**УВАЖАЕМЫЕ СТУДЕНТЫ! Изучите теоретические сведения к лабораторной работе, выполните пример и задание согласно вашему варианту.**

**Результаты работы, фотоотчет, предоставить преподавателю на e-mail:** igor-gricenko-95@mail.ru **в тчении ТРЕХ дней**

**Требования к отчету:**

**Отчет предоставляется преподавателю в электронном варианте и должен содержать:**

- **название работы, постановку цели, вывод;**
- **ответы на контрольные вопросы, указанные преподавателем.**

**При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)132-63-42,**

***ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).***

# Лабораторная работа 7

## Тема: Создание простых сценариев на языке JavaScript

## 1. ЦЕЛЬ РАБОТЫ

Целью работы является приобретение навыков создания и тестирования (исследования) сценариев на языке JavaScript. Рассматриваются проблемы добавления динамики в документы, а также обеспечения правильной реакции на действия пользователя, позволяющей в удобном для восприятия виде предоставить необходимый минимум информации о возможностях взаимодействия.

## 2. СВЕДЕНИЯ ИЗ ТЕОРИИ

### 2.1. Язык JavaScript

JavaScript появился в 1995 году, когда язык Java был уже достаточно широко известен. К тому времени, однако, существовал прототип языка JavaScript — язык описания скриптов (сценариев) LiveScript, встроенный в браузер Netscape Navigator 2.0. Впоследствии компания Netscape отказалась от названия LiveScript и в сотрудничестве с компанией Sun Microsystems, создавшей язык программирования Java, начала разработку нового языка, получившего название JavaScript [6,7].

Несмотря на некоторое сходство в синтаксисе и методах создания объектов, языки Java и JavaScript мало похожи. Определение языка JavaScript как "облегченной версии" языка Java является неточным. Многие утверждают, что язык JavaScript – это язык *описания сценариев,* а не язык программирования. Однако описание сценариев и программирование тесно связаны между собой, несмотря на то, что скрипты и программы применяются для различных целей. Создание скриптов часто рассматривают как "упрощенное программирование", т.е. средство для решения простых задач, хотя фактически именно скрипты являются основой многих программных продуктов, работающих в Internet. Подобно программам на языке Java, JavaScript-программы компилируются во внутреннее представление, известное как *байт-код,* который затем выполняется интерпретатором. Главной целью языка JavaScript является поддержка активного взаимодействия документов HTML с пользователем. Этот язык не претендует на то, чтобы быть полномасштабным средством программирования, таким как Java или C++. Скорее, он является расширением языка HTML, облегчающим работу пользователя с конкретным браузером. Язык JavaScript расширяет возможности стандартных элементов HTML, позволяя конструкциям веб-страницы взаимодействовать с объектами и свойствами языка JavaScript. С введением обработчиков событий HTML-документы стали более "живыми", поскольку разработчики получили возможность определять их поведение в зависимости от действий пользователя. Важен тот факт, что JavaScript-программы действительно являются выполняемым содержимым документов; они физически находятся внутри HTML-документов, в отличие от Java-апплетов, которые существуют вне документов, их активизирующих. Пример реакции системы на действия человека приведен на рис. 7.1. Курсор мыши в своем движении по экрану остановился на символе сноски. В результате обработки произошедшего события поверх основного текста документа всплыл текст сноски. Читателю не нужно использовать прокрутку документа, чтобы добраться до сноски, она сама появляется перед ним и сама исчезает, если курсор мыши сдвинуть в сторону от символа сноски.

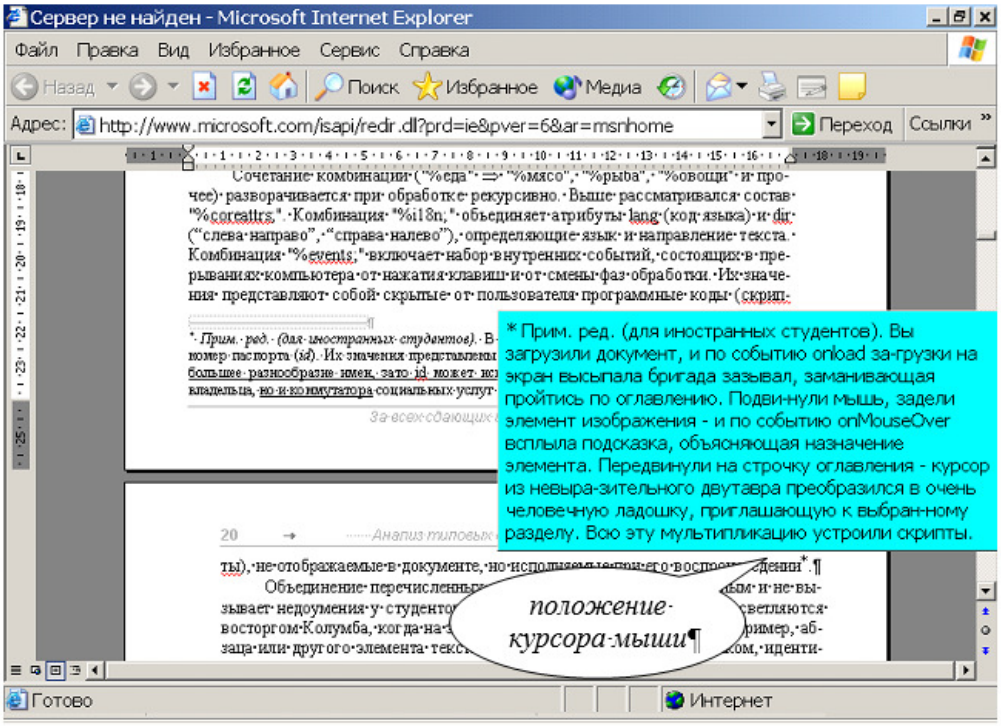



Рис 7.1 - Реакции системы на действия человека

Все события, происходящие в браузере Navigator, например, нажатие кнопки или переход к другой странице, обнаруживает и обрабатывает операционная система, передавая результирующие параметры JavaScript-программам. Важность этих событий состоит в том, что они позволяют программным элементам документов и браузеру более тесно взаимодействовать друг с другом. Например, JavaScript-программа включается, когда пользователь покидает страницу, и выполняет при этом необходимые, запланированные автором действия. JavaScript-программы могут обрабатывать множество других событий, таких как выбор нового элемента списка или инициализация формы. Кроме того, язык JavaScript хорошо приспособлен для проверки правильности описания и заполнения форм, обработки строк и динамического создания HTML-элементов. Он позволяет создавать их динамически, в процессе взаимодействия с пользователем, что называется "на лету". Почти во всех сложных приложениях HTML для управления внешним видом документа используются JavaScript-программы, динамически создающие HTML-элементы и даже целые документы. В частности, чтобы Web-страница приобрела новый внешний вид, ее автору не нужно вручную изменять исходный текст, это делают скрипты.

В результате, с помощью JavaScript-программ вы можете:

- формировать HTML-документы "на лету";

- производить проверку правильности данных HTML-формы перед передачей их на сервер;
- предоставить пользователю возможность вводить локальные данные для управления работой JavaScript-программы, а также выборочно выполнять различные операции;
- создавать окна сообщений и диалоговые окна для вывода предупреждающих сообщений и ввода данных;
- создавать документы с расширенными возможностями навигации, используя фреймы и автономные окна;
- обнаруживать Java-апплеты и подключаемые модули (plug-in) браузера и взаимодействовать с ними.

## 2.2. Основы грамматики JavaScript

Большинство программ на языке JavaScript распространяются по сети упакованными в документах, запрашиваемых пользователями. До тех пор пока ресурс не загружен в браузер, невозможно определить, содержит он JavaScript-программу или нет. Чтобы браузер мог обнаружить такую программу, в язык HTML введен элемент <SCRIPT>. Имеется также дополняющий элемент <NOSCRIPT>, позволяющий авторам HTML-документов выяснить, когда язык JavaScript использовать нельзя. Скрипты могут находиться в любом месте документа. Обратное неверно: теги элементов языка HTML нельзя помещать внутри JavaScript-программы. Не забывайте заключать JavaScript-программу в теги <SCRIPT>. . .</SCRIPT>, кроме тех случаев, когда она используется как обработчик события. Событием может явиться нажатие кнопки, передача заполненной формы, загрузка новой страницы и т.д. При встрече с открывающим тегом <SCRIPT>, браузер построчно анализирует содержимое документа до тех пор, пока не будет достигнут закрывающий тег </SCRIPT>. После этого производится проверка скрипта на наличие ошибок и компиляция JavaScript-программы в формат, пригодный для выполнения на компьютере пользователя. Если при проверке или компиляции программы обнаруживаются ошибки, Navigator выводит на экран окна с предупреждениями. Чтобы продолжить работу,

необходимо нажать кнопку ОК в каждом таком окне.

Главная часть JavaScript-программы может быть помещена в контейнер <HEAD>. . .</HEAD>, поскольку он считывается в программу Navigator при загрузке HTML-документа одним из первых. Теоретически скрипт можно размещать в любом месте HTML-документа, хотя лучше это делать перед контейнером <BODY>.. .</BODY>, т.е. в заголовке документа, чтобы функции языка JavaScript загружались в память сразу же после загрузки документа в браузер. Однако некоторые авторы любят размещать скрипты в конце программы. Тогда их не нужно разыскивать по тексту разметки. Окончательный выбор остается за вами. Описания функций лучше всего помещать в заголовок документа, хотя скрипты, выполняемые "на лету" и формирующие новые документы HTML при обращении к определенным частям текущего документа, могут быть размещены там, где они необходимы. Синтаксис элемента <SCRIPT> следующий:

<SCRIPT [*language*="JavaScript"] [*src*=URI]>

[JavaScript-statements (предложения языка JavaScript)...]

</SCRIPT>

Как известно, необязательный атрибут *language* элемента SCRIPT устанавливает язык скриптов, используемый в документе. Это может быть «родной» для компании Microsoft язык VBScript, наследующий конструкции Visual Basic. Здесь значением атрибута *language* является строка "JavaScript". Как правило, то же значение присваивается атрибуту по умолчанию. Далее, с помощью атрибута *src* (source, т.е. ресурс, источник) определяется адрес файла, содержащего скрипт. Заметим, что регистр символов в данном случае не имеет значения, как и при указании тегов языка HTML. Если атрибут *src* определен, то тело скрипта [JavaScript-statements...] пропускается, потому что скрипт будет загружен из определенного атрибутом *src* файла. Если значение атрибута *src* не указывается явно, то предполагается, что программа содержится в контейнере <SCRIPT>.. .</SCRIPT>. Операторы языка JavaScript подробно рассматриваются в [6,7].

Для того чтобы включить JavaScript-программу в HTML-документ, используйте открывающий тег <SCRIPT> без определения атрибута *src*. Например, при загрузке следующего HTML-документа на экран выводится окно сообщения:

```
<HTML>
<!--Пример сценария 1-1 -->
<HEAD>
<SCRIPT language=" JavaScript ">
alert {"Добро пожаловать !"} ;
</SCRIPT>
</HEAD>
---------- тело документа ----------
</HTML>
```

Метод alert() – это один из инструментов языка JavaScript. При его выполнении на экране появляется окно сообщения, в котором отображаются заданный текст, кнопка ОК и пиктограмма с восклицательным знаком. Такие предупреждения используются для привлечения внимания пользователя. Приведенная программа выполняется немедленно после загрузки включающего ее документа, хотя в языке JavaScript можно определить и функции, содержащие последовательности операторов – аналогов обработчиков прерываний в пакетах визуального программирования. Такие функции активизируются только при обращении к ним из основной программы.

## 2.3. Система событий языка JavaScript

Использование языка JavaScript при обработке событий значительно расширило возможности языка HTML. В элементы HTML, определяющие гиперсвязи и компоненты формы, добавлены необходимые атрибуты. Чаще всего

сценарии создаются для контроля реакции системы на любые действия пользователя, от движения курсора мыши по экрану до обработки информации, вводимой в поля форм. Возможности управления элементами форм обеспечиваются главным образом за счет функций обработки событий, которые задаются для всех элементов документа, в частности, для формы. Форма представляет собой контейнер, содержащий поля ввода, области текста, списки и кнопки. Для каждого из этих элементов определяются программы обработки событий, что существенно повышает степень интерактивности документа. События делятся на несколько категорий:

- события, связанные с документами в целом – загрузка и выгрузка;
- события, связанные с гиперссылками – активизация гиперссылки;
- события, связанные с формой – щелчки мыши на кнопках (button), группах кнопок выбора варианта (radiobutton), переключателях (checkbox), кнопках передачи данных и восстановления исходных значений элементов, получение и потеря фокуса ввода, а также изменение содержимого полей ввода, областей текста и списков, выделение текста в полях ввода и областях текста;
- события, связанные с рисунками – загрузка рисунка, ошибка загрузки рисунка, прерывание загрузки рисунка;
- события, связанные с клавиатурой – нажатие, давление, отпускание любой клавиши компьютера;
- события, связанные с мышью – помещение указателя мыши на гиперссылку и активизация гиперссылки, движение курсора по экрану, щелчки и двойные щелчки кнопкой мыши.

События, связанные с документами, возникают при загрузке и выгрузке документа, в то время как события гиперсвязей возникают при их активизации или при помещении на них указателя мыши. Очень популярно использование программ на языке JavaScript для обработки событий форм. События, связанные с рисунками, позволяют выполнять ответные действия как в процессе загрузки рисунка, так и при возникновении ошибок. Рисунки теперь тоже представляются как объекты с программными свойствами, которые можно описать массивами. Например, выражение document.images[0].src соответствует атрибуту *src* (адрес

URI) в открывающем теге первого элемента <IMG> текущего документа.

Обработчики событий конкретного объекта задаются в HTML-теге, определяющем этот объект. Например, обработчик события, связанного с рисунком, задается в теге <IMG>, обработчик события гиперсвязи – в теге <A> и т.д. В целях «перехвата» события программируют функции-обработчики событий. Ими могут оказаться достаточно объемные коды, или только группы из одного или нескольких операторов, разделенных точкой с запятой (;). В таблице 7.1 перечислены имена большинства событий, используемых в языке разметки гипертекстов, и условия их возникновения. В среднем столбце таблицы сосредоточены названия атрибутов, обеспечивающих правильную реакцию на события.

Таблица 7.1

| Имя события | Атрибут | Условие возникновения события |
|---|---|---|
| Blur | onBlur | Потеря фокуса ввода элементом формы |
| Change | onChange | Изменение содержимого поля ввода или области текста, выбор нового элемента списка |
| Click | onClick | Щелчок мыши на элементе HTML |
| Focus | onFocus | Получение фокуса ввода элементом формы |
| Load | onLoad | Завершение загрузки документа |
| MouseOver | onMouseOver | Помещение указателя мыши на элемент |
| MouseOut | onMouseOut | «Уход» указателя мыши из контура элемента |
| KeyPress | onKeyPress | Нажатие клавиши на клавиатуре компьютера |
| Select | onSelect | Выделение текста в поле ввода или во фрагменте текста |
| Submit | onSubmit | Передача данных формы |
| Unload | onUnLoad | Выгрузка текущего документа (начало загрузки в браузер нового документа) |

## 3. ПОДГОТОВКА К ВЫПОЛНЕНИЮ РАБОТЫ

3.1. Подготовить сменный носитель (дискету, flash-накопитель) с результатами работ №№ 1 и 2. На созданном в них сайте будет производиться доработка дизайна, составляющая содержание данного занятия. Результаты будут сохранены с целью их дальнейшего использования в работе № 3.

3.2. По рекомендованной литературе [6, 7] ознакомиться с принципами написания скриптов на языке JavaScript.

3.3. Подготовить и обосновать предложения по внесению динамики в сайт, разработанный на занятиях 1 и 2. Определить основные события, на которые должна реагировать система.

3.4. Составить программные коды обработчиков прерываний по п. 3.3.

3.5. По настоящим методическим указаниям составить план действий за компьютером с распределением обязанностей между членами бригады.

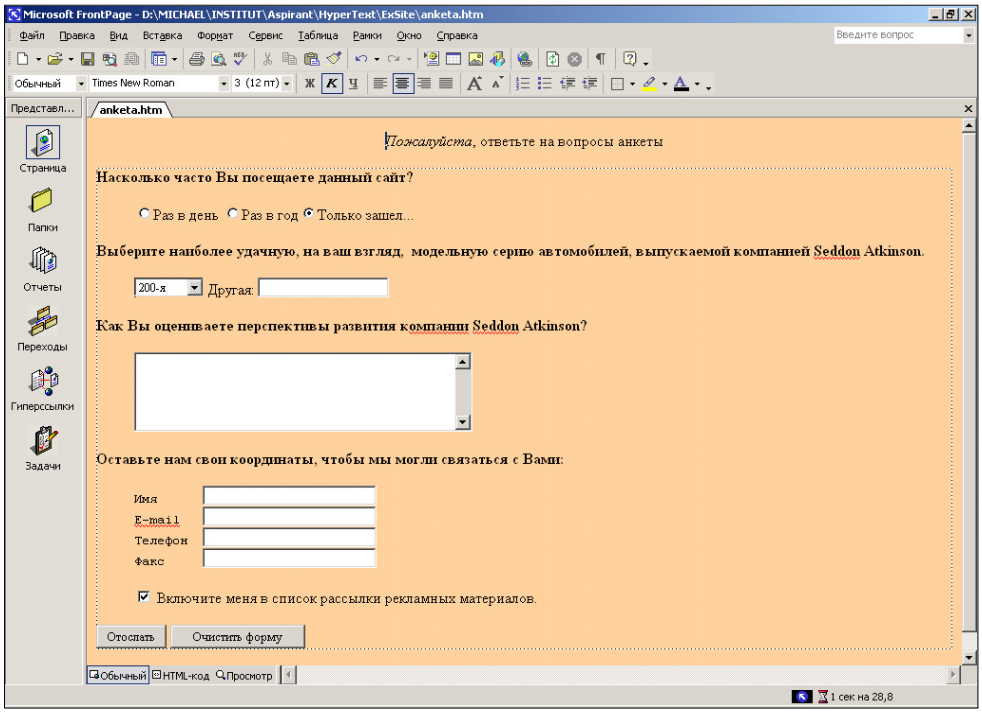
3.6. Подготовить текст теоретических разделов отчета о работе.

3.7. Ответить на контрольные вопросы настоящих методических указаний.

## 4. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

В процессе выполнения работы поощряются инициативные предложения студентов по разработке сценариев для придания динамики проектируемому сайту. По согласованию с преподавателем индивидуальные задания значительно отличаются от рекомендуемого ниже текста. Их состав должен быть подготовлен в соответствии с п. 3.3 настоящих методических указаний. Допускаются варианты с использованием технологии активных серверных страниц, реализация которой позволяет, например, производить запрос по многим ключам к удаленной базе данных, с указанием таких критериев поиска, как фамилия автора, год издания, издательство, ключевые слова, язык документа и т.д. Ниже изложены задания, указывающие минимальные требования к сложности задания.

4.1. ПРОЕКТИРОВАНИЕ СТРАНИЦЫ АНКЕТЫ. Создайте пустую веб-страницу. Настройте цвет фона, шрифты и другие свойства страницы, аналогичные ранее размеченным страницам сайта. Добавьте на страницу объект «форма». Поместите в форму группу кнопок выбора (radiobutton), комбинированный список с выпадающими ответами, несколько строк ввода (обязательны поля ввода адреса e-mail и номера телефона), список с возможностью выбора нескольких вариантов ответа и область ввода текста. Дайте




форме и ее объектам понятные («говорящие») имена. Не следует называть форму “f”, радиокнопки “r”, “rr” или “rrr”, а область ввода текста “a”. Оформите страницу в соответствии с правилами дизайна. Пример анкеты показан на рис. 7.2.

Добавьте ссылку на анкету в меню сайта.

4.2. СОЗДАНИЕ СКРИПТОВ ДЛЯ ПРОВЕРКИ ПРАВИЛЬНОСТИ ЗАПОЛНЕНИЯ ПОЛЕЙ ФОРМЫ.

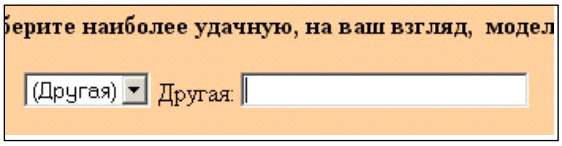
4.2.1. Самый надежный способ ненавязчивого контроля правильного ввода пользователем необходимой вам информации – создать обработчик события onBlur (элемент формы из активного превращается в неактивный). Тогда проверка производится после заполнения поля, в момент перехода к следующему. Можно использовать обработчик onChange, реагирующий на каждый вводимый символ,

но это лишит вашего потенциального клиента права на опечатку или на создание экзотического псевдонима. Постоянно всплывающие в процессе ввода информации сообщения могут заставить даже лояльного пользователя отказаться от сотрудничества с вами. Более того, часто такой способ просто неприемлем (например, для проверки правильности ввода адреса e-mail). Дополнительно организуйте функцию общей повторной проверки всех полей формы по событию submit (отправка формы). Опыт показывает, что «изобретательные» корреспонденты легко находят способ обойти событие onBlur.

4.2.2. Для поля ввода имени (см. рис. 3.2) разрешенными символами являются буквы и дефис. Если желательно допустить ввод фамилии, инициалов или имени и отчества, то следует разрешить пробелы и точки. Существуют два варианта скрипта проверки правильности ввода имени: один проверяет наличие разрешенных символов, другой – наличие запрещенных. В обоих случаях в функции проверки задается строка запрещенных или разрешенных для использования в имени символов, а затем в цикле проверяются значения символов из поля ввода имени. С помощью функции indexOf или search проверяется присутствие или отсутствие текущего символа в запрещающей или разрешающей строке. По факту обнаружения несоответствия символа в поле ввода имени указанным ограничениям на экран с помощью функции alert выдается соответствующее сообщение («Обнаружен неверный символ»), курсор помещается в поле ввода имени (метод focus), а вся строка выделяется (метод select), чтобы дать возможность пользователю удалить ее одним нажатием клавиши.

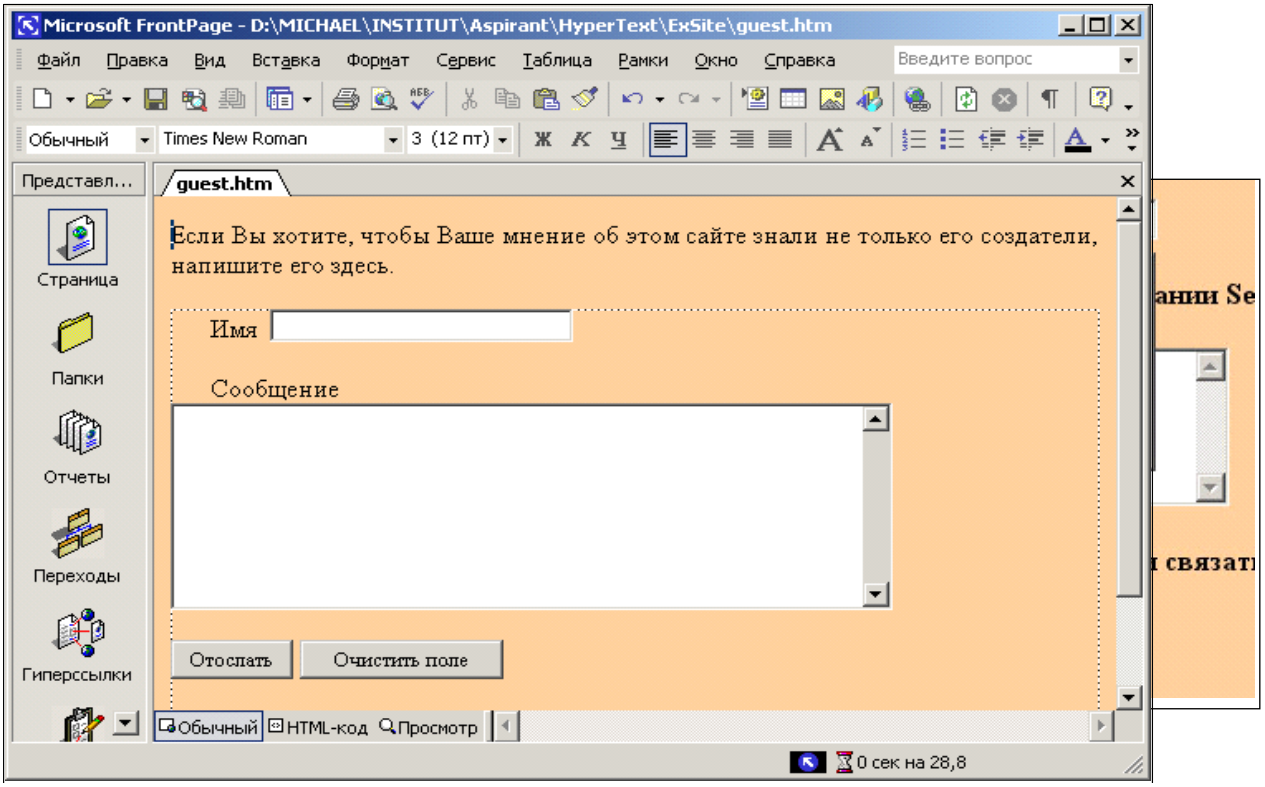
4.2.3. Функции проверки ввода номера телефона или факса принципиально не отличаются от проверки ввода имени. Единственным отличием является то, что при таком виде проверки проще задать строку разрешающих символов, так как их будет заведомо меньше, чем запрещенных. В номерах телефона и факса разрешено указывать цифры, а также скобки, дефис и плюс. Действия системы по факту обнаружения нарушений аналогичны перечисленным в пункте 4.2.2.

4.2.4. Содержимое комбинированного списка для выбора альтернативного варианта ответа в контроле не нуждается, оно



введено автором на этапе дизайна. Нужно проверить, введено ли что-нибудь в строку альтернативного варианта ответа (рис. 7.3). Правила дружественного интерфейса советуют при выборе варианта «Другая модель» автоматически – с помощью обработки события onChange элемента ComboBox – переводить фокус ввода на строку альтернативного варианта. В этом поле ввода допустимые комбинации символов должны принадлежать множеству всех известных моделей фирмы.

4.2.5. Функция проверки ввода адреса e-mail может быть реализована




несколькими способами. Самый простой из них – использовать функцию search. Более надежный и трудоемкий – последовательно перемещаться по строке, аналогично процедуре проверки имени, выставляя флажки-признаки наличия встреченных символов, проверяя их повторение и т.д. В обозначении e-mail адреса необходимо удостовериться в использовании только разрешенных символов. Это буквы, цифры, символы «_» и «-». Символ «@» должен присутствовать только один раз. Устанавливается наличие символа «.» (точка), отсутствие точки непосредственно после или перед символом «@», отсутствие лидирующей и замыкающей точки в обозначении адреса, отсутствие следующих друг за другом точек (многоточия), а также возможность указания доменов третьего и более высоких уровней (комбинации вида ghf56@hj.dhh.aaa.des). Действия по факту обнаружения нарушений аналогичны указанным выше (см. рис. 7.4). Отличие состоит в том, что сообщения об ошибках должны четко указывать, что именно некорректно в строке, например, «В адресе e-mail встречено два символа @».

4.2.6. Поле ввода текста может быть заполнено произвольным набором

символов. Следует проверить, не осталось ли оно пустым.

4.3. СТРАНИЦА ГОСТЕВОЙ КНИГИ. Создайте пустую страницу. Настройте ее свойства по аналогии с имеющимся страницами сайта и разместите на ней объект «форма» с полями ввода (рис. 7.5).

## СОДЕРЖАНИЕ ОТЧЕТА

Отчет о работе должен содержать следующие материалы в соответствии с полученным индивидуальным заданием:

4.4. Краткие формулировки назначения, области применения и функциональных характеристик проектируемого сайта (по материалам индивидуального задания).

4.5. Обоснование эффективности использования скриптов в разметке страниц создаваемого сайта (по материалам п.2 данного пособия).

4.6. Обоснование выбора визуальных элементов создаваемых документов (собственного или по п. 4 настоящего пособия).

4.7. Таблица (по аналогии с табл.3.1 пособия), представляющая информацию о событиях прерывания, используемых в лабораторной работе.

4.8. Программные коды обработчиков прерываний по событиям с комментариями или блок-схемами (по материалам п. 2.2 настоящего пособия).

4.9. Содержание выполненных индивидуальных занятий.

4.10. Ответы на контрольные вопросы и выводы по выполненной работе.

## 5. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Каковы назначение и функции скриптов (сценариев) в языке разметки гипертекстов? К какой категории языка они относятся - это элементы, атрибуты, комментарии, ссылки, или они не входят ни в какие другие категории HTML?

6.2. В каком разделе документа HTML предпочтительно располагать элемент <SCRIPT>?

6.3. Каким образом прерывание от внешнего устройства или программного процесса активизирует выполнение скриптов? Постройте универсальную схему обработки прерываний.

6.4. Какие задачи решаются с помощью скриптов для придания динамики

работе с гиперссылками? С какой целью программный код скриптов заключается в теги, воспринимаемые устаревшими браузерами как комментарии?

6.5. Элемент <A> (якорь) предназначен как для идентификации цели гиперссылки (атрибуты *id* и *name*), так и для указания адреса связанного ресурса (атрибут *href*). Воспримет ли система документ, в разметке которого не определены значения ни одного из этих атрибутов?

6.6. Какое противоречие содержит в себе рис. 3.1? Какие элементы представленного изображения несовместимы в реальной ситуации?

6.7. Могут ли программные коды сценариев составлять содержание таблиц стилей и можно ли с их помощью автоматически настраивать таблицы стилей в зависимости от конкретных характеристик компьютеров пользователей?