Назначение и применение JavaScript

Вступление

Гипертекстовая информационная система состоит из множества информационных узлов, множества гипертекстовых связей, определенных на этих узлах и инструментах манипулирования узлами и связями. Технология World Wide Web — это технология ведения гипертекстовых распределенных систем в Internet, и, следовательно, она должна соответствовать общему определению таких систем. Это означает, что все перечисленные выше компоненты гипертекстовой системы должны быть и в Web.

Web как гипертекстовую систему можно рассматривать с двух точек зрения. Во-первых, как совокупность отображаемых страниц, связанных гипертекстовыми переходами (ссылками — контейнер `<A>`). Во-вторых, как множество элементарных информационных *объектов*, составляющих отображаемые страницы (текст, графика, мобильный код и т.п.). В последнем случае множество гипертекстовых переходов страницы — это такой же информационный фрагмент, как и встроенная в текст картинка.

При втором подходе гипертекстовая сеть определяется на множестве элементарных информационных *объектов* самими HTML-страницами, которые и играют роль гипертекстовых *связей*. Этот подход более продуктивен с точки зрения построения отображаемых страниц "на лету" из готовых компонентов.

При генерации страниц в Web возникает дилемма, связанная с архитектурой "клиент-сервер". Страницы можно генерировать как на стороне клиента, так и на стороне сервера. В **1995** году специалисты компании Netscape создали механизм управления страницами на клиентской стороне, разработав язык программирования JavaScript.

Таким образом, ***JavaScript*** — это язык управления сценариями просмотра гипертекстовых страниц Web на стороне клиента. Если быть более точным, то JavaScript — это не только язык программирования на стороне клиента. Liveware, прародитель JavaScript, является средством подстановок на стороне сервера Netscape. Однако наибольшую популярность JavaScript обеспечило программирование на стороне клиента.

Основная идея JavaScript состоит в возможности изменения значений атрибутов HTML-контейнеров и свойств среды отображения в процессе просмотра HTML-страницы пользователем. При этом перезагрузки страницы не происходит. На практике это выражается в том, что можно, например, изменить цвет фона страницы или интегрированную в документ картинку, открыть новое окно или выдать предупреждение.

Название "JavaScript" является зарегистрированным товарным знаком компании Sun Microsystems. Реализация языка, осуществленная разработчиками Microsoft, официально называется ***JScript***. Версии JScript совместимы (если быть совсем точным, то не до конца) с соответствующими версиями JavaScript, т.е. JavaScript является подмножеством языка JScript. В данный момент JavaScript полностью занимает нишу браузерных языков. На синтаксис JavaScript оказал влияние язык Java, откуда и произошло название JavaScript; как и Java, язык JavaScript является объектным. Однако на этом их связь заканчивается: Java и JavaScript — это разные языки, ни один не является подмножеством другого.

Стандартизация языка была инициирована компанией Netscape и осуществляется ассоциацией ECMA (European Computer Manufacturers Association — Ассоциация

европейских производителей компьютеров). Стандартизированная версия имеет название *ECMAScript* и описывается стандартом ECMA-262 (доступна в сети: на английском, на русском).

Первая версия стандарта (принята в 1997 г.) примерно соответствовала JavaScript 1.1. На данный момент (2008 г) вышла уже третья редакция стандарта (принята в декабре 1999 г), включающая мощные регулярные выражения, улучшенную поддержку строк, новые управляющие конструкции, обработку исключений try/catch, конкретизированное определение ошибок, форматирование при численном выводе и другие изменения. Ведется работа над расширениями (источник) и четвертой редакцией стандарта. Отметим, что не все реализации JavaScript на сегодня полностью соответствуют стандарту ECMA. В рамках данного курса мы во всех случаях будем использовать название JavaScript.

Размещение кода JavaScript на HTML-странице

Главный вопрос любого начинающего программиста: "Как оформить программу и выполнить ее?". Попробуем на него ответить как можно проще, но при этом не забывая обо всех способах применения JavaScript-кода.

Во-первых, исполняет JavaScript-код браузер. В него встроен *интерпретатор* JavaScript. Следовательно, выполнение программы зависит от того, когда и как этот интерпретатор получает управление. Это, в свою очередь, зависит от функционального применения кода. В общем случае можно выделить четыре способа функционального применения JavaScript:

- гипертекстовая ссылка (схема URL);
- обработчик события (в атрибутах, отвечающих событиям);
- подстановка (entity);
- вставка (контейнер `<SCRIPT>`).

Ниже мы рассмотрим их по очереди. В учебниках по JavaScript описание применения JavaScript обычно начинают с контейнера `<SCRIPT>`. Но с точки зрения понимания сути взаимодействия JavaScript и HTML это не совсем правильно, поскольку такой порядок не дает ответа на ключевой вопрос: как JavaScript-код получает управление? Другими словами, каким образом вызывается и исполняется программа, написанная на JavaScript и размещенная в HTML-документе?

В зависимости от профессии автора HTML-страницы и уровня его знакомства с основами программирования возможны несколько вариантов начала освоения JavaScript. Если вы программист классического толка (C, Fortran, Pascal и т.п.), то проще всего начинать с программирования внутри тела документа. Если вы привыкли программировать под Windows, то в этом случае начинайте с программирования обработчиков событий. Если же вы имеете только опыт HTML-разметки или давно не писали программ, то тогда лучше начать с программирования гипертекстовых переходов.

Примечание 1. Все последующие примеры Вы можете проверять на работоспособность в Вашем браузере самостоятельно. Для этого скопируйте текст примера в файл (скажем, `primer.htm`); если текст примера состоит только из JavaScript-кода, то заключите его в тэги `<SCRIPT>` и `</SCRIPT>`. Получившийся файл можно просматривать в браузере.

Примечание 2. В данной вводной лекции примеры даются без разбора деталей всех использованных конструкций — воспринимайте их пока интуитивно. Последующие лекции все прояснят. Опишем лишь два важнейших оператора, встречающихся почти в каждом примере. Оператор `alert(строка)` выводит эту строку на экран в *окне*

предупреждения, пример такого окна изображен на рис. 1.1. Оператор `document.write(строка)` записывает указанную строку в текущий HTML-документ. Например, следующие два фрагмента HTML-документа равносильны:

Простой HTML-документ	Использование `document.write()`

`<HTML>`
`<BODY>`
`<H1>Заголовок</H1>`
`</BODY>`
`</HTML>`
	`<HTML>`
	`<BODY>`
	`<SCRIPT>`
	`document.write('<H1>Заголовок</H1>');`
	`</SCRIPT>`
	`</BODY>`
	`</HTML>`

Способ 1: URL-схема "JavaScript:"

Схема URL (Uniform Resource Locator) — это один из основных элементов Web-технологии. Каждый информационный ресурс в Web имеет свой уникальный URL. URL указывают в атрибуте `HREF` контейнера `A`, в атрибуте `SRC` контейнера `IMG`, в атрибуте `ACTION` контейнера `FORM` и т.п. Все URL подразделяются на *схемы доступа*, которые зависят от протокола доступа к ресурсу, например, для доступа к FTP-архиву применяется схема `ftp`, для доступа к Gopher-архиву — схема `gopher`, для отправки электронной почты — схема `mailto`. Тип схемы определяется по первому компоненту URL, например:

```
http://intuit.ru/directory/page.html
```

В данном случае URL начинается с `http` — это и есть задание схемы доступа (схема `http`).

Основной задачей языка программирования гипертекстовой системы является программирование гипертекстовых переходов. Это означает, что при выборе той или иной гипертекстовой ссылки вызывается программа реализации гипертекстового перехода. В Web-технологии стандартной программой, вызываемой при гипертекстовом переходе, является программа загрузки страницы (т.е. при клике по ссылке загружается страница с указанным URL). JavaScript позволяет поменять стандартную программу на программу пользователя. Для того чтобы отличить стандартный переход по протоколу HTTP от перехода, программируемого на JavaScript, разработчики языка ввели новую схему URL — `JavaScript`:

```
<A HREF="JavaScript:код_программы">...</A>
<FORM ACTION="JavaScript:код_программы" ...> ... </FORM>
```

В данном случае текст "`код_программы`" обозначает программу-обработчик на JavaScript, которая вызывается при выборе гипертекстовой ссылки в первом случае и при отправке данных формы (нажатии кнопки Submit) — во втором. Например, при нажатии на гипертекстовую ссылку "`Кликни здесь`" можно получить окно предупреждения:

```
<A HREF="JavaScript:alert('Внимание!!!');">Кликни здесь</A>
```

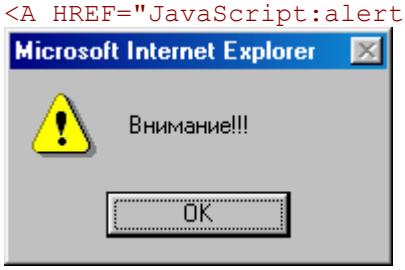

Рис. 1.1. Окно предупреждения

А при нажатии на кнопку типа `submit` в форме можно заполнить текстовое поле этой же формы:

```
<FORM METHOD=post NAME="form1"
  ACTION="JavaScript:form.e.value='Нажали кнопку: Заполнить';void(0);">
<INPUT TYPE=text NAME=e SIZE=30 VALUE=""><BR>
<INPUT TYPE=submit VALUE="Заполнить">
<INPUT TYPE=reset VALUE="Очистить">
</FORM>
```

Пример 1.1. Заполнение поля при нажатии кнопки (html, txt)

В URL можно размещать сложные программы и вызовы функций. Таким образом, при программировании гипертекстового перехода JavaScript-интерпретатор получает управление после того, как пользователь "кликнул" по гипертекстовой ссылке.

Способ 2: обработчики событий

Такие программы, как *обработчики событий*, указываются в атрибутах контейнеров, с которыми эти события связаны. Например, при нажатии на кнопку происходит событие `Click` и соответственно вызывается обработчик этого события `onClick`:

```
<FORM><INPUT TYPE=button VALUE="Кнопка"
     onClick="alert('Вы нажали кнопку');"></FORM>
```

А в момент завершения полной загрузки документа (он связан с контейнером `<BODY>`) происходит событие `Load` и, соответственно, будет вызван обработчик этого события `onLoad`:

```
<BODY onLoad="alert('Приветствуем!');">
...
</BODY>
```

Способ 3: подстановки

Подстановки (entity) поддерживаются только браузером Netscape Navigator 4.0. Они встречаются на Web-страницах довольно редко. Тем не менее это достаточно мощный инструмент генерации HTML-страницы на стороне браузера. Подстановки имеют формат: `&{код_программы};` и используются в качестве значений атрибутов HTML-контейнеров. В следующем примере поле ввода `INPUT` будет иметь, в качестве значения по умолчанию, адрес текущей страницы, а размер поля будет равным количеству символов в этом адресе.

```
<HTML>
<HEAD>
<SCRIPT>
function l()
{
  str = window.location.href;
  return(str.length);
}
</SCRIPT>
</HEAD>
<BODY>
<FORM><INPUT TYPE=text SIZE="&{l()};"
   VALUE="&{window.location.href};">
</FORM>
```

```
</BODY>
</HTML>
```

В случае подстановки JavaScript-интерпретатор получает управление в момент разбора браузером (компонент *парсер*) HTML-документа. Как только парсер встречает конструкцию `&{..};` у атрибута контейнера, он передает управление JavaScript-интерпретатору, который, в свою очередь, после исполнения кода это управление возвращает парсеру. Таким образом, данная операция аналогична подкачке графики на HTML-страницу.

Очевидно, что размещать в заголовке документа генерацию текста страницы бессмысленно — он не будет отображен браузером. Поэтому в заголовок помещают декларации общих переменных и функций, которые будут затем использоваться в теле документа. При этом браузер Netscape Navigator более требовательный, чем Internet Explorer. Если не разместить описание функции в заголовке, то при ее вызове в теле документа можно получить сообщение о том, что данная функция не определена.

В Internet Explorer подстановки не поддерживаются, поэтому пользоваться ими следует аккуратно. Прежде чем выдать браузеру страницу с подстановками, нужно проверить тип этого браузера. Альтернативой подстановкам в Internet Explorer можно считать *динамические свойства* стиля. Например, следующий фрагмент создаст поле ввода, ширина которого в пикселях (`px`) равна количеству символов в адресе страницы, умноженному на 10:

```
<INPUT TYPE=text style="width:expression(10*location.href.length+'px')">
```

Мы не будем подробно останавливаться на этом способе использования JavaScript-кода.

Способ 4: вставка (контейнер <SCRIPT>)

Контейнер `SCRIPT` — это развитие подстановок до возможности генерации текста документа JavaScript-кодом. В этом смысле применение `SCRIPT` аналогично Server Side Includes, т.е. генерации страниц документов на стороне сервера. Однако здесь мы забежали чуть вперед. При разборе документа HTML-парсер передает управление JavaScript-интерпретатору после того, как встретит тег начала контейнера `<SCRIPT>`. Интерпретатор получает на исполнение весь фрагмент кода внутри контейнера `SCRIPT` и возвращает управление HTML-парсеру для обработки текста страницы после тега конца контейнера `</SCRIPT>`.

Помещать JavaScript-код на HTML-странице с помощью контейнера `<SCRIPT>` можно двумя способами. Первый состоит в написании текста кода непосредственно внутри этого контейнера:

```
<SCRIPT>
 a = 5;
</SCRIPT>
```

Второй способ состоит в том, чтобы вынести код JavaScript в отдельный файл, например, `myscript.js` (расширение может быть любым), и затем включить его в HTML-страницу следующим образом:

```
<SCRIPT SRC="myscript.js"></SCRIPT>
```

Этот способ удобен, когда один и тот же скрипт планируется использовать на разных HTML-страницах. Обратите внимание, что при наличии атрибута `SRC` содержимое контейнера `<SCRIPT>` пусто, и это не случайно: согласно спецификации HTML, если

скрипт подключается из внешнего файла, то скрипт, написанный между тэгами <SCRIPT> и </SCRIPT>, если таковой имеется, будет проигнорирован браузером.

Здесь уместно небольшое замечание, которое позволит Вам избежать одной ошибки начинающих программистов. Между тэгами <SCRIPT> и </SCRIPT> не должно встречаться последовательности символов </SCRIPT> в любом контексте. Например, следующий пример работать не будет:

```
<SCRIPT>
alert('</script>');
</SCRIPT>
```

Дело в том, что специфика разбора HTML-документа браузером такова, что он сначала определяет границы скрипта, а потом уже передает его интерпретатору JavaScript. В нашем случае браузер посчитает, что код скрипта завершился на первой же встретившейся ему последовательности символов "</script>", т.е. не на той, на которой было нужно нам. Чтобы пример заработал, достаточно, например, написать alert('<\/script>') (т.к. комбинация "\/" выводит на экран символ "/"), либо разбить строчку на две: alert('</scr'+'ipt>').

Контейнер SCRIPT выполняет две основные функции:

- **размещение кода** внутри HTML-документа;
- **условная генерация** HTML-разметки на стороне браузера.

Первая функция аналогична декларированию переменных и функций, которые потом можно будет использовать в качестве программ переходов, обработчиков событий и подстановок. Вторая — это подстановка результатов исполнения JavaScript-кода в момент загрузки или перезагрузки документа.

Размещение кода внутри HTML-документа

Собственно, особенного разнообразия здесь нет. Код можно разместить либо в заголовке документа (внутри контейнера HEAD) либо в теле документа (внутри контейнера BODY). Последний способ и его особенности будут рассмотрены в разделе "Условная генерация HTML-разметки на стороне браузера". Поэтому обратимся к заголовку документа.

Код в заголовке документа размещается внутри контейнера SCRIPT. В следующем примере мы декларировали функцию time_scroll() в заголовке документа, а потом вызвали ее как обработчик события Load в теге начала контейнера BODY.

```
<HTML>
<HEAD>
<SCRIPT>
function time_scroll()
{
  var d = new Date();
  window.status = d.getHours()
          + ':' + d.getMinutes()
          + ':' + d.getSeconds();
  setTimeout('time_scroll()',1000);
}
</SCRIPT>
</HEAD>
<BODY onLoad="time_scroll()">
<H1>Часы в строке статуса</H1>
</BODY>
```

```
</HTML>
```
Пример 1.2. Часы в поле статуса окна (html, txt)

Функция `time_scroll()` вызывается по окончании полной загрузки документа (обработчиком `onLoad`). Она заносит текущую дату и время (`new Date`) в переменную `d`. Затем записывает текущее время в формате ЧЧ:ММ:СС в `window.status`, тем самым оно будет отображаться в поле статуса окна браузера (подробнее о нем рассказано в лекции "Программируем свойства окна браузера"). Наконец, она откладывает (`setTimeout`) повторный вызов самой себя на 1000 миллисекунд (т.е. 1 секунду). Таким образом, каждую секунду в поле статуса будет отображаться новое время.

Условная генерация HTML-разметки на стороне браузера

Всегда приятно получать с сервера страницу, подстроенную под возможности нашего браузера или, более того, под пользователя. Существует только две возможности генерации таких страниц: на стороне сервера или непосредственно у клиента. JavaScript-код исполняется на стороне клиента (на самом деле, серверы компании Netscape способны исполнять JavaScript-код и на стороне сервера, только в этом случае он носит название LiveWire-код; не путать с LiveConnect), поэтому рассмотрим только генерацию на стороне клиента.

Для генерации HTML-разметки контейнер `SCRIPT` размещают в теле документа, т.е. внутри контейнера `BODY`. Простой пример — встраивание в страницу локального времени:

```
<BODY>
...
<SCRIPT>
d = new Date();
document.write('Момент загрузки страницы: '
+ d.getHours() + ':'
+ d.getMinutes() + ':'
+ d.getSeconds());
</SCRIPT>
...
</BODY>
```
Пример 1.3. Точное время загрузки страницы (html, txt)

Комментарии в HTML и JavaScript

Несколько слов о различных видах комментариев. В программе JavaScript можно оставлять ***комментарии***, которые игнорируются JavaScript-интерпретатором и служат как пояснения для разработчиков. Однострочные комментарии начинаются с символов `//`. Текст начиная с этих символов и до конца строки считается комментарием. Многострочный комментарий заключается между символами `/*` и `*/` и может простираться на несколько строк.

```
<SCRIPT>

 a=5;  // однострочный комментарий

/* Многострочный
   комментарий    */

</SCRIPT>
```

Для скрытия JavaScript-кода от интерпретации старыми браузерами, не поддерживающими JavaScript (у высокого начальства еще встречаются), весь

JavaScript-код между тэгами <SCRIPT> и </SCRIPT> приходится заключать в HTML-комментарии <!-- и -->. Можно предположить, что эти комбинации символов, не являясь полноценными операторами JavaScript, могут быть неверно поняты JavaScript-интерпретатором и порождать ошибки. Однако этого не происходит, так как разработчики языка ввели соглашение: комбинация символов <!-- считается началом однострочного комментария (наряду с //). Со второй комбинацией (-->) такой трюк невозможен (т.к. двойной минус имеет специальное значение в JavaScript), и ее приходится комментировать символами //, что иллюстрирует следующий пример.

```
<SCRIPT>
<!-- Скрываем JavaScript-код от старых браузеров

 a = 5;

// -->
</SCRIPT>
```

Однако в данном курсе мы не будем загромождать примеры такого рода HTML-комментариями, переложив эту обязанность на пользователя. К тому же, все реже можно встретить браузеры, которые вместо выполнения JavaScript-кода выдают его текст в окно браузера.

Указание языка сценария

Контейнер <SCRIPT> имеет необязательный атрибут LANGUAGE, указывающий язык, на котором написан содержащийся внутри контейнера скрипт. Значение атрибута не чувствительно к регистру. Если этот атрибут опущен, то его значением по умолчанию считается "JavaScript". Поэтому все наши примеры можно записывать следующим образом:

```
<SCRIPT LANGUAGE="JavaScript">
...
</SCRIPT>
```

В качестве альтернативы атрибут LANGUAGE может принимать значения "JScript" (упоминавшаяся выше разновидность языка JavaScript, разработанная компанией Microsoft), "VBScript" или "VBS" (оба указывают на язык программирования VBScript, основанный на Visual Basic и тоже являющийся детищем Microsoft; поддерживается преимущественно браузером Internet Explorer) и другие. Кроме того, для JavaScript бывает необходимо указать версию языка, например, LANGUAGE="JavaScript1.2". Потребность в этом может возникнуть, если нужно написать разные участки кода для браузеров, поддерживающих разные версии языка.

Следует также иметь в виду, что в настоящей версии языка HTML (т.е. 4.0 и выше) атрибут LANGUAGE контейнера <SCRIPT> считается устаревшим и нерекомендуемым к использованию (deprecated). Вместо него в контейнере <SCRIPT> рекомендуется использовать атрибут TYPE. Его значениями, также не чувствительными к регистру, могут быть "text/javascript" (значение по умолчанию), "text/vbscript" и другие. Например, все наши примеры можно оформлять так:

```
<SCRIPT TYPE="text/javascript">
...
</SCRIPT>
```

Некоторые старые браузеры не понимают атрибут TYPE, поэтому можно задавать оба атрибута одновременно — LANGUAGE и TYPE. Атрибут TYPE имеет высший приоритет, т.е. если браузер распознает значение TYPE, то значение LANGUAGE игнорируется.

Поскольку в любом случае значение по умолчанию соответствует языку JavaScript, в наших примерах эти атрибуты будут опускаться.

Регистр символов

Как Вы, наверное, знаете, язык HTML является ***регистро-независимым***. Вследствие этого, контейнер `<SCRIPT>` можно писать как `<script>`, его атрибуты — как `Type`, `LANGuage` и `src`, значение атрибутов, указывающих язык, — как "`JavaSCRIPT`" и "`TEXT/JavaScript`". Разумеется, значение атрибута `SRC`, т.е. имя файла, следует писать точно так, как файл назван в операционной системе.

Напротив, язык же JavaScript — ***регистро-зависимый***. Это означает, что все переменные, функции, ключевые слова и т.п. должны набираться в том же регистре, в каком они заданы в языке или в программе пользователя. Например, если Вы объявили переменную `var myText='Привет'`, то в дальнейшем ее можно использовать только как `myText`, но не `MyText`. В этом кроется частая ошибка, которую допускают программисты на JavaScript. Она усугубляется еще и тем, что JavaScript не требует явно декларировать переменные, и встретив `MyText`, интерпретатор может решить, что это новая (но не объявленная) переменная.

Это касается и всех встроенных объектов, свойств и методов языка. Например, объектом является `document`. Вызов `document.write()` нельзя записать как `Document.write()` или `document.Write()`. К свойству объекта `document`, задающему цвет фона Web-страницы, можно обратиться только как `document.bgColor`, а метод этого же объекта, выдающий элемент с заданным идентификатором "`id5`", можно вызвать только как `document.getElementById("id5")`.

Названия событий, такие как `Click` (щелчок мышью), `DblClick` (двойной щелчок мышью), `Load` (окончание загрузки документа) и т.п. сами по себе не являются элементами синтаксиса. Обработчики же соответствующих событий могут появляться в двух контекстах:

- внутри кода JavaScript — в этом случае регистр имеет значение. Например, чтобы при возникновении события `Load` вызывалась функция `myFunction`, мы должны написать: `window.onload = myFunction`. Названия обработчиков событий `onload`, `onmouseover` и т.п. в таком контексте должны быть написаны маленькими буквами;
- как атрибут какого-либо HTML-контейнера — в этом случае регистр не важен. Например, чтобы обработчик события `onLoad` вызывал функцию `myFunction`, мы можем написать в HTML-исходнике: `<BODY onLoad="myFunction()">` либо `<BODY ONLOAD="myFunction()">`.