Вадим Дунаев

Web программирование для всех

bhv

УДК 681.3.068
ББК 32.973.26-018.1
Д83

Дунаев В. В.

Д83 Web-программирование для всех. — СПб.: БХВ-Петербург, 2008. — 560 с.: ил.

ISBN 978-5-9775-0197-2

В книге в доступной форме в виде диалогов между Простаком, Занудой и Профессором показано, что такое Web-программирование и в каких случаях его необходимо применять. Изложены основы языков JavaScript и PHP. Описано создание клиентских сценариев на JavaScript, начиная с создания новых окон браузера и заканчивая применением технологии AJAX. Рассмотрены серверные сценарии на языке PHP для работы с файлами и папками, взаимодействия с базами данных и многие другие. Материал сопровождается простыми практическими примерами, которые поддерживаются всеми современными браузерами, такими как Internet Explorer, Mozilla Firefox и Opera.

Для Web-разработчиков

УДК 681.3.06
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Леонид Кочин*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.04.08.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 45,15.
Тираж 2500 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.002108.02.07 от 28.02.2007 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0197-2

Оглавление

Глава 3

Основы языка JavaScript

В данной главе мы рассмотрим основные понятия и синтаксис языка JavaScript. На этом языке обычно пишут так называемые клиентские сценарии, выполняемые Web-браузером на компьютере пользователя. Кроме того, сценарии на JavaScript пригодны для выполнения Web-сервером, а также встроенным интерпретатором Windows Scripting Host на локальном компьютере.

3.1. Немного истории о версиях и стандартах

Современное состояние чего-либо, и, в частности, языков и технологий программирования, легче понять в исторической ретроспективе. Сначала HTML-документы содержали лишь команды (теги) форматирования текста, ссылки на другие документы и вставки графических изображений. Нередко мы сначала ограничиваем себя слишком скромными целями, то ли от неуверенности в собственных силах, то ли от непонимания перспективы. При этом мы говорим: хорошо бы реализовать хотя бы это немногое, а там посмотрим. И в самом деле, на первых порах этого немногого оказалось вполне достаточно для публикации пользователями Всемирной сети своих творений, содержащих самое главное — тексты с иллюстрациями и ссылки на другие документы того же рода. Таким образом, возможность практически неограниченного распространения чего бы то ни было была реализована.

Вспомните недавнее прошлое, когда была лишь одна возможность публикации, — напечатать сначала на механической пишущей машинке, а затем попытаться разместить свой текст на бумаге, выпускаемой каким-либо официальным издательством. На издание небольшой статьи уходило несколько месяцев. Однако, как почти всегда и бывает, освоенное не только порождает удовлетворенность или, напротив, разочарование достигнутым состоянием, но и вызывает новые безудержные желания (вспомните сказку о рыбаке

и золотой рыбке). HTML (язык разметки гипертекста) был прост и доступен даже тем, кто не имел никакого опыта программирования и, тем более, создания программных проектов, т. е. программ, предназначенных для более или менее широкой потребительской аудитории. Простота и доступность HTML сыграли свою главную роль — обеспечить доступ к всемирным информационным ресурсам всех желающих. Лиха беда начало!

Вскоре выяснилось, что добавление к уже существующему еще чего-то простого и в то же время мощного может раскрыть ворота в мир почти неограниченного могущества. Где же была предполагаемая точка атаки и последующей экспансии? Первоначальные HTML-документы содержали лишь команды представления на экране некоторых информационных ресурсов первостепенной важности: текстов, графики, а затем звуков и видео. Тогда можно было лишь надлежащим образом разместить информацию на страницах, но управлять ею или обрабатывать динамически (т. е. во время экспозиции) было невозможно. Решить эти и многие другие задачи можно было с помощью языков программирования, отличных от HTML и похожих на уже известные, посредством которых пишут программы для обычных компьютерных приложений. Таким образом, необходимо было обеспечить вставку в HTML-код фрагментов на другом языке программирования.

В 1995 г. компания Netscape впервые предложила язык LiveScript для своего браузера Netscape Navigator 2.0. Главное назначение этого языка было весьма скромным — проверка и обработка данных, введенных пользователем в форму на Web-странице, чтобы исключить передачу на сервер заведомо неправильных или пустых значений. Вскоре он был переименован: префикс Live был заменен на Java. Видимо, это произошло по коммерческим соображениям. По крайней мере, JavaScript не является какой бы то ни было упрощенной версией известного мощного, но совсем другого языка Java. Итак, Netscape ввела в обиход язык программирования для своего браузера. Однако в это время уже существовали и развивались браузеры других конкурирующих фирм. Естественно, они не могли не сделать чего-то подобного.

Вскоре за Netscape корпорация Microsoft представила свою редакцию JavaScript под немного отличающимся названием JScript для своего браузера Internet Explorer 3.0. Название JScript было выбрано, видимо, чтобы избежать проблем с авторскими правами и лицензированием. Тем не менее, идея родилась и воплотилась. Так началось развитие скриптового языка (т. е. языка сценариев), который теперь называют JavaScript, а название JScript употребляют, когда хотят подчеркнуть, что имеется в виду язык JavaScript с множеством фирменных расширений, поддерживаемых преимущественно браузерами Microsoft Internet Explorer. Microsoft немало потрудилась, чтобы обеспечить программисту как можно больше возможностей и удобств программирования.

Поэтому браузеры других фирм были вынуждены поддерживать в порядке исключения кое-что из созданного Microsoft. И в самом деле, нельзя же полностью игнорировать массу программного кода, написанного исключительно для чрезвычайно широко распространенного браузера Microsoft Internet Explorer. Заметим попутно, что браузер Opera стал поддерживать JavaScript, начиная с версии 3.0.

Со временем все основные производители браузеров создали собственные расширения языка JavaScript, позволяющие решать существенно более сложные и интересные задачи, чем первоначально поставленные. Этапы развития JavaScript фиксировались в виде его версий, которые были тесно связаны с типами и версиями браузеров. Действительно, любой язык жив, пока существует система, способная его интерпретировать. Такой системой для языков программирования является компилятор или интерпретатор. Сейчас мы не будем выяснять, в чем разница между ними. Достаточно лишь указать, что скриптовые языки воспринимаются интерпретаторами, а интерпретатор JavaScript встроен в Web-браузер, который в принципе может выполнять скрипты (сценарии). Современные браузеры это могут делать, но пока с некоторыми различиями.

В табл. 3.1 приведена информация о поддержке JavaScript некоторыми наиболее популярными браузерами.

Таблица 3.1. *Поддержка JavaScript браузерами*

Браузеры	**Версии JavaScript**
Netscape 2.*x*	JavaScript 1.0
Netscape 3.*x*	JavaScript 1.1
Netscape 4.0 — 4.05	JavaScript 1.2
Netscape 4.06 — 4.08, 4.5*x*, 4.6*x*, 4.7*x*	JavaScript 1.3
Netscape 6.*x*, 7.*x*	JavaScript 1.5
Mozilla (варианты)	JavaScript 1.5
Internet Explorer 3.0	JScript 1.0
Internet Explorer 4.0	JScript 3.0
Internet Explorer 5.0	JScript 5.0
Internet Explorer 5.5	JScript 5.5
Internet Explorer 6.0, 7.0	JScript 5.6

Нетрудно заметить, что версии и редакции (клоны) языка JavaScript плотно связаны с поддерживающими их браузерами различных фирм. Браузеры

содержали в себе встроенные интерпретаторы этого языка, а их разработчики могли видоизменять эти языки и расширять их как угодно. Однако произвол такого творчества, разумеется, ограничен. Всемирная паутина по своему замыслу должна предоставлять пользователю информационные ресурсы инвариантно относительно браузеров. Поэтому творцы языка должны были, так или иначе, стремиться к некоторому "общему знаменателю". Тем не менее, разночтение все же возникло и породило весьма запутанную проблему совместимости программных кодов (сценариев), написанных для различных браузеров.

Разумеется, синтаксис и основные объекты языка остаются незыблемыми, поддерживаемыми всеми современными браузерами, которые в принципе способны выполнять сценарии для Web-страниц. Однако в языке предусмотрены еще и объекты браузера и документа, состав и/или реализация которых варьируют от версии к версии, а также зависят от типа браузера. Именно эти объекты и приводят к несовместимости браузеров различных производителей.

Сначала Netscape и Microsoft использовали одинаковые стратегии развития своих браузеров. А именно в конкурентной борьбе они создавали свои фирменные расширения первоначальной версии языка JavaScript, которые, к сожалению, не были совместимыми. На этом пути были созданы браузеры Netscape Navigator 4.*x* и Microsoft Internet Explorer 4.*x*. Между JavaScript от Netscape и Jscript от Microsoft было много общего, но имелись и существенные различия, хотя и Netscape, и Microsoft вместе с другими крупнейшими производителями программного обеспечения участвовали в работе организации World Wide Web Consortium (W3C, Консорциум Всемирной паутины), цель которой — выработка рекомендаций и стандартов для Web. Казалось бы, они вполне могли бы договориться об "общем знаменателе", но этого не произошло.

В дальнейшем стратегии Netscape и Microsoft существенно разошлись. Компания Netscape в своих браузерах версий 6.*x* и 7.*x* отказалась от создания, поддержки в полной мере уже существующих и наращивания фирменных дополнительных возможностей JavaScript в пользу приверженности стандартам, при этом она также отказалась от принципа сохранения обратной совместимости своих новых продуктов. Иначе говоря, в новых браузерах Netscape не поддерживались в полной мере сценарии, написанные для предыдущих версий. И это в условиях, когда уже существовало огромное количество кода, написанного для прежних версий браузеров Netscape! В результате, что нетрудно было предвидеть, линейка браузеров Netscape Navigator 4.*x* стала терять популярность среди пользователей и в настоящее время почти исчезла. Вместе с тем, в 1998 г. Netscape представила исходный код браузера Mozilla. Это был важный этап реализации новой стратегии, часть которой

уже воплотилась в браузерах Netscape 6*x*, 7*x*. Название "Mozilla" сначала использовалось в качестве внутреннего имени программного кода браузеров компании Netscape. В настоящее время оно относится к браузеру как продукту, платформе и организации. Mozilla — это одновременно наименование и кроссплатформенного браузера с открытым кодом, и платформы — каркаса, на базе которого разработаны как сам браузер, так и другие кроссплатформенные приложения, и организации — независимой группы разработчиков, занимающейся поддержкой и усовершенствованием данных программных продуктов. Открытый программный код Mozilla теперь может быть использован в качестве основы (с соблюдением определенных лицензионных соглашений) для создания любых собственных браузеров теми, кто возьмется за это дело. Так например, Mozilla и Mozilla Firefox — конкретные браузеры, созданные на базе одного и того же программного кода Mozilla, но являющиеся разными продуктами, поскольку имеют некоторые отличия.

Microsoft, напротив, продолжила создавать расширения языка, дающие опытному разработчику большие возможности, а неопытному — автоматическое исправление синтаксических ошибок и отклонений от соблюдения стандартов. Так что для Internet Explorer можно позволить себе некоторую небрежность в написании программных кодов: браузер многое исправит сам. При этом, что очень важно, Microsoft сохранила для своих браузеров Internet Explorer 5.*x*, 6.0 и 7.0 обратную совместимость и в то же время обеспечила поддержку современных стандартов, хотя и не на 100%. Нередко можно услышать, что хуже всех соответствует стандартам браузер Microsoft Internet Explorer. Это, на мой взгляд, неверно. Internet Explorer поддерживает стандарты примерно так же, как и его современные конкуренты, но в отличие от них он сохраняет совместимость со своими более ранними версиями, а также поддерживает множество других, мощных и полезных возможностей, которые в стандарты не вписываются. Многие разработчики используют расширения, которые функционируют неверно или совсем не работают в других браузерах. Однако это совсем не доказывает, что Internet Explorer далек от стандартов, к которым якобы близки другие браузеры. Да, он выполняет коды, написанные вне стандартов, но это его дополнительные возможности, которые никому не навязываются насильно. Если вы хотите создавать Web-документы, инвариантные относительно всех современных браузеров, то пишите тексты программ с соблюдением стандартов, не прибегая к дополнительным фирменным возможностям браузеров. А при разработке приложений исключительно для пользователей Internet Explorer (например, для интрасети) ничто не мешает вам применить его фирменные расширения языка JScript.

Теперь немного о собственно стандартах. Рано или поздно, во избежание кризиса "вавилонского столпотворения", причиной которого было, как хорошо

известно, разноязычие строителей, приходится задуматься о соответствии пестрого разнообразия языков каким-нибудь общим рамкам.

Чтобы создать основу для обеспечения межбраузерной совместимости в отношении языка JavaScript, была выработана стандартная спецификация ECMAScript (ECMA-262) для языка сценариев, встраиваемых в Web-страницы. В настоящее время известны три редакции (edition) данного стандарта. Все современные браузеры теперь объявляют полную поддержку стандарта ECMA-262 Edition 3, но некоторые из них, как уже отмечалось ранее, имеют и дополнительные возможности (расширения). Больше всех фирменных (нестандартных) возможностей предоставляет, пожалуй, браузер Microsoft Internet Explorer 5.*x*, но и другие делают что-то в этом направлении. Так что разработчикам Web-приложений следует быть внимательным и готовым к тестированию своей продукции на нескольких наиболее популярных браузерах. Соответствие стандартам версий JavaScript (Netscape) и JScript (Microsoft) представлено в табл. 3.2.

Таблица 3.2. *Соответствие версий языка стандартам*

Netscape	Microsoft	Стандарт	Примечания
JavaScript 1.0 (Netscape 2.0), JavaScript 1.2 (Netscape 4.0 — 4.05)	JScript 1.0 (Internet Explorer 3.0)	Приблизительное соответствие ECMA-262 Edition 1	Много исключений, имеются дополнения
JavaScript 1.3 (Netscape 4.06)	JScript 3.0 (Internet Explorer 4.0)	Строгое соответствие ECMA-262 Edition 1	Имеются дополнения
JavaScript 1.5 (Netscape 6*x*, 7*x*, Mozilla)	JScript 5.5, 5.6 (Internet Explorer 5.5, 5.6)	Строгое соответствие ECMA-262 Edition 3	То же

В данной главе мы рассмотрим те конструкции языка JavaScript, которые инвариантны относительно браузеров — базовые определения и встроенные объекты, избегая, где только это возможно, обращения к объектам браузера и загруженного в него документа.

3.2. Общая характеристика языка

В данном разделе приводятся наиболее общие сведения о JavaScript, которые могут оказаться полезными для тех, кто уже имеет опыт работы с каким-нибудь другим языком. Новички при первом чтении книги могут опустить этот материал.

Отличительные особенности JavaScript:

- Исходный программный код интерпретируется.
- Операторы разделяются точкой с запятой. Вместе с тем, за завершенным (т. е. полным) оператором, расположенным в одной строке, разделитель можно не указывать: в этом случае переход на другую строку работает так же, как и точка с запятой.
- Язык регистрозависимый. Новички об этом нередко забывают, возможно, потому, что HTML, с которым JavaScript обычно используется совместно, не зависит от регистра (имена тегов и атрибутов HTML можно писать как строчными, так и прописными буквами).
- Внутристрочный комментарий (неинтерпретируемый, игнорируемый текст) выделяется символом //, а многострочный — парой символов /* и */.
- К данным применяется слабый (динамический) контроль типов. В операторах с разнотипными данными последние автоматически приводятся к требуемому типу. Типы данных могут быть примитивными и составными. Примитивные типы содержат простые однородные значения, такие данные можно передавать функциям в качестве параметров по значению, а не по ссылке. Составные типы содержат разнородные данные (в том числе и составные), их передают функциям только по ссылке.
- Язык объектно-ориентированный, однако он основан на прототипах, а не на классах. Имеются четыре типа объектов: встроенные объекты, объекты браузера (интерпретирующей системы), объекты документа и объекты пользователя (программиста).
- Ввод-вывод в основном ограничен взаимодействием с документами и пользователями. По умолчанию предполагается, что доступ к локальной файловой системе запрещен. Однако браузеры могут предоставлять специальные объекты, с помощью которых обеспечивается работа с файловой системой пользователя, хотя и с выдачей предупреждений об опасности выполнения файловых операций.

3.3. Как создавать и отлаживать сценарии на JavaScript

Сценарии (программы) на языке JavaScript взаимодействуют с объектами HTML-документа (Web-страницы). Для этого они собственно и создаются. Чтобы реализовать такое взаимодействие, их необходимо, прежде всего, внедрить (вставить) в текущий HTML-документ.

3.3.1. Вставка сценариев в HTML-документ

Сценарии можно вставить в HTML-документ несколькими способами, но чаще всего их размещают внутри контейнерного тега <SCRIPT>, т. е. между дескрипторами <SCRIPT> и </SCRIPT>. Сам контейнер <SCRIPT> находится в HTML-документе. Программный код сценария можно писать непосредственно в HTML-документе, а также в отдельных текстовых файлах, которые вызываются из него. Проще всего размещать сценарий непосредственно в HTML-документе, именно так обычно и поступают. Мы рассмотрим оба варианта.

Встречая тег <SCRIPT> в загруженном HTML-документе, браузер "понимает", что за ним начинается код сценария. Заключительный тег </SCRIPT> указывает браузеру, что код сценария закончился. Все, что находится вне этих тегов, браузер воспринимает как HTML-код. Контейнер <SCRIPT> может располагаться в любом месте HTML-документа, и даже не один раз. От его расположения в теле HTML-документа иногда может зависеть функционирование всей Web-страницы, но об этом будет сказано позже.

Контейнерный тег <SCRIPT>, объемлющий код сценария, может содержать следующие атрибуты:

- LANGUAGE — указывает язык сценария; возможные значения:
  - "JavaScript", "JScript". Однако второе значение правильно воспринимается только браузером Internet Explorer. Для сценариев, рассчитанных на различные браузеры, следует указывать LANGUAGE = "JavaScript" или "javascript";
  - "VBScript", "VBS". Эти значения воспринимаются только браузером Internet Explorer.

Пример

```
<SCRIPT TYPE = "javascript">
 ...  // код сценария
</SCRIPT>
```

ЗАМЕЧАНИЕ

Если атрибут LANGUAGE не указан, то Internet Explorer подразумевает JScript. Вместе с тем, любая опечатка в значении атрибута LANGUAGE может привести к игнорированию браузером всего содержимого тега <SCRIPT>.

Атрибут LANGUAGE не соответствует стандарту W3C, хотя он широко используется на практике. Вместо него рекомендуется применять атрибут TYPE (см. далее).

- `TYPE` – указывает MIME-тип используемого языка. Для JavaScript следует задавать значение `"text/javascript"`.

Пример

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

Замечание

В теге `<SCRIPT>` возможно указание одновременно и атрибута `LANGUAGE`, и `TYPE`. Однако настоятельно рекомендуется указывать атрибут `TYPE`.

- `SRC` — указывает имя или URL-адрес текстового файла, содержащего код сценария. Этот атрибут необходим в том случае, если сценарий расположен не непосредственно в HTML-документе, а в отдельном файле. Расширение файла со сценарием может быть каким угодно, но обычно используют js:

```
<SCRIPT LANGUAGE = "JScript" SRC = "myscripts.js"></SCRIPT>.
```

Замечание

Если сценарий располагается в отдельном файле, то в нем, разумеется, теги `<SCRIPT>` и `</SCRIPT>` не пишут.

Сценарий, загруженный из внешнего файла, можно представить себе просто как его вставку в HTML-документ.

Внимание!

В примерах, приводимых в настоящей книге, мы ради краткости не будем указывать атрибуты `LANGUAGE` и `TYPE`.

Старые браузеры, появившиеся раньше JavaScript, игнорируют теги `<SCRIPT>` и `</SCRIPT>`, а все, что находится между ними, интерпретируют как текстовое содержимое HTML-документа. Результат при этом может быть самым неожиданным. Чтобы уменьшить вероятность отображения кода сценария в окне старого браузера, следует заключить его в дескрипторы комментария `<!--` и `-->`. Новые браузеры, поддерживающие сценарии, будут опускать эти символы, выполняя код сценария, а старые браузеры (не понимающие сценарии), наоборот, проигнорируют код сценария. Пример такого подхода приведен в листинге 3.1.

Листинг 3.1

```
<SCRIPT LANGUAGE = "JavaScript">
 <!--
  // код сценария
 //-->
</SCRIPT>
```

Обратите внимание на заключительные символы тега комментария, перед которыми стоят две косые черты. При их отсутствии браузер будет пытаться интерпретировать символы `-->` как окончание комментария HTML, а при наличии он просто их проигнорирует.

В браузерах, потенциально поддерживающих сценарии, эту функцию можно отключить. Кроме того, существуют браузеры, принципиально не поддерживающие сценарии. В данной ситуации желательно хотя бы вывести сообщение о том, что произошло. С этой целью в HTML-документе используют контейнерный тег `<NOSCRIPT>`, в котором размещают, например, текст предупреждающего сообщения. Все браузеры, поддерживающие сценарии, проигнорируют содержимое тегов `<NOSCRIPT>` кроме тех случаев, когда поддержка сценариев отключена. Браузеры, принципиально не поддерживающие сценарии, напротив, содержимое тега `<SCRIPT>` (если оно заключено в теги `<!--` и `//-->`) опустят, а тега `<NOSCRIPT>` — отобразят. Пример приведен в листинге 3.2.

Листинг 3.2

```
<HTML>
<HEAD>
<TITLE>Пример применения тега NOSCRIPT</TITLE>
</HEAD>
<SCRIPT TYPE="text/JavaScript">
<!--
   alert("Поддержка JavaScript включена");
//-->
</SCRIPT>
<NOSCRIPT>
   <H2>Ваш браузер не поддерживает JavaScript
   или его поддержка отключена</H2>
</NOSCRIPT>
</HTML>
```

Здесь был использован метод `alert(сообщение)` для вывода сообщений в диалоговом окне. Подробнее об этом и других методах вывода сообщений будет рассказано в *разд. 3.4*.

> ***Внимание!***
>
> В примерах, приводимых в настоящей книге, мы не будем указывать символы комментария для адаптации к старым браузерам из соображений экономии времени и места.

Как уже отмечалось, в отдельных файлах обычно размещают библиотеки функций (определения функций), а также сценарии, использующиеся в нескольких HTML-документах одного или нескольких сайтов. Пример сценария из нескольких разделов иллюстрирует листинг 3.3.

Листинг 3.3

```
<HTML>
...
 <SCRIPT>
  function myfunc(){
   ...
  }
 </SCRIPT>
 <SCRIPT SRC = "myprog1.js"></SCRIPT>
 <SCRIPT SRC = "myprog2.js"></SCRIPT>
...
</HTML>
```

Здесь в HTML-документе расположены три раздела (секции) сценария, два из которых загружаются из отдельных файлов. В первом разделе сценария дано определение некоторой функции.

Сценарий можно также писать в виде строки операторов, разделенных точкой с запятой. Такая строка, взятая в кавычки, служит в качестве значения атрибута-события: `<IMG SRC="mypicture.gif" ONCLICK = "alert('Привет!')"`.

Вызовы функций и их определения могут следовать в произвольном порядке, только если они расположены в одном и том же контейнере `<SCRIPT>`. Если вы размещаете их в разных контейнерах `<SCRIPT>`, то необходимо, чтобы определение функции предшествовало ее вызову. Аналогично, если определения функций находятся в отдельном файле, то его необходимо загрузить

в HTML-документ раньше вызовов этих функций. В листинге 3.4 приведены примеры правильного и неправильного расположения сценариев в HTML-документе.

Листинг 3.4

Правильно

```
<HTML>
 ...
 <SCRIPT>
  function myfunc(){
   ...
  }
 </SCRIPT>
 ...
 <SCRIPT>
  ...
  myfunc()
  ...
 </SCRIPT>
 ...
</HTML>
```

Неправильно

```
<HTML>
 ...
 <SCRIPT>
  ...
  myfunc()
  ...
 </SCRIPT>
 ...
 <SCRIPT>
  function myfunc(){
   ...
  }
 </SCRIPT>
 ...
</HTML>
```

При попытке загрузить и выполнить неправильный вариант HTML-кода появится диалоговое окно с сообщением "Ошибка: Предполагается наличие объекта". Браузер интерпретирует теги HTML последовательно. Так, встретив выражение вызова функции `myfunc()` в одном контейнере `<SCRIPT>`, браузер еще не имеет в памяти определения этого объекта (функции), расположенного в другом контейнере `<SCRIPT>`. Если же определение функции и ее вызов находятся в одном и том же контейнере `<SCRIPT>`, то его содержимое сначала загружается в память, а затем анализируется. Когда интерпретатор встречает вызов функции, то он ищет ее определение в памяти и в случае удачи выполняет код этой функции. Следующие два варианта, приведенные в листинге 3.5, являются правильными, потому что определение функции и ее вызов находятся в одной секции сценария (в одном контейнере `<SCRIPT>`).

Если необходимо, чтобы сценарий оказался в браузере прежде, чем загрузятся элементы HTML-документа, то его следует расположить в верхней части HTML-кода, а еще лучше в контейнере `<HEAD>` (в заголовке документа).

Листинг 3.5

Вариант 1

```
<HTML>
 ...
 <SCRIPT>
  ...
  function myfunc(){
   ...
  }
  ...
  myfunc()
  ...
 </SCRIPT>
 ...
</HTML>
```

Вариант 2

```
<HTML>
 ...
 <SCRIPT>
  ...
  myfunc()
  ...
  function myfunc(){
   ...
  }
  ...
 </SCRIPT>
 ...
</HTML>
```

Если требуется, чтобы сценарий загрузился после всех элементов HTML-документа, то возможны следующие два варианта.

1. Расположить сценарий в конце HTML-документа.
2. Указать атрибут-событие `ONLOAD` в контейнерном теге `<BODY>`, который задает основную часть HTML-документа. В данном случае значением атрибута `ONLOAD` обычно является строка, содержащая имя функции. Определение этой функции, как правило, содержится в контейнере `<SCRIPT>`, размещенном в контейнере заголовка HTML-документа `<HEAD>`. Обратите внимание, что при использовании атрибута-события `ONLOAD` в теге `<BODY>` обработчик этого события выполняется по завершении загрузки всех элементов, определенных в контейнере `<BODY>`, а не в процессе их загрузки. Пример такого подхода иллюстрирует листинг 3.6.

Листинг 3.6

```
<HTML>
 <HEAD>
  <SCRIPT>
   function myfunc(){
    ...
   }
  </SCRIPT>
```

```
    </HEAD>
    ...
    <BODY ONLOAD =" myfunc()">
     <! Теги тела документа >
    </BODY>
    ...
   </HTML>
```

3.3.2. Подготовка, запуск и отладка сценариев

В данной книге мы не будем рассматривать мощные и удобные для пользователя редакторы визуальной разработки Web-страниц и сайтов, такие как Dreamweaver, FrontPage др. Как HTML-документы, так и сценарии для них можно разрабатывать с помощью обыкновенного текстового редактора, например, Блокнота Windows.

Откройте какой-нибудь текстовый редактор (например, Блокнот Windows) и создайте в нем заготовку файла, который вы будете потом редактировать, вводя экспериментальные выражения или даже целые программы. Введите в рабочее поле редактора текст, приведенный в листинге 3.7.

Листинг 3.7

```
<HTML>
 <HEAD><TITLE>Примеры</TITLE></HEAD>
 <SCRIPT>

 </SCRIPT>
</HTML>
```

По существу, это HTML-документ, определяющий пустую Web-страницу. Сохраните файл на диске, например, под именем proba.htm. Расширение имени файла должно быть htm или html, поскольку мы хотим, чтобы он исполнялся в Web-браузере. В этом файле мы написали теги языка HTML. Выражения на языке JavaScript следует записывать между тегами `<SCRIPT>` и `</SCRIPT>`. При изучении данной главы записывайте в каждой строке не более одного выражения на языке JavaScript. Заканчивайте строку нажатием клавиши <Enter>, чтобы перейти к новой строке. Если в одной строке потребуется написать несколько выражений, то их следует отделять друг от друга точкой с запятой. В листинге 3.8 приведен пример простейшей программы.

Листинг 3.8. Пример программы с использованием сценария

```
<HTML>
 <HEAD><TITLE>Примеры</TITLE></HEAD>
 <SCRIPT>
  x = 5;
  y = x + 4;
  alert(y);
 </SCRIPT>
</HTML>
```

При выполнении учебных примеров обычно требуется отобразить на экране окончательные или промежуточные данные. Это можно реализовать методом `alert()`, указав в круглых скобках то, что требуется вывести на экран. В приведенном листинге 3.8 метод `alert(y)` выведет на экран диалоговое окно, в котором появится значение переменной `y` (в данном случае — число 9), как показано на рис. 3.1. Если не вставлять строку `alert(y)`, то программа выполняется без отображения результата.

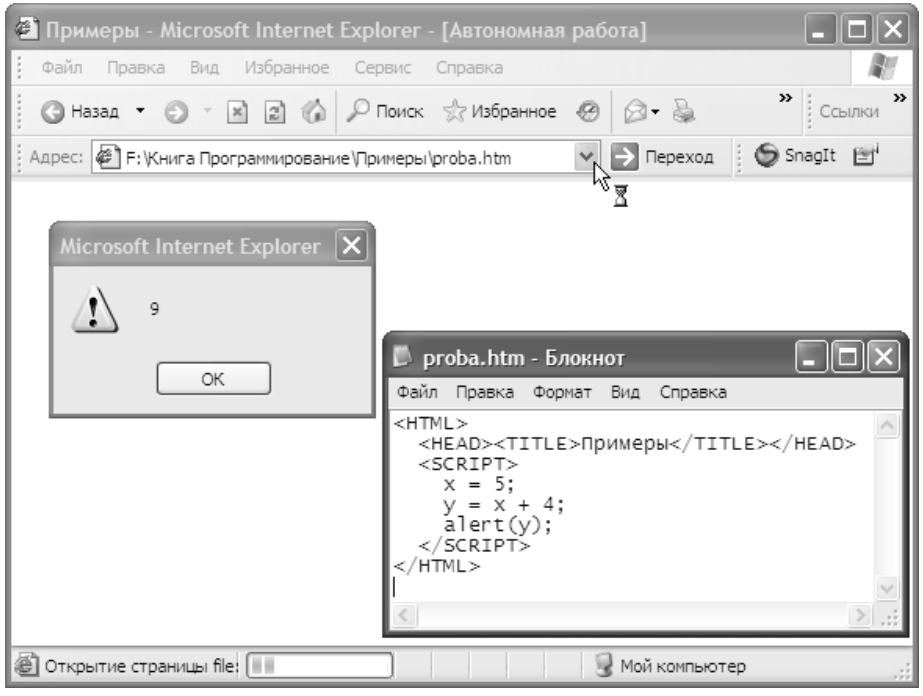

Рис. 3.1. HTML-код в текстовом редакторе и результат его выполнения браузером