



ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
УПРАВЛЕНИЕ ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ
КВАЛИФИКАЦИИ

Кафедра «Программное обеспечение вычислительной тех-
ники и автоматизированных систем»

Учебно-методическое пособие по дисциплине

«ВЕБ-ТЕХНОЛОГИИ»

Автор
Чугунный К.А.

Ростов-на-Дону, 2018

Аннотация

Учебно-методическое пособие предназначено для студентов очной формы обучения по направлениям 02.03.03 «Математическое обеспечение и администрирование информационных систем», 09.03.04 «Программная инженерия».

Авторы

Ст.преподаватель
каф. ПОВТиАС
Чугунный К.А.



Оглавление

1. ЛАБОРАТОРНАЯ РАБОТА №1: БАЗОВЫЙ HTML	5
2. ЛАБОРАТОРНАЯ РАБОТА №2: КЛИЕНТСКИЕ СЦЕНАРИИ. ИСПОЛЬЗОВАНИЕ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ 6	
2.1. Обработка событий в JavaScript	6
2.2. Регулярные выражения	10
2.3. Использование регулярных выражений в JavaScript	14
2.4. Порядок выполнения лабораторной работы	16
2.5. Контрольное задание.....	17
3. ЛАБОРАТОРНАЯ РАБОТА №3: ПРОГРАММНАЯ ОБРАБОТКА XML ДОКУМЕНТОВ С ПОМОЩЬЮ XML DOM ...17	
3.1. Структурный анализ (парсинг) XML	18
3.2. Программный интерфейс XML DOM.....	20
3.3. Перемещение между узлами дерева	21
3.4. Игнорирование пустых текстовых узлов	22
3.5. Изменение значения атрибута	23
3.6. Свойства и методы объекта Node	23
3.7. Порядок выполнения лабораторной работы	25
3.8. Контрольное задание.....	29
4. ЛАБОРАТОРНАЯ РАБОТА №4: ФОРМАТИРОВАНИЕ И ИМПРЕОБРАЗОВАНИЕ XML ДОКУМЕНТА С ПОМОЩЬЮ XSL. XSLT ПРЕОБРАЗОВАНИЕ XML ДОКУМЕНТА	30
4.1. Объявление XSL.....	31
4.2. Реализация преобразования с помощью сценария.	32
4.3. Порядок выполнения лабораторной работы	32
4.4. Контрольное задание.....	38
5. ЛАБОРАТОРНАЯ РАБОТА №5: РЕАЛИЗАЦИЯ АСИНХРОННОГО ВЗАИМОДЕЙСТВИЯ ВЕБ-БРАУЗЕРА С ВЕБ-СЕРВЕРОМ С ПОМОЩЬЮ ТЕХНОЛОГИИ AJAX	39
5.1. Порядок выполнения лабораторной работы	40
5.2. Контрольное задание.....	45
6. ЛАБОРАТОРНАЯ РАБОТА №6: Разработка CGI-	

приложений на Perl и PHP45

- 6.1. Основы разработки сценариев на языке Perl45
- 6.2. Основы разработки сценариев на языке PHP52
- 6.3. Порядок выполнения лабораторной работы59
- 6.4. Контрольные задания61

7. ЛАБОРАТОРНАЯ РАБОТА №7: Примеры разработки

RSs-источников и RSS-ридеров.....62

- 7.1. Теоретический материал62
- 7.2. Порядок выполнения лабораторной работы67
- 7.3. Контрольное задание.....71

1. ЛАБОРАТОРНАЯ РАБОТА №1: БАЗОВЫЙ HTML

Цель работы:

Получить навыки разработки статического WEB-интерфейса с пользовательским меню и областью отображения.

Порядок выполнения работы:

Написание кода разметки меню -> написание кода разметки страниц -> отображения рисование или поиск необходимых графических элементов интерфейса -> объединение элементов в один гипертекстовый документ с двумя фреймами -> разработка корневого меню - > поиск графических изображений товаров -> включение ссылок на изображения в разработанный интерфейс.

Задание для выполнения:

Создать не менее 6 статических HTML-страниц, в которых будет представлен список товаров в соответствии с вариантом задания. Общая структура страниц должна состоять из двух фреймов. В одном фрейме - меню. В другом должна выводиться соответствующая информация. Титульная страница должна содержать в верхней части графическое меню, в нижней должны выводиться разделы товара. Необходима страница, содержащая информацию о фирме и ее реквизитах. Необходима страница, содержащая данные о товаре, у которого возможен просмотр. Страницы должны содержать графические изображения выбранных товаров, различные виды списков, таблицы и листы стилей для оформления.

Варианты конкретизирующие задания:

Тип товара
Сотовые телефоны
Одежда
Аудио-видео аппаратура
Компьютеры
Книги
Кондиционеры

Содержание отчёта:

Титульный лист, цель работы, порядок выполнения работы с последовательными выводами, привести теги с подобранными атрибутами и описать их действие. Выводы по работе относительно достоинства и недостатков интерфейса, полученного изученным способом.

2. ЛАБОРАТОРНАЯ РАБОТА №2: КЛИЕНТСКИЕ СЦЕНАРИИ. ИСПОЛЬЗОВАНИЕ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

Цель работы:

- 1) Получить представление об общих принципах обработки в JavaScript событий, связанных с окном веб-браузера, веб-страницей, содержащейся в браузере и элементами документа.
- 2) Научиться использовать простейшие элементы регулярных выражений для поиска подстрок, структура и содержание которых описывается нетривиальным шаблоном.

2.1. Обработка событий в JavaScript

Популярность JavaScript во многом обусловлена именно тем, что написанный на нем сценарий может реагировать на действия пользователя и другие внешние события. Каждое из событий связано с тем или иным объектом: формой, гипертекстовой ссылкой или даже с окном, содержащим текущий документ.

В качестве примеров внешних событий, на которые могут реагировать объекты JavaScript, можно привести следующие.

- окончание загрузки документа в окно (или окончание загрузки документов во все фреймы окна). Это событие связано с объектом window;
- щелчок мышью на объекте. Это событие может быть связано с интерактивным элементом формы или с гипертекстовой ссылкой;
- получение объектом фокуса ввода. Это событие может быть связано с объектами типа Text, Password и с другими

- интерактивными элементами;
- передача на сервер данных, введенных пользователем с помощью интерактивных элементов. Связывается с формой.

Обработка события производится с помощью специально предназначенного для этого фрагмента кода, называемого обработчиком события. Для каждого события JavaScript предоставляет свой обработчик. Однако при построении сценария можно создавать собственный обработчик события и использовать его вместо обработчика, заданного по умолчанию.

Имя обработчика определяет, какое событие он должен обрабатывать. Так, для того чтобы сценарий нужным образом отреагировал на щелчок мышью, используется обработчик с именем *onClick*, для обработки события, заключающегося в получении фокуса ввода, - обработчик *onFocus*.

Для того чтобы указать интерпретатору JavaScript на то, что обработкой события должен заниматься обработчик, необходимо включить в HTML-дескриптор следующее выражение:

имя_обработчика="команды_обработчика"

Это выражение включается в тэг, описывающий объект, с которым связано событие.

Например, если необходимо обработать событие, заключающееся в получении фокуса полем ввода, дескриптор, описывающий этот интерактивный элемент, должен иметь примерно следующий вид:

```
<input type="text" name="Inform" onFocus="handleFocus();">
```

Имя обработчика является одним из атрибутов HTML-дескриптора, а команды, предназначенные для обработки события, выступают в роли значения этого атрибута. В данном случае обработка события производится в теле функции *handleFocus()*. В принципе, обработчиком может быть не только функция, но и любая последовательность команд JavaScript в виде составного оператора.

Следующий пример демонстрирует обработку события, связанного с наведением курсора мыши на гиперссылку:

```
<a href = "http://www.myhp.edu" onmouseover="alert('An
onMouseOver event'); return false">
    
</a>
```

Ниже приводится полный текст HTML документа с JavaScript сценарием, в котором обрабатывается событие нажатия кнопки мыши, и определяется, какая именно из них была нажата:

Пример 1.

```
<html>
    <head>
    <script language = "javascript">
    function whichButton(event)
    {
        if (event.button == 2)
        {
            alert("Вы щелкнули правой кнопкой
            мыши!");
        }
        else
        {
            alert("Вы щелкнули левой кнопкой
            мыши!");
        }
    }
    </script>
    </head>

    <body onmousedown="whichButton(event)">
    <p>Щелкните любой кнопкой мыши в любом
    месте документа</p>
    </body>

</html>
```

Таким образом, для того чтобы обработать какое-либо стандартное событие в браузере, необходимо добавить в

подходящий HTML тэг атрибут, соответствующий этому событию, указав в качестве значения атрибута имя JavaScript функции. Список атрибутов, которые определены для HTML тэгов приводится ниже:

IE: *Internet Explorer*, **F:** *Firefox*, **O:** *Opera*, **W3C:** *стандарт*

Атрибут	Описание	Номер версии браузера			W3C
		IE	F	O	
onabort	Прерванная загрузка изображения	4	1	9	Да
onblur	утрата фокуса элементом	3	1	9	Да
onchange	Изменение содержимого в поле ввода	3	1	9	Да
onclick	Щелчок мыши на объекте	3	1	9	Да
ondblclick	Двойной щелчок мыши на объекте	4	1	9	Да
onerror	Ошибка при загрузке изображения или документа	4	1	9	Да
onfocus	Получение фокуса элементом	3	1	9	Да
onkeydown	Нажатие клавиши	3	1	Нет	Да
onkeypress	Клавиша нажата	3	1	9	Да
onkeyup	Отжатие клавиши	3	1	9	Да
onload	Завершение загрузки страницы или изображения	3	1	9	Да

Веб-технологии

onmousedown	Нажатие кнопки мыши	4	1	9	Да
onmousemove	Перемещение курсора мыши	3	1	9	Да
onmouseout	Смещение курсора мыши с объекта	4	1	9	Да
onmouseover	Наведение курсора мыши на объект	3	1	9	Да
onmouseup	Отжатие кнопки мыши	4	1	9	Да
onreset	Кнопка "Reset" нажата	4	1	9	Да
onresize	Изменение размера окна	4	1	9	Да
onselect	Выделение текста	3	1	9	Да
onsubmit	Кнопка "Submit" нажата	3	1	9	Да
onunload	Уход с веб-страницы	3	1	9	Да

2.2. Регулярные выражения

Регулярные выражения — система поиска текстовых фрагментов в электронных документах, основанная на специальной системе записи образцов для поиска.

Образец, задающий правило поиска, называется «шаблоном». Применение регулярных выражений принципиально преобразило технологии электронной обработки текстов.

С помощью регулярных выражений можно задавать структуру искомого шаблона и его позицию внутри строки (например, в начале или в конце строки, на границе или не на границе слова).

При описании структуры шаблона используются:

гибкая система квантификаторов (операторов повторения); операторы описания наборов символов и их типа (числовые, нечисловые, специальные).

Для того, чтобы задать положение искомого фрагмента внутри строки, можно использовать один из следующих операторов:

Представление	Позиция
\wedge	Начало строки
$\$$	Конец строки
$\backslash b$	Граница слова
$\backslash B$	Не граница слова
(?=шаблон)	Искомая строка следует после указанной строкой (с просмотром вперед)
(?!=шаблон)	Искомая строка не следует после указанной строки (с просмотром вперед)
(?<=шаблон)	Искомая строка следует после указанной строкой (с просмотром назад)
(?!<шаблон)	Искомая строка не следует после указанной строки (с просмотром назад)

Кроме того, язык регулярных выражений предоставляет набор квантификаторов, позволяющих указать число повторений шаблона:

Представление	Число повторений
$\{n\}$	Ровно n
$\{m, n\}$	От m до n включительно
$\{m, \}$	Не менее m
$\{,n\}$	Не более n

Имеются и более простые квантификаторы:

Представление	Число повторений	Эквивалент
*	Ноль или более	$\{0, \}$
+	Одно или более	$\{1, \}$
?	Ноль или одно	$\{0, 1\}$

Для задания внутри шаблона группы символом можно использовать следующие операторы:

Оператор	Описание
[xyz]	Любой символ из указанного множества
[^xyz]	Любой символ не входящий в указанное множество
[x-z]	Любой символ из указанного диапазона
[^x-z]	Любой символ не входящий в указанный диапазон
. (точка)	Любой символ кроме символов разрыва или переноса строки
\w	Любой буквенно-цифровой символ, включая символ подчеркивания
\W	Любой не буквенный символ
\d	Любая цифра
\D	Любой нецифровой символ
\s	Любой не отображаемый символ
\S	Любой символ (кроме неотображаемых символов)

Для группировки отдельных частей шаблона можно использовать следующие операторы:

Оператор	Описание
()	Поиск группы символов внутри скобок и сохранение найденного соответствия
(?:)	Поиск группы символов внутри скобок без сохранения найденного соответствия
	Комбинирование частей в одно выражения с последующим поиском любой из частей в отдельности. Аналогично оператору «ИЛИ».

Если шаблон поиска включает специальные (как

правило неотображаемые) символы, для их описания можно использовать следующие обозначения:

Обозначение	Описание
<code>\0</code>	Символ с нулевым кодом
<code>\n</code>	Символ новой строки
<code>\r</code>	Символ начала строки
<code>\t</code>	Символ табуляции
<code>\v</code>	Символ вертикальной табуляции
<code>\xxx</code>	Символ, имеющий заданный восьмеричный ASCII код <i>xxx</i>
<code>\xdd</code>	Символ, имеющий заданный шестнадцатиричный ASCII код <i>dd</i>
<code>\uxxxx</code>	Символ, имеющий ASCII код выраженный ЮНИКОДОМ <i>xxxx</i>

Квантификаторам в регулярных выражениях соответствует максимально длинная строка из возможных (т.е. квантификаторы являются «жадными»). Это может приводить к некоторым проблемам. Например, шаблон (`<.*>`) описывающий на первый взгляд теги HTML на самом деле будет выделять более крупные фрагменты в документе.

Например, строка вида

```
<p><font color='blue'><i>Регулярные
выражения</i></font> - удобный инструмент
для поиска в строках </p>
```

формально соответствует указанному выше шаблону

Для решения данной проблемы можно использовать два подхода.

В регулярном выражении учитываются символы, не соответствующие желаемому образцу (например, `<[^>]*>` для вышеописанного случая).

Определение квантификатора как нежадного (ленивого) - большинство реализаций позволяют это сделать, добавив после него знак вопроса.

Например, по шаблону (<.*?>) будут найдены все теги из рассмотренной строки.

Таким образом, получаются следующие «нежадные» модификации квантификаторов:

Квантификатор	Описание
*?	«не жадный» эквивалент *
+?	«не жадный» эквивалент +
{n,}?	«не жадный» эквивалент {n,}

Следует, однако, иметь в виду, что использование «ленивых» квантификаторов может привести к ситуации, когда выражению соответствует слишком короткая, в частности, пустая строка.

2.3. Использование регулярных выражений в JavaScript

При поиске по тексту можно использовать шаблон, описывающий подстроку. В JavaScript такой шаблон может быть описан с помощью объекта RegExp. В простейшем случае такой шаблон описывает отдельный символ, однако имеет смысл его использовать для регулярных выражений.

Следующий ниже код описывает RegExp объект с именем `pttn`, содержащий регулярное выражение, описывающее целое десятичное число:

```
var pttn = new RegExp("[0-9]+");
```

Объект RegExp имеет 3 встроенных метода: `test()`, `exec()` и `compile()`.

Метод `test()` выполняет поиск по шаблону:

```
var pttn = new RegExp("[0-9]+");
document.write(pttn.test("38 попугаев"));
```

Результат:

```
true
```

Метод `exec()` выполняет поиск подстроки по шаблону и возвращает найденные соответствия; если соответствий нет, возвращается значение `null`:

```
var pattn=new RegExp("[0-9]+");  
document.write(pattn.exec("38 попугаев"));
```

Результат:

```
38
```

Если необходимо найти все соответствия, то при вызове конструктора `RegExp` следует указать дополнительный параметр `"g"`, указывающий на необходимость глобального поиска.

Пример 2

```
var pattn = new RegExp("[0-9]+", "g");  
do  
{  
    result = pattn.exec("1 попугай, 2 попугая,..., 38  
    попугаев");  
    document.write(" " + result);  
}  
while (result != null)
```

Результат:

```
1 2 38 null
```

Метод `compile()` применяется для изменения ранее созданного шаблона.

Пример 3

```
var pattn = new RegExp("[0-5]+");  
document.write(pattn.exec("38 попугаев"));  
  
pattn.compile("[6-9]+");  
document.write("; " + pattn.exec("38 попугаев"));
```

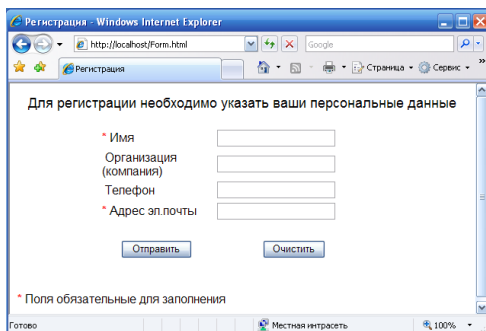
Результат:

3;8

2.4. Порядок выполнения лабораторной работы

Проверка значений, введенных пользователем в поля формы для регистрации.

- 1) Для выполнения лабораторной работы необходимо создать веб-страницу, содержащую форму с полями, следующего вида:



Регистрация - Windows Internet Explorer

http://localhost/Form.html

Регистрация

Для регистрации необходимо указать ваши персональные данные

* Имя

Организация (компания)

Телефон

* Адрес эл. почты

* Поля обязательные для заполнения

Готово Местная intrасеть 100%

- 2) В тэге `<form>` добавьте обработчик события отправки данных вида:

```
onSubmit = "CheckData(); return false;"
```

В данном случае указана функция обработчик `CheckData()`. Оператор `return false;` предотвращает автоматическую отправку данных после выполнения функции-обработчика. Отправка данных будет выполняться из обработчика.

- 3) Добавьте на странице секцию JavaScript кода, описывающего функцию-обработчик:

```
function CheckData()
{
    var ans;
    ans = confirm("Вы уверены, что хотите отправить введенные данные ?");
    if (ans) submit();
}
```

Как это видно из кода, функция `CheckData()` в случае подтверждения со стороны пользователя самостоятельно вызывает

метод `submit()` для передачи данных из формы.

- 4) Теперь необходимо добавить проверку значений, введенных в поля формы пользователем.

Прежде всего, необходимо убедиться в том, что заполнены все поля, обязательные для ввода. Для этого можно использовать проверку на равенство нулю длины строки (свойство `length`), являющейся значением узла дерева документа, соответствующего полю ввода, например:
`document.getElementById("uname").value.length`.

Следующая проверка должна контролировать структуру и содержимое полей. Для этого можно использовать объект `RegExp`, например:

```
var validEMail, pattn;

pattn = new RegExp("^[|.|_A-Za-z0-9]+?@[|.|_A-Za-z0-9]+?.|[A-Za-z0-9]{2,6}$");

validEMail = pattn.test(document.getElementById("email").value));
```

В данном фрагменте описана проверка структуры электронного адреса из поля формы с идентификатором `"email"`. Для проверки был использован шаблон на основе регулярного выражения.

2.5. Контрольное задание

Самостоятельно постройте регулярное выражение, описывающее шаблон для проверки номера телефона, и внесите все необходимые изменения и дополнения в функцию `CheckData()`.

3. ЛАБОРАТОРНАЯ РАБОТА №3: ПРОГРАММНАЯ ОБРАБОТКА XML ДОКУМЕНТОВ С ПОМОЩЬЮ XML DOM

Цель работы

Ознакомление с основными принципами XML DOM и методами программной обработки XML документов путем манипулирования узлами дерева документа.

Теоретический материал

XML DOM определяет объекты и свойства всех XML элемен-

тов и методы (интерфейс) для доступа к ним. Иначе говоря, XML DOM описывает каким образом необходимо получать, изменять, добавлять и удалять XML элементы.

В соответствии с моделью DOM все, что содержится внутри XML документа - является узлом. То есть XML документ представляется в виде дерева узлов, которыми являются элементы, атрибуты и текст.

Поскольку структуры HTML и XML документов очень похожи, а HTML DOM и XML DOM являются частями более общего стандарта DOM, то и многие аспекты HTML DOM легко переносятся в XML DOM. Поэтому основное внимание будет уделено именно специфическим особенностям именно XML DOM. Рекомендуется предварительное ознакомление с лабораторной работой 3.

3.1. Структурный анализ (парсинг) XML

Все современные браузеры имеют встроенные XML анализаторы (парсеры) для чтения и обработки XML. Анализатор считывает XML документ, размещает его в памяти и преобразует в XML DOM объект, доступный для языков программирования. Все примеры здесь приведены на JavaScript.

Имеются некоторые отличия между анализаторами в Microsoft и в других браузерах. Первый поддерживает как загрузку XML файлов, так и текстовых строк, содержащих XML код, в то время как в других браузерах используются отдельные анализаторы. При этом все анализаторы имеют функции для перемещения по дереву XML документа, доступа, вставки и удаления узлов в дереве.

Рассмотрим пример загрузки XML объектов (файлов и строк) с помощью XML анализатора Microsoft.

```
xmlDoc=new ActiveX-  
Object("Microsoft.XMLDOM");  
xmlDoc.async="false";  
xmlDoc.load("timetable.xml");
```

В первой строке программы создается пустой объект XML документа Microsoft. Далее для предотвращения работы сценария до полной загрузки документа флаг асинхронности устанавливается в "false". В третьей строке содержится инструкция загрузить XML файл "timetable.xml".

В следующем пример происходит загрузка строки с XML кодом для последующего анализа.

```
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async="false";
xmlDoc.loadXML(txt);
```

Следует обратить на разницу между методами *load()* и *loadXML()* по их назначению.

Замечание. Современные браузеры не допускают междоменные обращения к файлам из соображений безопасности, т.е. сама веб-страница (с программным кодом) и XML файл должны физически находиться на одном сервере. В противном случае браузер выдаст сообщение об ошибке доступа.

Ниже приведены также кроссплатформенные реализации загрузки XML файла и XML строки соответственно.

```
<html>
<body>
<script type="text/javascript">
try //Internet Explorer
{
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
}
catch(e)
{
try //Firefox, Mozilla, Opera, etc.
{

xmlDoc=document.implementation.createDocument("", "", null);
}
catch(e) {alert(e.message)}
}
try
{
xmlDoc.async=false;
xmlDoc.load("timetable.xml");
document.write("xmlDoc is loaded, ready for use");
}
catch(e) {alert(e.message)}
</script>
</body>
</html>
```

```

<html>
<body>
<script type="text/javascript">
text="<timetable>";
text=text+"<lesson>";
text=text+"<timeFrom>08.00</timeFrom>";
text=text+"<subject>Deutsch</subject>";
text=text+"<teacher>Borisova</teacher>";
text=text+"</lesson>";
text=text+"</timetable>";

try //Internet Explorer
{
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async="false";
xmlDoc.loadXML(text);
}
catch(e)
{
try //Firefox, Mozilla, Opera, etc.
{
parser=new DOMParser();
xmlDoc=parser.parseFromString(text,"text/xml");
}
catch(e) {alert(e.message)}
}
document.write("xmlDoc is loaded, ready for use");
</script>
</body>
</html>
    
```

3.2. Программный интерфейс XML DOM

В рамках DOM модели **XML** можно рассматривать как множество узловых *объектов*. Доступ к ним осуществляется с помощью *JavaScript* или других языков программирования. Программный интерфейс DOM включает в себя набор стандартных *свойств* и *методов*.

Свойства представляют некоторые сущности (например, <day>), а *методы* - действия над ними (например, добавить <les-

son>).

В XML DOM используются практически те же свойства и методы, что и в HTML DOM.

Например, результатом выполнения следующего ниже JavaScript кода будет текстовой содержимое элемента <subject> в файле *timetable.xml*.

```
txt =  
xmlDoc.getElementsByTagName("subject")[0].childNodes[0].nodeValue;
```

Результат: "Deutsch".

В рамках DOM XML возможны 3 способа доступа к узлам:

1. С помощью метода *getElementsByTagName(name)*. При этом возвращаются все узлы с указанным именем тэга (в виде индексированного списка). Первый элемент в списке имеет нулевой индекс.
2. Путем обхода узлов дерева с использованием циклических конструкций.
3. Путем перемещения по дереву с использованием отношений между узлами.

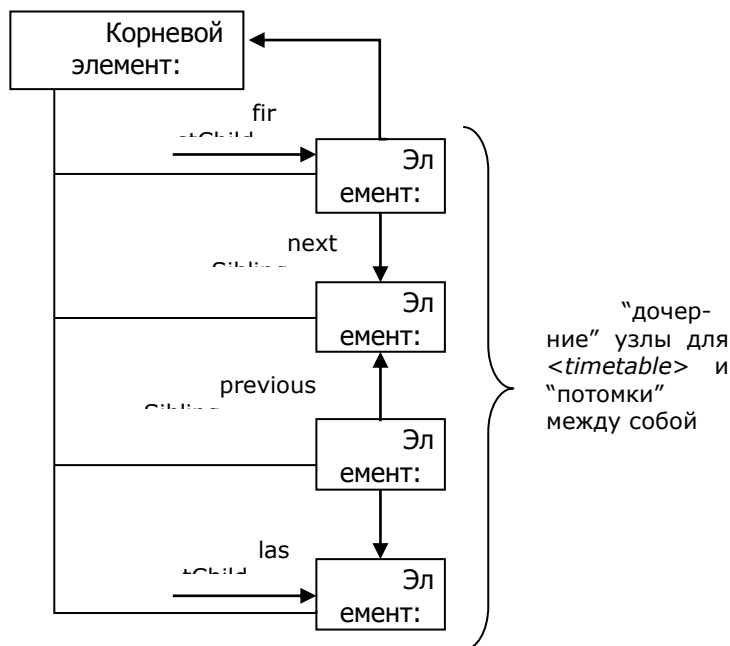
Для определения длины списка узлов используется свойство *length*.

3.3. Перемещение между узлами дерева

В XML DOM отношения между узлами определены в виде следующих свойств узлов:

- parentNode
- childNodes
- firstChild
- lastChild
- nextSibling
- previousSibling

Характер отношений между узлами представлен на следующем рисунке:



3.4. Игнорирование пустых текстовых узлов

Firefox и некоторые другие браузеры воспринимают неотображаемые символы как текстовые узлы (в отличие от *Internet Explorer*). Такая ситуация приводит к проблемам при использовании свойств *firstChild*, *lastChild*, *nextSibling*, *previousSibling*. Для того, чтобы игнорировать такие пустые текстовые узлы можно использовать следующий прием:

```

function get_nextSibling(n)
{
    y = n.nextSibling;
    while (y.nodeType!=1)
    {
        y = y.nextSibling;
    }
    return y;
}
    
```

Поскольку узлы *элементов* имеют тип 1, то в том случае, когда узел-потомок не является узлом *элемента*, будет происходить перемещение к следующему узлу до тех пор, пока не будет найден узел *элемента*.

3.5. Изменение значения атрибута

Узлы атрибутов могут принимать текстовые значения. Изменение этого значения реализуется либо через метод *setAttribute()*, либо через свойство узла атрибута *nodeValue*

Метод *setAttribute()* изменяет значение существующего атрибута или создает новый атрибут.

Например:

```
xmlDoc = loadXMLDoc("timetable.xml");
x = xmlDoc.getElementsByTagName('lesson');
x[0].setAttribute("type", "lab");
```

Свойство *nodeValue* можно использовать для изменения значения атрибута узла:

```
xmlDoc = loadXMLDoc("timetable.xml");
x = xmlDoc.getElementsByTagName("lesson")[0];
y = x.getAttributeNode("type");
y.nodeValue = "lab";
```

Удаление узла из дерева реализуется с помощью метода *removeChild()*:

```
xmlDoc=loadXMLDoc("timetable.xml ");
y = xmlDoc.getElementsByTagName("lesson")[0];
xmlDoc.documentElement.removeChild(y);
```

3.6. Свойства и методы объекта Node

IE: Internet Explorer, **F:** Firefox, **O:** Opera, **W3C:** (Стандарт)

Свойство	Описание	IE	F	O	W3C
		Версия			
baseURI	Возвращает абсолютный URI узла	Нет	1	Нет	Да
childNodes	Возвращает свойство NodeList (список доч.узлов)	5	1	9	Да
firstChild	Возвращает первый дочерний узел	5	1	9	Да
lastChild	Возвращает последний дочерний узел	5	1	9	Да
localName	Возвращает локальную часть имени узла	Нет	1	9	Да

namespaceURI	Возвращает URI узла в пространстве имен	Нет	1	9	Да
nextSibling	Возвращает следующий дочерний узел	5	1	9	Да
nodeName	Возвращает имя узла в зависимости от типа	5	1	9	Да
nodeType	Возвращает тип узла	5	1	9	Да
nodeValue	Устанавливает или возвращает значение узла в зависимости от типа	5	1	9	Да
ownerDocument	Возвращает корневой элемент (объект document) для узла	5	1	9	Да
parentNode	Возвращает родительский узел	5	1	9	Да
prefix	Устанавливает или возвращает префикс пространства имен узла	Нет	1	9	Да
previousSibling	Возвращает непосредственно предшествующий узел	5	1	9	Да
textContent	Устанавливает или возвращает текстовое содержимое узла	Нет	1	Нет	Да
xml	Возвращает XML код узла	5	Нет	Нет	Нет

Методы объекта *Node*

Метод	Описание	IE	F	O	W3C
appendChild()	Добавить новый узел в конец списка дочерних узлов	5	1	9	Да
cloneNode()	Клонирование узла	5	1	9	Да
compareDocumentPosition()	Сравнение позиций двух узлов	Нет	1	Нет	Да
getFeature(feature, version)	Возвращает объект DOM, реализующий специализированный API			Нет	Да

getUserData(key)	Возвращает объект, ассоциирующийся с ключем текущего узла. Перед этим объект должен быть ассоциирован с текущим узлом путем вызова setData с тем же ключем			Нет	Да
hasAttributes()	Возвращает истинное значение, если узел имеет атрибуты	Нет	1	9	Да
hasChildNodes()	Возвращает истинное значение, если узел имеет дочерние узлы	5	1	9	Да
insertBefore()	Вставляет новый узел перед существующим узлом	5	1	9	Да
isDefaultNamespace(URI)	Определяет, является ли указанный namespaceURI значением по умолчанию			Нет	Да
isEqualNode()	Проверяет равенство двух узлов	Нет	Нет	Нет	Да
isSameNode()	Проверяет идентичность двух узлов	Нет	1	Нет	Да
isSupported(feature, version)	Определяет - поддерживается ли указанная характеристика узлом			9	Да
removeChild()	Удаляет дочерний узел	5	1	9	Да
replaceChild()	Заменяет дочерний узел	5	1	9	Да
setData(key,data,handler)	Ассоциирует объект с ключем в узле			Нет	Да

3.7. Порядок выполнения лабораторной работы

При выполнении данной лабораторной работы потребуется XML документ *timetable.xml*:

```
<?xml version="1.0"?>
<timetable>
<day dayOfWeek="Monday">
  <lesson type="practical">
    <timeFrom>08.00</timeFrom>
    <timeTo>09.30</timeTo>
    <subject>Deutsch</subject>
    <teacher>Borisova</teacher>
    <room>216</room>
  </lesson>
  <lesson type="lecture">
    <timeFrom>09.40</timeFrom>
    <timeTo>11.10</timeTo>
    <subject>SAP Administration</subject>
    <teacher>Egorov</teacher>
    <room>384</room>
  </lesson>
  <lesson type="practical">
    <timeFrom>11.20</timeFrom>
    <timeTo>12.50</timeTo>
    <subject>SAP Administration</subject>
    <teacher>Petrov</teacher>
    <room>384</room>
  </lesson>
</day>
</timetable>
```

1. Создание JavaScript сценария загрузки XML документа.

Создайте текстовый файл *loadxmldoc.js*, содержащий описание функции загрузки XML документа:

```
function loadXMLDoc(dname)
{
try //Internet Explorer
{
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
}
catch(e)
{
try //Firefox, Mozilla, Opera, etc.
{

xmlDoc=document.implementation.createDocument("", "",null);
}
catch(e) {alert(e.message)}
}
try
{
xmlDoc.async=false;
xmlDoc.load(dname);
return(xmlDoc);
}
catch(e) {alert(e.message)}
return(null);
}
```

и сохраните его в той же папке, где находится файл *timetable.xml*.

Код вызова этой функции может выглядеть следующим образом:

```
<html>
<head>
<script type="text/javascript" src="loadxmldoc.js">
</script>
</head>

<body>
<script type="text/javascript">
xmlDoc=loadXMLDoc("timetable.xml");
document.write("xmlDoc is loaded, ready for use");
</script>
</body>
</html>
```

2. Перемещение по дереву узлов.

Подготовьте следующую HTML страницу:

```
<html>
<head>
<script type="text/javascript" src="loadxml.doc.js">
</script>
</head>
<body>
<script type="text/javascript">
xmlDoc = loadXMLDoc("timetable.xml");
x = xmlDoc.getElementsByTagName("subject");
for (i=0; i<x.length; i++)
{
document.write(x[i].childNodes[0].nodeValue);
document.write("<br />");
}
</script>
</body>
</html>
```

После загрузки страницы в браузере можно будет увидеть следующий результат:

Deutsch	Administration
SAP	
SAP Administration	

3. Изменение значения элемента.

Следующий пример демонстрирует изменение значения элемента `<subject>`:

```
xmlDoc=loadXMLDoc("timetable.xml");
x=xmlDoc.getElementsByTagName("subject")[0].childNodes[0];
x.nodeValue="Java programming";

x = xmlDoc.getElementsByTagName("subject");
for (i=0; i<x.length; i++)
{
document.write(x[i].childNodes[0].nodeValue);
document.write("<br />");
}
```

Внесите соответствующие изменения в предыдущую страницу и загрузите ее в браузере.

4. Перемещение по узлам дерева с использованием отношений между ними.

Следующий код показывает, как используя отношения *firstChild* и *nextSibling* можно получить для текущего узла список его дочерних узлов:

```

x = xmlDoc.getElementsByTagName("lesson")[0].childNodes;
y = xmlDoc.getElementsByTagName("lesson")[0].firstChild;

for (i = 0; i < x.length; i++)
{
    if (y.nodeType == 1)
    {
        document.write(y.nodeName + "<br />");
    }
    y=y.nextSibling;
}
    
```

Внесите необходимые изменения в *html* страницу и загрузите ее в браузер.

3.8. Контрольное задание

В приведенном ниже XML документе описана экзаменационная ведомость:

```

<gradeReport id="120851">
<date>10-06-2008</date>
<subject>Computer Science Fundamentals</subject>
<examiner>prof.Litvinov</examiner>
  <gradeList>
    <gradeRecord id="1">
      <student>Ivanov</student>
      <grade>4</grade>
    </gradeRecord>
    <gradeRecord id="2">
      <student>Petrov</student>
      <grade>3</grade>
    </gradeRecord>
    <gradeRecord id="3">
      <student>Sidorov</student>
      <grade>5</grade>
    </gradeRecord>
  </gradeList>
</gradeReport>
    
```

1. Используя методы DOM XML, сформируйте HTML страницу, содержащую таблицу из трех столбцов: *номер, студент, оценка*.
2. Используя методы DOM XML, замените цифровые значения оценок их словесными эквивалентами, например "4" на "good".

4. ЛАБОРАТОРНАЯ РАБОТА №4: ФОРМАТИРОВАНИЕ И ПРЕОБРАЗОВАНИЕ XML ДОКУМЕНТА С ПОМОЩЬЮ XSL. XSLT ПРЕОБРАЗОВАНИЕ XML ДОКУМЕНТА

Цель работы

Знакомство с методами форматирования и преобразования XML документов на основе XSLT преобразований.

Теоретический материал

XSLT можно определить следующим образом:

- XSLT обозначает *XSL Transformations*.
- XSLT является самой важной частью XSL преобразования
- XSLT позволяет преобразовывать один XML в другой XML документ.
- XSLT использует XPath для перемещения по структуре XML документа.
- XSLT является W3C рекомендацией.

XSLT используется для преобразования XML документа в другой XML документ либо в документ другого распознаваемого браузерами типа, например HTML или XHTML. Обычно XSLT делает это, преобразуя каждый XML элемент в соответствующий ему (X)HTML элемент.

С помощью XSLT можно добавлять или удалять элементы и атрибуты в результирующем документе. Также возможна перегруппировка и сортировка элементов, фильтрация элементов при отображении и многое другое.

Фактически XSLT преобразует исходное XML дерево в результирующее XML дерево.

XSLT использует XPath для поиска информации в XML документе, т.е. XPath является инструментом навигации по элементам

и атрибутам XML документов.

В процессе преобразования XSLT использует XPath для поиска частей исходного документа, соответствующих одному или более заданных шаблонов. Когда соответствие найдено XSLT преобразует найденную часть исходного документа в соответствующую часть результирующего документа.

XSLT является W3C рекомендацией с ноября 1999 года.

подавляющее большинство браузеров имеют поддержку XML и XSLT.

Internet Explorer

Начиная с 6 версии, *Internet Explorer* поддерживает XML, пространства имен, CSS, XSLT и XPath. Версия 5 не совместима с официальной *W3C XSL Рекомендацией*.

Mozilla Firefox

Начиная с версии 1.0.2, *Firefox* поддерживает XML и XSLT (CSS).

Mozilla

Mozilla содержит *Expat for XML* парсер поддерживает отображение XML+CSS. Также имеет поддержка пространства имен. Реализует XSLT преобразования.

Netscape

Начиная с версии 8, *Netscape* использует в качестве ядра *Mozilla*, и поэтому имеет такую же поддержку XML/XSLT.

Opera

Начиная с версии 9, *Opera* поддерживает XML и XSLT (CSS). Версия 8 поддерживает только XML+CSS.

4.1. Объявление XSL

Корневым элементом, указывающим на то, что документ является XSL таблицей стилей является следующий:

```
<xsl:stylesheet>
или полностью равноценный ему
<xsl:transform>
```

В соответствии с W3C XSLT Рекомендацией корректный способ объявления таблицы стилей XSL выглядит следующим образом:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

или

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
```

Для того чтобы XSLT элементы, атрибуты и характеристики были доступны в начале документа необходимо объявить пространство имен XSLT:

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

указывающее на официальное пространство имен W3C XSLT. При этом также следует указать атрибут *version="1.0"*.

4.2. Реализация преобразования с помощью сценария.

XSLT преобразование из XML в XHTML выполняемое самими браузерами на основе таблицы стилей XSL является не всегда желательным, поскольку может поддерживаться не всеми браузерами.

Использование в качестве альтернативы JavaScript позволяет:

- Выполнять проверку типа браузера
- Использовать подходящие таблицы стилей в зависимости от типа браузера и потребностей пользователей.

Другим решением для браузеров, не поддерживающих XSLT является преобразование XML в XHTML на веб-сервере.

4.3. Порядок выполнения лабораторной работы

1. Реализация XSLT преобразования XML документа в XHTML средствами браузера.

- 1) Создайте файл *ttable.xml* следующего содержания:


```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="ttable.xsl"?>
<timetable>
  <lesson>
    <timeFrom>09.30</timeFrom>
    <timeTo>11.00</timeTo>
    <subject>Deutsch</subject>
    <teacher>Borisova</teacher>
  </lesson>
  <lesson>
    <timeFrom>11.10</timeFrom>
    <timeTo>12.20</timeTo>
    <subject>SAP Administration</subject>
    <teacher>Petrov</teacher>
  </lesson>
  <lesson>
    <timeFrom>12.40</timeFrom>
    <timeTo>14.00</timeTo>
    <subject>SAP Administration</subject>
    <teacher>Ivanov</teacher>
  </lesson>
  <lesson>
    <timeFrom>14.00</timeFrom>
    <timeTo>15.20</timeTo>
    <subject>Wen-technology</subject>
    <teacher>Loktev</teacher>
  </lesson>
</timetable>
```

- 2) Подготовьте для него соответствующий файл таблицы стилей (*ttable.xsl*):

```

<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet                                version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
    <h2>My academical timetable</h2>
    <table border="1">
      <tr bgcolor="#a5abfa">
        <th align="left">Time From</th>
        <th align="left">Subject</th>
        <th align="left">Teacher</th>
      </tr>
      <xsl:for-each select="timetable/lesson">
        <tr>
          <td><xsl:value-of select="timeFrom"/></td>
          <td><xsl:value-of select="subject"/></td>
          <td><xsl:value-of select="teacher"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>
    
```

Первая строка в файле *ttable.xml* вида:

```
<?xml-stylesheet type="text/xsl" href="ttable.xsl"?>
```

представляет собой ссылку на подходящую таблицу стилей.

В данном примере для описания шаблона был использован элемент `<xsl:template>`.

Атрибут *match* применяется для связывания XML элемента с шаблоном. Значение атрибута *match* является выражение *XPath*. В данном случае *match="/"* указывает на весь документ.

Содержимое элемента `<xsl:template>` описывает фрагмент HTML кода в выходном документе. Элемент `<xsl:value-of>` может быть использован для извлечения значения XML элемента и добавления его в выходной поток при преобразовании. Значение атрибута *select* является выражением *XPath*, которое напоминает нотацию, используемую в файловой системе; знак *(/)* указывает

```
<html>
<head>
<script>
function loadXMLDoc(fname)
```

на обращение к подкаталогу.

Элемент `<xsl:for-each>` позволяет выбирать каждый элемент XML указанного множества узлов.

3) При наличии подходящего браузера можно будет увидеть следующий результат после загрузке файла *ttable.xml*:

My academical timetable

Time From	Subject	Teacher
09.30	Deutsch	Borisova
11.10	SAP Administration	Petrov
12.40	SAP Administration	Ivanov
14.00	Wen-technology	Loktev

Для фильтрации вывода элементов можно добавить атрибут элемента `<xsl:for-each>`, задающий критерий отбора элементов. Например:

```
<xsl:for-each select="timetable/lesson[subject='SAP Administration']">
```

Проверьте работу данного фильтра.

Допустимо использование следующих операторов для описания фильтра:

- **=** (равно)
- **!=** (не равно)
- **<** (меньше чем)
- **>** (больше чем)

2. Реализация преобразования с помощью *JavaScript*.

1) Подготовьте *html* файл, содержащий код:

```
{
var xmlDoc;
// Код для IE
if (window.ActiveXObject)
{
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
}
// Код для Mozilla, Firefox, Opera и др.
else if (document.implementation
&& document.implementation.createDocument)
{
xmlDoc=document.implementation.createDocument("", "", null);
}
else
{
alert('Your browser cannot handle this script!');
}
xmlDoc.async=false;
xmlDoc.load(fname);
return(xmlDoc);
}

function displayResult()
{
xml=loadXMLDoc("ttable.xml");
xsl=loadXMLDoc("ttable.xsl");
// Код для IE
if (window.ActiveXObject)
{
ex=xml.transformNode(xsl);
document.getElementById("example").innerHTML=ex;
}
// Код для Mozilla, Firefox, Opera и др.
else if (document.implementation
&& document.implementation.createDocument)
{
xsltProcessor=new XSLTProcessor();
xsltProcessor.importStylesheet(xsl);
resultDocument = xsltProcessor.transformToFragment(xml,document);
document.getElementById("example").appendChild(resultDocument);
}
}
```

```
</script>
</head>
<body id="example" onLoad="displayResult()" >
</body>
</html>
```

Функция *loadXMLDoc()* загружает XML и XSL файлы в зависимости от типа браузера.

Функция *displayResult()* используется для отображения XML файла в стиле, задаваемом XSL файлом. Она выполняет следующие действия:

- загружает XML и XSL.
 - определяет тип браузера.
 - если браузер поддерживает *ActiveX* объекты:
 - с помощью метода *transformNode()* таблица стилей XSL применяется к XML документу.
 - формируется тело текущего документа.
 - если браузер клиента не поддерживает *ActiveX* объекты:
 - создается новый объект *XSLTProcessor* и в него импортируется XSL файл.
 - с помощью метода *transformToFragment()* таблица стилей XSL применяется к XML документу.
 - формируется тело текущего документа.
- 2) Загрузите этот документ в веб-браузере.

3. Реализация преобразования с помощью JavaScript

- 1) Подготовьте файл, содержащий следующий код сценария на языке ASP:

```
<%
'Load XML
set xml = Server.CreateObject("Microsoft.XMLDOM")
xml.async = false
xml.load(Server.MapPath("ttable.xml"))

'Load XSL
set xsl = Server.CreateObject("Microsoft.XMLDOM")
xsl.async = false
xsl.load(Server.MapPath("ttable.xsl"))

'Transform file
Response.Write(xml.transformNode(xsl))
%>
```

В начале сценария создается экземпляр парсера *Microsoft XML parser* (XMLDOM), и XML файл загружается в память. Далее создается еще один экземпляр парсера, и XSL файл загружается в память. В последней строке выполняется собственно преобразование XML файла с использованием XSL файла в XHTML, который отправляется обратно браузеру.

2) Проверьте работу сценария в веб-браузере.

4.4. Контрольное задание

В приведенном ниже XML документе описана экзаменационная ведомость:

```
<gradeReport id="120851">
<date>10-06-2008</date>
<subject>Computer Science Fundamentals</subject>
<examiner>prof.Litvinov</examiner>
  <gradeList>
    <gradeRecord id="1">
      <student>Ivanov</student>
      <grade>4</grade>
    </gradeRecord>
    <gradeRecord id="2">
      <student>Petrov</student>
      <grade>3</grade>
    </gradeRecord>
    <gradeRecord id="3">
      <student>Sidorov</student>
      <grade>5</grade>
    </gradeRecord>
  </gradeList>
</gradeReport>
```

Постройте для него XSL файл, необходимый для XSLT преобразования исходного XML документа в HTML страницу, содержащую заголовок ведомости и таблицу оценок со столбцами: *номер, студент, оценка*.

5. ЛАБОРАТОРНАЯ РАБОТА №5: РЕАЛИЗАЦИЯ АСИНХРОННОГО ВЗАИМОДЕЙСТВИЯ ВЕБ-БРАУЗЕРА С ВЕБ-СЕРВЕРОМ С ПОМОЩЬЮ ТЕХНОЛОГИИ AJAX

Цель работы

Ознакомление с принципами асинхронного взаимодействия между веб-клиентом и веб-сервером в рамках технологии AJAX.

Теоретический материал

AJAX = Asynchronous JavaScript and XML.

AJAX – не является новым языком программирования, но технология создания улучшенных, более быстрых и в большей степени интерактивных веб-приложений.

JavaScript сценарий посредством AJAX может непосредственно взаимодействовать с сервером с помощью объекта *XMLHttpRequest*. Использование данного объекта обмен данными с веб-сервером могут происходить без перезагрузки страницы.

AJAX позволяет веб-страницам запрашивать небольшие объемы информации с сервера нежели целиком всю страницу в результате асинхронной передачи данных (в рамках HTTP протокола) между браузером и сервером.

AJAX не зависит от программного обеспечения веб-сервера и основан на следующих веб-стандартах:

- JavaScript
- XML
- HTML
- CSS

Поскольку эти веб-стандарты четко определены и имеют поддержку в наиболее распространенных веб-браузерах, то AJAX приложения являются браузеро- и платформи-независимыми.

Популярность AJAX связана с появлением сервиса *Google Suggest* в 2005 году. Данный сервис на основе объекта *XMLHttpRequest* предоставляет в распоряжение пользователя достаточно динамический веб-интерфейс. В процессе ввода символов пользователем в поле поискового запроса JavaScript отправляет их на сервер и получает от него список подсказок.

Объект *XMLHttpRequest* поддерживается в *Internet Explorer* (начиная с 5 версии и выше), *Safari 1.2*, *Mozilla 1.0 | Firefox*, *Opera 8+* и *Netscape 7*.

5.1. Порядок выполнения лабораторной работы

В данной лабораторной работе рассматривается пример системы, имитирующей работу сервиса *Google Suggest* на основе AJAX.

1. Реализация клиентской части

Предполагается, что пользователь может вводить в текстовое поле формы название автомобильной марки, получая при этом динамически список вариантов названий, соответствующих уже введенным символам, без перезагрузки страницы.

1) Создайте следующую веб-страницу:

```
<html>
<head>
<script src="chint.js"></script>
</head>
<body>
<form>
First Name:
<input type="text" id="txt1" onkeyup="showHint(this.value)">
</form>
<p>Suggestions: <span id="txtHint"></span></p>
</body>
</html>
```

Как видно из кода, при наступлении события *onkeyup* (отжатие клавиши) вызывается обработчик *showHint()*.

2) В файле *chint.js* сохраните следующий код обработчика:

```
var xmlhttp;
function showHint(str)
{
if (str.length==0)
{
document.getElementById("txtHint").innerHTML="";
return;
}
xmlhttp=GetXmlHttpRequest();
if (xmlhttp==null)
{
alert ("Your browser does not support AJAX!");
return;
}
```



```

var url = "ghint.php";
url = url + "?q=" + str;
url = url + "&sid=" + Math.random();
xmlHttp.onreadystatechange = stateChanged;
xmlHttp.open("GET", url, true);
xmlHttp.send(null);
}
function stateChanged()
{
if (xmlHttp.readyState==4)
{
document.getElementById("txtHint").innerHTML =
xmlHttp.responseText;
}
}
function GetXmlHttpRequest()
{
var xmlHttp=null;
try
{
// Firefox, Opera 8.0+, Safari
xmlHttp = new XMLHttpRequest();
}
catch (e)
{
// Internet Explorer
try
{
xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
}
catch (e)
{
xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
}
}
return xmlHttp;
}
    
```

Из кода видно, что каждый раз, когда вводится символ, вызывается функция-обработчик. Если при этом содержимое текстового поля формы непустое ($str.length > 0$), функция выполняет следующие действия:

- Формируется *url* для отправки веб-серверу
- Добавляется значение параметра *q*, равное содержанию текстового поля, к *url*
- Добавляется к *url* случайное число для предотвращения кеширования
- Создается объект *XMLHTTP*, при этом указывается функция (*stateChanged*) подлежащая исполнению при наступлении события ввода символа
- Открывается объект *XMLHTTP* с указанным значением *url*
- Отправляется HTTP запрос веб-серверу

Если поле ввода пустое, происходит очистка содержимого раздела *txtHint* на веб-странице.

Ключевым моментом в данной системе является использование объекта *XMLHttpRequest*.

Данный объект по-разному создается в различных браузерах. Так, *Internet Explorer* для этого использует *ActiveXObject*, в то время как остальные браузеры используют встроенный в JavaScript объект *XMLHttpRequest*.

Для поддержки работы системы в разных браузерах использован оператор *"try-catch"*.

- Сначала делается попытка создать объект *XMLHttpRequest* для браузеров *Firefox*, *Opera* или *Safari*:
xmlHttp = new XMLHttpRequest();
- В случае неудачи, делается следующая попытка создания объекта для *Internet Explorer 6.0+* :
xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
- Если это также не удастся, то делается попытка создания объекта уже для *Internet Explorer 5.5+* :
xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
- В случае, если ни одна из этих попыток не принесла успеха, выдается сообщение об отсутствии поддержки AJAX браузером.

2. Реализация серверной части.

```
<?php
header("Cache-Control: no-cache, must-revalidate");
// Прошедшая дата
header("Expires: Mon, 1 Sep 2008 07:30:00 GMT");
// Инициализация массива названий
$a[]="Audi";
```

```

$a[]="BMW";
$a[]="Buick";
$a[]="Chevrolet";
$a[]="Citroen";
$a[]="Dodge";
$a[]="Ferrari";
$a[]="Fiat";
$a[]="Ford";
$a[]="Honda";
$a[]="Hyundai";
$a[]="Cherokee";
$a[]="Cherry";
$a[]="Lada";
$a[]="Lamborghini";
$a[]="Lincoln";
$a[]="Mazda";
$a[]="Mercedes";
$a[]="Mitsubishi";
$a[]="Nissan";
$a[]="Opel";
$a[]="Peugeot";
$a[]="Plymoth";
$a[]="Pontiac";
$a[]="Renault";
$a[]="Rover";
$a[]="Saab";
$a[]="Subaru";
$a[]="Suzuki";
$a[]="Toyota";
$a[]="Volkswagen";
$a[]="Volvo";
//получение параметра q из URL
$q = $_GET["q"];
//поиск соответствий из массива если длина q > 0
if (strlen($q) > 0)
{
    $hint = "";
    for($i = 0; $i<count($a); $i++)
    {
        if (strtolower($q) == strtolower(substr($a[$i],0,strlen($q))))
        {
            if ($hint == "")

```

```

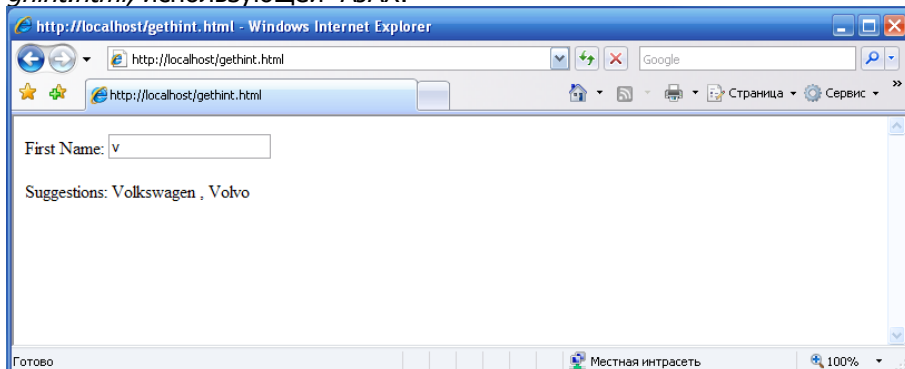
        {
            $hint=$a[$i];
        }
    else
    {
        $hint=$hint." , ".$a[$i];
    }
    }
}
}
}
// Возврат строки "нет вариантов" если соответствий не
найдено
// либо найденное соответствие
if ($hint == "")
{
    $response = "no suggestion";
}
else
{
    $response = $hint;
}
//вывод результата
echo $response;
?>

```

3. Проверка работоспособности системы.

Проверьте с помощью подходящего веб-браузера работу системы.

На скриншоте показана работа в браузере с веб-страницей *ghint.html*, использующей AJAX:



5.2. Контрольное задание

С использованием PHP-сценария разработайте модифицированный вариант калькулятора для четырех арифметических операций, в котором происходит асинхронная передачи данных между браузером и сервером (с помощью AJAX).

6. ЛАБОРАТОРНАЯ РАБОТА №6: РАЗРАБОТКА CGI-ПРИЛОЖЕНИЙ НА PERL И PHP

Цель работы

Ознакомление с:

- 1) основами языков разработки веб-сценариев на языках Perl и PHP;
- 2) синтаксисом языков Perl и PHP;
- 3) реализацией обработки данных, полученных от клиентского приложения, на стороне веб-сервера.

6.1. Основы разработки сценариев на языке Perl

Программа на языке Perl состоит из *деклараций* и *операторов*. Любой текст, начиная с символа "#" и до конца строки, считается *комментарием* и игнорируется.

Для *переменных* деклараций не требуется. До тех пор пока пока им не будет присвоено какое-либо конкретное значение, они просто содержат неопределенное значение *undef*. *Декларации* могут располагаться в любом месте программы, т. к. обрабатываются на этапе компиляции, предшествующем этапу исполнения программы.

Операторы языка Perl подразделяются на *простые* и *составные*. *Составные* операторы состоят из блоков, заключенных в фигурные скобки. В отличие от языка C, фигурные скобки в составных операторах обязательны, даже если в них заключен только один оператор. Операторы разделяются точкой с запятой.

Знакомство с Perl можно начать со сценария *сценарий 1*.

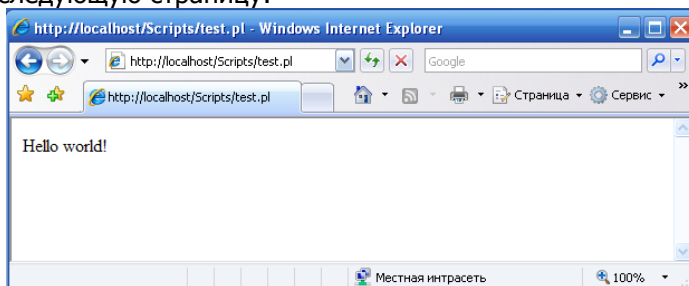
Сценарий 1. Вывод строки приветствия на Perl.

```
#!c:/perl/bin/perl
print "Content-type:text/html\n\n";
print "Hello world!";
```

Первая строка сценария, оформленная в виде комментария, указывает на физическое размещение интерпретатора языка Perl. Остальные строки фактически формируют ответ веб-сервера:

- Во второй строке в выходной поток передается поле *Content-type* заголовка ответа сервера, и в конце вставляется пустая строка, отделяющая заголовок от тела ответа сервера.
- В последней строке помещается содержимое тела ответа сервера.

В результат выполнения сценария данного сценария получим следующую страницу:



В следующем примере будет рассмотрен сценарий, считывающий и обрабатывающий данные, полученные веб-сервером из запроса клиента. Исходные данные должны вводиться пользователем в поля формы веб-страницы, загруженной в веб-браузере.

Сценарий 2. *Вывод списка параметров, полученных сервером в запросе от клиента.*

```

#!c:/perl/bin/perl

print "Content-type: text/html\n\n";
print "<HTML><BODY>\n";

$method = $ENV{'REQUEST_METHOD'};

if ($method eq 'POST')
{
    $length = $ENV{'CONTENT_LENGTH'};
    read(STDIN, $qstr, $length);
}
else
{
    if ($method eq 'GET') { $qstr = $ENV{'QUERY_STRING'}; }
    else
    {
        print "Method \"$method\" is not supported
</BODY></HTML>";
        #exit(0);
    }
}

print "<P>Метод = ", $method;
print "<p>Строка параметров: <p>\n";
print $qstr;

# обратная перекодировка

$qstr =~ tr/+/ /;
$qstr =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;

# выделение списка параметров

print "<p>А теперь отдельные поля:<p>";
@pars = split(/&/, $qstr);

# определение размера массива

$n_pars = @pars;

# выделение имени и значения для каждого параметра

for ($i=0; $i<$n_pars; $i++)
{

```

```
# выделение списка из двух переменных $name и $value

($name, $value) = split(/=/, $pars[$i]);
print "Параметр <B>", $name, "</B> равен <I>", $value,
"</I><br>";
}
print "</HTML></BODY>\n";
```

Данный сценарий ориентирован на передачу данных из веб-формы одним из основных методов: *GET* или *POST*. Поскольку передача данных в этих методах отличается, то в сценарии сначала определяется метод передачи данных путем обращения к переменной окружения *REQUEST_METHOD*, значение которой доступно сценарию через одноименный элемент ассоциативного массива (хэша) *ENV*, содержащего значения всех переменных окружения.

После определения использовавшегося в запросе клиента метода, выбирается адекватный способ чтения параметров, полученных из веб-формы:

- В случае метода *POST* определяется общий размер переданных данных (в байтах) из переменной окружения *CONTENT_LENGTH*, а затем блок данного размера считывается из входного потока *STDIN* с помощью функции *read*.
- В случае метода *GET* данные доступны в переменной окружения *QUERY_STRING*.
- Если метод запроса не совпадает ни с одним из рассмотренных выше или его значение не определено, то происходит принудительное завершение сценария с выдачей соответствующего сообщения.

Поскольку в рамках протокола HTTP символы, отличные от латинских букв и цифр передаются в виде шестнадцатиричных кодов (пробелы заменяются на '+'), требуется предварительное обратное преобразование полученных данных с помощью операторов замены *s///* и *tr//* с использованием шаблонов в виде регулярных выражений.

Далее с помощью функции *split* происходит разделение

блока символов на подстроки по заданному разделителю:

- *Пары*, описывающие *имя* параметра и его *значение* разделяются с помощью символа `&`. Результат разделения помещается в скалярный массив @pairs.
- Внутри пары *имя* и *значение* разделяются символом `='`.

Рассмотренный пример может быть взят за основу для любого сценария, обрабатывающего данные из веб-формы, поскольку позволяет получить исходные данные, полученные от клиента. Расширение сценария сводится к добавлению кода обработки полученных данных и формирования итогового документа, возвращаемого веб-сервером клиенту.

Рассмотрим для примера сценарий, выполняющий четыре арифметические операции над целыми числами, которые пользователь вводит через поля веб-формы в браузере.

Сценарий 3. *Калькулятор арифметических операций для целых операндов.*

```

#!c:/perl/bin/perl

print "Content-type: text/html\n\n";
print "<HTML><BODY>\n";

$method = $ENV{'REQUEST_METHOD'};

if ($method eq 'POST')
{
    $length = $ENV{'CONTENT_LENGTH'};
    read(STDIN, $qstr, $length);
}
else
{
    if ($method eq 'GET') { $qstr = $ENV{'QUERY_STRING'}; }
    else
    {
        print "Method ".$method." is not supported
</BODY></HTML>";
        exit(0);
    }
}

$qstr =~ tr/+// ;
$qstr =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;

@pars = split(/&/, $qstr);
$n_pars = @pars;

foreach $par (@pars)
{
    ($name, $value) = split(/=/, $par);
    $opers{$name} = $value;
}

$op1 = int($opers{'A'});
$op2 = int($opers{'B'});
$op = $opers{'op'};
    
```

```

switch:
{
    if ($op eq '+') { $res = $op1 + $op2; last switch; }
    if ($op eq '-') { $res = $op1 - $op2; last switch; }
    if ($op eq '*') { $res = $op1 * $op2; last switch; }
    if ($op eq '/') {
        if ($op2 == 0)
        {
            print "    Divide    by    zero!
</BODY></HTML>";
            exit(0);
        }
        else
        {
            $res = $op1 / $op2; last switch;
        }
    }
    {
        print "Operator "$op." is not supported
</BODY></HTML>";
        exit(0);
    }
}

print "Result: ".$op1.$op.$op2.' = '.$res;
print "                "<p><a
href="\$.ENV{'HTTP_REFERER'}."\'>Back</a></p>";

print "</HTML></BODY>\n";
    
```

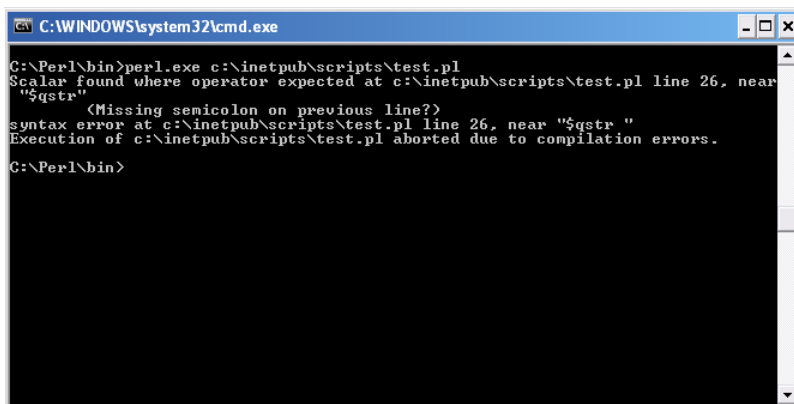
В данном примере из строки запроса извлекаются 3 параметра с именами "A", "B" и "op", после чего в зависимости от значения третьего параметра выполняется соответствующая арифметическая операция над первыми двумя параметрами.

Operand1:	<input type="text" value="34"/>
Operand2:	<input type="text" value="12"/>
Operation:	<input type="text" value="*"/>
<input type="button" value="Calculate!"/>	

В итоге формируется *html* страница с результатом вычисления и обратной ссылкой на исходную страницу с формой. Для определения адреса исходной страницы используется значение переменной окружения *HTTP_REFERER*:

Result: 34*12 = 408
[Back](#)

Для поиска синтаксических ошибок в сценариях на Perl можно использовать непосредственный запуск интерпретатора в командной строке, который находится в установочном каталоге Perl:



```

C:\WINDOWS\system32\cmd.exe
C:\Perl\bin>perl.exe c:\inetpub\scripts\test.pl
Scalar found where operator expected at c:\inetpub\scripts\test.pl line 26, near
"$qstr"
  <Missing semicolon on previous line?>
syntax error at c:\inetpub\scripts\test.pl line 26, near "$qstr "
Execution of c:\inetpub\scripts\test.pl aborted due to compilation errors.
C:\Perl\bin>
    
```

6.2. Основы разработки сценариев на языке PHP

PHP - сценарии могут размещаться в отдельном файле (с расширением *.php*) или встраиваются непосредственно в HTML документ.

Существует несколько способов внедрения кода PHP в HTML документы:

- С помощью открывающего тега `<?php` и закрывающего тега `?>`.
- С помощью коротких тегов `<? и ?>`. Данная возможность доступна только при специальной настройке.
- С помощью тэгов `<script language="php">` и `</script>`
- Путем использования echo тэгов в стиле ASP: `<% и %>`. Такая возможность доступна при соответствующей конфигурационной настройке.

В дальнейшем в примерах будет использоваться первый из вариантов внедрения PHP кода.

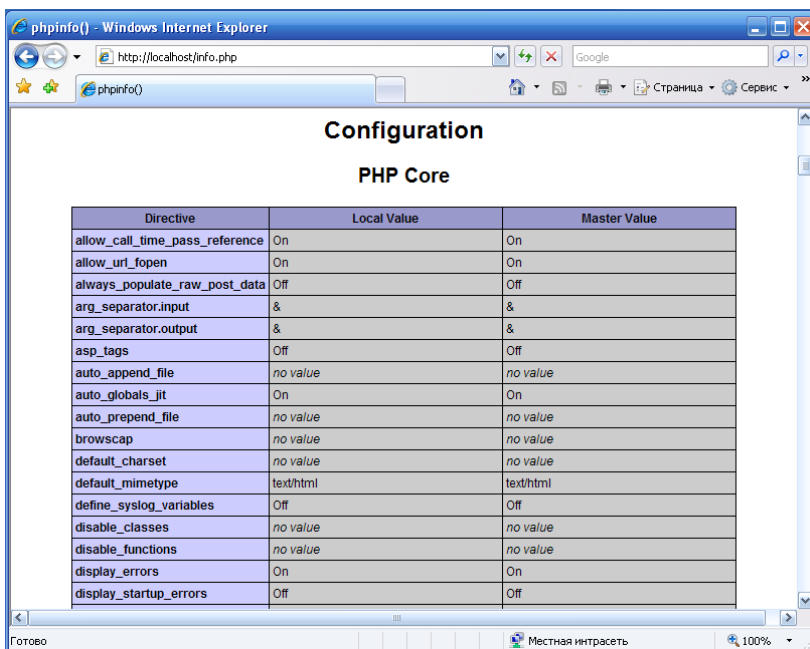
Код, который находится внутри указанных тэгов, обрабатывается интерпретатором PHP, весь остальной код остается неизменным.

Для того, чтобы увидеть текущие настройки PHP, и для проверки его работоспособности полезно использовать специальную функцию `phpinfo()`:

Сценарий 4. Использование `php`-функции `phpinfo()`.

```
<?php  
phpinfo();  
?>
```

После выполнения этого кода, в веб-браузере можно будет увидеть примерно следующее (показана небольшая часть):



Directive	Local Value	Master Value
allow_call_time_pass_reference	On	On
allow_url_fopen	On	On
always_populate_raw_post_data	Off	Off
arg_separator.input	&	&
arg_separator.output	&	&
asp_tags	Off	Off
auto_append_file	no value	no value
auto_globals_jit	On	On
auto_prepend_file	no value	no value
browscap	no value	no value
default_charset	no value	no value
default_mimetype	text/html	text/html
define_syslog_variables	Off	Off
disable_classes	no value	no value
disable_functions	no value	no value
display_errors	On	On
display_startup_errors	Off	Off

В целом же листинг содержит информацию об установленных опциях и расширениях PHP, версии PHP, информацию о веб-сервере и переменных окружения, информацию о версии ОС, путях, настройках конфигурационных переменных, полях заголовка HTTP и PHP лицензии.

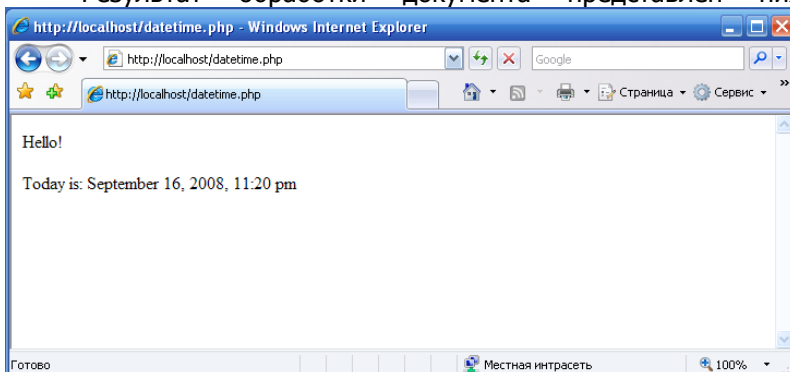
Следующий пример демонстрирует вариант с внедрением PHP кода в HTML:

Сценарий 5. Внедрение PHP кода в HTML документ.

```

<html>
<body>
<p>Hello! </p>
<p>Today is:
<?php
    $today = date("F j, Y, g:i a");
    echo($today);
?>
</p>
</body>
</html>
    
```

Результат обработки документа представлен ниже



Одной из главных задач, решаемых с помощью PHP, является обработка данных, получаемых от пользователя через веб-формы. Рассмотрим, каким образом в PHP реализуется такая обработка.

Сценарий 6. *Вывод списка параметров, полученных сервером в запросе от клиента.*

```
<?php
    $method = $_SERVER["REQUEST_METHOD"];

    if ($method == "GET") $query = "_GET";
    elseif ($method == "PUT") $query = "_PUT";
    else die("$method is not supported!");

    print "<p><b>Method</b>: $method </p>";
    print "<p><u>Params:</u></p>";

    foreach ($$query as $name => $value)
    {
        print "<b>$name</b> = <i>$value</i> <br>";
    }
?>
```

В первой строке сценария с помощью переменной окружения *REQUEST_METHOD* из глобального ассоциативного массива *\$_SERVER* определяется метод передачи данных в запросе клиента. В зависимости от выбранного метода переданные данные будут извлекаться либо из глобальной переменной *\$_GET* либо из *\$_PUT*. Если метод отличается от *GET* или *PUT*, либо неопре-

делен, то происходит принудительное завершение работы сценария с выдачей сообщения через вызов функции *die()*. В принципе, можно также использовать глобальный массив *\$_REQUEST*, содержащий внутри себя массивы *\$_GET*, *\$_POST* и *\$_COOKIE*, позволяющий избавиться от проверки метода передачи.

Конструкция вида *\$\$query* демонстрирует косвенное обращение к переменной, т.е. переменная *\$query* содержит идентификатор другой переменной, и для обращения к ней необходимо добавить еще один знак *\$*.

Обе переменные *\$_GET* и *\$_POST* являются ассоциативными массивами, поэтому для перебора элементов был использован специальный оператор

foreach (*имя_массива as ключ => значение*)

Следует обратить внимание на то, что внутри строковых констант, ограниченных символами " " можно вставлять переменные. После обработки такой строки интерпретатором вместо переменной вставляется ее фактическое значение. Также для конкатенации строк можно использовать оператор *`.`* В целом можно сказать, что в PHP можно использовать операторы ветвления, выбора и циклов аналогичные тем, что используются в языке C.

Код сценария, реализующего четыре арифметические операции над целыми числами представлен в листинге ниже:

Сценарий 7. *Калькулятор арифметических операций для целых операндов.*


```
<?php
    $method = $_SERVER["REQUEST_METHOD"];

    if ($method == "GET") $query = "_GET";
    elseif ($method == "PUT") $query = "_PUT";
    else die("$method is not supported!");

    $q = $$query;

    $a = $q["A"];
    $b = $q["B"];
    $op = $q["op"];

    switch ($op)
    {
        case '+': $result = (int)$a + (int)$b; break;
        case '-': $result = (int)$a - (int)$b; break;
        case '*': $result = (int)$a * (int)$b; break;
        case '/': {
            if ($b == '0') die("divide by zero!");
        }
        default: die("operator $op is not defined");
    }
    print "<p>Result: $a $op $b = $result </p>";
    print " <p><a
href=".$_SERVER["HTTP_REFERER"].">Back</a></p>";
?>
```

В PHP имеется широкий диапазон функция для работы с файлами. Например, следующий пример демонстрирует чтение файла, в котором находится выполняемый PHP код:

Сценарий 8. Чтение файла, содержащего PHP код, выполняемого сценария.

```
<?php
$fh = fopen("read.php","r");
if (!$fh) die("Cannot open file");

while (!feof ($fh))
{
    $line = fgets($fh);
    echo $line,"<br>";
}
fclose($fh);
?>
```

В данном примере используются практически те же самые функции, что и в языке С:

- **fopen** (*путь_к_файлу, тип_доступа*) – открытие файла;
- **feof** (*указатель_на_файл*) – проверка на наличие признака конца файла;
- **fgets** (*указатель_на_файл*) – чтение строки из файла;
- **fclose** (*указатель_на_файл*) – закрытие файла.

Для записи данных в файл можно использовать функцию **fputs** (*указатель_на_файл, строка*). В следующем примере сгенерированные функцией *rand()* псевдослучайные числа сохраняются в файле *rand.dat*:

Сценарий 9. *Сохранение в файле последовательности псевдослучайных чисел.*

```
<?php
$n = 10;
$fh = fopen("rand.dat","w");
if (!$fh) die("Cannot open file");
srand();
for ($i=0; $i < $n; $i++)
{
    $d = rand(0,100);
    fputs($fh,"$d\n");
}
fclose($fh);
?>
```

Перед выполнением данного сценария следует правильно настроить права доступа для веб-сервера к директории, в которой сохраняется файл *rand.dat*.

6.3. Порядок выполнения лабораторной работы

Для практического изучения примеров в данной лабораторной работе необходимо наличие установленных и правильно сконфигурированных интерпретаторов языков *Perl* и *PHP*.

Часть 1. Язык Perl.

1. Создайте файл с текстом *сценария 1*. Файл должен иметь расширение *pl*, и размещаться в директории *Scripts* (или *cgi-bin*). Проверьте настройки доступа к папке для веб-сервера, который должен иметь право на выполнение сценариев в этой папке.

При сохранении файла в редакторе также следует также выбрать правильную кодировку символа переноса строки. Кроме того, первая строка сценария должна содержать правильный путь к директории, в которой установлен интерпретатор языка Perl (обычно это файл *perl.exe* или подобный ему).

Если текст сценария содержит синтаксические ошибки, то после попытки его запустить на выполнение веб-сервером, последний вернет клиенту ответ, содержащий код внутренней ошибки сервера. Поэтому перед запуском сценария рекомендуется выполнить его проверку. Для проверки сценария на наличие синтаксических ошибок удобно использовать непосредственный запуск интерпретатора вручную. Для этого в командной строке из директории, указанной в первой строке сценария, необходимо запустить исполняемый модуль (обычно *perl.exe*) с аргументом, являющимся именем файла (с указанием пути), содержащим текст сценария. При наличии синтаксических ошибок в сценарии интерпретатор выдаст сообщения с указанием соответствующих номеров строк в файле, в которых эти ошибки обнаружены. Исправляйте ошибки до тех пор, пока интерпретатор не перестанет выдавать сообщения об ошибках.

Для того, чтобы посмотреть работу сценария, необходимо в браузере набрать его URL по HTTP-протоколу.

2. Подготовьте файл с текстом *сценария 2*.

Для проверки работоспособности данного сценария можно выполнить его непосредственный запуск в веб-браузере через его URL с добавлением строки параметров, например:

http://localhost/Scripts/test.pl?a=2&b=14

Подготовьте HTML страницу, содержащую форму с полями для ввода данных. Вставьте в тэге *<FORM>* атрибут *ACTION* со значением, равным URL сценария, в качестве метода выполнения

запроса укажите в атрибуте *METHOD* значение *GET*. Проверьте работу формы. Сделайте то же самое, но для метода *POST*.

3. Подготовьте файл с текстом *сценария 3* и HTML страницу, содержащую форму с полями для ввода операндов (простые поля для ввода текста с именами 'A' и 'B') и выбора арифметической операции (поле типа 'select' с именем 'op'). Добавьте кнопку типа 'submit' и атрибут *ACTION* со значением, равным URL сценария, в тэге *<FORM>*:

```

<html>
<body>
  <form action='http://localhost/Scripts/test.pl'>
    <p>Operand1:      <input      type='text'
name='A'></p>
    <p>Operand2:      <input      type='text'
name='B'></p>
    <p>Operation:<br>
    <select name='op'>
      <option value='+'>+</option>
      <option value='-'>-</option>
      <option value='*'>*</option>
      <option value='/'>/</option>
    </select></p>
    <input type='submit' value='Calculate!'>
  </form>
</body>
</html>
    
```

Проверьте работу сценария.

Часть 2. Язык PHP.

4. Подготовьте текстовый файл с расширением *PHP* и разместите его в директории в соответствии с конфигурационными настройками интерпретатора *PHP* (параметр *doc_root*).

После запуска сценария вы увидите страницу конфигурации, в которой вы увидите значения *переменных ядра PHP*, установленные библиотеки функций, значения *переменных окружения* и *глобальных переменных PHP*.

Данный сценарий можно использовать для проверки списка передаваемых данных от клиента, если URL сценария использовать в форме (атрибут *ACTION*) или вызывать его напрямую в веб-браузере.

5. Подготовьте файл с текстом *сценария 6* и HTML страницу с

формой для проверки его работы.

Проверьте работу сценария для методов *GET* и *POST*.

6. Подготовьте файл с текстом *сценария 7* и соответствующую HTML страницу с формой, как в аналогичном примере на языке Perl.
7. Подготовьте файл с текстом *сценария 8*. В качестве первого аргумента функции *forep* укажите имя этого файла. Проверьте работу сценария.
8. Подготовьте файл с текстом *сценария 9*. Для выполнения сценария необходимо правильно настроить права доступа для веб-сервера к директории, в которой будет сохраняться файл (должен быть разрешен доступ на запись). Из соображений безопасности рекомендуется для записи создавать отдельную директорию.

Запустите сценарий. Убедитесь, что был создан файл *rand.dat* и просмотрите его содержимое.

6.4. Контрольные задания

1. Извлечение списка слов из текста (сценарий на языке Perl).

1) Подготовьте веб-страницу с формой, содержащей поле для ввода текста и кнопку типа *submit*. Для атрибута *Action* в форме укажите в качестве значения *URL* perl-сценария, например <http://localhost/Scripts/wcount.pl>.

2) Подготовьте сценарий на языке Perl, который извлекает из текста, полученного от клиентского приложения, список слов с помощью функции *split*. В качестве первого аргумента при вызове функции необходимо указать регулярное выражение, задающее список разделителей слов. Также сценарий должен показать общее число найденных слов.

2. Извлечение списка слов из текста (сценарий на языке PHP).

1) Измените веб-страницу с формой, разработанную в предыдущем задании: для атрибута *Action* в форме укажите в качестве значения *URL* PHP-сценария, например <http://localhost/wcount.php>.

2) Подготовьте сценарий на языке PHP, который извлекает из текста, полученного от клиентского приложения, список слов с помощью функции *preg_split*, которая имеет сле-

дующий синтаксис:

```
array preg_split(string pattern, string subject)
```

т.е. возвращает массив, состоящий из подстрок заданной строки *subject*, которая разбита по границам, соответствующим шаблону *pattern*. Шаблон описывается с помощью подходящего регулярного выражения. Также сценарий должен показать общее число найденных слов.

7. ЛАБОРАТОРНАЯ РАБОТА №7: ПРИМЕРЫ РАЗРАБОТКИ RSS-ИСТОЧНИКОВ И RSS-РИДЕРОВ

Цель работы

Введение в технологию RSS. Изучение структуры RSS документов, их генерации и публикации.

7.1. Теоретический материал

RSS – это метод распространения веб-контента с веб-сайта на других веб-сайтах. RSS позволяет выполнять быстрый просмотр новостей и изменений.

На кого ориентирован формат RSS? В первую очередь RSS предназначен для использования на очень часто обновляемых веб-сайтах, например:

- Новостные сайты (списки новостей).
- Сайты компаний (списки новостей и продуктов).
- Календари (списки предстоящих событий и знаменательных дней).
- Изменения сайтов (список обновленных и новых страниц).

Официального стандарта RSS как такового не существует. Реально используются следующие форматы:

- Порядка 50% RSS-потоков используют RSS 0.91.
- Порядка 25% используют RSS 1.0.
- Оставшиеся 25% RSS-потоков распределены между RSS 0.9 и RSS 2.0.

Как работает RSS?

1. Создается RSS документ в виде файла с расширением .xml.
2. Этот файл размещается на веб-сайте.
3. RSS-поток регистрируется в RSS-агрегаторах.

1. Пример RSS документа

Рассмотрим пример RSS-документа.

```

<?xml version="1.0" encoding="windows-1251" ?>
<rss version="2.0">
  <channel>
    <title>My Home Page</title>
    <link>http://www.MyHP.edu</link>
    <description>Free web building tutorials</description>
  <item>
    <title>RSS Demo</title>
    <link>http:// www.MyHP.edu/rss</link>
    <description>New RSS demo on my home
page</description>
  </item>
  <item>
    <title>XML Demo</title>
    <link> www.MyHP.edu/xml</link>
    <description>New XML demo on my home
page</description>
  </item>
</channel>
</rss>
    
```

- В первой строке размещено объявление версии XML и кодировки документа.
- Вторая строка идентифицирует данный документ как RSS документ версии 2.0.
- В третьей строке содержится элемент *<channel>*, описывающий RSS поток. Он в свою очередь содержит дочерние элементы:
 - *<title>* - описывает заголовок RSS канала.
 - *<link>* - описывает гиперссылку на канал.
 - *<description>* - содержит краткую характеристику канала.

Каждый элемент *<channel>* может содержать один или более *<item>* элементов. Каждый элемент *<item>* описывает отдельную статью RSS источника. В свою очередь каждый элемент *<item>* имеет три обязательных дочерних элемента:

- *<title>*
- *<link>*
- *<description>*

Кроме обязательных элементов *<channel>* может содержать некоторые дополнительные дочерние элементы.

- Элемент *<category>* описывает категорию потока и используется RSS агрегаторами для группировки сайтов на

основе категорий.

- Элемент `<copyright>` информирует об авторском праве на данный документ.
- Элемент `<image>` предназначен для отображения изображения при показе пока агрегатором. В свою очередь он содержит три обязательных дочерних элемента:
 - `<url>` - URL изображения
 - `<title>` - альтернативный текст, отображаемый при невозможности показа изображения
 - `<link>` - содержит гиперссылку на веб-сайт, содержащий канал

Пример:

```

<image>

<url>http://www.myhp.edu/images/me.gif</url>
  <title>My home page</title>
  <link>http://www.myhp.edu</link>
</image>
    
```

Элемент `<language>` информирует о языке документа. Может быть использован агрегаторами для группировки сайтов по языку.

2. Список дочерних элементов для элемента `<channel>`

Элемент	Описание	Статус
<code><category></code>	Указывается одна или несколько категорий потока	Необязательный
<code><cloud></code>	Регистрирует процессы, получающие немедленное извещение в случае обновления потока	Необязательный
<code><copyright></code>	Извещает об авторском праве на документ	Необязательный
<code><description></code>	Краткое описание канала	Обязательный
<code><docs></code>	Указывает URL документов по форматам, используемым в потоке	Необязательный
<code><generator></code>	Указывает на программу, генерирующую поток	Необязательный

<image>	Позволяет показывать изображение, представляющее поток в агрегаторе	Необязательный
<language>	Указывается язык потока	Необязательный
<lastBuildDate>	Определяется дата последнего обновления содержимого потока	Необязательный
<link>	Указывается гиперссылка на канал	Обязательный
<managingEditor>	Содержится почтовый адрес редактора содержимого потока	Необязательный
<pubDate>	Определяет дату последней публикации содержимого потока	Необязательный
<rating>	Содержится PICS рейтинг потока	Необязательный
<skipDays>	Указывает дни, в которые агрегаторы не должны обновлять поток.	Необязательный
<skipHours>	Указывается количество часов в течение которых агрегаторы не должны обновлять поток	Необязательный
<textInput>	Описывается текстовое поле ввода, отображаемой в потоке	Необязательный
<title>	Размещается имя канала	Обязательный
<ttl>	Указывается количество минут в течение которых поток может находиться в кэше до обновления из источника	Необязательный
<webMaster>	Помещается адрес электронной почты веб-дизайнера потока	Необязательный

3. Список дочерних элементов для элемента <item>

Элемент	Описание	Статус
<author>	Адрес электронной почты автора темы	Необязательный
<category>	Описывает одну или более категорий для данной темы	Необязательный
<comments>	Позволяет связать тему с комментариями по данной теме	Необязательный

<description>	Краткое описание темы	Обязательный
<enclosure>	Позволяет включить мультимедиа файл в данную тему	Необязательный
<guid>	Содержит уникальный идентификатор темы	Необязательный
<link>	Содержит гиперссылку на тему	Обязательный
<pubDate>	Содержит последнюю дату публикации темы	Необязательный
<source>	Указывает внешний источник для данной темы	Необязательный
<title>	Название темы	Обязательный

4. Публикация RSS файла

Создание RSS потока не ограничивается разработкой RSS документа. Необходимо еще опубликовать этот файл. Для этого потребуется выполнить следующую последовательность действий:

1. Выбрать подходящее название для RSS файла. Расширение должно быть *.xml*.
2. Проверить RSS файл на правильность с помощью подходящей программы-валидатора, например, взятой по адресу <http://www.feedvalidator.org>.
3. Разместить RSS файл в подходящем веб-каталоге веб-сервера.
4. Скопировать одну из «кнопок»: **RSS** или **XML** в веб-каталог.
5. Вставить выбранную «кнопку» на исходящую страницу RSS потока в виде гиперссылки на RSS файл, например:

```

<a href = "www.myhp.edu/rss/myrss.xml">
  < img src="www.myhp.edu/rss/rss.gif" width="35"
height="15">
</a>
    
```

6. Разместить созданный RSS поток в популярных Каталогах RSS потоков, при этом URL потока должен ссылаться именно на сам XML файл потока, т.е. <http://www.myhp.edu/rss/myrss.xml>.
7. Можно также зарегистрировать поток в популярных поисковых системах, например:

- *Yahoo* - http://publisher.yahoo.com/rss_guide/submit.php
 - *Google* - <http://www.google.com/ig>
 - *MSN* - <http://w.moreover.com/site/products/ind/pingserver.html>
8. Периодическое обновление потока.




В принципе, можно самостоятельно заниматься формированием данных в соответствующем формате RSS для работы потока. Однако имеются достаточно удобные высокоуровневые средства для автоматизации данной работы. Например:




- *MyRSSCreator* - <http://www.myrsscreator.com/>
- *FeedFire* - <http://www.feedfire.com/site/index.html>

Для чтения RSS потоков используются специальные программы чтения – RSS ридеры. Некоторые из браузеров имеют встроенные RSS ридеры. В частности, веб-браузер *MS Internet Explorer 7* может интерпретировать XML файлы RSS потоков и корректно их отображать.

5. Добавление RSS-канала с помощью Microsoft *Internet Explorer* версии 7

При использовании MS Internet Explorer версии 7 и Office Outlook 2007 можно добавлять и просматривать RSS-каналы с помощью любой из этих программ.

1. При просмотре веб-страницы, содержащей RSS, в *Internet Explorer* рядом с кнопкой  (**Домашняя страница**) отображается кнопка .
2. Нажмите кнопку .
3. На веб-странице отобразится список доступных RSS-каналов.
4. Выберите RSS-канал, который необходимо добавить.

Можно также нажать кнопки ,  или , расположенные на веб-странице.

7.2. Порядок выполнения лабораторной работы

1. **Создание RSS документа с помощью PHP сценария и с использованием интерфейса DOM XML.**
 - 1) Подготовьте файл, содержащий сценарий на языке PHP:

```

<?php
// Создается новый XML документ.
$dfeed = new DOMDocument('1.0', 'utf-8');

// Создание корневого элемента <rss>
$erss = $dfeed->createElement('rss');
$erss->setAttribute('version', '2.0');

// И добавление его в дерево документа
$dfeed->appendChild($erss);

// Создание элемента <channel> и добавление его к <rss>
$echannel = $dfeed->createElement('channel');
$erss->appendChild($echannel);

// Создание и добавление в channel
// узлов <title>, <link>, <description>, <language>, <pub-
Date>
$echannel->appendChild( $dfeed->createElement('title', 'RSS-
channel title') );
$echannel->appendChild( $dfeed ->createElement('link',
'http://www.myhp.edu') );
$echannel->appendChild( $dfeed ->createElement('description',
'my RSS demo') );
$echannel->appendChild( $dfeed ->createElement('language',
'en') );
$echannel->appendChild( $dfeed ->createElement('pubDate',
date('r')) );

// Добавление к узлу <channel> 2 узлов <item>
for ( $i = 1; $i <= 2; $i++ )
{
    $eitem = $dfeed->createElement('item');
    $echannel->appendChild($eitem);

    // Создание дочерних элементов для <item>
    $eitem->appendChild( $dfeed->createElement('title',
'Item'. $i) );
    $eitem->appendChild( $dfeed->createElement('link',
'http://www.myhp.edu/rss/'. $i. '.xml') );
    $eitem->appendChild( $dfeed->createElement('description',
'Description for '. $i. ' item') );

```

```

    }

    // Сохранение документа в файле demo.rss
    $dfeed->save('demo.rss');
    ?>
    
```

- 2) Созданный файл разместите на веб-сервере, настройте права доступа (право на запись) для веб-сервера к директории, в которой размещается сценарий, либо к специальной директории, в которой будет создан файл *demo.rss* (это более безопасно).
- 3) Выполните сценарий по запросу с веб-браузера.
- 4) В результате выполнения сценария получится следующий документ:

```

<?xml version="1.0" encoding="utf-8" ?>
<rss version="2.0">
  <channel>
    <title>RSS-channel title</title>
    <link>http://www.myhp.edu</link>
    <description>my RSS demo</description>
    <language>en</language>
    <pubDate>Thu, 06 Nov 2008 01:50:53
+0300</pubDate>
  <item>
    <title>Item1</title>
    <link>http://www.myhp.edu/rss/1.xml</link>
    <description>Description for 1 item</description>
  </item>
  <item>
    <title>Item2</title>
    <link>http://www.myhp.edu/rss/2.xml</link>
    <description>Description for 2 item</description>
  </item>
</channel>
</rss>
    
```

2. **Чтение RSS документа с помощью PHP сценария и с использованием интерфейса DOM XML.**
- 1) Подготовьте файл, содержащий сценарий на языке PHP:

```

<?php
// Загрузка документа
$dfeed = DOMDocument::load('demo.rss');

// Чтение элемента <channel>
$echannel = $dfeed->firstChild->firstChild;

// Чтение первого дочернего элемента узла <channel>
т.е.< title>
    $ccchild = $echannel->firstChild;

// Вывод содержимого дочерних элементов узла
<channel> до элемента <item>:

print "<strong><br>";
while ( $ccchild->tagName != 'item' )
{
    print $ccchild->nodeValue. "<br>";
    $ccchild = $ccchild->nextSibling;
}
print "</strong><br>";

// Получение списка узлов <item>
$litems = $echannel->getElementsByTagName('item');

print "<br>";
foreach ( $litems as $eitem )
{
    $title = $eitem->firstChild->nodeValue;
    $link = $eitem->firstChild->nextSibling->nodeValue;
    $descr = $eitem->lastChild->nodeValue;

    print "<a href=\".$link.>\".$title.</a> :: ".$descr."<br>";
}
print "<br>";

?>
    
```

В методе *load()* в качестве параметра укажите путь к RSS-файлу.

- 2) Созданный файл разместите на веб-сервере, настройте права доступа (право на запись) для веб-сервера к директории, в которой размещается сценарий, либо к

специальной директории, в которой будет создан файл *demo.rss* (это более безопасно).

- 3) Выполните сценарий по запросу с веб-браузера.
- 4) Результат должен выглядеть следующим образом:

RSS-channel	title
http://www.myhp.edu	
my	RSS
en	demo
Thu, 06 Nov 2008 02:15:55 +0300	
Item1	:: Description for 1 item
Item2	:: Description for 2 item

7.3. Контрольное задание

Разработайте PHP-сценарий, генерирующий RSS документ, содержащий описание веб-страниц на вашем локальном веб-сайте. В качестве источника информации для сценария можно использовать либо заранее подготовленный текстовый файл, либо функции PHP из библиотеки для работы с файловой системой (см., например: <http://www.softtime.ru/bookphp/help.php>)