Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический университет имени П. О. Сухого»

Кафедра «Информационные технологии»

Д. А. Литвинов, А. В. Ковалев

ПРОГРАММИРОВАНИЕ В INTERNET

КУРС ЛЕКЦИЙ
по одноименной дисциплине для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной и заочной форм обучения

Электронный аналог печатного издания

Гомель 2013

УДК 004.42+004.738.5(075.8)
ББК 32.973-018я73
Л64

Рекомендовано к изданию научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(*протокол № 9 от 26.03.2012 г.*)

Рецензент: зав. каф. «Информатика» ГГТУ им. П. О. Сухого канд. техн. наук, доц.
А. И. Рябченко

Литвинов, Д. А.

Л64 Программирование в Internet : курс лекций по одноим. дисциплине для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» днев. и заоч. форм обучения / Д. А. Литвинов, А. В. Ковалев. – Гомель : ГГТУ им. П. О. Сухого, 2013. – 63 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://library.gstu.by/StartEK. – Загл. с титул. экрана.

ISBN 978-985-535-140-6.

Предназначен для обучения Internet-программированию на стороне клиента и сервера. Рассматриваются основные технологии прикладного программирования в Интернет, создание динамических Web-страниц. Изучаются языки программирования JavaScript и PHP.

Для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной и заочной форм обучения.

УДК 004.42+004.738.5(075.8)
ББК 32.973-018я73

ISBN 978-985-535-140-6

ЯЗЫК JAVASCRIPT

1. Назначение JavaScript

JavaScript – это язык управления сценариями просмотра гипертекстовых страниц Web на стороне клиента. Основная идея JavaScript состоит в возможности изменения значений атрибутов HTML-контейнеров и свойств среды отображения в процессе просмотра HTML-страницы пользователем, без перезагрузки страницы. Основные свойства:

1. Исходный код программ встраивается непосредственно в HTML-документ либо загружается из независимых файлов.

2. Программа выкладывается на сервер в виде исходного кода в текстовой форме и в дальнейшем интерпретируется (без предварительной компиляции) браузером после загрузки ее с сервера.

3. Структура объектов не задается однозначно устройством класса, а является динамической и может меняться на этапе выполнения программы. Объекты могут динамически получать новые поля и методы или изменять любые параметры старых.

4. Свободная типизация: элементарные типы данных переменных не описываются, при присваивании тип левой части всегда определяется по результату присваивания.

5. Динамическое связывание кода с объектами: ссылки на объекты проверяются во время выполнения программы.

1.1. Встраивание кода JavaScript в документ

Код JavaScript может быть встроен в документ HTML следующими основными способами:

1. Включение фрагментов сценария внутри тега script.

```
<script type="text/JavaScript"> ... инструкции ...</script>
```

2. Включение файлов со сценариями JavaScript.

```
<script type="text/JavaScript" src="myFunc.js"></script>
```

3. Определение как обработчика события внутри атрибута тега.

```
<span onClick="this.style.color='red;'">Click me!</span>
```

4. URL типа JavaScript. В гиперссылке в качестве значения атрибута **href**.

```
<a href="javascript:
  window.open('next.htm','','width=300,height=200'); void(0);">
</a>
```

2. Объектная модель JavaScript. Объекты браузера

Браузер предоставляет доступ к иерархии объектов, которые используются для создания клиентских сценариев. Сценарии работают с так называемой объектной моделью браузера (BOM) и загруженного в него документа (DOM). Все объекты языка JavaScript можно разделить на три группы:

1. Внутренние объекты и функции **JavaScript** – встроенные объекты и сам язык в соответствии со стандартом ECMA-262.

2. Объектная модель браузера (**Browser Object Model – BOM**) – объекты, предназначенные для работы с окном и организации доступа к документу.

3. Объекты, связанные с тегами языка HTML (**Document Object Model – DOM**) – интерфейс, позволяющий получать доступ к содержимому документа.

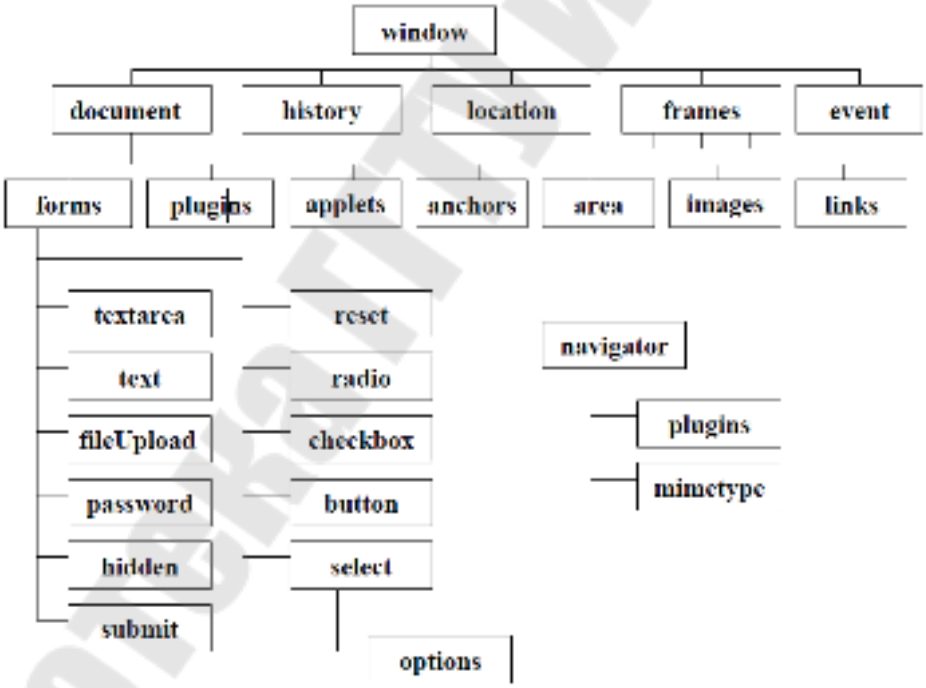


Рис. 2.1. Иерархия основных объектов браузера

Объекты браузера представляют из себя иерархическую структуру, во главе ее стоит объект **window**. Браузер создает экземпляр объекта **window** при открытии страницы, остальные объекты являются его свойствами. Структура объектной модели браузера представлена на рис. 2.1. Рассмотрим основные объекты более подробно.

2.1. Объект window

Объект **window** всегда является объектом верхнего уровня. Набор свойств, методов и событий объекта представлен в табл. 2.1.

Таблица 2.1

Основные свойства, методы и события объекта window

Свойства	name, parent, self, opener, location, history, navigator, document, screen
Методы	alert, confirm, prompt, open, close, focus, blur, scroll, setInterval, setTimeout, print
События	OnLoad, OnUnload, OnFocus, OnBlur, OnError, OnResize, OnScroll

Рассмотрим основные свойства и методы управления окном браузера:

– **window.name** – имя окна. Имя окна используется преимущественно для установки атрибута **target** гиперссылок;

– **window.parent** – возвращает ссылку на родительское окно;

– **window.self** – ссылка на текущее окно;

– **window.opener** – ссылка на родительское окно открытое, вызовом **window.open**. Если окно открыто из другого, то оно сохраняет ссылку на него, если такого окна нет, то **window.opener = null**;

– **window.focus()** – переводит фокус на текущую страницу;

– **window.blur()** – метод уводит фокус из окна. При этом неясно, куда фокус будет передан. Лучше целенаправленно передать фокус, чем просто его потерять;

– **window.print()** – печать документа;

– **window.alert(сообщение)** – окно предупреждения;

– **window.confirm(сообщение)** – окно запроса ответа на сообщение. Щелчок по «ОК» возвращает true, по «Отмена» – false;

– **window.prompt(Сообщение, Значение по умолчанию)** – окно ввода данных;

– **window.open()** – метод предназначен для создания новых окон. В общем случае его синтаксис выглядит следующим образом:

```
open("URL","window_name","param, param,...", replace);
```

где **URL** – страница, которая будет загружена в новое окно, **window_name** – имя окна, которое можно использовать в атрибуте **TARGET** в контейнерах **A** и **FRAME**, **param** – параметры создаваемого окна (табл. 2.2).

Таблица 2.2

Основные свойства окна

Параметры	Назначение
width	Ширина окна в пикселах
height	Высота окна в пикселах
toolbar	Создает окно с системными кнопками браузера
location	Создает окно с полем location
directories	Создает окно с меню предпочтений пользователя
status	Создает окно с полем статуса status
menubar	Создает окно с меню
scrollbars	Создает окно с полосами прокрутки
resizable	Создает окно, размер которого можно будет изменять
top, left	Положение по вертикали (горизонтали) относительно верхнего (левого) края экрана

– **window.close()** – метод предназначен для закрытия окна. Если необходимо закрыть текущее, то: **window.close()**; или **self.close()**. Для закрытия родительского окна: **window.opener.close()**. Для закрытия произвольного окна используют указатель, полученный при его создании: **win_id=window.open(); ... win_id.close();**

– **window.setTimeout(func|code, delay)** – метод **однократно** выполняет код(или функцию), указанный в первом аргументе, асинхронно, с задержкой в **delay** миллисекунд;

– **window.clearTimeout(timeID)** – позволяет уничтожить поток, созданный методом **setTimeout()** до его выполнения. **setTimeout** – Идентификатор, возвращенный **timeID**;

– **window.setInterval(func|code, delay)** – метод **периодически** выполняет код(или функцию), указанный в первом аргументе, асинхронно, с задержкой в **delay** миллисекунд;

– **window.clearInterval(intervalID)** – позволяет уничтожить поток, созданный методом **setTimeout()** до его выполнения. **intervalID** – идентификатор, возвращенный **setInterval**.

Пример. Создание окон с заданными свойствами.

```
window.open('about:blank','', 'directories=yes, height=200, location=yes, menubar=yes, resizable=yes, scrollbars=yes, toolbar=yes, width=200');
```

2.2. Объект location

Объект позволяет получить доступ к URL-документа, отображаемого в окне браузера. Его свойства доступны не только для чтения, но и для изменения, что позволяет загружать нужный документ.

Для получения доступа к объекту служит свойство – **window.location**. Свойства и методы объекта **location** представлены в табл. 2.3.

Таблица 2.3

Основные свойства и методы объекта location

Свойство	Описание
href	Полная строка URL
protocol	Протокол
hostname	Имя или адрес сервера
port	Порт
host	Hostname + port
pathname	Путь и имя файла
assign(URL)	Загрузка документа с заданным URL. Или простым присвоением: window.location.href = url
replace(URL)	Загрузка документа с заданным URL (замена текущего документа)
reload(параметр)	Параметр = true | false. True – загрузка с сервера. False – загрузка из кэша браузера

Пример. Свойства объекта **location** для адреса:
http://gstu.gomel.by:80/main/index?search#mark
window.location.href = http://gstu.gomel.by:80/main/index?search#mark
window.location.hostname = gstu.gomel.by;
window.location.pathname = /main/index;

2.3. Объект history

Объект позволяет перемещаться по истории просмотра браузера. Объект указывает на массив URL-страниц, которые пользователь посещал ранее. Свойства и методы объекта представлены в табл. 2.4.

Таблица 2.4

Основные методы и свойства объекта history

Имя	Описание
length	Количество посещенных сайтов (длина списка истории)
back(номер)	Загрузка предыдущего адреса из списка. Необязательный параметр указывает номер элемента в списке
forward()	Загрузка следующего адреса из списка
history.go(параметр)	Если параметр – число, то метод работает также, как back, если строка, то это посещенный адрес. {back() = history.go(–1)}

2.4. Объект navigator

Объект **Navigator** содержит информацию о браузере. Что позволяет использовать и создавать сценарии под различные версии браузера. Свойства и методы объекта представлены в табл. 2.5. Свойства только для чтения.

Таблица 2.5

Основные методы и свойства объекта navigator

Имя	Описание
appName	Название браузера
appVersion	Версия браузера
userAgent	Поле userAgent протокола HTTP
browserLanguage	Язык браузера
cookieEnabled	Проверка разрешения использования cookie
javaEnabled()	True | false. Проверка возможности использования java (апплетов)

Пример. Определение информации о браузере.

```
alert('Кодовое имя: ' + window.navigator.appCodeName + '\n' +
'Название браузера: ' + window.navigator.appName + '\n' +  'Версия: ' +
window.navigator.appVersion + '\n' + 'Поддержка cookie: ' +
window.navigator.cookieEnabled + '\n' +  'Строка агента: ' +
window.navigator.userAgent );
```

2.5. Объект screen

Объект **screen** описывает возможности используемого устройства отображения – разрешение клиентского экрана, глубину цвета и другие. Основные свойства и методы описаны в табл. 2.6. Свойства только для чтения.

Таблица 2.6

Основные методы и свойства объекта screen

Имя	Описание
height	Вертикальное разрешение экрана
width	Горизонтальное разрешение экрана
availHeight	Возвращает высоту полезной области экрана без панели задач и подобных ей элементов графического интерфейса системы
availWidth	Возвращает ширину полезной области экрана без панели задач и подобных ей элементов графического интерфейса системы
colorDepth	Глубина цвета. Количество битов, используемое для кодировки цвета одного пиксела (True Color=24)

Пример. Проверка разрешения клиентского экрана.

```
var goodW = (window.screen.width >= 800 && window.screen.width < 2048)
var goodH = (window.screen.height >= 600 && window.screen.height < 1024)
if(goodW && goodH) alert('Подходящее разрешение');
```

2.6. Управление документом. Объект document

Объект **document** определяет интерфейс по изменению и управлению видимой частью документа (контейнер <**body**>). Все HTML-объекты являются свойствами объекта **document**. При загрузке HTML-документа, браузер создает его объектную модель. Прежде всего создается объект окна **window** и его подобъекты **location**, **screen** и др. Затем создается объект **docment** и наполняется объектами, представляющими отдельные элементы HTML-документа.

Объект **document** имеет достаточно много свойств и методов, основные из которых представлены в табл. 2.7.

Таблица 2.7

Основные методы и свойства объекта document

Имя	Описание
activeElement	Ссылка на элемент (объект) страницы находящийся в данный момент в фокусе
lastModified	Дата последнего изменения документа
parentWindow	Ссылка на родительский объект Window
readyState	Возвращает текущее состояние документа: complete – документ полностью загружен; interactive – загружен не полностью но доступен для просмотра и управления; loading – загружается; uninitialized – недоступен, загружается
referrer	Возвращает адрес Web–страницы, с которой пользователь перешел на текущую страницу
URL	Возвращает адрес текущей Web–страницы
body	Возвращает ссылку на объект BODY
write()	Запись данных в HTML в документ
close()	Закрытие потока вывода документа
open()	Открытие потока ввода в уже созданное окно. Документ очищается

2.7. Обращение к объектам документа. Объект document

Объектная модель документа представляется в виде дерева и образует иерархическую структуру с отношениями типа родитель – потомок.

Чтобы указать конкретный объект, требуется перечислить все содержащие его объекты. Для доступа к свойствам и методам используют следующий синтаксис:

объект_1. ... объект_N.свойство объект_1. ... объект_N. метод()

Для доступа к объектам Web-страницы применяют встроенные методы объекта **document** и коллекции его объектов.

document.**getElementById**("**elementID**") – свойство возвращает объект документа с заданным атрибутом **ID**. Если таких элементов несколько, то возвращается первый найденный.

document.elementFromPoint(X, Y) – свойство возвращает объект документа, находящийся по координатам **X** и **Y**.

В объектной модели документа объекты также, сгруппированы в так называемые *коллекции*. **Коллекция** – это структура данных (объект), похожая на массив, проиндексированный как по индексам элементов, так и по их именам, и имеющая свои свойства и методы. Основные коллекции объектной модели представлены в табл. 2.8.

Таблица 2.8

Основные оллекции объектной модели документа

Коллекция	Описание
document.all[]	Все объекты, отвечающие контейнерам внутри контейнера <BODY>
document.anchors[]	Все якоря – т. е. объекты, отвечающие контейнерам <A>
document.forms[]	Все формы – т. е. объекты, отвечающие контейнерам <FORM>
document.images[]	Все картинки – т. е. объекты, отвечающие контейнерам <IMG>
document.links[]	Все ссылки – т. е. объекты, отвечающие контейнерам <A HREF="..."> и <AREA HREF="...">
document.f.elements[]	Все элементы формы с именем f – т. е. объекты, отвечающие контейнерам <INPUT> и <SELECT>
document.f.s.options[]	Все опции (контейнеры <OPTION>) в контейнере <SELECT NAME = s> в форме <FORM NAME = f>

Элементы коллекции нумеруются начиная с нуля, в порядке их появления в HTML-документе. Доступ к элементам коллекций осуществляется либо по индексу (в квадратных скобках), либо по имени (в квадратных скобках либо через точку). Последний способ удобен, когда имя элемента коллекции хранится в качестве значения переменной. Как и у обычных массивов, у коллекций есть свойство **length**, которое позволяет узнать количество элементов в коллекции.

Рассмотрим способы обращения к коллекциям объектов: если обрабатывается документ, загруженный в текущее окно, то объект **window** можно не упоминать, а сразу начинать с ключевого слова **document**.

```
window_obj.document.коллекция.объект_id
window_obj.document.коллекция["объект_id"]
window_obj.document.коллекция[индекс_объекта],
```

где **window_obj** – указатель на объект окна, к документу которого осуществляется доступ; **объект_id** – значение атрибута **ID** в теге; **индекс_объекта** – число, указывающее порядковый номер объекта в коллекции.

Пример. Способы обращения к элементам документа на примере коллекции **all**. Для определения тега, на основании которого был создан объект, можно использовать свойство коллекций – **tagName**.

```
<!DOCTYPE HTML PUBLIC "–//W3C//DTD HTML 4.0 Transitional//EN" >
<script type="text/JavaScript">
function win_out() {
 alert("объектов в коллекции="+document.all.length);
 alert("1 "+document.all[0].tagName + "\n2
"+document.all["my_h4"].tagName + "\n3
"+document.all.my_h4.tagName + "\n4 "+document.all.my_h4.id);
var id="my_p";
alert("доступ с использованием
переменной\n"+document.all[id].tagName);
}
</script>
<BODY onclick="win_out();">
<H4 id="my_h4"> тестовая страница – 1</H4> <p id="my_p">Текст
...</p>
</BODY>
```

Помимо стандартных коллекций можно создавать коллекции, созданные посредством заданного тега.

document.getElementsByTagName("имя_тега") – возвращает коллекцию элементов, созданных посредством заданного тега. Метод можно вызывать не только для **document**, но и для любого объекта, созданного на основе тега. В этом случае поиск осуществляется только внутри объекта.

```
my_obj = document.getElementsByTagName("H3"); // коллекция заголовков
```

document.getElementsByName("имя_элемента") – возвращает коллекцию элементов с атрибутом **name**, равным заданному.

```
<form> Введите:<INPUT id="data_text"/></form>
document.getElementsByName("data_text")[0].value
```

3. Программирование форм

3.1. Объект Form. Обращение к объектам формы

В HTML-документах формы применяются для ввода данных и взаимодействия с пользователем. Объект **form** является свойством объекта **document**, а также элементом его коллекции **forms**. Свойства и методы объекта представлены в табл. 3.1. В зависимости от того какой объект **form** используется, обращаться можно как по имени (атрибут **name** формы), так и по индексу.

```
document.имя_формы (не рекомендуется)
document.forms.имя_формы (предпочтительнее)
document.forms[индекс_формы]
document.forms["имя_формы"]
document.forms.имя_формы
```

Таблица 3.1

Основные свойства, методы и события объекта form

Свойства	length, action, method, target, encoding, elements[]
Методы	reset(),submit()
События	Reset, Submit

3.2. Свойство объекта Form

Рассмотрим основные свойства и методы объекта **form** более подробно. Свойство **length** – содержит число элементов коллекции **forms**[]. Свойство **action** – задает URL адрес файла скрипта, который получает данные, отправленные браузером. Так же можно задавать вызов локальных скриптов.

```
ACTION="http://ya.ru/yandsearch"  ACTION="javascript:win_out(); void(0);"
```

Применение **void(0)**, отменяет стандартное событие **submit**, браузер не обращается к серверу для передачи данных.

Свойство **method** – определяет метод доступа к ресурсам HTTP-сервера из браузера – **GET** или **POST**.

```
document. my_form.method='post';
```

При обработке браузером страницы для каждого объекта **Form** создается связанный с ним массив (коллекцию) полей формы. Свойство **elements[]** содержит ссылку на коллекцию элементов формы. Обратиться к элементам управления (к полям) формы можно как по имени (значение атрибута **name**), так и по индексу в коллекции:

ссылка_на_форму.elements.имя_элемента

ссылка_на_форму.elements[индекс_элемента]

ссылка_на_форму.elements["имя_элемента"]

Также свойствами формы могут быть следующие объекты **button, checkbox, fileUpload, hidden, password, radio, reset, select, submit, text, textarea**. В этом случае для обращения к элементу можно использовать следующий вид:

document.имя_формы.имя_элемента (не рекомендуется)

document.forms.имя_формы.имя_элемента (предпочтительнее)

Пример. Пример обращения к элементам формы. Свойство **value** – содержимое текстового поля.

```
<FORM name="my_form"> Имя: <INPUT name="my_name" size=8>
</FORM>
```

Число элементов на форме**: document.forms[0].elements.length**

Примеры обращения к элементам формы**:**

document.my_form.my_name.value

document.forms[0].elements["my_name"].value

3.3. Методы объекта Form

Метод **reset()** позволяет установить значения полей формы по умолчанию. Действие метода аналогично кнопке типа **Reset**.

document.forms["my_form"].reset();

Метод **submit**() позволяет инициировать передачу введенных в форму данных на сервер, после чего страница перезагружается. Действие метода аналогично кнопке типа **Submit**.

Пример. Отправка данных на сервер ya.ru по нажатию на ссылку.

```
<script type="text/JavaScript">
function check_send(form_name) {
 if(document.forms[form_name].text.value !="") // проверка на пустое поле
       document.forms[form_name].submit();
 else  alert("Задайте строку поиска");
}</script>
<FORM name="my_form" METHOD=get
ACTION="http://ya.ru/yandsearch">
 Введите строку поиска:<INPUT NAME="text" SIZE=50 MAXLENGTH=50>
</FORM>
<A HREF="javascript:void(check_send('my_form'))">Отправить данные </A>
```

3.4. События объекта Form

Событие **onReset** – возникает при нажатии пользователем на кнопку типа **reset** или при выполнении метода **reset**(). Событие **onSubmit** возникает при нажатии на кнопку **Submit**, графическую кнопку **image** или при вызове метода **submit()**.

Обработчики события задаются в контейнере **FORM** в атрибутах **onReset** и **onSubmit.** Функции, определенные в этих атрибутах, будут выполняться перед действиями браузера по умолчанию. Если обработчику события вернуть **false**, то действие по умолчанию выполняться не будет. Этот прием называется перехватом события. Общая ошибка при возвращении значения обработчику событий заключается в возвращении значения только в строку кода в обработчике событий, а не в обработчик событий непосредственно.

Не верно	Верно
onSubmit ="validForm(this);"	onSubmit ="return validForm(this);"

Пример. Обработка событий **onSubmit**:

```
<script type="text/JavaScript">
function validForm(form_obj) {
 if(form_obj.text.value !="" && Number(form_obj.text.value).toString() !=
"NaN")    return true; // выполнение действий по умолчанию
 else  {  alert("Введите число!"); form_obj.reset(); return false; }
}</script>
<FORM name="my_form" ACTION="http://ya.ru/yandsearch"
onSubmit="return validForm(this);">
 Введите число для поиска:<INPUT NAME="text" SIZE=10>
 <INPUT TYPE="submit" VALUE="Искать"></FORM>
```

3.5. Элементы форм. Объект кнопка (Button)

В HTML-формах может использоваться четыре вида кнопок:

```
<INPUT TYPE=button VALUE="Кнопка типа button">
<INPUT TYPE=image SRC="URL">
<INPUT TYPE=submit VALUE="Кнопка отправки">
<INPUT TYPE=reset VALUE="Кнопка сброса">
```

Первая кнопка является универсальной и обычно используется, для организации взаимодействия с пользователем. Остальные специализированные и используются только для отправки данных (**image,**

submit) или очистки формы (**reset**). Общие свойства, методы и события объекта **button** представлены в табл. 3.2.

Таблица 3.2

Основные свойства, методы и события объекта button

Свойства	form, name, type, value
Методы	focus(), blur(), click()
События	onBlur, onClick, onFocus, onMouseUp, onMouseDown

form – ссылка на родительский объект форму; **name** – значение свойства **name** объекта; **type** – значение свойства **type**; **value** – значение свойства **value**; **blur**() – убирает фокус с кнопки; **click**() – симулирует щелчок мыши по кнопке; **focus**() – передает фокус кнопке.

Обработчики событий задаются либо в атрибутах кнопки, либо в атрибуте формы – **onSubmit** и **onReset**. Форма также имеет методы **submit**() и **reset**(). Рассмотрим, как взаимодействуют методы друг с другом и с соответствующими обработчиками событий:

– при вызове метода **click**() кнопки вызывается и обработчик события **onClick** этой кнопки;

– при вызове метода **submit**() формы **не вызывается** обработчик события **onSubmit** формы;

– при вызове метода **reset**() формы вызывается и обработчик события **onReset** формы.

Пример. Задание обработчика события для объекта **button.** Обработчику события **Test**() передается ссылка (**this**) на объект кнопки, вызвавшей событие. В обработчике по свойству **form** можно определить родительскую форму и получить доступ к ее элементам.

```
<INPUT TYPE=button value="Кнопка" name="my_but" onClick="Test(this)">
function Test(obj){
 if(obj.form.my_text.value == "")
     obj.form.my_text.style.border="2px solid red";
 else obj.value="Данные введены!";
}
```

Для специализированных кнопок нажатие приводит к следующей последовательности действий браузера:

– вызов обработчика события **onClick** у данной кнопки;

– вызов специального обработчика события (**onSubmit**, **onReset**) формы;

– действие браузера по умолчанию (если не отменено).

Графическая кнопка – это разновидность кнопки отправки (**Submit**), но с картинкой вместо текста. При нажатии на сервер отправятся не только данные, но и координаты указателя мыши относительно левого верхнего угла изображения в момент нажатия (перехватить скриптом нельзя).

3.6. Элементы форм. Объект текстовое поле (Text)

Поля ввода (контейнер **INPUT** типа **TYPE=text**) являются наиболее часто используемыми объектами формы. С каждым текстовым полем ввода связан объект **Text**, который является свойством формы, в которой он определен. Объект характеризуется свойствами, методами и событиями, представленными в табл. 3.3.

Таблица 3.3

Основные свойства, методы и события объекта text

Свойства	form, name, type, value, defaultValue, maxLength, size, disabled, readOnly
Методы	focus(), blur(), select()
События	onBlur, onClick, onFocus, onChange, onSelect, onDblClick, onMouseDown, onMouseUp, onMouseOver, onMouseOut, onKeyPress

defaultValue – значение свойства **value** по умолчанию, определенное при создании формы; **value** – содержимое поля; **maxLength** – максимальное число символов, которое можно присвоить значению данного поля; **size** – число видимых в поле символов; **disabled** – разрешение установки фокуса на этом поле; **readOnly** – разрешение изменять значение поля; **select**() – выделяет весь введенный текст.

3.7. Элементы форм. Объект список (Select и Option)

В HTML-формах для реализации списков используют контейнер **select**, который включает в себя контейнеры **option**. Для управления списками в JavaScript используется объект **Select** и дочерний объект **Option**. Рассмотрим основные свойства, методы и события, характеризующие эти объекты (табл. 3.4, 3.5).

Таблица 3.4

Основные свойства, методы и события объекта Select

Свойства	options[], form, name, type, size, length, selectedIndex
Методы	focus(), blur()
События	onBlur, onChange, onFocus

options[] – ссылка на массив объектов **options**; **length** – количество элементов (строк) в списке; **selectedIndex –** возвращает номер выбранной строки списка. Если ни одна строка не выбрана – 1; **type** – определяет возможность одновременного выбора нескольких пунктов списка (значения "**select-multiple**" или "**select-one**").

Таблица 3.5

Основные свойства объекта Option

Свойства	length, defaultSelected, selected, selectedIndex, text, value

text – представляет собой отображаемый в списке; **value** – значение, которое передается серверу при выборе пункта меню. Если оно не определено, передается содержимое **text**; **defaultSelected** – пункт меню выбран по умолчанию (true/false); **selected** – пункт списка выбран (true/false); **selectedIndex** – индекс выбранного **Option** объекта в массиве элементов **options[]**.

Методов и событий у объекта **option** нет. Объект **option** имеет конструктор, что позволяет программно создать элементы списка.

```
opt_obj = new Option([ text, [ value, [ defaultSelected, [ selected ] ] ] ]);
```

При программировании списков следует учитывать, что объект **option** не имеет свойства **name**. К его элементам можно обращаться только как к элементам массива **options**[].

Пример. Рассмотрим примеры работы со списками.

```
<form>  <select name="City">
   <option value="Moscow">Москва</option>
   <option value="Kiev">Киев</option>
   <option value="Minsk">Минск</option>
 </select> </form>
var objSel = document.forms["City"];
```

Добавление нового элемента в конец списка:

```
objSel.options[objSel.options.length] = new Option("Гомель", "Gomel");
```

Управление числом элементов списка:

```
objSel.options.length=2;   // усечение списка
objSel.options[1] = null;    // удаление 1 элемента списка
objSel.options.length = 0; // удалить все элементы списка
```

Доступ к элементам списка:

```
var text  = objSel.options[2].text;
var value = objSel.options[2].value;
```

Определение выбранного элемента списка:

```
var text = objSel.options[objSel.selectedIndex].text;
```

В случае множественного выбора элементов свойство selectedIndex содержит индекс первого. Чтобы найти все выделенные элементы, необходимо проверить в цикле их свойство selected.

```
function getSelectedIndexes (oListbox){
 var arrIndexes = new Array;
 for (var i=0; i < oListbox.options.length; i++)
      if (oListbox.options[i].selected) arrIndexes.push(i);
 return arrIndexes;
};
```

Функция **getSelectedIndexes** принимает в качестве аргумента объект списка и возвращает массив индексов выделенных элементов.

Выделение элементов списка. Свойства selectedIndex и selected доступны как для чтения, так и для записи, поэтому, присваивая им значения, можно выделять нужные элементы.

```
objSel.selectedIndex = 1;
```

Для выбора нескольких элементов:

```
objSel.options[1].selected=true;  objSel.options[3].selected=true;
```

3.8. Элементы форм. Переключатели (Radio и CheckBox)

Переключатели применяются для организации выбора одной или нескольких возможностей. При обработке формы с переключателями браузер для каждого создает отдельные объекты. Все переключатели, имеющие одинаковые значения атрибута **NAME**, группируются в массив, который помещается в массив **elements[]** соответствующего объекта **Form**. Если одна форма имеет несколько наборов таких переключателей, массив **elements[]** будет иметь несколько объектов. Для определения, какой из переключателей находится во включенном состоянии, необходимо проверить его свойство **checked**. Переключатели **checkbox** могут иметь и разные имена. Доступ к переключателям осуществляется через массив **elements[]** по индексу или имени.

Таблица 3.6

Свойства, методы и события объектов Radio, CheckBox

Свойства	form, name, type, value, checked, defaultChecked, length
Методы	click(), focus(), blur()
События	onBlur, onClick, onFocus

checked – свойство, отражающее состояние переключателя (true/false). Позволяет изменять состояние переключателя.

defaultChecked – свойство, отражающее значение атрибута **CHECKED** (true/false).

Пример. Определение выбранного **radio** переключателя:

```
function Check(idForm, idRadio){
 var obj=document.forms[idForm].elements[idRadio]; // ссылка на объект
 for(var i=0; i<obj.length; i++) // определения выбранного поля
   if(obj[i].checked) необходимые действи действия
}
```

Пример вызова для формы с именем «**Test**» и группой переключателей с атрибутом **NAME="Song"**: Check('Test','Song');

3.9. Элементы форм. Текстовые области (Textarea)

Поле многострочного ввода текста на HTML-форме. Объект создается HTML-тэгом **TEXTAREA** и помещается в массив **elements** соответствующего объекта **form**.

Таблица 3.7

Основные свойства, методы и события объекта textarea

Свойства	form, name, type, value, defaultValue, cols, rows, disabled, readOnly
Методы	focus(), blur(), select()
События	onBl ur, onFocus, onChange, onSelect, onKeyDown, onKeyPress, onKeyUp

cols (rows) – Возвращает или задает значение атрибута **cols (rows)** элемента.

4. DHTML. Объектная модель документа

4.1. Объектная модель документа (DOM)

Объектная модель документа (Document Object Model, **DOM**) – это прикладной программный интерфейс (API), определяющий порядок доступа к объектам, из которых состоит документ, и манипуляции ими. Стандарт DOM появился благодаря желанию иметь общий прикладной интерфейс (API) для DHTML программирования. Он представляет документы HTML (и XML) в виде дерева, каждый элемент которого соответствует элементу HTML (XML). DOM – это модель HTML- и XML-документов, независимая от платформы и языка программирования, которая определяет:

– интерфейсы для обхода, просмотра и изменения узлов дерева;

– технологии определения обработчиков событий;

– работу с таблицами стилей и областями документа.

На сегодняшний день W3C стандартизовал DOM первого и второго уровней (DOM 1 и DOM 2); в стадии рабочего проекта находится DOM 3. DOM 2 Core содержит два набора интерфейсов, каждый из которых обеспечивает полный доступ ко всем элементам документа. Первый набор представляет объектно-ориентированный подход со следующей иерархией наследования: документ – составляющие его элементы – их атрибуты и текстовое содержимое. При таком рассмотрении дерево документа представляет собой **иерархию объектов**. Второй подход рассматривает все составляющие документа как равноправные узлы (**Nodes**), т. е. представляет собой иерархию узлов. DOM 2 состоит из следующих групп взаимосвязанных интерфейсов:

– **Core** – определяющие представление документа в виде дерева;

– **View** – описывающие возможные отображения документа;

– **Event** – определяющие порядок генерации и обработки событий;

– **Style** – определяющие применение к документам таблиц стилей;

– **Traversal** & **Range** – интерфейсы, определяющие прохождение дерева документа и манипулирование областями его содержимого;

– **HTML** – определяющие представление документа в виде дерева.

DOM 2 **Core** представляет **XML**-документы в виде деревьев, состоящих из узлов, которые, в свою очередь, также являются объектами и реализуют более специализированные интерфейсы. Одни типы узлов могут иметь дочерние элементы, т. е. сами являться поддеревьями, другие являются только листьями (рис. 4.1).

```
<html>
 <head>
  <title>The document</title>
 </head>
<body>
  <div>Data</div>
  <ul>
    <li>Warning</li>
    <li></li>
  </ul>
  <div>Top Secret!</div>
</body>
</html>
```

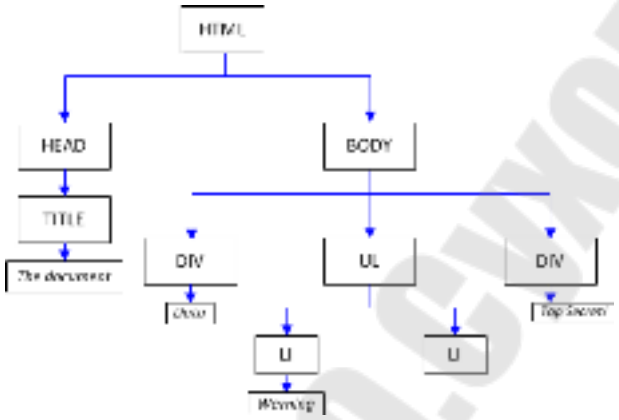


Рис. 4.1. Пример объектной модели документа

Пробелы и переносы строки между тегами также включаются в **DOM** и становятся пробельными узлами. В табл. 4.1 представлены возможные типы узлов дерева **HTML**-документа.

Таблица 4.1

Типы узлов дерева HTML-документа

Интерфейс	Описание	Дети
Document	Документ	Element (не более одного), Comment, DocumentType (не более одного)
DocumentFragment	Фрагмент документа	Element, Comment, Text
DocumentType	Тип документа	детей не имеет
Element	Элемент	Element, Comment, Text
Attr	Атрибут	Text
Comment	Комментарий	детей не имеет
Text	Текст	детей не имеет

4.2. Древовидная структура документа

DOM представляет HTML-документ в виде дерева объектов **Node** различных типов. Интерфейс **Node** определяет свойства и методы для перемещения по дереву и манипуляций с ним. К базовым программным интерфейсам (API) модели DOM, относятся: **Node**, **Element**, **Document, Attr**, **CharacterData** и др. Все составляющие документа реализуются интерфейсами, которые являются их потомками.

Рассмотрим структуру HTML-документа. Корневым узлом DOM дерева является объект **Document**. Его свойство **document.documentElement** ссылается на объект **Element**, представляющий корневой элемент доку-

мента (тег <**html**>). Свойство **document.body** ссылается на объект **Element**, представляющий корневой элемент начала видимой части документа (тег < **body** >). Его дочерние элементы образуют дерево, узлами которого в основном являются объекты **Element**, которые представляют теги, а также объекты **Text** и их текстовое содержимое.

4.3. Интерфейс Document. Доступ к элементам документа

Интерфейс **Document** является основой для доступа к содержанию документа и для создания его элементов. Основные методы:

– **document.createAttribute(name)** – создает новый объект типа Attr и возвращает указатель на него. Аргумент name задает имя создаваемого атрибута;

– **document.createDocumentFragment()** – создает новый пустой объект типа DocumentFragment и возвращает указатель на него;

– **document.createElement(tagName)** – создает новый объект типа Element и возвращает указатель на него. Аргумент tagName задает тег создаваемого элемента;

– **document.createTextNode(data)** – создает новый объект типа Text и возвращает указатель на него. Аргумент data задает текст содержимого узла.

Примеры использования рассмотренных методов будут приведены в следующих разделах.

4.4. Интерфейс Node

Интерфейс **Node** соответствует абстрактному узлу дерева документа. Основными узлами дерева элементов являются – Attr, Element, Text. Рассмотрим интерфейсы работы с узлами документа.

4.4.1. Интерфейс Node. Свойства узла

Основными свойствами DOM-узлов являются – тип, имя и содержимое узла:

– **узел.nodeName** – возвращает строку, содержащую имя узла, согласно табл. 4.2;

– **узел.nodeType** – возвращает тип узла согласно табл. 4.2 (всего существует 12 типов). В Internet Explorer (в зависимости от версии)

свойство реализовано для узлов **Element** и **Text**. Для остальных типов узлов оно не определено;

– **узел.nodeValue** – доступ к значению узла. Значение узла зависит от его типа (табл. 4.2).

Таблица 4.2

Основные типы узлов DOM

Интерфейс	nodeType	nodeValue	nodeName
Element	1	null	Тег элемента
Text	3	Содержимое узла	#text
Document	9	null	#document
DocumentFragment	11	null	#document-fragment
Attr	2	Значение атрибута	Имя атрибута

Для узлов типа **Element** определены дополнительно определены следующие свойства:

– **узел.tagName** – возвращает строку, содержащую имя тега данного элемента;

– **узел.innerHTML** – HTML содержимое узла в виде строки. Свойство доступно для изменения. Присвоение свойству нового значения, содержащего **HTML**-код, удаляет старое значение и интерпретирует новое. Если новое добавляется к старому содержимому элемента, то оно будет перезагружено, т. е. все содержимое узла создается заново. Не стоит этим пользоваться для больших изменений, поскольку операция по перестроению документа достаточно ресурсоемкая. Предпочтительнее создавать фрагмент, наполнять его содержимым и затем присоединять к дереву документа.

4.4.2. Интерфейс Node. Обход DOM

При навигации по дереву элементов обязательным требованием является полная загрузка документа браузером. Используя свойства объекта **Node**, можно перемещаться по дереву элементов. Рассмотрим необходимые свойства:

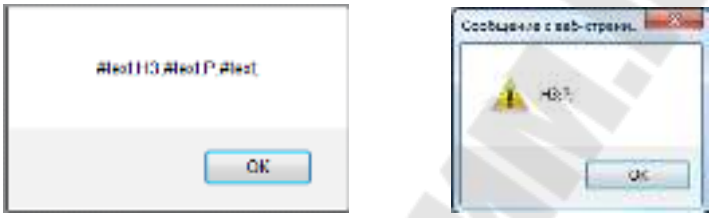
– **узел.childNodes** – возвращает объект типа **NodeList**, содержащий список всех детей данного узла. Если узел не имеет детей, то возвращается список нулевой длины;

– **узел.children** – возвращает объект типа **NodeList**, содержащий список **только** дочерних узлов – элементов (соответствующих тегам).

Пример. Определение дочерних элементов узла:

```
<script type="text/JavaScript">
function test(){
    var b = document.getElementById("b1"); var list = b.childNodes; str="";
    for (var i = 0; i < list.length; i++) { str+=list[i].nodeName + ";"; }
    document.write(str);
}</SCRIPT>
<BODY id="b1"><h3>Заголовок</h3>
<P onclick='test()' my="text">Абзац
<b>жирный</b>продолжение</P></BODY>
```

FireFox 10.0 **IE 9.0**



Браузер IE возвращает только узлы элементы. Для того чтобы не учитывать другие узлы, можно добавить проверку на тип узла. В этом случае результат одинаков для всех браузеров.

```
if (list[i].nodeType == 1)  str+=list[i].nodeName + ";";
```

Для перемещения по узлам дерева существуют специальные свойства, указывающие на соседние элементы. Каждый DOM-узел содержит массив всех детей (**childNodes**), отдельно – ссылки на первого и последнего ребенка и еще ряд дополнительных свойств (рис. 4.2). Все навигационные ссылки – только для чтения. При изменениях DOM, добавлении или удалении элементов они обновляются автоматически.

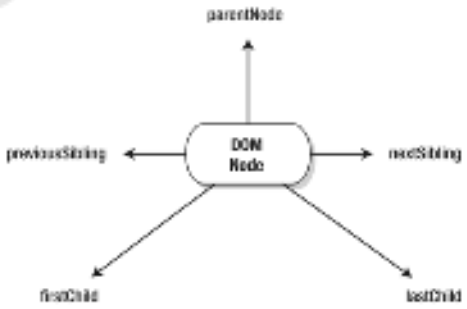


Рис. 4.2. Свойства ссылки на соседние элементы DOM

– **узел.firstChild** – возвращает указатель на первый узел, являющийся сыном данного узла. Если узел не имеет детей, то возвращается **null**. Свойство эквивалентно выражению **узел.childNodes**[0];

– **узел.lastChild** – возвращает указатель на последний узел, являющийся сыном данного узла. Если узел не имеет детей, то возвращается **null**. Это свойство эквивалентно выражению: **узел.childNodes[узел.childNodes.length-1]**;

– **узел.nextSibling** – возвращает указатель на узел, непосредственно следующий в документе за данным узлом. Если такого узла нет, то возвращается **null**;

– **узел.previousSibling** – возвращает указатель на узел, непосредственно предшествующий в документе данному узлу. Если такого узла нет, то возвращается **null**;

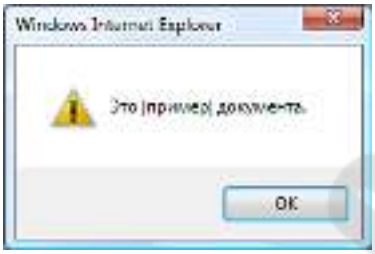
– **узел.parentNode** – возвращает указатель на узел, являющийся отцом данного узла. Если узел не имеет отца, то возвращается **null**.

Пример. Обход всех элементов документа с использованием ссылок. Функция **getText**(n) отыскивает все узлы заданного типа, вложенные в корневой **n**, объединяет их содержимое и возвращает результат в виде строки. Операция объединения строк очень ресурсоемка, потому сначала содержимое текстовых узлов помещается в массив, затем выполняется операция конкатенации элементов массива в одну строку.

```
<script type="text/JavaScript">
function getText(n) {
   var strings = [];
   getStrings(n, strings); return strings.join("|");
   // Эта рекурсивная функция отыскивает все текстовые узлы
   // и добавляет их содержимое в конец массива.
   function getStrings(n, strings) {
      if (n.nodeType == 1) strings.push(n.nodeName);
      if (n.nodeType == 1 ) {
         for(var m=n.firstChild; m != null; m=m.nextSibling)
           { getStrings(m, strings);
         }
      }
   }
}</script>
<body onload="alert(getText(document.body))">
<p>Это <i>пример</i> документа.</p></body>
```

```
if (n.nodeType == 1)
strings.push(n.nodeName);
```


```
if (n.nodeType == 3)
strings.push(n. nodeValue);
```



4.4.3. Интерфейсы Node. Добавление, удаление узлов

Динамическое изменение DOM является основой создания «живых» страниц. Рассмотрим, как создавать новые элементы «на лету» и заполнять их данными. Для создания элементов DOM используются методы документа, рассмотренные в разделе 4.3. В большинстве случаев создаются узлы двух видов: элементы или текстовый. Для присоединения созданных узлов к DOM и удаления существующих используются следующие методы объекта **Node**:

– **узел.appendChild(newChild)** – добавляет узел **newChild** в конец списка детей данного узла и возвращает его в качестве результата. Если узел **newChild** уже был в списке, то он сначала удаляется, а затем добавляется;

– **узел.insertBefore(newChild, refChild)** – вставляет узел **newChild** в список детей данного узла перед узлом **refChild** и возвращает его в качестве результата. Если узел **newChild** уже был в списке, то он сначала удаляется, а затем добавляется;

– **узел.cloneNode(deep)** – создает копию данного узла и возвращает ее. Если аргумент **deep** равен **true**, то создается копия поддерева документа, начиная с данного узла; если он равен **false**, то копируется только сам узел и его атрибуты, если это **Element**;

– **узел.removeChild(oldChild)** – удаляет узел **oldChild** из списка детей данного узла и возвращает его в качестве результата;

– **узел.replaceChild(newChild, oldChild) –** заменяет узел **oldChild** в списке детей данного узла на узел **newChild** и возвращает **oldChild** в качестве результата. Если **newChild** уже был в списке детей, то он удаляется из него до замены;

Документ HTML – это сложная структура, связанная с CSS и с механизмом отображения. При вставке узла происходят разные внутренние скрытые события и обновления данных, распространяющиеся по всему дереву. Поэтому вставка в элемент, не привязанный к доку-

менту, всегда быстрее. Структура данных проще, зависимостей меньше. Таким образом, можно рекомендовать следующую последовательность – создать элемент, наполнить данными, а затем присоединить к документу. Для создания крупных ветвей лучше использовать объект documentFragment, наполнять его и после добавлять к дереву. Когда documentFragment вставляется в DOM – то он **исчезает**, а вместо него вставляются его потомки.

Пример. Создание узла документа и добавление к дереву:

```
<script type="text/JavaScript">
function test(){
  var text = '<b>Добавляемый текст!</b>';
      var parentElem = document.getElementById("b1");
      elem = document.createElement('div');
      elem.innerHTML = text;
      parentElem.appendChild(elem);
      elem.appendChild(document.createTextNode(text));
      parentElem.appendChild(elem);
}</SCRIPT>
<BODY id="b1">
<h3>Заголовок</h3><P onclick='test()' my="text">Начальный текст.</P>
</BODY>
```

Начальный вид документа

Вид после добавления

Как видно из примера, содержимое текстового узла не интерпретируется браузером. Создание и вставка текстового узла работают значительно быстрее **innerHTML**.

Пример. Рекурсивная функция **createTree**() создает вложенный список (**UL**/**LI** дерево) из объекта:

```
<script type="text/JavaScript">
var data = {
    "Рыбы":{"Форель":{},"Щука":{} },
    "Деревья":{
    "Хвойные":{"Лиственница":{},"Ель":{} },
```

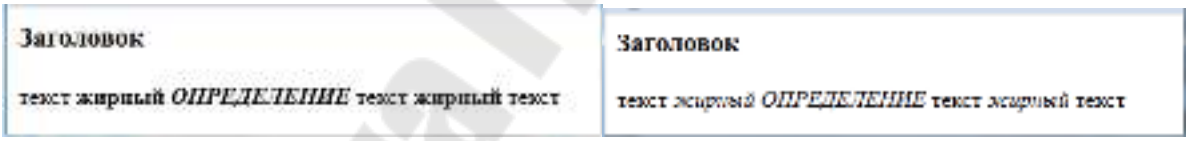
```
    "Цветковые":{"Береза":{}, "Тополь":{} } }
  };
function createTree(container, obj) {
      var ul = document.createElement('ul');
      var container = document.getElementById('container');
      container.appendChild( createTreeDom(obj) );
  }
  function createTreeDom(obj) {
      // если нет детей, то рекурсивный вызов ничего не возвращает
      // так что вложенный UL не будет создан
      if (obj.length==0) return;
      var ul = document.createElement('ul');
      for (var key in obj) {
          var li = document.createElement('li');  li.innerHTML = key;
          var childrenUl = createTreeDom(obj[key]);
          if (childrenUl) li.appendChild(childrenUl);
          ul.appendChild(li);
      }
      return ul;
  }
  var container = document.getElementById('container');
  createTree(container, data);
</script>
<body onclick="createTree('container', data);">
Это пример документа. <div id="container"></div> </body>
```

Пример. Функция **change**() заменяет один контейнер на другой. Важным моментом является то, что мы не создаем новых дочерних узлов, а копируем их из созданного клона. Причем добавление элемента из клона к новому объекту **NewElem** вызывает его автоматиче-

ское удаление из клона. Та же особенность и при замене, поскольку вставляемый **NewElem** затирает заменяемый контейнер, следовательно, следующий замещаемый контейнер будет снова иметь номер 0.

```
<script type="text/JavaScript">
function change(tagNameOld, tagNameNew){
    list = document.getElementsByTagName(tagNameOld);
    for (var i = 0; i < list.length; ) {
      var NewElem = document.createElement(tagNameNew);
      var clon=list[0].cloneNode(true);
     //Добавление узла в NewElem вызывает его удаление из clone.
      while(clon.firstChild) NewElem.appendChild(clon.firstChild);
      // или так  for(var m=clon.firstChild; m != null; m=clon.firstChild)
      // NewElem.appendChild(m);
       var par = list[0].parentNode;
       par.replaceChild(NewElem,list[0]);
 }
}</SCRIPT>
<BODY><h3>Заголовок</h3>
<P id="id1" onclick='change("b", "i")'>текст <b>жирный
<em>ОПРЕДЕЛЕНИЕ</em></b> текст <b> жирный</b> текст</P>
</BODY>
```

Начальный вид документа **Вид после добавления**



4.5. Интерфейсы работы с атрибутами

4.5.1. Интерфейс Node

Для работы с атрибутами используются следующие свойства и методы объекта **Node**:

– **узел.attributes** – возвращает именованный список (объект NamedNodeMap) всех атрибутов узла, если узел имеет тип Element, и null – в противном случае. В Internet Explorer этот список не содержит атрибута **style**;

– **узел.hasAttributes()** – метод возвращает true, если узел является элементом и имеет хотя бы один атрибут. Эквивалентно (**узел.attributes.length != 0**). В IE не работает.

4.5.2. Интерфейс Attr. Атрибуты

Интерфейс **Attr** соответствует атрибуту элемента **XML**- или **HTML**-документа. **Attr** представляет собой атрибут объекта **Element**. Он наследует все методы и свойства объекта **Node**. В модели **DOM** объект **Attr** не является частью дерева документа. Поэтому значения свойств **parentNode**, **previomSibling** и **nextSibling** равны **null**. Рассмотрим свойства объекта.

– **атрибут.name** – возвращает строку, содержащую имя данного атрибута. Введение этого свойства в DOM не вполне обосновано, так как его значение всегда совпадает со значением свойства nodeName. Свойство неизменяемое;

– **атрибут.ownerElement** – возвращает указатель на узел Element, атрибутом которого является данный атрибут, или null, если такого элемента нет. Свойство неизменяемое;

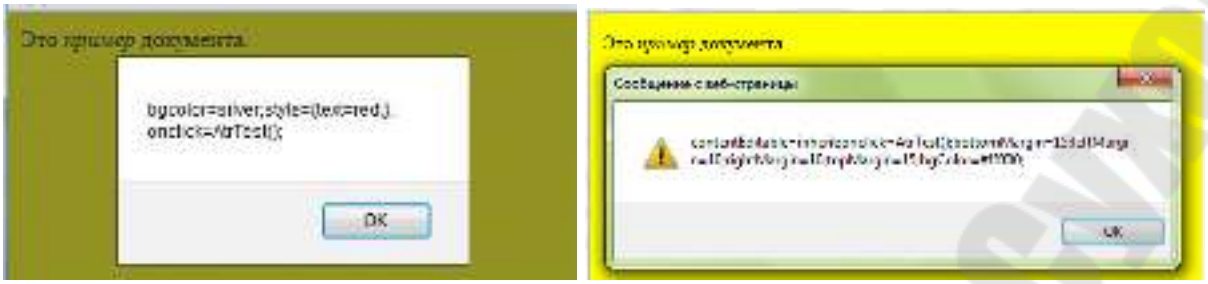
– **атрибут.specified** – возвращает **true**, если значение атрибута было явно задано в тексте документа или в сценарии, и **false** – в противном случае. Свойство неизменяемое;

– **атрибут.value** – позволяет задавать и считывать значение данного атрибута. В HTML его значение всегда совпадает со значением свойства **nodeValue**.

Пример. Работа с атрибутами через интерфейс **Attr**:

```
<script type="text/JavaScript">
function AtrTest() {
alert(document.body.hasAttributes()); // в IE не работает
var attrs = document.body.attributes; // создание списка атрибутов
var str="";
for (var i = 0; i < attrs.length; i++)
  if (attrs[i].nodeValue) str+=attrs[i].name + "="+ attrs[i].value + ";";
  // или так if (attrs[i].nodeValue) str+=attrs[i].nodeName + "="+
attrs[i].nodeValue + ";";
attrs["bgColor"].value="yellow"; // изменение атрибута
// или так attrs["bgColor"].nodeValue="yellow"; // изменение атрибута
alert(str);
}</script>
<body bgcolor="silver" onclick="AtrTest()" style="{text=red;}">
<p>Это <i>пример</i> документа.</p>
</body>
```

FireFox 10.0 IE9.0



4.5.3. Интерфейс Element. Атрибуты

Объект **Element** представляет собой элемент XML- или HTML-документа. Объект Element наследует все методы и свойства объекта **Node**. Основные свойства и методы объекта:

– **элемент.tagName** – возвращает строку, содержащую имя тега данного элемента. Свойство является избыточным, так как его значение всегда совпадает со значением свойства nodeName;

– **элемент.getAttribute(name)** – метод возвращает значение атрибута, имеющего имя name. Если атрибут не имеет значения, то возвращается пустая строка;

– **элемент.getAttributeNode(name)** – метод возвращает указатель на объект Attr, соответствующий атрибуту с именем name. Если такого атрибута нет, то возвращается null;

– **элемент.getElementsByTagName(tagname)** – метод **getElementsByTagName** создает новый объект **NodeList**, состоящий из всех потомков данного элемента типа **Element**, имеющих заданное имя, и возвращает указатель на него. Аргумент **tagname** задает искомый тег; если он равен "*", то возвращается список всех элементов документа. В IE можно использовать нестандартное свойство **element.all**;

– **элемент.hasAttribute(name)** – метод возвращает true, если элемент имеет атрибут с именем name, и false – в противном случае. Не поддерживается IE, но можно так: **элемент.getAttribute(name) != null**;

– **элемент.removeAttribute(name)** – метод удаляет атрибут с именем name. Если атрибут имеет значение по умолчанию, то он немедленно появляется вновь, со значением по умолчанию;

– **элемент.removeAttributeNode(oldAttr)** – метод удаляет заданный узел **oldAttr** и возвращает указатель на него. Если атрибут имеет значение по умолчанию, то он немедленно появляется вновь, со значением по умолчанию;

– **элемент.setAttribute(name, value)** – метод создает новый атрибут с именем name и значением **value**. Если атрибут с таким именем уже есть, то ему просто присваивается новое значение;

– **элемент.setAttributeNode(newAttr)** – метод добавляет к данному элементу новый узел **newAttr**. Если атрибут с именем **newAttr**.**nodeName** уже есть, то он заменяется на новый узел, а указатель на старый узел возвращается в качестве результата. Если такого атрибута нет, то возвращается **null**.

Пример. Работа с атрибутами через интерфейс **Element**:

```
<script type="text/JavaScript">
function AtrTest() {
 var elem = document.body;
 // число всех элементов документа
 var List = elem.getElementsByTagName("*"); alert(List.length);
 // число всех элементов внутри первого абзаца
List = elem.getElementsByTagName("p")[0].getElementsByTagName("*");
alert(List.length);
 alert(document.body.getAttribute("text")); // значение атрибута text
 // значение атрибута bgcolor, через ссылку на узел атрибута
 var attr = document.body.getAttributeNode("bgColor"); alert(attr.nodeValue);
 //alert(elem.hasAttribute("bgColor")); //Определяет наличие атрибута.
                                        // В IE не работает
 attr.nodeValue="yellow"; // установка нового значения атрибута bgcolor
 // удаление атрибута из узла (узел остается)
 elem.removeAttribute("bgColor"); alert(elem.getAttribute("bgColor"));
 // удаление узла атрибута
 elem.removeAttributeNode(elem.getAttributeNode("text"));
 // создание атрибута в узле.
 elem.setAttribute("bgColor","#FFCC66");
 // создание узла атрибутов пользователя
 var attr = document.createAttribute("temp");
 attr.value = "temporary"; document.body.setAttributeNode(attr);
 alert(document.body.getAttribute("temp"));
 // создание узла атрибутов цвета текста документа
 var attr = document.createAttribute("text");
 attr.value = "blue"; document.body.setAttributeNode(attr);
 alert(document.body.getAttribute("text"));
}
</script>
<body bgColor="silver" onclick="AtrTest()" text="red">
<p color="blue">Это <i>пример</i> документа.</p>
</body>
```

4.5.4. DOM-свойства и атрибуты

У DOM-элементов в javascript есть свойства и атрибуты. И те, и другие имеют имя и значение, но это не одно и то же. Между ними есть соответствие, но оно не однозначное. С точки зрения **javascript,** узлы DOM являются объектами. У объектов есть свойства, поэтому любому узлу можно назначить свойство, используя обычный синтаксис.

```
var elem = document.getElementById('MyElement'); elem.myProperty = 5;
```

Если посмотреть на DOM-элемент с другой стороны, то, являясь элементом HTML, DOM-элемент может иметь любое количество атрибутов. В следующем примере элемент имеет атрибуты **id**, **class** и нестандартный (валидатор будет ругаться) атрибут **alpha**.

```
<div id="MyElement" class="big" alpha="omega"></div>
```

Атрибуты можно добавлять, удалять и изменять рассмотренными ранее методами. Имя атрибута является **регистронезависимым**. Значением атрибута может быть только строка.

Создатели javascript решили (с лучшими намерениями) запутать ситуацию и создать искусственное соответствие (синхронизацию) между свойством и атрибутом. То есть браузер синхронизирует значения ряда свойств с атрибутами. Если меняется атрибут, то меняется и свойство с этим именем, и наоборот.

```
document.body.id = 5; alert(document.body.getAttribute('id')); // результат 5
document.body.setAttribute('id', 'NewId');
alert(document.body.id); // результат 'NewId'
```

Такая синхронизация гарантируется для всех основных стандартных атрибутов. При этом атрибуту с именем **class** соответствует свойство **className**, так как ключевое слово **class** зарезервировано в **javascript**. Для нестандартизированных атрибутов браузер ничего не гарантирует. Название атрибута не зависит от регистра. Атрибуты с именами "**abc**" и "**ABC**" – один и тот же атрибут.

```
document.body.setAttribute('abc', 1); document.body.setAttribute('ABC', 5);
alert(document.body.getAttribute('abc')) // результат 5
```

Но свойства в разных регистрах - два разных свойства.

```
document.body.abc = 1;document.body.ABC = 5;
alert(document.body.abc); // результат 1
```

В IE версии до 8.0 с этим проблемы. Этот браузер старается по возможности уравнять свойства и атрибуты. Но как быть, если свойства – регистрозависимы, а атрибуты – нет? Создатели IE поступили хитро: **setAttribute** ставит оба свойства (abc и ABC). А **getAttribute** возвращает первое попавшееся из них, с учетом регистронезависимо-

сти. Если таких свойств несколько, то невозможно сказать, какое именно он вернет.

Атрибут можно установить любой, а свойство – нет. Например, можно установить для тэга <**body**> атрибут **tagName**, но соответствующее свойство – только для чтения, поэтому оно не изменится. Строго говоря, браузер не гарантирует синхронизацию атрибута и свойства.

ЯЗЫК PHP

PHP (*Hypertext Preprocessor*) – язык программирования для создания *скриптов*, работающих на стороне WEB-сервера. Скрипты вставляются в HTML-страницы и выполняются на стороне сервера перед выдачей страницы браузеру. Область применения – динамическое формирование и обработка HTML-страниц. PHP поддерживается практически на всех известных платформах, операционных системах и WEB-серверах. В PHP сочетаются две парадигмы программирования – объектная и процедурная.

Основные возможности: работа с текстовыми данными, поддержка различных БД, работа с файловой системой, аутентификация доступа и др. PHP поддерживает взаимодействие с различными сервисами посредством соответствующих протоколов: протокол управления доступом к директориям LDAP, протокол работы с сетевым оборудованием SNMP, протоколы передачи сообщений IMAP, NNTP и POP3, протокол передачи гипертекста HTTP и т. д.

Получив исходный код PHP анализирует его и затем транслирует в байт – код, который уже исполняется (интерпретируется). Таким образом, PHP составлен из двух почти независимых блоков – **транслятора** и **интерпретатора**. Такое разделение выполнено для повышения быстродействия. Основные настройки работы PHP содержатся в файле конфигурации **php.ini**.

1. Синтаксис PHP. Типы, переменные, константы

1.1. Вставка PHP-кода в HTML-документ

При обработке файла PHP передает его текст на выход без изменений, пока не встретит один из специальных тегов, после которого начинает работу транслятор. Такой механизм позволяет вставлять произвольное число PHP-блоков и отделять их от HTML-кода. Существует четыре набора тегов для обозначения PHP-кода:

1. <?php **код** ; ?> 2. <script language="php"> **код** ; </script>

3. <? **код** ; ?> 4. <% **код** ; %>

Первый и второй способы основные и доступны всегда, другие включаются в конфигурационном файле «**php.ini**».

1.2. Операторы включения

Инструкции включения **include** и **require** позволяют разбить программу на несколько файлов. Синтаксис операторов одинаков:

include 'имя_файла'; **include** $file_name; **include** ("имя_файла");

Оператор **include** позволяет включать код, содержащийся в указанном файле, и выполнять его каждый раз при включении. Код включаемого файла наследует ту же область видимости переменных, что и строка, на которой произошло включение. Если включение происходит внутри функции включающего файла, тогда весь код, содержащийся во включаемом файле, будет вести себя так, как будто он был определен внутри этой функции.

Поиск файла для вставки происходит в следующем порядке (include_path – задается в файле конфигурации **php.ini**):

– в **include_path** относительно текущей рабочей директории;

– в **include_path** относительно директории текущего скрипта.

При обнаружении **include** транслятор прерывает выполнение, переходит в режим HTML и начинает трансляцию включаемого файла (поэтому в подключаемом файле обязательно присутствие **<?php** , в начале). Затем снова возвращается в главный файл. Таким образом, **require** работает гораздо быстрее. Оператор включения **require** вставляет файл единожды на этапе трансляции, т. е. до выполнения сценария.

Конструкцию **require** целесообразнее использовать там, где не требуется динамическое включение файлов в сценарий (например, шаблонных страниц с HTML-кодом), а **include** только с целью динамического включения файлов в код PHP-скрипта. Обе конструкции также поддерживают включение удаленных файлов (если разрешено настройками PHP, директива **allow_url_fopen**).

При использовании операторов включения существует вероятность ошибки включения одного и того же блока через другие подключаемые модули. Для гарантирования однократного включения необходимо использовать операторы: **require_once** и **include_once**.

1.3. Основные типы данных. Задание переменных и констант

PHP относится к языкам со *свободной типизацией данных.* При использовании разнотипных переменных в выражениях PHP автоматически приводит данные к нужному типу. Если целое число превышает пределы типа, оно будет интерпретировано как с плавающей

точкой. В PHP практически не существует ограничений на размер строк. Основные типы данных: **string**, **integer**, **float**, **Boolean**, **Array**, **Object**, **Null**, **Resourse**.

Переменные в PHP определяются именем со знаком доллара (**$name**). Имя переменной чувствительно к регистру. Одной переменной можно присваивать различные типы данных:

$a = -123; **$a** = 1.234; **$var** = "Bob"; **$Var** = 'Bob';

PHP позволят задавать два вида ссылок – **жесткие** и **символические**. Для создания жесткой ссылки используется символ **&**. Так, в выражении **$x = &$y** переменная **$x,** ссылается на **$y**. Любое изменение одной переменной приводит к изменению другой. **Символическая ссылка** – это строковая переменная, хранящая имя другой переменной. Для создания используется двойной **$$**:

$a = "hello"; $$a = "world";

echo "$a ${$a}"; или echo "$a $hello"; – **Результат**: hello world

echo "$$a"; – **Результат**: $hello

Константы могут задаваться только скалярного типа данных. Создаются функцией: define("имя_константы", значение).

При использовании констант приставка знака доллара ($) не нужна. Проверить, определена ли константа, можно функцией:

boolean defined(имя_константы1, имя_ константы2,...имя_ константыN)

1.4. Операторы вывода

Для вывода строк и значений переменных используется оператора **echo**. Синтаксис:

echo $varl, $var2,..., $varN; **echo** "Переменные: $varl, $var2,...,$varN";

echo 'Переменные: $varl, $var2,...,$varN';

В первых двух случаях вместо переменных будут выведены их значения. В третьем случае (одинарные кавычки) строка выводится без подстановок значений переменных. Для форматирования числовых данных можно использовать функцию:

number_format(число, точность, разделитель1, разделитель2)

Функция возвращает строку, содержащую отформатированное число, и принимает следующие параметры: **число** – число (**float**), которое требуется отформатировать; **точность** – целое число знаков в дробной части числа; **разделитель1** – строка с символом для разделения целой и дробной частей; **разделитель2** – строка с символом для разделения групп по три разряда в целой части числа.

number_format(123456.789, 2, "рублей", " "); **Результат**: 123 456р79

Для вывода информации о переменных, которая может понадобиться при отладке программы, применяются функции:

string **var_dump**(список_переменных); - подробная информация

string **print_r**(список_переменных); – только значения переменных

1.5. Преобразование типов

В PHP тип переменной определяется по контексту, в котором она используется. В выражениях и операторах тип преобразуется автоматически. При преобразование строк в числа значение определяется по начальной части строки. Если строка начинается с верного числового значения, будет использовано это значение. Иначе значением будет 0 (ноль). Строка будет распознана как **float**, если она содержит любой из символов '**.**', '**e**', или '**E**'. Иначе она будет определена как целое.

При преобразовании в логический тип следующие значения рассматриваются как **FALSE**: 0 или 0.0, пустая строка и строка "0", пустой массив, тип **NULL**. Все остальные значения рассматриваются как **TRUE**, включая отрицательные числа.

```
$foo = 4 + "10.2 Kg";  – результат $foo = float (14.2)
$foo = (bool) array();  – результат $foo = false
$foo = (bool) -2;    – результат $foo =  true
```

Явное приведение типов в PHP работает так же, как и в C. Допускаются следующие приведения типов: (**int**), (**integer**), **intval** – приведение к целому числу; (**bool**), (**boolean**) – приведение к булеву типу; (**float**), **floatval** – приведение к числу с плавающей точкой; (**string**), **strval** – приведение к строке; (**array**) – приведение к массиву; (**object**) – приведение к объекту.

```
$foo = 10;   $bar = (boolean) $foo;   $bar = intval(4.2);
```

1.6. Функции преобразования и проверки типов

Для преобразования типа можно использовать функцию:

bool **settype** (mixed &$var, string $type)

var – имя переменной; **type** – строка задающая тип: "**boolean**", "**integer**", "**float**", "**string**", "**array**", "**object**", "**null**".

Возвращает **TRUE** в случае успешного завершения.

Для проверки типа – функция: string **gettype** (mixed $var). Возвращает строку, соответствующую типу переменной. Для проверки, определена ли переменная, служит функция: **isset**(имя_переменной1,

имя_переменной2,…,имя_переменнойN). Если все переменные в списке существуют, то функция возвращает **true**, в противном случае – **false**. Для очистки переменной от значения используется функция: **unset** (имя_переменной). В результате выполнения функции переменная приобретает значение **null**.

Для определения, является ли значение переменной пустым, используется функция: **empty**(имя_переменной). Возвращает **true**, если значение переменной равно 0 (целое число), "" (пустая строка), "0", **false**, **null**, переменная не объявлена или является пустым массивом. В противном случае возвращается **false**.

Для проверки конкретного типа можно использовать функции: **is_string, is_int, is_float, is_numeric, is_null, is_array**, которые возвращают **true** при соответствии типа.

Синтаксис операторов и управляющих конструкций аналогичен языку Си.

2. Массивы. Работа с массивами

Массивы в PHP являются основным типом данных. Многие встроенные функции используют его в качестве аргументов или для возврата результата. Для обращения к элементам используются индексы или пары ключ–значение. Массив оптимизирован в нескольких направлениях, поэтому его можно использовать как обычный массив, ассоциативный массив, список (вектор), хеш-таблицу, стек, очередь и т. д.

2.1. Определение массивов

Для задания массивов применяются следующие способы:

1. С помощью конструктора **array()**:

```
$name = array( value1, value2, ... )
$name = array([key1] => value1, [key2] => value2, ... )
```

Ключ **key** может быть любого типа, кроме массив и объект. Числовой ключ является **индексом**.

```
$books = array ("php" =>"PHP users guide", 12 => true);
$arr1 = array(5 => 43, 32, 56, "b" => 12);
```

2. С помощью синтаксиса квадратных скобок:

$name["new_key"]="new_value", $name[]="new_value" – новый элемент имеет числовой ключ, на единицу больший максимального су-

ществующего. Если массив, в который добавляется значения, еще не существует, он будет создан. Нумерация элементов массива с нуля.

3. Многомерные массивы определяются через одномерные, элементами которых являются массивы. Обращение к элементам многомерного массива: имя_массива [индекс_уровня1]..[индекс_уровняN];

```
$сотрудники= array (array("Иван", 100), array("Василий", 200));
echo $сотрудники[1][0]; Результат: Василий
```

2.2. Операции с массивами

При выводе элемента массива в составе текстовой строки его следует заключить в фигурные скобки.

```
$myarray=array("Иван", "Федоров", 125.7);
echo "Сотрудник {$myarray[1]} получает {$myarray[2]} руб.";
```

Для удаления элемента массива используется функция **unset**().

```
unset($books[12]); Результат – удаление элемента с ключом 12
unset ($books); Результат – удаление массива полностью
```

Копирование массива выполняется с помощью оператора присваивания. Изменение значения элемента в одном из них никак не отразится на соответствующем элементе в другом.

```
$a=array(5, 2, 4, 3); $x=$a; // Результат – копия массива
```

Объединение массивов выполняется оператором «+». Результатом объединения двух массивов $a и $b будет массив $c, состоящий из элементов $a, к которым справа дописаны элементы массива $b с ключами, не встречающимися в $a. Если встречаются совпадающие ключи, то в результирующий массив включается элемент из первого массива. При объединении массивов важен порядок слагаемых.

С массивами можно выполнять операции сравнения. **Равенство массивов** – совпадение всех пар **ключ=>значение**. **Эквивалентность** – это равенство массивов с совпадением порядка элементов. Равенство значений проверяется оператором «==», а эквивалентность – «===».

2.3. Перемещение по массиву

Для перебора элементов массивов можно использовать операторов цикла **while**, **for** или специальный **foreach**, который имеет две синтаксические формы:

```
foreach ($array as $value) { блок_выполнения }
foreach ($array as $key => $value) { блок_выполнения }
```

Конструкция **foreach** последовательно перебирает все элементы массива, присваивая их значения переменной $**value**, а индексы – $**key**, которые доступны в блоке выполнения.

foreach($a as $k => $v) { print "\$a[$k] => $v.\n"; }

foreach($a as $v1) { foreach ($v1 as $v2) { print "$v2\n"; } } – для 2-хмерного

Перебор элементов массива можно выполнить и с помощью специальных функций:

– **current***(*массив*)* – возвращает значение текущего элемента;

– **each**(массив) – возвращает текущую пару ключ/значение, перемещая указатель на следующий элемент. Пара возвращается в виде **массива** из 4 элементов: [**0**] = ключ, [**1**] = значение, [**key**] = ключ, [**value**] = значение;

– **next**(массив) – перемещает указатель на следующий элемент;

– **prev**(массив) – перемещает указатель на предыдущий элемент;

– **end**(массив) – перемещает указатель на последний элемент;

– **reset**(массив) – перемещает указатель на начальный элемент.

Функции **prev()** и **next()** при выходе за пределы массива возвращают **false**. Лучше использовать проверку жесткого равенства «===».

$foo =array("bob", "Robert" => "Bob")

$b= each ($foo); Результат: [0] = 0 [1] => 'bob' [key] => 0 [value] => 'bob'

$b=each($foo); Результат:[0]='Robert' [1]='Bob' [key]='Robert' [value]= 'Bob'

Для присвоения списку переменных значений массива используется оператор **list**. Работает только с числовыми массивами и подразумевает, что числовые индексы начинаются с 0.

list(список_переменных) = массив;

Для обработки элементов массива можно использовать следующие конструкции:

```
reset ($arr);
while (list(, $value) = each ($arr)) {
  echo "Value: $value<br>\n";
}
```

```
reset ($arr);
while (list ($key, $value) = each ($arr)) {
  echo "Key: $key; Value: $value<br>\n";
}
```

2.4. Функции работы с массивами

Рассмотрим основные функции, используемые при обработке массивов:

– int **count**(mixed $var [, int $mode]) – число элементов массива. Для многомерного массива задается параметр **mode** = **COUNT_RECURSIVE**, устанавливающий рекурсивное применение;

– **bool** in_array**(mixed "искомое значение", array "массив", [bool "по типу"])** – определяет, содержится ли в массиве искомое значение. Третий аргумент задает поиск с совпадением типа. Если искомое значение – строка, то сравнение чувствительно к регистру. Искомым значением может быть массив;

– bool **array_search**(mixed"искомое значение",array "массив",[bool"по типу"]) – возвращает ключ, если элемент найден, и **false** – если нет;

– array **array_keys**(array "массив", [mixed "значение для поиска"]) – возвращает массив строковых и/или числовых ключей. Поиск регистрозависимый. Если второй параметр не задан, возвращаются все ключи;

– **array** array_unique**(array "массив")** – возвращает новый массив без повторяющихся значений, с сохранением ключей. При сравнении элементы массива рассматриваются как строки (elem1 === $elem2);

– bool **sort** (array массив [, int флаги]); bool **rsort** (array массив [, int флаги]) – сортировки по возрастанию и убыванию значений, соответственно. Функции удаляют существовавшие ключи, заменяя их числовыми индексами, соответствующими новому порядку элементов. Флаг сравнения: **SORT_REGULAR** – сравнивать элементы обычным образом; **SORT_NUMERIC** – сравнивать элементы как числа; **SORT_STRING** – сравнивать элементы как строки;

– bool **asort**(array массив [, int флаги]); bool **arsort**(array массив [, int флаги]) – сортировки по возрастанию и убыванию значений с сохранением индексов;

– bool **ksort**(array массив [, int флаги]); bool **krsort**(array массив [, int флаги]) – сортировки по возрастанию и убыванию ключей;

– string **implode**(string разделитель, array массив) – возвращает строку, полученную объединением строковых представлений элементов массива, со вставкой разделителя между ними;

– array **explode**(string разделитель, string строка [,int количество]) – возвращает массив строк, полученных разбиением исходной строки, с использованием заданного разделителя. Аргумент «**количество**» определяет максимальное число элементов, при этом последний элемент будет содержать остаток исходной строки;

– int **extract**(array массив [, int тип [, string префикс]]) – создает переменные из символьных индексов массива и присваивает им значения соответствующих элементов массива, согласно параметрам тип и префикс, возвращает количество созданных переменных. Параметр

«**тип**» задает и определяет способ обработки коллизии имен: **EXTR_OVERWRITE**, **EXTR_SKIP**, **EXTR_PREFIX_SAME** и т. д.;

– array **compact**(mixed переменная [, mixed переменная...]) – создает массив, содержащий имя переменных в качестве ключа и ее значения.

2.5. Глобальные предопределенные массивы

В PHP имеются предопределенные массивы с глобальной областью видимости. Их еще называют **автоглобальными** или **суперглобальными массивами**, поскольку они доступны из любого сценария на языке PHP. Ниже приведен их список основных: **$GLOBALS** – содержит все глобальные переменные, как предопределенные, так и созданные (например: **$GLOBALS["MYVAR"]**); **$_GET**, **$_POST** – содержит данные, переданные в сценарий методом **GET** или **POST**; **$_SERVER** – содержит переменные, установленные Web-сервером.

В более ранних версиях PHP использовались глобальные массивы с длинными именами: **$HTTP_GET_VARS**, **$HTTP_POST_VARS** и др. В **PHP** 5 они доступны, хотя и не рекомендуются к использованию.

3. Строки. Работа со строками

Строки в **PHP** – одни из самых универсальных и наиболее часто используемых объектов, при создании Web-сайта. Строки не имеют специального маркера конца строки. Длина строки ограничена только свободной памятью.

3.1. Задание строк. Основные операции

Для задания строки используются одинарные или двойные кавычки. Последовательность, заключенная в одинарные кавычки, хранится без преобразований. В строках в двойных кавычках выполняется обработка переменных и специальных символов, и лишь после этого может использоваться сама строка.

```
$name = "Сэм"; $outputl = "$name";
```

Для обращения к отдельным символам можно использовать синтаксис **строка{номер символа}**: $string{3};

Для конкатенации строк используется символ «**.**».

```
$a = "20"; $b = "10"; Sc = $a.$b;  Результат: 2010.
```

Для сравнения строк лучше использовать оператор сравнения ===, или явное приведение типа, поскольку пустая строка "" прежде всего трактуется как 0 (ноль) и уж затем – как "пусто".

int **strcmp**(string строка1, string строка2) – посимвольное сравнение строк без учета регистра. Результат отрицательное число, если строка1 меньше строки2; положительное, если строка1 больше строки2, и 0, если равны. Для сравнения с учетом регистра – функция **strcasecmp**.

3.2. Функции работы со строками

Для преобразования символьных строк в **PHP** имеется большое количество встроенных функций. Рассмотрим наиболее важных из них:

– int **strlen**(string строка) – длина строки с учетом служебных символы;

– int **strpos**(string строка, string подстрока [, int смещение]) – возвращает позицию первого вхождения подстроки в строке, начиная с заданного смещения. Если подстрока не найдена, возвращает **FALSE**;

– string **trim**(string строка [, string список_символов]) – возвращает строку с удаленными из начала и конца строки пробелами. Второй аргумент задает список (можно диапазон) символов для.

Пример. Удалить управляющие символы (**ASCII код 0 до 31**) из начала и конца строки: **trim**($binary, "\x00..\x1F");

– string **strstr**(string строка, string подстрока) – возвращает часть строки, начиная с заданной подстроки и до конца строки. Если подстрока не найдена, возвращает **FALSE**. Поиск регистрозависимый (**stristr()** – регистронезависимый);

– string **substr**(string строка, int начало [, int длина]) – возвращает подстроку с заданной длиной, начиная с заданной позиции;

– mixed **str_replace**(mixed поиск,mixed замена,mixed объект) – возвращает строку или массив, в котором все вхождения «**поиск**» заменены на «**замена**», для исходных данных – «**объект**». Аргумент может быть массивом. Функция **str_replace** чувствительна к регистру, ее регистронезависимый аналог – **str_ireplace()**. Если объект, в котором производится поиск и замена, является массивом, то эти действия выполняются для каждого элемента массива, результат – новый массив;

– string **strip_tags** (string строка [, string допустимые теги]) – возвращает строку, из которой удалены HTML- и PHP-тэги. Второй аргумент используется для указания тэгов, которые не должны удаляться;

– string **htmlspecialchars**(string строка [, int $quote_style [, string $charset]]) – преобразует **специальные символы** в их HTML-представление. Второй аргумент **quote_style** определяет режим обработки кавычек: **ENT_COMPAT** – преобразуются только двойные кавычки; **ENT_QUOTES** – преобразуются двойные и одинарные; **NOQUOTES** – без преобразования:

$new = **htmlspecialchars**("Test", ENT_QUOTES);

Результат: <a href='test'>Test</a&gt

– string **htmlentities**(string строка [, int $quote_style [, string $charset]]) – преобразует **все** символы в их HTML-представление. Обратное преобразование: **html_entity_decode**();

– string **addslashes**(string строка) – возвращает сроку, в которой перед каждым спецсимволом добавлен обратный слэш. Функция обычно применяется при использовании данных в запросах к базе данных. Обратное преобразование: **stripslashes**().

4. Функции в PHP

4.1. Определение функции. Передача параметров

В общем виде объявление функции имеет следующий синтаксис:

```
function Имя_функции ($параметр1, $параметр2, ... $параметрN) {
   Тело функции;  return "значение возвращаемое функцией";
}
```

Имена функций нечувствительны к регистру. Функция может использоваться раньше ее определения. Исключение составляют функции, определяемые условно, т. е. внутри условных операторов или других функций. Данные в функцию можно передавать тремя различными способами: по значению; по ссылке; значения аргументов по умолчанию. Для передачи по ссылке в определении функции перед именем аргумента записывается знак «&».

```
function Sqr(&$data){ $data*= $data; }
```

Значения аргументов, используемые по умолчанию, должны быть константными (можно массивы и NULL) и задаются **после всех** остальных аргументов в определении функции.

```
function makecoffee($types = array("cappuccino"), $coffeeMaker = NULL)
{ $device = is_null($coffeeMaker) ? "hands" : $coffeeMaker;
 return "Making a cup of ".join(", ", $types)." with $device."; }
echo makecoffee();  Результат:  Making a cup of cappuccino with hands.
```

```
echo makecoffee(array("cappuccino", "lavazza"), "teapot");
```

Результат: Making a cup of cappuccino, lavazza with teapot.

Для использования внешних глобальных переменных внутри функции их нужно объявить в теле функции, с ключевым словом **global**. Пример: **global** $a;

4.2. Списки аргументов переменной длины

PHP позволяет создавать функции с переменным числом аргументов. Для работы с аргументами применяются функции PHP:

– int **func_num_args**(void) – возвращает число аргументов, переданных в текущую функцию;

– mixed **func_get_arg**(int $arg_num) – возвращает аргумент из списка переданных в функцию, номер которого задан параметром **arg_num**;

– array **func_get_args**(void) – возвращает массив, состоящий из списка аргументов, переданных функции.

4.3. Возврат значения функцией. Переменные функции

Возвращаемое функцией значение может быть любого типа, включая массивы и объекты. При возврате массива в программе удобно использовать функцию **list**(), которая присваивает значения указанным переменным.

Также функция может возвращать ссылку. Для этого необходимо объявить имя функции со знаком амперсанд (**&**) и указывать его при вызове функции. Обычно функция возвращает ссылку на глобальную переменную (или ее часть) или ссылку на один из аргументов, если он был также передан по ссылке.

Пример

```
$a = 3; $b = 2;
function & ref($par){
global $a, $b;   if ($par % 2 == 0) return $b;  else return $a; }
$var =& ref(4); echo $var, " и ", $b"<br>"; // выведет 2 и 2
$b = 10; echo $var, " и ", $b"<br>"; // выведет 10 и 10
```

PHP поддерживает концепцию **переменных функций**. Если **имя переменной** заканчивается круглыми скобками, то PHP ищет функцию с таким же именем и пытается ее выполнить. Эту возможность можно использовать для реализации обратных вызовов, таблиц

функций и др. Переменные функции не работают со встроенными функциями (необходимо использовать свою функцию-обертку).

```
$func = "Show_text"; $func("Привет!");   // вызов функции Show_text
```

5. Объекты и классы в PHP

Язык PHP5 является объектно-ориентированным языком программирования. В пятую версию внесены значительные изменения для более полной поддержки ООП – конструкторы, ссылки на объекты, интерфейсы, абстрактные классы и др. В PHP класс определяется с помощью следующего синтаксиса:

```
class Имя_класса{
/*список свойств*/
        var $имя_свойства;
/*список методов*/
        function имя_метода( ){ /* определение метода */  }
}
```

Внутри *класса* можно использовать псевдопеременную **this** для обращения к текущему экземпляру. Определение класса нельзя разносить по разным блокам php-кода. Инициализировать значения переменных можно автоматически и при создании экземпляра класса. Для задания начальных значений свойствам класса существует два метода. Можно инициализировать значения с помощью оператора **var** или с помощью функции **конструктора**. С помощью **var** можно инициализировать **только константные значения**. В PHP4 функция-**конструктор** должна иметь имя, совпадающее с именем всего **класса**. В PHP5 конструктор класса именуется **__construct**. Если PHP5 не может его обнаружить, то вызывается конструктор по имени класса. Объект нужного класса создается с помощью оператора **new**.

```
$имя_объекта = new имя_класса;
```

Для обращения к свойствам и методам объекта используется синтаксис: $имя_объекта->свойство; $имя_объекта->метод(аргументы).

Пример. Создать класс, описывающий категорию статей.

```
class Articles {
   var $title;   var $author;   var $data;
   function __construct ($t, $a){
      $this->title = $t; $this->author = $a;
      $this->data = date("Y-m-d"); // текущая дата
   }
```

```
    function show_article(){
        $art = $this->title . "<br>" . $this-> data."<br>Автор: " . $this->author;
        echo $art;
    }
}
$art = new Articles("PHP 5","А. Костарев"); // создание объекта
$art->show_article(); // вызов метода
```

Обычно каждое определение класса выполняется в отдельном файле. Если в проекте используется много классов, то подгружать их в начале файла неудобно. Для подключения классов в PHP 5 можно определить функцию **__autoload**. Когда скрипт пытается создать объект или обращается к статическому методу ранее не подключенного класса, автоматически запускается функция __autoload().

```
function __autoload($name) { include_once "mylib/".$name.".php"; }
```

В качестве $name в метод передается имя класса, которое должно совпадать в регистре с именем файла.

6. Взаимодействие браузера и PHP-скриптов

6.1. Способы передачи данных на WEB-сервер

Наиболее распространенными методами передачи данных между браузером и PHP-скриптами являются **GET** и **POST**. Передаваемые пакеты можно формировать как программно, с помощью JavaScript, так и с использованием возможностей языка **HTML-**форм. Программно формируемые пакеты передаются только методом **GET**. Для передачи данных на сервер методом **GET** скрипту на стороне браузера необходимо сформировать пары **переменная=значение**, разделяя их символом **&**, и добавить к URL-адресу PHP-скрипта через символ **?**.

Пример

```
http://www.host.com/script.php?name=Vasya&age=20
```

Данные, передаваемые методом **GET**, легко подделать, поэтому не стоит применять метод для передачи параметров, влияющих на безопасность. **POST**-запросы имеют большую безопасность и функциональность. При передачи данных из форм строка формируется автоматически браузером. Метод передачи задается атрибутом **method** формы. Имена элементов формы, атрибут **name**, необходимо задавать без использования кириллицы и спецсимволов. Поскольку в PHP они буду использоваться как имена переменных.

6.2. Обработка параметров запросов. Переменные окружения

При отправке данных на сервер передаются не только сами данные, но и ряд переменных, называемых переменными окружения и характеризующих клиента. Рассмотрим некоторые из них: **REMOTE_ADDR** – IP-адрес хоста, отправляющего запрос; **REMOTE_HOST** – имя хоста, с которого отправлен запрос; **HTTP_REFERER** – URL-ресурса, который ссылается на наш сервер (например, если пользователь перешел на сайт со страницы http://www.somehost.com/page.php, то значение поля **HTTP_REFERER** будет http://www.somehost.com/page.php); **REQUEST_METHOD** – метод, который был использован при отправке запроса; **QUERY_STRING** – информация, находящаяся в URL после знака вопроса; **SCRIPT_NAME** – виртуальный путь к программе, которая должна выполняться; **HTTP_USER_AGENT** – информация о браузере, который использует клиент; **HTTP_HOST** – имя виртуального хоста, которому адресован запрос; **REMOTE_USER** – имя пользователя, прошедшего аутентификацию; **QUERY_STRING** – строка переданных серверу параметров.

PHP имеет несколько способов получения доступа к данным, переданным клиентом. До версии PHP 4.1 доступ к таким данным осуществлялся по именам переданных переменных, которые создавались автоматически. Из-за проблем с безопасностью, начиная с версии PHP 4.1 для обращения к переменным, используются только специальные глобальные массивы – $**_REQUEST**, $**_POST** и $**_GET**. Массив $**_REQUEST** содержит данные, переданные методами **POST** и **GET**, а также с помощью **cookies**. Для импортирования переданных GET/POST/Cookie можно использовать встроенную функцию: **import_request_variables**(метод передачи (g, p, c), 'префикс').

Пример. Обращение к переменной **username**, переданной скрипту.

Через массивы: $**_GET**['username']; $**_REQUEST**['username'];

Через регистрацию: **import_request_variables**('gp', 'g_'); echo $g_username;

Для определения значения переменной окружения используется функция **getenv**('имя переменной'). **Пример**:

использованный метод: echo **getenv**('REQUEST_METHOD');

переданные параметры: echo **getenv**('QUERY_STRING');

адреса страницы отправительницы: echo **getenv**('HTTP_REFERER');

Определение адреса страницы, с которой посланы данные, полезно для предотвращения ее подмены.

6.3. Обработка элементов форм

Для доступа к содержимому тестового поля или текстовой области используется значение атрибута "**NAME**", в качестве ключа элемента массива $**REQUEST**["Name"], $**_GET**["Name"], $**_POST** ["Name"]. Для вывода многострочного текста с учетом форматирования необходимо заключить его в теги <**pre**> или применить функцию **nl2br()** – заменяющую символы перевода строки на тег <**br**> .

Пример. Доступ к данным, введенным в форме:

```
<INPUT NAME="Input" TYPE="TEXT">
<TEXTAREA NAME="Text" COLS="50" ROWS="4">
echo Текстовое поле='.$_REQUEST["Input"].'<br>'.$_POST["Input "].'<br>';
echo '<pre>'.$_POST["Text"].'</pre>'; echo nl2br($_POST["Text"]);
```

Для определения состояния выключателей также используется значение атрибута NAME в качестве ключа глобальных массивов. Если переключатель включен, то соответствующий ему элемент массива принимает значение, определенное в атрибуте VALUE. Если выключен, то значение соответствующего элемента массива не определено (NULL). Для проверки, существует ли заданный элемент массива, может использоваться функция **isset** или **empty**. **Пример:**

```
<INPUT NAME="Check1" TYPE="CHECKBOX" VALUE="яблоко">яблоко<BR>
<INPUT NAME="Check2" TYPE="CHECKBOX" VALUE="груша"> груша <BR>
if (isset ($_REQUEST ["Check1"])) echo $_REQUEST ["Check1"]."<BR>";
if (!empty ($_REQUEST ["Check2"])) echo $_REQUEST ["Check2"]."<BR>";
```

Значение переключателей типа **checkbox** и **radiobutton** можно передавать как массив. В этом случае всем элементам задаются одинаковые значения атрибута **NAME**. При передаче скрипту первый отмеченный переключатель записывается в первый элемент массива, второй отмеченный переключатель – во второй и т. д.

Пример

```
<INPUT NAME="Group1" TYPE="RADIO" VALUE="яблоко"> яблоко <BR>
<INPUT NAME="Group1" TYPE="RADIO" VALUE="груша"> груша <BR>
if (!isset($_REQUEST["Group1"]))  echo 'Не выбран ни один пункт!<br>';
else echo $_REQUEST["Group1"];
```

Для доступа к выбранным элементам списка используется массив, имя которого задает атрибут **NAME** с значениями задаваемыми атрибутом **VALUE**. При нажатии на кнопке-изображении в скрипт передаются координаты точки. Имена переменных с координатами щелчка создаются автоматически дописыванием **_x** и **_y**, к имени, заданном атрибутом **NAME**. Для определения того, что данные переда-

ны в сценарий, после заполнения формы можно проанализировать нажатие кнопки **SUBMIT**:

<input type="**submit**" name="**okbutton**" value="**OK**">

if($_POST['**okbutton**']== 'OK') { } или if(isset($_POST['**okbutton**'])) { }

Для обработки данных, введенных пользователем, удаления команд, тегов, экранирования или замены спецсимволов можно использовать следующие функции работы со строками: **htmlspecialchars**, **stripslashes**, **htmlentities**, **strip_tags**, **trim**. А также функции фильтрации данных:

– bool **filter_has_var**(int $type , string $variable_name) – проверка существования переменной указанного типа;

– mixed **filter_var**(mixed $variable [, int $filter = FILTER_DEFAULT [, mixed $options]]) – фильтрует переменную с помощью выбранного фильтра;

– mixed **filter_var_array**(array $data [, mixed $definition]) – фильтрация массива переменных;

– mixed **filter_input**(int $type, string $variable [, int $filter = FILTER_DEFAULT [, mixed $options]]) – фильтрация внешних переменных;

– mixed **filter_input_array** (int $type [, mixed $definition]) – фильтрация массива внешних переменных, где:

type – один из **INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER или INPUT_ENV**; **data** – массив со строковыми ключами, содержащий данные для фильтрации; **variable_name** – значение переменной для фильтрации; **options** – ассоциативный массив параметров, либо логическое ИЛИ флагов. Если фильтр принимает параметры, флаги могут быть указаны в элементе массива "flags"; **definition** – массив, определяющий аргументы. Допустимый ключ – строка **string**, содержащая имя переменной, и допустимое значение – или тип **filter**, или массив **array**, при необходимости определяющий фильтр, флаги и параметры. Если значение является массивом, допустимыми ключами являются filter, который определяет (тип фильтра), **flags**, который определяет любые флаги, применяемые к фильтру, и **options**, который определяет любые параметры, применяемые к фильтру; **filter** – идентификатор применяемого фильтра. Фильтры валидации данных: FILTER_VALIDATE_IP, FILTER_VALIDATE_EMAIL, FILTER_VALIDATE_FLOAT, FILTER_VALIDATE_INT, FILTER_VALIDATE_BOOLEAN, FILTER_VALIDATE_URL. Очищающие фильтры: FILTER_SANITIZE_NUMBER_FLOAT, FILTER_SANITIZE_URL, FILTER_SANITIZE_NUMBER_INT, FILTER_SANITIZE_STRING,

FILTER_SANITIZE_FULL_SPECIAL_CHARS и др. Более подробное описание – *http://www.php.ru/manual/filter.filters.validate.html.*

Пример

Проверка емала: **filter_var**('bob@ya.ru', FILTER_VALIDATE_EMAIL);

Проверка URL: **filter_var**($value01,FILTER_VALIDATE_URL);

Очистка целого числа: **filter_var**($var, FILTER_SANITIZE_NUMBER_INT);

Проверка наполненности поля формы, с фильтрацией данных.

```
if(!filter_has_var(INPUT_POST, "site_url"))
{ echo("Это поле формы не было заполнено"); }
else{ $site_url = filter_input(INPUT_POST, "site_url", FILTER_SANITIZE_URL); }
```

Если форма состоит из нескольких полей и не все нуждаются в фильтрации, можно воспользоваться обработкой массива переменных. В массиве **filters** определяются имена переменных, подлежащих проверке, назначенные для них фильтры и вспомогательные опции.

```
$filters = array ("name" => array("filter"=>FILTER_SANITIZE_STRING),
"age" => array("filter"=>FILTER_VALIDATE_INT,
                "options"=>array("min_range"=>16,"max_range"=>120)),
"email"=> FILTER_VALIDATE_EMAIL,);
$result = filter_input_array(INPUT_GET, $filters);
```

Пример. Сценарий, реализующий формирование формы и ее обработку, без повторного заполнения правильно введенных полей и с защитой от опасных данных.

$_SERVER['PHP_SELF'] – адрес текущего скрипта относительно корня сайта. Для автоматического формирования ссылки на сценарий-обработчик.

```
<html> <body> <?php
  $data_err=false;  $jobs=array('не работаю','учусь','менеджер','инженер');
  if(isset($_POST['name'], $_POST['year'], $_POST['job'])) {
    $year_now =  date("Y");
    $name =  htmlspecialchars(trim($_POST['name']));
    $year =  htmlspecialchars(trim($_POST['year']));
    $job =  htmlspecialchars(trim($_POST['job']));
    if($name == ''){ echo 'Укажите имя!<br>'; $data_err=true;}
    if(!is_numeric($year)){ $year='';  $data_err=true;}
    if ($year < 1900 || $year > $year_now) {
     $data_err=true; echo "Укажите год рождения! Допустимый диапазон значений: 1980..$year_now <br>";
    }
  } else $data_err=true;
  if(!$data_err){
   echo 'Здравствуйте, ' . $name . '!<br>';  echo 'Занятие, ' . $job . '!<br>';
    $age = $year_now - $year;  echo 'Вам ' . $age . ' лет<br>';  echo '<hr>';
```

```
  } else{
 ?>
 <form method="post" action="<?php echo $_SERVER['PHP_SELF']?>">
  Введите Ваше имя: <input type="text" name="name"
  value="<?php echo stripslashes($name)?>">
<br> Ваше занятие:  <select name='job'>
  <?php
   foreach ($jobs as $val){
    $sel= $job==$val ? " selected='selected' ":'';
    echo "<option value='$val' $sel>$val</option>";
   }  ?>
  </select> <br/>
  Введите Ваш год рождения: <input type="text" name="year"
  value="<?php echo $year?>"> <input type="submit" value="OK"> </form>
 <?php }?> </body>  </html>
```

7. Работа с файловой системой

7.1. Создание, открытие, закрытие файла

PHP предоставляет большой набор средств по работе с файлами и файловой системой. Рассмотрим подробнее некоторые функции:

resource **fopen**(string имя_файла, string тип_доступа [, bool use_include_path]) – функция возвращает указатель (типа ресурс) на открытый файл. Параметр **имя_файла** – локальный путь к файлу или URL – адрес файла в сети (если разрешено директивой **allow_url_fopen** в **php.ini**). Чтобы открыть локальный файл, нужно, чтобы PHP имел соответствующие права доступа к этому файлу. Параметр **тип_доступа** задает режим доступа к файлу и может принимать одно из следующих значений: **r** – открыть файла только для чтения; **r+** – открыть файл одновременно на чтение и запись; **w** – создать новый пустой файл. Если на момент вызова уже существует такой файл, то он уничтожается; **w+** – аналогичен **r+**, только если на момент вызова файла такой существует, его содержимое удаляется; **a** – открыть существующий файл в режиме записи, при этом указатель сдвигается на конец файла; **a+** – открыть файл в режиме чтения и записи, указатель сдвигается на конец файла. В любом режиме доступа может существовать дополнительный параметр: **b** – бинарный режим (по умолчанию) или **t** – текстовый, устанавливается режим автоматической трансляции перевода строки.

Аргумент **use_include_path** определяет, искать (**TRUE**) ли данный файл в директориях **include_path.**

При невозможности создать или открыть файл с помощью **fopen PHP** генерирует предупреждение, а функция **fopen** возвращает значение **false**. Предупреждения можно запретить с помощью символа **@** перед командой:

– bool **fclose**(указатель на файл) – закрытие соединения с файлом. Возвращает **TRUE** в случаи успеха;

– bool **unlink**(string имя_файла) **–** удаление файла. Возвращает **TRUE** в случае успеха.

7.2. Запись данных в файл

Для записи данных в файл применяются следующие функции.

– int **fwrite**(указатель_на_файл, string строка [,int длина]) – записывает строку в файл, на который ссылается указатель. Длина задает количество символов для записи. Функция возвращает число записанных байтов или **false**, в случае ошибки. Функция имеет псевдоним **fputs**();

– int **file_put_contents**(string имя_файла, mixed $data [, int $flags]) **–** запись строки или массива. Функция идентична последовательному вызову функций **fopen**(), **fwrite**() и **fclose**(). Параметр **flags** определяет режим доступа к файлу: **FILE_APPEND** – добавление данных; **LOCK_EX** – эксклюзивный доступ к файлу на время записи.

```
$h = fopen("my_file.html", "w");  $text = "Этот текст запишем в файл. \r\n";
if (fwrite($h, $text)) echo "Ок."; else echo "Ошибка при записи данных";
fclose($h);
file_put_contents($file, $text, FILE_APPEND | LOCK_EX);
```

– int **fputcsv**(указатель_на_файл [, массив [, string $delimiter [, string $enclosure]]]) – запись данных, имеющих табличную структуру (CSV – файл). Функция форматирует массив **fields** в виде **CSV** и записывает в файл **handle**. Возвращает длину записанной строки или **FALSE** при неудаче. Параметр **delimiter** задает разделитель полей (один символ). По умолчанию это запятая. Параметр **enclosure** устанавливает ограничитель полей (один символ). По умолчанию это двойная кавычка.

```
$list = array ('aaa,bbb,ccc,dddd',
'123,456,789', '"aaa","bbb"');
$fp = fopen('file.csv', 'w');
foreach ($list as $line) { fputcsv($fp, split(',', $line));}
fclose($fp);
```

Результат:

```
aaa,bbb,ccc,dddd
123,456,789
"""aaa""","""bbb"""
```

7.3. Чтение данных из файла

Рассмотрим функции чтения данных из файлов:

– string **fread**(указатель_на_файл, длина) – чтение данных из файла, пока не встретится конец файла или не будет прочитано указанное параметром **длина** число байт. Для вычисления длины файла можно использовать функцию – **filesize**(имя файла)**.**

```
$h = fopen("my_file.html", "r+"); $content = fread($h, filesize("my_file.html"));
```

Функция **filesize**() кэширует результаты своей работы. Если изменить содержимое файла **my_file.html** и снова запустить приведенный выше скрипт, то результат его работы не изменится. Чтобы этого избежать, необходимо очистить статический кэш – **clearstatcache**().

Для чтения содержимого бинарных файлов рекомендуется открывать файл с помощью дополнительного параметра доступа **b**. При получении данных из сетевых потоков чтение останавливается после того, как пакет станет доступным. Поскольку мы не можем знать заранее длину файла, чтение выполняется блоками (**8192** байт), с последующей их сборкой. Для определения конца файла используется функция – **feof**(имя файла).

```
$handle = fopen("http://www.example.com/", "rb"); $contents = '';
while (!feof($handle)) { $contents .= fread($handle, 8192); } fclose($handle);
```

– string **fgets**(указатель_на_файл [, int длина]) – чтение строки текста из файла. Синтаксис аналогичен **fread**, но длину считываемой строки указывать необязательно. Чтение оканчивается, если прочитано (**длина-1**) символов, найден символ перевода строки или конец файла. Если длина не указана, считывается **1 Кбайт** текста (**1024** символа);

– string **fgetss**(указатель_на_файл, длина [,string допустимые теги]) – считывает строку из указанного файла и удаляет все встретившиеся **html**- и **PHP**-теги, за исключением разрешенных.

Пример. В считываемых строках, удалив все теги, кроме <**b**> и <**i**>:

```
$h = fopen("my_file.html", "r");
while (!feof($h)) {$content = fgetss($h, 1024, '<b><i>'); echo $content."<br>";}
fclose($h);
```

– string **fgetc**(указатель_на_файл) – функция посимвольного чтения данных. Возвращает символ из файла, на который ссылается указатель на файл, и значение, **FALSE**, если встречен конец файла. Для проверки значения необходимо использовать оператор «===»;

– array **fgetcsv**(указатель_на_файл [, array массив [, string $delimiter [, string $enclosure]]]) – функция чтения данных из CSV файла. Функция

производит анализ строки на наличие записей в формате CSV и возвращает найденные поля в качестве массива. При ошибке или по достижению конца файла возвращает **FALSE**.

Пример

```
$row = 1;
if (($handle = fopen("test.csv", "r")) !== FALSE) {
   while (($data = fgetcsv($handle, 1000, ",")) !== FALSE) {
        $num = count($data);
        echo "<p> $num полей в строке $row: <br /></p>\n"; $row++;
        for ($c=0; $c < $num; $c++) { echo $data[$c] . "<br />\n"; }
   } fclose($handle);
```

Для чтения содержимого файла, без его предварительного открытия, можно применить специальные функции **readfile**(), **file**() и **file_get_contents**();

– int **readfile**(string имя_файла [,bool use_include_path]) – считывает файл с заданным именем и записывает его в буфер вывода. Функция возвращает число считанных байтов (символов) файла, а в случае ошибки – **FALSE**. Сообщения об ошибке можно подавить оператором **@**. Функцию можно использовать для доступа к удаленным файлам.

Пример. Вывести на экран содержимое и размер заданного файла.

```
$n = @readfile ("my_file1.html");
if (!$n) echo "Error in readfile"; else echo $n
```

Для предварительной обработки загруженных данных удобно использовать функции считывания в переменную;

– array **file**(string имя_файла [,bool use_include_path]) – возвращает массив, каждый элемент которого является строкой в файле, информацию из которого мы считываем. Символ новой строки тоже включается в каждый из элементов массива. В случае ошибки функция возвращает **FALSE**. Работает с удаленными файлами.

Пример. Вывести на экран содержимое файла, с нумерацией строк.

```
$arr = file ("my_file.html");
foreach($arr as $i => $a)  echo $i,": ".htmlspecialchars($a)."<br>";
```

– string **file_get_contents** (string имя_файла [, bool use_include_path]) – возвращает содержимое файла в виде. В случае неудачи вернет **FALSE**. Может считывать информацию из удаленных файлов.

7.4. Специальные функции работы с файлами

Для проверки существования или доступности файла для чтения или записи используются специальные функции: **file_exists**(string имя),

is_writable(string имя), **is_readable**(string имя). Функции возвращают логическое значение TRUE при успехе проверки. В качестве имени может выступать путь к файлу или папке:

– int **fseek** (указатель_на_файл, int $offset [, int $whence = SEEK_SET]) – устанавливает смещение в файле. Новое смещение, в байтах от начала файла, получается путем прибавления параметра **offset** к позиции, указанной в параметре **whence**, значения которого определяются следующим образом: **SEEK_SET** – смещение от начала файла; **SEEK_CUR** – смещение от текущей позиции; **SEEK_END** – смещение от конца файла;

– bool **rewind** (указатель_на_файл) – устанавливает в начало файловый указатель;

– bool **ftruncate**(указатель_на_файл, int size) – урезает файл до указанной длины.

Параллельное выполнение скриптов с записью в файл может нарушить целостность информации. Для безопасного выполнения записи необходимо использовать механизм блокировки файла;

– bool **flock** (указатель_на_файл, int operation) – блокирует доступ из других PHP-скриптов (но не из других процессов ОС) к файлу. Другие скрипты при попытке открыть файл остановятся и будут ждать снятия блокировки. Параметр operation задает режим блокировки: **LOCK_SH**=1 – другие процессы могут отрыть только в режиме чтения; **LOCK_EX**=2 – блокировка; **LOCK_UN**=3 – снятие блокировки. Функция работает не со всеми файловыми системами. При необходимости реализации параллельного доступа на изменение данных лучше использовать СУБД.

Пример.

```
$fp = fopen("counter.txt", 'a+');
flock($fp, LOCK_EX); // Блокирование файла для записи
операции с файлом
flock($fp, LOCK_UN); // Снятие блокировки
fclose($fp);
```

8. Авторизация доступа. Сессии

8.1. Механизм сессий. Настройка

Сессии – механизм, позволяющий создавать и использовать переменные, сохраняющие свое значение в течение всего времени рабо-

ты пользователя с сайтом. Для каждого пользователя создаются свои переменные, которые могут использоваться на любой странице сайта до выхода пользователя из системы. При входе (регистрации) пользователь получает уникальный номер (**SID**, Session IDentifier – 32 символа), позволяющие идентифицировать его в течение этого сеанса (сессии) работы с сайтом. Идентификатор передается на сервер вместе с каждым запросом клиента и возвращается обратно вместе с ответом сервера. Существует несколько способов передачи идентификатора сессии: cookies или параметры командной строки.

Cookies были созданы специально как метод однозначной идентификации клиентов и представляют собой расширение протокола HTTP. Идентификатор сессии сохраняется во временном файле на компьютере клиента. Многие пользователи отключают поддержку cookies из-за проблем с безопасностью. Второй способ передачи идентификатора используется автоматически, если у браузера выключены cookies (при соответствующей настройке PHP). В этом случае идентификатор сессии автоматически встраивается во все URL, после формирования страницы. Например:

Index **превращается в** Index

Настройки сессий в PHP, выполняется в файле php.ini.

session.save_path – папка на сервере для хранения данных сессии;

session.use_cookies – разрешение cookies при работе с сессиями;

session.use_trans_sid – разрешение добавления SID к URL;

session.auto_start – автоматический запуск сессии.

8.2. Работа с сессиями

Если в настройках сервера не настроен автоматический запуск сессий, то любой скрипт, в котором нужно использовать данные сессии, должен начинаться с команды: **session_start**(). После чего сервер создает новую сессию или восстанавливает текущую, основываясь на идентификаторе сессии, переданном по запросу. Интерпретатор PHP ищет переменную, в которой хранится идентификатор сессии (по умолчанию это PHPSESSID) сначала в cookies, потом в переменных, переданных с помощью POST и GET запросов. Если идентификатор найден, то пользователь считается идентифицированным и URL проходят автозамену, в противном случае пользователь считается новым, для него генерируется новый идентификатор.

Функцию **session_start**() нужно вызывать во всех скриптах, в которых предстоит использовать переменные сессии, причем до вы-

вода каких-либо данных в браузер. Получить идентификатор текущей сессии можно с помощью функции **session_id**(). Имя сессии можно получить функцией **session_name**().

echo **session_id**(); – 9ebca8bd62c830d3e79272b4f585ff8f

echo **session_name**(); – PHPSESSID

Для сохранения в течение сессии собственных переменных их необходимо зарегистрировать: **session_register**(список переменных). Регистрируются не значения, а **имена** переменных (без знака **$**). Зарегистрировать переменную достаточно один раз на любой странице, где используются сессии. Все зарегистрированные переменные становятся глобальными. Зарегистрировать переменную также можно, записав ее значение в ассоциативный массив $_SESSION:

$_**SESSION**['имя_переменной'] = 'значение_переменой';

Не рекомендуется одновременно использовать оба метода регистрации. Пример регистрации переменной, переданной из формы:

$_**SESSION**['login'] = $_**GET**['login']; // регистрируем переменную login

Для удаления переменных сессии функция **session_unregister** (имя переменной) или **unset**(переменная).

session_unregister('login'); или **unset**($_**SESSION**['login']);

Проверить, зарегистрирована какая-либо переменная в текущей сессии, можно функцией: **session_is_registered(имя переменной).**

Для сброса значения всех переменных сессии – функция **session_unset**(). Уничтожить текущую сессию целиком можно командой **session_destroy**() – уничтожает все данные, ассоциируемые с текущей сессией.

8.3. Использование механизма сессий

Для авторизации доступа с помощью сессий необходимо использование страницы для регистрации пользователя (index.php). Если введенные данные верны, то сессия регистрируется и пользователь получает доступ к закрытым страницам сайта. В противном случае снова выводится форма регистрации.

index.php

```
<?php function draw_form($bad_login = false) {  ?>
  <form action="" method="post">
     <input type="text" name="login"></input><br/>
     <input type="password" name="pass"></input>
     <input type="submit" name="submit"></input>
  </form>
  <?php
```

```
    if ($bad_login)  echo 'неправильный логин и/или пароль';
}
/* Проверку логина и пароля */
function check_login($login, $pass) {
    return ($_POST['login'] == 'admin') && ($_POST['pass'] == 'qwerty');
}
session_start(); // Создание сессии
if (isset($_GET['logout'])) {  // если выполняется 'logout' – выход с сайта
    session_unset(); session_destroy();
    header("Location: index.php"); // перенаправление на index.php
    exit(); // после передачи редиректа всегда нужен exit или die
            // иначе выполнение скрипта продолжится.
}
if (!isset($_SESSION['login'])) { // если мы не авторизированы
    $login = $_POST['login'];   $pass = $_POST['pass'];
    if (count($_POST) <= 0)  draw_form();
    else {
        if (check_login($login, $pass)) $_SESSION['login'] = $login;
        else  draw_form(true); // неправильный пароль
    }
}
// теперь, чтобы понять, авторизирован ли пользователь, достаточно
// проверить, содержится ли в переменной $_SESSION['login'] его ник. Или
// объявлена она или нет. Это можно сделать при помощи isset()
isset($_SESSION['login']) or die(); // если функция вернула false то die()
echo 'Здравствуйте, ' . $_SESSION['login'];
?>
<br/>это секретный контент<br/>
<a href="page.php">другая страница</a><br/>
<a href="index.php?logout">выход</a>
```

Для доступа к страницам сайта, требующим авторизации, необходимо в начале сценариев открывать сессию и проверять зарегистрированные переменные:

page.php

```
<? php session_start();
    isset($_SESSION['login']) or die('вы не вошли');
    echo 'Здравствуйте, ' . $_SESSION['login'];
?>
<br/>это секретный контент 2<br/>
<a href="index.php">на главную</a><br/>
<a href="index.php?logout">выход</a>
```

Литература

1. Олищук, А. В. Разработка WEB-приложений на PHP 5. Профессиональная работа / А. В. Олищук, А. Н. Чаплыгин. – М. : Вильямс, Диалектика, 2006.

2. Рейсиг, Дж. JavaScript. Профессиональные приемы программирования / Дж. Рейсиг. – СПб. : Питер, 2008.

3. Котеров, Д. PHP 5 / Д. Котеров, А. Костарев. – СПб. : БХВ-Петербург, 2008.

4. Флэнаган, Д. JavaScript. Подробное руководство / Д. Флэнаган. – Символ-Плюс, 2008.

5. Гудман, Д. JavaScript. Библия пользователя / Д. Гудман, М. Моррисон. – М. : Вильямс, Диалектика, 2006.

6. Зервас, К. Web 2.0. Создание приложений на PHP / К. Зервас. – Вильямс, Диалектика, 2009.

7. Колесниченко, Д. Н. Самоучитель PHP 5 / Д. Н. Колесниченко. – СПб. : Наука и техника, 2004.

8. Кузнецов, М. В. PHP. Практика создания Web-сайтов / М. В. Кузнецов, И. В. Симдянов. – СПб. : БХВ-Петербург, 2009.

9. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL и JavaScript / Р. Никсон. – СПб. : Питер, 2011.

10. Стефанов, С. Описание книги JavaScript. Шаблоны / С. Стоянов. – Символ-Плюс, 2011.

11. Тиге, Дж. К. DHTML и CSS для Internet / Дж. К. Тиге. – М. : НТ Пресс, 2005.

Содержание

Учебное электронное издание комбинированного распространения

Учебное издание

Литвинов Дмитрий Александрович
Ковалев Алексей Викторович

ПРОГРАММИРОВАНИЕ В INTERNET

Курс лекций
по одноименной дисциплине для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной и заочной форм обучения

Электронный аналог печатного издания

Редактор *А. В. Власов*
Компьютерная верстка *М. В. Кравцова*

Подписано в печать 20.03.13.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 3,72. Уч.-изд. л. 4.
Изд. № 38.
http://www.gstu.by

Издатель и полиграфическое исполнение:
Издательский центр учреждения образования
«Гомельский государственный технический университет
имени П. О. Сухого».
ЛИ № 02330/0549424 от 08.04.2009 г.
246746, г. Гомель, пр. Октября, 48.