18. Скрипты

18.1 Введение в скрипты

Клиентский *скрипт* - это программа, которая может сопровождать документ HTML или непосредственно быть внедренной в него. Эта программа выполняется на клиентской машине при загрузке документа или в другое время, например, когда активизируется ссылка. Поддержка скриптов в HTML не зависит от языка скрипта.

Скрипты предлагают авторам средства усиления интерактивности документов HTML. Например:

- Скрипты могут оцениваться во время загрузки документа и динамически изменять содержимое документа.
- Скрипты могут использоваться в форме для обработки вводимых данных. Дизайнеры могут динамически заполнять поля формы в зависимости от значений других полей. Они могут проверять, попадают ли введенные данные в предопределенный диапазон значений, соответствие полей и т.д.
- Скрипты могут включаться событиями, оказывающими влияние на документ, например, загрузкой, выгрузкой, фокусом элемента, перемещением мыши и т.д.
- Скрипты могут связываться с управляющими элементами формы (например, с кнопками) для представления элементов пользовательского интерфейса.

Авторы могут прикреплять к документу HTML два типа скриптов:

- Скрипты, выполняющиеся один раз при загрузке документа агентом пользователя. Скрипты, описанные в элементе SCRIPT, выполняются при загрузке документа. Для агентов пользователя, не обрабатывающих скрипты, авторы могут определить альтернативное содержимое с помощью элемента NOSCRIPT.
- Скрипты, выполняемые каждый раз, когда происходит определенное событие. Эти скрипты могут назначаться ряду элементов с помощью атрибутов внутренних событий.

Примечание. *Более подробная информация приводится в разделах о макросах скриптов.*

18.2 Разработка документов для агентов пользователя, поддерживающих скрипты

Следующие разделы относятся к агентам пользователей, поддерживающих скрипты.

18.2.1 Элемент SCRIPT

```
<!ELEMENT SCRIPT - - %Script;           -- выражение script -->
<!ATTLIST SCRIPT
  charset     %Charset;       #IMPLIED  -- кодировка символов связанного ресурса --
  type        %ContentType;   #REQUIRED - тип содержимого языка скрипты --
  language    CDATA           #IMPLIED  -- предопределенное имя языка серипта --
  src         %URI;           #IMPLIED  -- URI внешнего скрипта --
  defer       (defer)         #IMPLIED  -- агент пользователя может отложить выполнение
скрипта --
  >
```

Начальный тег: ***обязателен****, Конечный тег:* ***обязателен***

Определения атрибутов

`src` = *uri* [CT]

Этот атрибут определяет местоположение внешнего скрипта.

`type` = *content-type* [CI]

Этот атрибут задает язык скрипта содержимого элемента и имеет приоритет над языком скрипта, заданным по умолчанию. Язык скрипта указывается как тип содержимого (например, "text/javascript"). Авторы должны указать значение этого атрибута. Значение по умолчанию для этого атрибута не задано.

`language` = *cdata* [CI]

Нежелателен. Этот атрибут определяет язык скрипта содержимого этого элемента. Его значением является идентификатор языка, но поскольку идентификаторы не стандартизованы, этот атрибут является нежелательным, вместо него нужно использовать

атрибут type.
defer [CI]
Если этот логический атрибут установлен, он обеспечивает для агента пользователя подсказку о том, что скрипт не будет генерировать содержимое документа (например, "document.write" в javascript) и таким образом агент пользователя может продолжать синтаксический разбор и представление документа.
Атрибуты, определяемые в любом другом месте

- charset (кодировка символов)

Элемент SCRIPT помещает скрипт в документа. Этот элемент может указываться в элементе HEAD или BODY документа HTML несколько раз.
Скрипт может определяться в содержимом элемента SCRIPT или во внешнем файле. Если не установлен атрибут src, агенты пользователя должны интерпретировать содержимое элемента как скрипт. Если для атрибута src установлено значение URI, агенты пользователей должны игнорировать содержимое этого элемента и загружать скрипт через URI. Обратите внимание, что атрибут charset относится к кодировке символов скрипта, назначаемого атрибутом src; он не относится к содержимому элемента SCRIPT.
Скрипты оцениваются *ядром скрипта,* которое агент пользователя должен знать.
Синтаксис данных скрипта зависит от языка скрипта.

18.2.2 Указание языка скрипта

Поскольку документ HTML не предполагает определенный язык скрипта, авторы документа должны явно сообщить агентам пользователей язык каждого скрипта. Это может выполняться с помощью объявления скрипта по умолчанию или с помощью локального объявления.

Язык скрипта по умолчанию

Авторы должны указывать язык скрипта по умолчанию для всех скриптов в документе, включив следующее объявление META в тег HEAD:

```
<META http-equiv="Content-Script-Type" content="type">
```

где "type" - тип содержимого, именующий язык скрипта. Примерами значений являются "text/tcl", "text/javascript", "text/vbscript".
Если отсутствует объявление META, значение по умолчанию может устанавливаться с помощью заголовка протокола HTTP "Content-Script-Type".

```
    Content-Script-Type: тип
```

где "тип" - тип содержимого, именующий язык скрипта.
Агенты пользователей должны определять язык скрипта по умолчанию для документа в соответствии со следующими действиями (приоритет от высшего к низшему):

1. Если в объявлении META задается "Content-Script-Type", язык скрипта по умолчанию задается последним таким объявлением в потоке символов.
2. В противном случае, если в заголовках HTTP задается "Content-Script-Type", язык скрипта по умолчанию задается последним таким объявлением в потоке символов.

Документы, в которых не указан язык скрипта по умолчанию, но содержатся элементы, задающие внутренние события, некорректны. Агенты пользователей могут предпринимать попытки интерпретировать некорректно заданные скрипты, но это не обязательно. Средства разработки должны генерировать информацию о языке скрипта по умолчанию во избежание создания некорректных документов.

Локальное объявление языка скрипта

Язык скрипта можно задавать в каждом элементе SCRIPT с помощью атрибута type. Если не указан язык скрипта по умолчанию, этот атрибут должен устанавливаться для каждого элемента SCRIPT. Если задан язык скрипта по умолчанию, атрибут type имеет приоритет над ним.

В этом примере мы объявляем язык скриптов по умолчанию: "text/tcl". В заголовок включается элемент SCRIPT, сам скрипт находится во внешнем файле и использует язык "text/vbscript". Кроме того, один элемент SCRIPT находится в теле документа и содержит другой скрипт, написанный на языке "text/javascript".

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
     "http://www.w3.org/TR/REC-html40/strict.dtd">
<HTML>
<HEAD>
<TITLE>Документ со скриптами</TITLE>
<META http-equiv="Content-Script-Type" content="text/tcl">
<SCRIPT type="text/vbscript" src="http://someplace.com/progs/vbcalc">
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT type="text/javascript">
...код JavaScript...
</SCRIPT>
</BODY>
</HTML>
```

Ссылки из скрипта на элементы HTML

Каждый язык скриптов имеет собственные соглашения относительно ссылок на объекты HTML в скрипте. В данной спецификации не определяется стандартный механизм для ссылки на объекты HTML.

Однако скрипты должны ссылаться на элементы в соответствии с назначенным им именем. Ядро скриптов должно соблюдать при идентификации элемента следующие правила приоритета: атрибут name имеет преимущество над атрибутом id, если установлены оба эти атрибута. В противном случае используется установленный атрибут.

18.2.3 Внутренние события

Примечание. *Сообщаем авторам документов HTML, что в области внутренних событий (например, в привязке скриптов к событиям) весьма вероятны изменения. Работа в этой области ведется членами рабочей группы по объектной модели документов W3C (W3C Document Object Model Working Group) (более подробную информацию можно найти на Web-сайте W3C по адресу http://www.w3.org/).*

Определения атрибутов

onload = *скрипт* [CT]

Событие onload происходит, когда агент пользователя заканчивает загружать окно или все фреймы элемента FRAMESET. Этот атрибут может использоваться в элементах BODY и FRAMESET.

onunload = *скрипт* [CT]

Событие onunload происходит, когда агент пользователя удаляет документ из окна или фрейма. Этот атрибут может использоваться в элементах BODY и FRAMESET.

onclick = *скрипт* [CT]

Событие onclick происходит при однократном щелчке кнопки указующего устройства на элементе. Этот атрибут может использоваться с большинством элементов.

ondblclick = *скрипт* [CT]

Событие ondblclick происходит при двойном щелчке клавиши указующего устройства на элементе. Этот атрибут может использоваться с большинством элементов.

onmousedown = *скрипт* [CT]

Событие onmousedown происходит при нажатии кнопки указующего устройства на элементе. Этот атрибут может использоваться с большинством элементов.

onmouseup = *скрипт* [CT]

Событие onmouseup происходит при отпускании кнопки указующего устройства на элементе. Этот атрибут может использоваться с большинством элементов.

onmouseover = *скрипт* [CT]

Событие `onmouseover` происходит при перемещении указующего устройства на элемент. Этот атрибут может использоваться с большинством элементов.

`onmousemove` = *скрипт* [CT]

Событие `onmousemove` происходит при перемещении указующего устройства, когда оно находится на элементе. Этот атрибут может использоваться с большинством элементов.

`onmouseout` = *скрипт* [CT]

Событие `onmouseout` происходит при перемещении указующего устройства за пределы элемента. Этот атрибут может использоваться с большинством элементов.

`onfocus` = *скрипт* [CT]

Событие `onfocus` происходит при получении элементом фокуса с помощью указующего устройства или последовательности перехода. Этот атрибут может использоваться со следующими элементами: `LABEL`, `INPUT`, `SELECT`, `TEXTAREA` и `BUTTON`.

`onblur` = *скрипт* [CT]

Событие `onblur` происходит при переходе фокуса с этого элемента с помощью указующего устройства или последовательности перехода. Оно может использоваться с теми же элементами, что и `onfocus`.

`onkeypress` = *скрипт* [CT]

Событие `onkeypress` происходит при нажатии и отпускании клавиши на элементе. Этот атрибут может использоваться с большинством элементов.

`onkeydown` = *скрипт* [CT]

Событие `onkeydown` происходит при нажатии клавиши на элементе. Этот атрибут может использоваться с большинством элементов.

`onkeyup` = *скрипт* [CT]

Событие `onkeyup` происходит при отпускании клавиши на элементе. Этот атрибут может использоваться с большинством элементов.

`onsubmit` = *скрипт* [CT]

Событие `onsubmit` происходит при отправке формы. Оно используется только в элементе `FORM`.

`onreset` = *скрипт* [CT]

Событие `onreset` происходит при сбросе формы. Оно используется только в элементе `FORM`.

`onselect` = *скрипт* [CT]

Событие `onselect` происходит при выделении пользователем некоторого текста в текстовом поле. Этот атрибут может использоваться с элементами `INPUT` и `TEXTAREA`.

`onchange` = *скрипт* [CT]

Событие `onchange` происходит при потере управляющим элементом фокуса ввода, *если* его значение было изменено с момента получения фокуса. Этот атрибут используется со следующими элементами: `INPUT`, `SELECT` и `TEXTAREA`.

Действие можно связать с определенным рядом событий, происходящих при взаимодействии пользователя с агентом. Значением каждого из перечисленных выше "внутренних событий" является скрипт. Этот скрипт выполняется, если это событие происходит для этого элемента. Синтаксис скрипта зависит от языка скрипта.

Элементы управления, такие как `INPUT`, `SELECT`, `BUTTON`, `TEXTAREA` и `LABEL` реагируют на внутренние события. Если они не отображаются в форме, они могут использоваться для улучшения графического интерфейса документа.

Например, авторы могут включить в документы кнопки, которые не используются для отправки формы, но при нажатии которых происходит некоторое взаимодействие с сервером.

В следующем примере показан возможное поведение управляющего элемента и интерфейса пользователя в зависимости то внутренних событий.

В этом примере userName является обязательным текстовым полем. Если пользователь не заполняет это поле, событие `onblur` вызывает функцию JavaScript для проверки значения поля userName.

```
<INPUT NAME="userName" onblur="validUserName(this.value)">
```

Вот еще один пример JavaScript:

```
<INPUT NAME="num"
    onchange="if (!checkNum(this.value, 1, 10))
        {this.focus();this.select();} else {thanks()}"
    VALUE="0">
```

Вот пример обработчика событий для текстового поля на языке VBScript:

```
    <INPUT name="edit1" size="50">
    <SCRIPT type="text/vbscript">
      Sub edit1_changed()
        If edit1.value = "abc" Then
          button1.enabled = True
        Else
          button1.enabled = False
        End If
      End Sub
    </SCRIPT>
```

Вот тот же пример с использованием Tcl:

```
    <INPUT name="edit1" size="50">
    <SCRIPT type="text/tcl">
      proc edit1_changed {} {
        if {[edit value] == abc} {
          button1 enable 1
        } else {
          button1 enable 0
        }
      }
      edit1 onChange edit1_changed
    </SCRIPT>
```

Вот пример привязки события внутри скрипта на языке JavaScript. Для начала простой обработчик щелчка мыши:

```
<BUTTON type="button" name="mybutton" value="10">
<SCRIPT type="text/javascript">
      function my_onclick() {
         . . .
      }
    document.form.mybutton.onclick = my_onclick
 </SCRIPT>
 </BUTTON>
```

Вот более интересный обработчик окна:

```
<SCRIPT type="text/javascript">
      function my_onload() {
         . . .
      }

      var win = window.open("some/other/URI")
      if (win) win.onload = my_onload
</SCRIPT>
```

На языке Tcl это выглядит следующим образом:

```
 <SCRIPT type="text/tcl">
     proc my_onload {} {
       . . .
     }
     set win [window open "some/other/URI"]
```

```
    if {$win != ""} {
        $win onload my_onload
    }
</SCRIPT>
```

Обратите внимание, что выражение "document.write" или эквивалентные выражения в обработчиках внутренних событий создают и выполняют запись в новый документ, а не изменяют текущий.

18.2.4 Динамическое изменение документов

Скрипты, выполняемые при загрузке документа, могут динамически изменять содержимое документа. Такая возможность зависит от самого языка скрипта (например, выражение "document.write" в объектной модели HTML поддерживается некоторыми производителями).

Динамическое изменение документа может моделироваться следующим образом:

1. Все элементы SCRIPT оцениваются по порядку при загрузке документа.
2. Оцениваются все конструкции скрипта в данном элементе SCRIPT, генерирующие SGML CDATA. Сгенерированный текст вставляется в документ вместо элемента SCRIPT.
3. Сгенерированные CDATA оцениваются повторно.

Документы HTML ограничиваются требованием соответствия HTML DTD до и после обработки элементов SCRIPT.

В следующем примере показано, как скрипты могут динамически изменять документ. Следующий скрипт:

```
<TITLE>Тестовый документ</TITLE>
<SCRIPT type="text/javascript">
    document.write("<p><b>Hello World!<\/b>")
</SCRIPT>
```

имеет тот же эффект, что и разметка HTML:

```
<TITLE>Тестовый документ</TITLE>
<P><B>Hello World!</B>
```

18.3 Разработка документов для агентов пользователей, не поддерживающих скрипты

В следующих разделах обсуждается создание документов для агентов пользователей, не поддерживающих скрипты.

18.3.1 Элемент NOSCRIPT

```
<!ELEMENT NOSCRIPT - - (%block;)+
  -- альтернативное содержимое для представления без скриптов -->
<!ATTLIST NOSCRIPT
  %attrs;                              -- %coreattrs, %i18n, %events --
  >
```

Начальный тег: ***обязателен****, Конечный тег:* ***обязателен***

Элемент NOSCRIPT позволяет авторам определять альтернативное содержимое, когда скрипт не выполняется. Содержимое элемента NOSCRIPT должно генерироваться агентам пользователей, поддерживающими скрипты, только в следующих случаях:

- Агент пользователя сконфигурирован так, чтобы не выполнять скрипты.
- Агент пользователя не поддерживает язык скриптов, используемый элементом SCRIPT.

Агенты пользователей, не поддерживающие клиентские скрипты, должны представлять содержимое этого элемента.

В следующем примере агент пользователя, выполняющий элемент SCRIPT, включит в документ динамически создаваемые данные. Если агент пользователя не поддерживает скрипты, пользователь может загрузить эти данные по ссылке.

```
<SCRIPT type="text/tcl">
```

```
 ...скирпт на языке Tcl для вставки данных...
</SCRIPT>
<NOSCRIPT>
 <P>Доступ к данным <A href="http://someplace.com/data">.</A>
</NOSCRIPT>
```

18.3.2 Как скрыть скрипт от агентов пользователей

Агенты пользователей, не распознающие элемент SCRIPT, могут представить его в виде текста. Некоторые ядра скриптов, включая ядра для языков JavaScript, VBScript и Tcl, позволяют включать выражения скриптов в комментарий SGML. Агенты пользователей, не распознающие элемент SCRIPT, будут игнорировать комментарии, а ядро скрипта обнаружит и выполнит его.

Другим решением этой проблемы является хранение скриптов во внешних документах и ссылка на них с помощью атрибута SRC.

Комментирование скриптов JavaScript

Ядро JavaScript допускает использование строки "<!--" а начале элемента SCRIPT и игнорирует дальнейшие символы, до конца строки. JavaScript интерпретирует символ "//" как начало комментария, продолжающегося до конца текущей строки. Это необходимо, чтобы строка "-->" не разбиралась синтаксическим анализатором JavaScript.

```
<SCRIPT type="text/javascript">
<!--  скрыть содержимое скрипта от старых браузеров
  function square(i) {
    document.write("Вызов передал ", i ," в функцию.","<BR>")
    return i * i
  }
  document.write("Функция вернула ",square(5),".")
// конец скрытого содержимого  -->
</SCRIPT>
```

Комментирование скриптов в VBScript

В VBScript комментарием считается весь текст, начиная с символа одиночной кавычки до конца строки. Это может использоваться для комментирования строки "-->" от VBScript, например:

```
   <SCRIPT type="text/vbscript">
     <!--
       Sub foo()
        ...
       End Sub
     ' -->
    </SCRIPT>
```

Комментирование скриптов в TCL

В Tcl комментарием считается текст от символа "#" до конца строки:

```
<SCRIPT type="text/tcl">
<!--  скрыть содержимое скрипта от старых браузеров
  proc square {i} {
    document write "Вызов передал $i в функцию.<BR>"
    return [expr $i * $i]
  }
  document write "Функция вернула [square 5]."
# конец скрытого содержимого  -->
</SCRIPT>
```

Примечание. *Некоторые браузеры считают концом комментариев первый символ ">". В этом случае Вы можете перенести операнды для операторов отношения и сдвига (например, используя "y < x" вместо "x > y") или использовать определенные в языке скрипта способы обхода символа ">".*

На начало документа
На содержание