll()">
Это пример прокрутки в строке состояния средствами JavaScript.
</body>
</html>
```

Большая часть функции scroll() нужна для вычленения той части текста, которая будет показана пользователю. Я не буду объяснять этот код подробно - Вам необходимо лишь понять, как вообще осуществляется эта прокрутка. Чтобы запустить этот процесс, мы используемся процедурой обработки события onLoad, описанной в тэге <body>. То есть функция *scroll()* будет вызвана сразу же после загрузки HTML-страницы.

Через посредство процедуры onLoad мы вызываем функцию scroll(). Первым делом в функции scroll() мы устанавливаем таймер. Этим гарантируется, что функция scroll() будет повторно вызвана через 100 миллисекунд. При этом текст будет перемещен еще на один шаг и запущен другой таймер. Так будет продолжаться без конца.

(В Netscape Navigator 2. x с таким типом скроллинга были некоторые проблемы - его выполнение иногда приводило к появлению ошибки 'Out of memory'. Я получил много писем, где объяснялось, что это возникает вследствие рекурсивного вызова функции scroll(), что в конце концов приводит к выходу за пределы памяти. Но это не так. Данный вызов функции не является рекурсивным! Рекурсию мы получим, если будем вызывать функцию scroll() непосредственно внутри самой же функции scroll(). А этого здесь мы как раз и не делаем. Прежняя функция, установившая таймер, заканчивается еще до того, как начинается выполнение новой функции. Проблема же состояла в том, что в действительности мы не могли в JavaScript выполнять коррекцию строк. И если Вы пробуете сделать это, то JavaScript просто-напросто создавал новый объект - но при этом не удалял старый. Именно таким образом происходило переполнение памяти.)

Скроллинг используется в Интернет довольно широко. И есть риск, что быстро он станет непопулярным. Я должен признаться, что и сам не очень его люблю. В большинстве страниц, где он применяется, особенно раздражает то, что из-за непрерывного скроллинга становится невозможным прочесть в строке состояния адрес URL. Эту проблему можно было бы решить, позаботившись о приостановке скроллинга, если происходит событие MouseOver - и, соответственно, продолжении, когда фиксируется onMouseOut. Если Вы хотите попытаться создать скроллинг, то пожалуйста не используйте стандартный его вариант - пробуйте привнести в него некоторые приятные особенности. Возможен вариант, когда одна часть текста приходит слева, а другая - справа. И когда они встречаются посередине, то в течение некоторых секунд текст остается неизменным. Воспользовавшись небольшой долей фантазии, Вы конечно же сможете найти еще несколько хороших альтернатив (некоторые примеры я привожу в своей книге).

e-mail:skoch@rumms.uni-mannheim.de
http://rummelplatz.uni-mannheim.de/~skoch/
Моя книга по JavaScript