В. В. Самсонов
А. Л. Єрохін

Методи та засоби Інтернет-технологій

В. В. Самсонов, А. Л. Єрохін

Методи та засоби Інтернет-технологій

Рекомендовано Міністерством освіти і науки України як навчальний посібник для студентів вищих навчальних закладів, які навчаються за напрямом «Комп'ютерні науки»

Харків
Компанія СМІТ
2008

УДК 681.3.06(07)
ББК 32.973.26-018.2
С 17

Рекомендовано Міністерством освіти і науки України як навчальний посібник для студентів вищих навчальних закладів, які навчаються за напрямом «Комп'ютерні науки» (лист № 14/18.2-900 від 05.04.2006 р.)

Видано за рахунок державних коштів
Продаж заборонено

Рецензенти:

Жолткевич Г. М., зав. каф. теоретичної та прикладної інформатики, директор Центру комп'ютерних технологій Харківського національного університету ім. В. Н. Каразіна, д-р техн. наук, професор;

Шаронова Н. В., проф. каф. автоматизованих систем управління Національного технічного університету «Харківський політехнічний інститут», д-р техн. наук, професор;

Коряк С. Ф., доц. каф. програмного забезпечення ЕОМ Харківського національного університету радіоелектроніки, канд. техн. наук, доцент

Самсонов, В. В.

С 17 Методи та засоби Інтернет-технологій : навч. посібник / В. В. Самсонов, А. Л. Єрохін. — Х. : Компанія СМІТ, 2008. — 264 с.

ISBN 978-966-8530-19-5

Метою навчального посібника є формування у студентів систематичного і наукового підходу до методології вибору засобів комп'ютерних інформаційних технологій на фізичному, математичному та семантичному рівнях і програмної реалізації алгоритмів розв'язання складних сучасних інженерних задач. Ставиться за мету на практиці вивчити застосовувані методи сучасних інформаційних технологій з ухилом на Інтернет-технології (об'єктну модель документа DOM, технології HTML, DHTML, JavaScript, VBScript, CSS, основи XML).

Для студентів вищих навчальних закладів, які навчаються за напрямом «Комп'ютерні науки».

УДК 681.3.06(07)
ББК 32.973.26-018.2

ISBN 978-966-8530-19-5

ЗМІСТ

Передмова

На сьогодні наше суспільство переживає інформаційну революцію. Розвиток інформаційних технологій характеризується швидкою зміною концептуальних уявлень про роль методів, технічних засобів і людей, зайнятих у цій галузі.

Інформація стає найважливішим ресурсом суспільства і визначальним чинником в економічній, технічній і науковій сферах діяльності людини. Сьогодні не можна вирішити жодної значної проблеми без опрацювання значних обсягів інформації. Комп'ютерні інформаційні технології на сьогодні є мало не єдиним засобом підсилення інтелекту людини.

Метою вивчення методів і засобів комп'ютерних інформаційних технологій є формування в студентів систематичного і наукового підходу до методології вибору засобів комп'ютерних інформаційних технологій на фізичному, математичному і семантичному рівнях і програмної реалізації алгоритмів розв'язання складних сучасних інженерних задач.

За результатами вивчення дисципліни студенти повинні:

знати:
- класифікацію сучасного програмного забезпечення інформаційних технологій;
- сучасні методи й засоби автоматизованої обробки текстової, графічної, мовної інформації, зображень;
- склад засобів комп'ютерних інформаційних технологій;
- технології HTML, DHTML, сценаріїв у Web, CSS, XML;
- основні перспективи і напрямки розвитку інформаційних технологій;

уміти:
- створювати web-документи;
- створювати сценарії мовами JavaScript та VBScript;
- проектувати та програмно реалізовувати web-сайт;

мати:
- навички програмної реалізації основних операцій з DOM;
- навички роботи з інструментами розробки документів для сучасних комп'ютерних систем.

Розділ 1
ВСТУП ДО МЕТОДІВ ТА ЗАСОБІВ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1. Класифікація методів та засобів сучасних комп'ютерних інформаційних технологій

Одним із найважливіших розділів комп'ютерних наук є *комп'ютерні інформаційні технології (КІТ), які можна визначити як сукупність технічних, програмних засобів та прийомів роботи, за допомогою яких виконуються операції з обробки інформації в усіх сферах людської діяльності.* З визначення випливає суто прикладний характер КІТ.

Предметом КІТ є інформація та інформаційні потоки у різних формах подання. Для КІТ необхідні як апаратні, так і програмні засоби, за допомогою яких досягається мета та виробляється кінцева продукція. Що ж є кінцевою продукцією КІТ?

По-перше, це *документи* у різних формах свого існування; по-друге, це *інформаційні потоки у вигляді сигналів.*

Документи, які є предметом КІТ, у свою чергу, можна умовно поділити на дві великі групи.

До першої групи відносять документи, які містять інформацію у звичній для людини формі:

1) текстові документи (містять тексти, написані природними мовами);

2) графічні документи (відео, фотографії, рисунки, креслення тощо);

3) табличні документи (фінансові, довідкові, зведені документи тощо).

Для обробки документів з першої групи сучасні КІТ використовують спеціалізовані програмні засоби: текстові, графічні, табличні редактори чи процесори.

До другої групи документів відносять документи, які містять інформацію у кодованому вигляді, необхідному для управління апаратними та програмними засобами КІТ (наприклад, об'єктний код програми, web-документ тощо).

Розглянемо *загальну класифікацію методів та засобів КІТ*:

1. Методи та засоби вводу-виводу аналогових та цифрових сигналів (аналого-цифрові перетворювачі, алгоритми перетворювання, методи зменшення похибок перетворення).

2. Методи та засоби обробки текстової інформації (кодові таблиці, методи та алгоритми розпізнавання текстів).

3. Методи та засоби обробки графічної інформації (методи кодування графічної інформації, методи обробки графічної інформації, методи та засоби обробки відеозображень, методи кодування зображень, методи покращення якості зображень).

4. Методи та засоби обробки мовної інформації (методи формування мовних повідомлень, методи розпізнавання мовних повідомлень, засоби мовного інтерфейсу).

5. Методи та засоби комп'ютеризованих систем зв'язку (методи кодування інформації та засоби передачі каналами зв'язку).

6. Методи та засоби збереження інформації (методи кодування та запису інформації).

7. Методи та засоби реалізації інтерфейсів користувача (засоби текстових та графічних інтерфейсів).

Найбільш перспективним на теперішній час є новий напрямок КІТ, пов'язаний з інтеграцією усіх семи напрямків в один, який називається *Інтернет-технології*. Саме методам і засобам Інтернет-технологій присвячено цей навчальний посібник.

1.2. Огляд технологій обробки даних

Розглянемо технології обробки даних виходячи з програмного забезпечення. В табл. 1.1 наведені класи задач, які розв'язуються за допомогою КІТ, класи програмного забезпечення та приклади програмних продуктів. Відповідні розширення файлів наведені в Додатку 5.

Таблиця 1.1

Задачі КІТ та класи програмного забезпечення

Задачі КІТ	Клас програмного забезпечення	Приклади програмних продуктів
1	2	3
Обробка текстових документів	Текстові редактори (процесори)	MS Word, Лексикон, Adobe PageMaker
Обробка графічних документів	Графічні редактори	Adobe Photoshop, Corel Draw, 3D Studio

Продовження табл. 1.1

1	2	3
Обробка табличних документів	Табличні редактори (процесори)	MS Excel, Quattro Pro, Lotus 1-2-3
Обробка різнорідних даних	Статистичні системи	Statgraphics, Statistica
Розв'язування математичних задач	Математичні системи	MathCAD, MATLAB
Збір, зберігання та обробка масивів даних	Бази даних	MS Access, FoxPro, dBase, Oracle, Paradox
Робота в мережі Інтернет	Браузери	MS Internet Explorer, Netscape Navigator, Opera, Mozilla
Розробка web-документів	HTML- та JavaScript-редактори	Dreamweaver, Homesite, Scrib, Flash
Розпізнавання образів, мовний інтерфейс, переклади на інші мови, перевірка орфографії	Системи штучного інтелекту	Speaking Mouse, Magic Gooddy, ABBYY Lingvo, Рута Плай, ABBYY Fine Reader
Імітація та моделювання	Системи віртуальної реальності	Віртуальне навчальне середовище www.vdll.kture.kharkov.ua
Електронні порадні системи	Експертні системи	Prospector, Expertax
Юридичні й економічні довідки	Інформаційно-довідкові та пошукові системи	ЛІГА, Юрист
Автоматизація офісної діяльності	Електронні організатори	MS Outlook

Закінчення табл. 1.1

1	2	3
Економічні та бухгалтерські задачі	Бухгалтерські системи	1С Бухгалтерія, Парус, БЕСТ
Рекламні технології, представлення результатів діяльності	Засоби створення мультимедійних презентацій	MS PowerPoint
Проектування	Системи автоматизованого проектування	AutoCAD, ArchiCAD
Моделювання роботи електронних пристроїв	Системи моделювання	Workbench, MicroCap, Simulink
Нестандартні задачі — створення нового програмного забезпечення	Інтегровані середовища розробки програмного забезпечення	Turbo Pascal, Visual C, Visual Basic, Visual Cafe, Rational Rose

1.3. Еволюція архітектури обчислювальних систем

На рис. 1.1 наведено схему розвитку технологій обробки інформації з точки зору архітектури обчислювальної системи, починаючи з зародження обчислювальної техніки (монолітні обчислювальні системи) і закінчуючи найсучаснішою технологією («тонкий клієнт — товстий сервер»). Як видно з рисунку, обчислювальні системи розвиваються спірально. На теперішній час знову активно розвиваються комп'ютерні системи з розподіленими компонентами, а це автоматично означає новий виток розвитку мережі Інтернет. Виходячи з цього, міжнародний консорціум з розвитку Інтернет-технологій WWW Consortium (або W3C) ініціював розробку стандарту для представлення документів у комп'ютерних системах. Цей стандарт отримав назву DOM — стандарт об'єктної моделі документа.

1.4. Основні поняття стандарту DOM

Розглянемо основні поняття об'єктної моделі документа (англ. DOM — *Document Object Model*).

DOM — це сукупність об'єктів, якою може бути поданий документ. Саме виходячи з об'єктної моделі документ генерується спеціальним програмним забезпеченням після передачі його каналами зв'язку на клієнтську сторону.

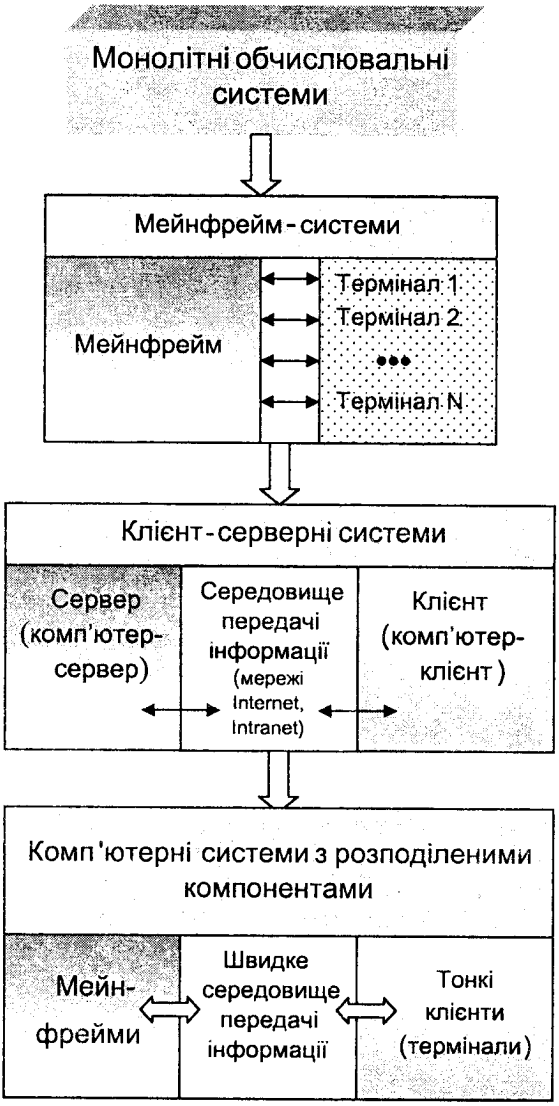


Рис. 1.1. Еволюція архітектури обчислювальних систем

Наприклад, після завантаження web-документа він генерується браузером (спеціальною програмою клієнтської сторони). При генерації документа браузер повинен забезпечувати ту чи іншу версію DOM.

На сьогодні DOM — прикладний інтерфейс програмування для HTML- і XML-документів.

Стандарт визначає логічну структуру документа і засоби доступу і маніпулювання документами. DOM розроблений з метою використання з будь-якими мовами програмування. На сьогоднішній день найбільш популярні реалізації стандарту OMG (*Object M Group*) та IDL (*Interface Definition Language*). Стандарт DOM специфікує інтерфейси, що можуть використовуватися для управління XML- або HTML-документами.

DOM подає документ як ієрархію об'єктів Node (вузли), що реалізують інші, більш спеціалізовані інтерфейси (*див.* табл. 1.2). Деякі типи вузлів можуть мати успадковані вузли різноманітних типів, інші ж Node є листками дерева і від них не може нічого успадковуватися.

Таблиця 1.2

Типи вузлів у DOM

Вузли (Node)	Успадковані вузли
1	2
Document	Element (максимально — 1), Comment, ProcessingInstruction, DocumentType
DocumentFragment	Comment, ProcessingInstruction, DocumentType, Text, CDataSection, EntityReference
DocumentType	Не має
Element	Element, Comment, ProcessingInstruction, Text, CDataSection, EntityReference
EntityReference	Element, Comment, ProcessingInstruction, Text, CDataSection, EntityReference
Attr	Text, EntityReference

Закінчення табл. 1.2

1	2
ProcessingInstruction	Не має
Comment	»
Text	»
CDATASection	»
Notation	»
Entity	Element, Comment, ProcessingInstruction, Text, CDataSection, EntityReference

Більшість API визначені не стільки як класи, скільки як інтерфейси.

Це означає, що реальне втілення інтерфейсу потребує розширення методів визначення імен специфікації операцій. Звідси випливає, що звичайні конструкції мов ООП не можуть бути використані для створення DOM-об'єктів. Відповідним рішенням для цього в ООП є визначення фабричних методів (*factory*-методів), що створюють сутності об'єктів.

Наприклад, у інтерфейсі *document* створюється деякий документ *X* за допомогою методу *create( )* інтерфейсу *document*, де *X* — це ім'я об'єкта.

Ядро DOM API розроблено з метою забезпечення сумісності широкого діапазону мов.

Більш докладно характеристики деяких інтерфейсів розглянемо в наступних розділах.

1.5. Доступ до об'єктної моделі документа в сучасних комп'ютерних системах

Розглянемо класифікацію способів доступу до об'єктної моделі:

1. За допомогою автономних програм.

2. За допомогою сценарних програм. *Сценарій* — це програмний код, який міститься в документі і виконується інтер-

претатором. Іноді сценарії називають *скриптами*. При цьому існує два способи виконання сценарних програм:

2.1. Сценарії, що знаходяться всередині документа.

2.2. Сценарії, що зберігаються у вигляді віддалених ресурсів (зовнішні сценарії).

3. За допомогою впроваджених програм-об'єктів.

3.1. За допомогою впроваджених аплетів мовою Java.

3.2. За допомогою впроваджених елементів керування (Active-об'єктів).

1.6. Контрольні запитання

1. Надайте визначення КІТ.
2. Які задачі можна вирішувати за допомогою КІТ?
3. Назвіть етапи еволюції архітектури обчислювальних систем.
4. Що таке DOM?
5. Назвіть основні елементи DOM.
6. Які існують типи вузлів у DOM?
7. Які існують засоби доступу до об'єктної моделі документа?
8. Що таке сценарій?

Розділ 2
ОСНОВИ ІНТЕРНЕТ-ТЕХНОЛОГІЙ

2.1. Основні поняття мережних технологій

Мережі комп'ютерів поділяються на глобальні та локальні (корпоративні). Локальні мережі припускають з'єднання комп'ютерів за різноманітними топологіями. Інформація, що передається між комп'ютерами, упаковується в пакет.

Розглянемо основні терміни сучасних мережних технологій.

1. Глобальна мережа Internet (Інтернет). Структура мережі Інтернет зображена на рис. 2.1.

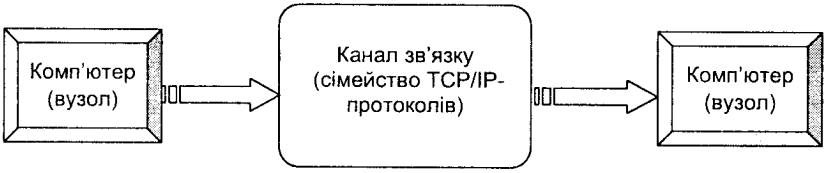


Рис. 2.1. Структура мережі Інтернет

Вузол — це комп'ютер, що знаходиться в мережі.

2. Intranet (Інтранет) — різновид локальної мережі, у якій інформація передається за протоколами TCP/IP. Це просто локальна мережа, що працює за Інтернет-стандартом.

3. Extranet (Екстранет). Сутність цієї мережі продемонстрована на рис. 2.2.

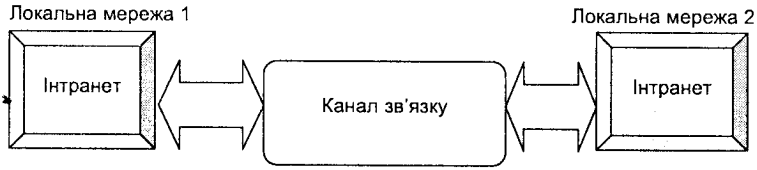


Рис. 2.2. Екстранет

Фізично комп'ютерні мережі можуть з'єднуватися за кількома топологіями.

Коротко розглянемо найпоширеніші топології мереж.

1. Топологія «Кільце» (рис. 2.3).

Для обміну пакетами в такій мережі використовується метод маркера. Пакет надходить більш надійно. Мережа вважається зайнятою, якщо в ній перебуває пакет із маркером.

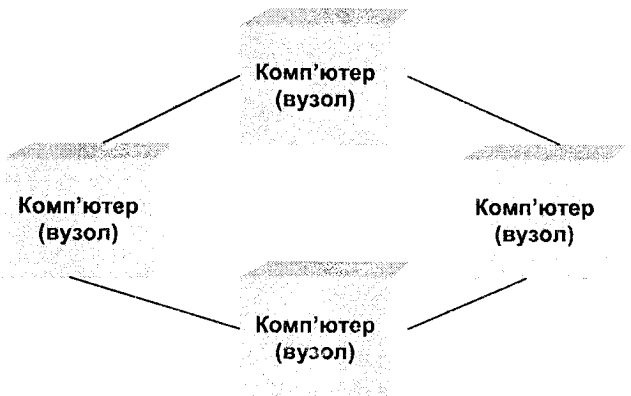


Рис. 2.3. Топологія «Кільце»

2. Топологія «Шина» (рис. 2.4).

Має такі недоліки: швидкість нижча, ніж в інших видах з'єднань; виникає необхідність у додаткових пристроях (термінаторах).

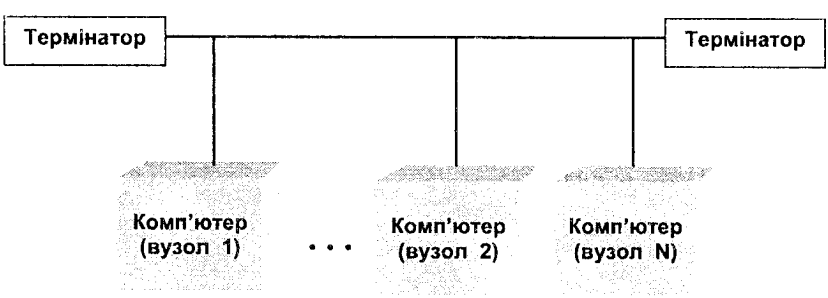


Рис. 2.4. Топологія «Шина»

3. Топологія «Зірка» (рис. 2.5). На теперішній час є найпоширенішою топологією в локальних мережах.

Переваги: висока надійність.

Має суттєвий недолік: висока вартість.

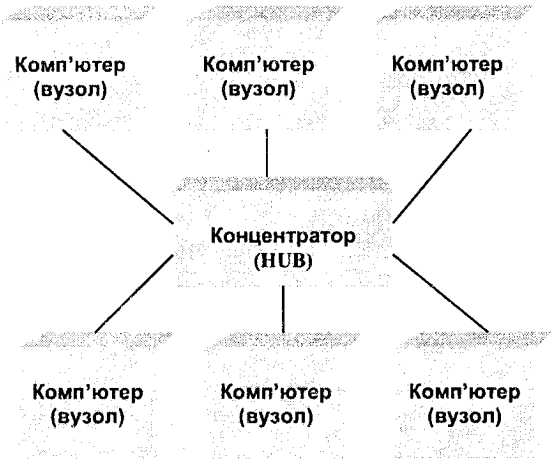


Рис. 2.5. Топологія «Зірка»

У мережах може бути виділений і невиділений файл-сервер.

Мережі без виділення сервера називаються одноранговими мережами. Якщо виділено головний комп'ютер, то він називається *хостом* (host) або *сервером*.

2.2. Основні поняття мережі Інтернет

Інтернет — це глобальна комп'ютерна мережа, побудована на базі протоколу TCP/IP та містить у собі різнорідні регіональні, корпоративні та локальні мережі (*див.* Додаток 1).

У 1995 році організацією Federal Networking Council USA, що покликана сприяти розвитку мережі, було дане до цього часу актуальне визначення Інтернет: «Інтернет — це глобальна інформаційна система, що складається з логічно взаємозалежних частин з унікальним адресним простором, заснованим на протоколі IP або його подальших розширень, здатна підтримувати зв'язок з використанням протоколів TCP/IP, їхніх наступних

розширень або інших IP-сумісних протоколів, а також яка забезпечує, використовує або робить доступним, публічно або приватним чином, комунікаційний сервіс та інфраструктуру високого рівня».

Підтримка роботи мережі здійснюється територіально розподіленими вузлами — Інтернет-серверами (*хостами*).

Якщо попросити пересічну людину знайти відмінності між словом «Інтернет» та абревіатурою WWW, то, скоріш за все, відповіддю буде: «Відмінностей немає». Але це не вірно. WWW — це абревіатура від «World Wide Web» («всесвітня павутина»), що означає один із найбільш популярних сервісів мережі, орієнтований на надання універсального доступу до документів з використанням технології гіпертексту.

Гіпертекстові документи пов'язані між собою за допомогою посилань, які однозначно адресують потрібний документ.

Переглядати web-ресурси та здійснювати навігацію між документами дозволяють спеціальні програми — браузери (або броузери). Браузери відрізняються за ступенем підтримки стандартів, додаткових можливостей, способом відображення сторінок та багатьма іншими параметрами. На теперішній час лідером за популярністю серед браузерів є Microsoft Internet Explorer, якому поступаються Netscape Navigator, Opera та Mozilla.

Основним вмістом Інтернету є так званий контент у вигляді різноманітних Інтернет-проектів, які розміщуються на *сайтах*.

Сайт — це сукупність інформаційних ресурсів, які поєднані один з одним за тематикою чи предметною галуззю.

Що ж входить до загального поняття «*Інтернет-проект*» (або *web-проект*)?

По-перше, це сайти комерційних організацій, які, як самі ці організації, спрямовані на отримання вигоди. Причина створення Інтернет-магазинів і сайтів, що надають усілякі платні сервіси, — отримання прямої вигоди, вираженої в збільшенні банківського рахунку. Як платні сервіси можуть виступати контекстний показ реклами на пошукових серверах, продаж банеропоказів, надання платного онлайн-доступу до деякої інформації і таке інше. Більшість сайтів комерційних організацій лише непрямим способом збільшують прибутки, вирішуючи задачу прямої реклами, надання інформації про компанію, її продукції чи послуги, просування торгових марок, PR (зв'язки з громадськістю) тощо. Багато компаній надають за допомогою

свого сайта технічну підтримку своїм споживачам або організують онлайнову довідкову службу (рис. 2.6)

По-друге, не комерційні сайти, які найчастіше є інформаційними. Вони надають інформацію про діяльність якоїсь некомерційної організації, партії, фонду або ж просто є тематичними ресурсами, присвяченими будь-якій сфері людської діяльності або предметній галузі. Окрім інформаційної функції, такі сайти нерідко виконують функції моніторингу цієї галузі, дозволяючи проводити різноманітні опитування, анкетування, збирати статистику активності відвідувачів.

Інтернет-сайт може брати участь в автоматизації окремих частин інформаційних потоків підприємства. Наприклад, робота з філіалами може будуватися не на телефонних дзвінках та листах, а на централізованій базі даних (БД) з web-інтерфейсом. Аналогічно за допомогою онлайнової системи можна побудувати взаємовідносини з постійними клієнтами.

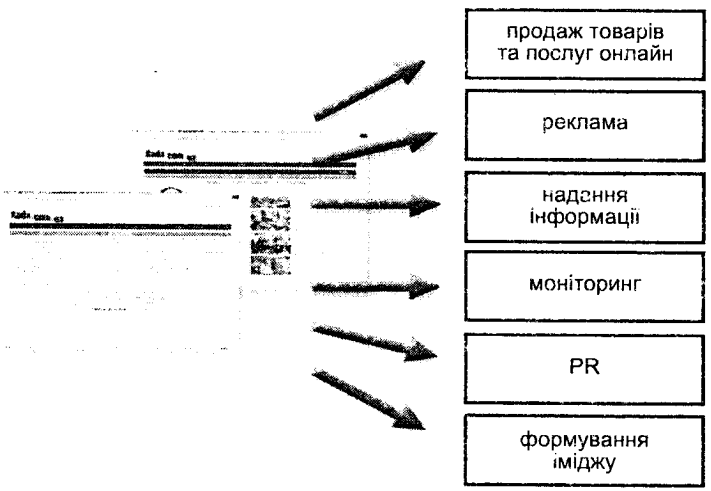


Рис. 2.6. Типові завдання комерційного Інтернет-проекту

Надання послуг Інтернет побудовано за схемою «*клієнт-сервер*» (*див.* рис. 1.1). Надання послуг здійснюється спільною роботою двох процесів: на комп'ютері користувача і на комп'ютері-сервері. Процес на комп'ютері користувача називається *клієнтом*, а на комп'ютері-сервері — *сервером*.

Клієнт і сервер є, по суті, частинами однієї програми, взаємодіючи по віртуальному зв'язку в мережі. Сервер за вказівками клієнта виконує відповідні дії, наприклад, пересилає клієнту файл. Для надання послуги необхідна наявність двох цих модулів — клієнта і сервера, та їх одночасна погоджена робота.

Взаємодія клієнта і сервера описується відповідними стандартними *протоколами*, тому програмне забезпечення *клієнта* і *сервера* може бути випущене різними виробниками і працювати на різнорідних комп'ютерах. Тому ж існує невелика проблема нестандартності інтерфейсу клієнта безпосередньо вже з користувачем. Ця взаємодія може мати зовсім різні форми: інтерактивну, командну тощо. Системи команд можуть мати відмінності, але від цього самі можливості не змінюються, оскільки клієнт і сервер завжди взаємодіють однаково — відповідно до протоколу.

Далі розглянемо, за якими протоколами працює сучасний Інтернет та які сервіси він надає користувачам.

2.3. Стандарти та сервіси мережі Інтернет

Коли йдеться про стандарти роботи в мережах, то зазвичай розглядаються протоколи (*див.* Додаток 1).

Сучасні мережі побудовані за багаторівневим принципом. Щоб організувати зв'язок двох комп'ютерів, потрібно спочатку створити набір правил їхньої взаємодії, визначити мову їхнього спілкування, тобто визначити, що означають сигнали, які посилаються ними, і т. д. Ці правила і визначення називаються *протоколом*.

Для роботи мереж необхідно використати багато різних протоколів: наприклад, керуючих фізичним зв'язком, установленням зв'язку по мережі, доступом до різних ресурсів і т. д. Ось чому протоколи побудовані за багаторівневою структурою.

Розглянемо рівні протоколів TCP/IP.

Ієрархія протоколів TCP/IP відповідає стандарту OSI.

Що ж таке *модель OSI*? Еталонна модель OSI, яку іноді називають *стеком OSI*, є семирівневою мережною ієрархією. Вона розроблена Міжнародною організацією зі стандартизації *ISO* (англ. — *International Standardization Organization*). Ця модель містить у собі по суті дві різні моделі:

- горизонтальну модель на базі протоколів, що забезпечують механізм взаємодії програм і процесів на різних вузлах;

- вертикальну модель на основі послуг, що забезпечуються сусідніми рівнями один одному на одному вузлі.

У горизонтальній моделі двом програмам потрібен загальний протокол для обміну даними. У вертикальній — сусідні рівні обмінюються даними з використанням інтерфейсів API (англ. — *Application Programming Interface* — інтерфейс прикладного програмування).

На рис. 2.7 показана загальноприйнята семирівнева структура згідно з ISO. Ця модель відома як *«еталонна модель ISO OSI»*. Вона дозволяє складати мережні системи з продуктів-модулів програмного забезпечення, які розроблені різними виробниками.

Взаємодія рівнів у цій моделі — субординарна. Кожен рівень може реально взаємодіяти тільки із сусідніми рівнями (верхнім і нижнім), віртуально — тільки з аналогічним рівнем на іншому кінці лінії.

Під *реальною* взаємодією мають на увазі безпосередню взаємодію, безпосередню передачу інформації, наприклад, пересилання даних в оперативній пам'яті з області, відведеної одній програмі, в область іншої програми. При безпосередній передачі дані залишаються незмінними весь час.

Під *віртуальною* взаємодією мають на увазі опосередковану взаємодію і передачу даних. Дані в процесі передачі можуть бути заздалегідь обговореним способом видозмінюватися.

Модель ISO OSI вимагає жорстокої стандартизації вертикальних міжрівневих взаємодій. Така стандартизація гарантує сумісність продуктів, що працюють за стандартом деякого рівня, із продуктами, що працюють за стандартами сусідніх рівнів, навіть у тому випадку, якщо вони випущені різними виробниками. Кількість рівнів може здаватися надлишковою, але така розбивка необхідна для досить чіткого поділу необхідних функцій, для запобігання зайвої складності та створення структури, що може відповідати завданню конкретного користувача, залишаючись у рамках стандарту.

Дамо короткий огляд рівнів.

Рівень 0 — пов'язаний з фізичним середовищем — передавачем сигналу і насправді не включається в цю схему, але дуже корисний для розуміння моделі в цілому. Цей рівень представляє посередників, що з'єднують кінцеві пристрої: кабелі, радіолінії тощо. Кабелів існує безліч різних видів і типів: екрановані

і неекрановані кручені пари, коаксіальні, на основі оптичних волокон і т. д. Оскільки цей рівень не включений до схеми, то він нічого і не описує, тільки вказує на середовище.

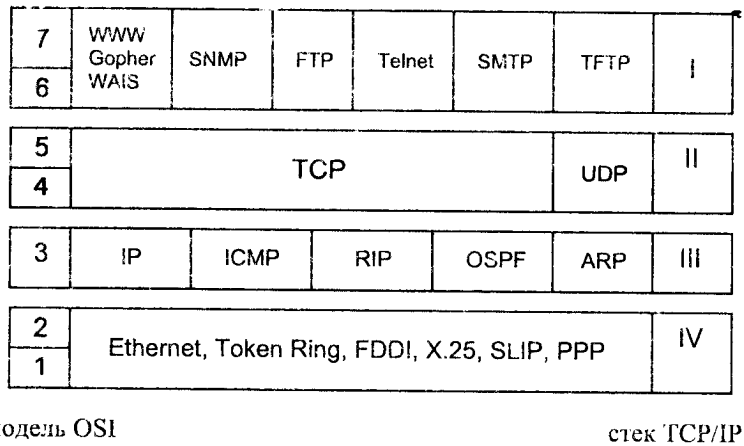


Рис. 2.7. Семирівнева модель OSI

Рівень 1 — фізичний. Містить фізичні аспекти передачі двійкової інформації з лінії зв'язку. Детально описує, наприклад, напруги, частоти, природу передавального середовища. Цьому рівню ставиться в обов'язок підтримувати зв'язок і прийом-передачу бітового потоку. Безпомилковість є бажаною, але не обов'язковою.

Рівень 2 — канальний. Відповідає за зв'язування даних. Забезпечує безпомилкову передачу блоків даних — *кадрів* (*frame*) через рівень 1. Цей рівень має визначати початок і кінець *кадру* в бітовому потоці, формувати з даних, переданих фізичним рівнем, кадри послідовності, включати процедуру перевірки наявності помилок і їхнього виправлення. Цей рівень (і тільки він) оперує такими елементами, як бітові послідовності, методи кодування, маркери. Він несе відповідальність за правильну передачу даних (*пакетів*) на ділянках між безпосередньо зв'язаними елементами мережі. Забезпечує керування доступом до середовища передачі.

Внаслідок його складності канальний рівень розділяють на два підрівні: MAC (*Medium Access Control*) — рівень керу-

вання доступом до середовища і LLC (*Logical Link Control*) — рівень керування логічним зв'язком (каналом). Рівень MAC керує доступом до мережі (з передачею маркера в мережах *Token Ring* чи розпізнаванням конфліктів (зіткнень передач) у мережах *Ethernet*) і власне мережею. Рівень LLC діє над рівнем MAC, саме він посилає й одержує повідомлення з даними.

Рівень 3 — мережний. Цей рівень користується можливостями, наданими йому рівнем 2, для забезпечення зв'язку двох будь-яких вузлів у мережі. Цей рівень здійснює проводку повідомлень мережею або ж множиною спільно працюючих мереж. Це вимагає *маршрутизації*, тобто визначення шляху, яким мають пересилатися дані.

Маршрутизація здійснюється на цьому ж рівні. Виконує обробку адреси, а також *демультиплексування*. Основною функцією програмного забезпечення на цьому рівні є вибірка інформації з джерела, перетворення її в *пакети* і правильна передача в точку призначення.

Існують два принципово різних способи роботи мережного рівня.

Перший — це метод *віртуальних каналів*. Він полягає в тому, що канал зв'язку встановлюється при виклику (початку сеансу — сесії зв'язку), по ньому передається інформація, і з закінченням передачі канал закривається (знищується). Передача пакетів відбувається зі збереженням вихідної послідовності, навіть якщо пакети пересилаються різними фізичними маршрутами, тобто віртуальний канал динамічно перенаправляється. При цьому пакети даних не містять адресу пункту призначення, тому що вона визначається під час установлення зв'язку.

Другий — метод *дейтаграм*. Дейтаграми є незалежними, тобто вони містять усю необхідну для їхнього пересилання інформацію. У той час, як перший метод надає наступному рівню (рівню 4) надійний канал передачі даних, вільний від перекручувань (помилок), і правильно доставляє пакети в пункт призначення, другий метод вимагає від наступного рівня роботи над помилками і перевірки доставки потрібному адресату.

Рівень 4 — транспортний. Регламентує пересилання пакетів повідомлень між процесами, виконуваними на комп'ютерах мережі. Завершує організацію передачі даних: контролює потік даних, що проходить маршрутом, визначеним на третьому рівні: правильність передачі блоків даних, правильність доставки

в потрібний пункт призначення, їхню комплектність, цілісність, порядок проходження. Збирає інформацію з блоків у її первісний вигляд. Або ж оперує з дейтаграмами, тобто очікує відгуку-підтвердження прийому з пункту призначення, перевіряє правильність доставки й адресації, повторює надсилання дейтаграми, якщо не прийшов відгук. У рамках транспортного протоколу передбачено п'ять класів якості транспортування і відповідні процедури керування. Цей же рівень має містити надійну схему адресації для забезпечення зв'язку через множину мереж і шлюзів.

Транспортний рівень приховує від усіх вищих рівнів будь-які деталі та проблеми передачі даних, забезпечує стандартну взаємодію попереднього рівня з прийомом-передачею інформації незалежно від конкретної технічної реалізації цієї передачі.

Рівень 5 — сеансовий. Координує взаємодію користувачів: установлює зв'язок, відновлює аварійно завершені сеанси. Цей же рівень відповідає за картографію мережі — він перетворює регіональні (доменні) комп'ютерні імена в числові адреси і навпаки. Він координує не комп'ютери і пристрої, а процеси в мережі, підтримує їхню взаємодію — керує сеансами зв'язку між процесами прикладного рівня.

Рівень 6 — рівень представлення даних. Цей рівень має справу із синтаксисом і семантикою переданої інформації, тобто тут установлюється взаєморозуміння двох сполучених комп'ютерів щодо того, як вони представляють і розуміють після одержання передану інформацію. Тут вирішуються такі задачі, як перекодування текстової інформації і зображень, стискання і розпакування інформації, підтримка мережних файлових систем, абстрактних структур даних тощо.

Рівень 7 — прикладний. Забезпечує інтерфейс між користувачем і мережею, робить доступними різні послуги. На цьому рівні реалізується, принаймні, п'ять прикладних служб: передача файлів, віддалений термінальний доступ, електронна передача повідомлень, служба довідника й управління мережею. У конкретній реалізації визначається користувачем (програмістом) відповідно до його задач і можливостей. Цей рівень має справу з багатьма різними протоколами термінального типу, яких існує більш ста.

Переважна більшість сучасних мереж з історичних причин лише приблизно відповідають еталонній моделі ISO OSI.

Розглянемо деякі з зазначених рівнів у їхній проекції на Інтернет докладніше. Слід пам'ятати, що проекція є трохи спотвореною внаслідок неповної відповідності технології Інтернет стандартові еталонної моделі.

1. *Пересилання бітів* відбувається на фізичному рівні схеми ISO OSI. На жаль, тут будь-яка спроба короткого і доступного опису приречена на провал. Потрібне введення величезної кількості спеціальних термінів, понять, описів процесів на фізичному рівні і т. д. Окрім того, існує настільки велике розмаїття приймальних і передавальних середовищ, що їх майже неможливо оглянути, але для розуміння роботи мереж цього і не потрібно. Для простоти вважатимемо, що просто є труба, по якій від краю до краю перекачуються біти. Саме біти, без усякого розподілу на які-небудь групи (байти, декади і т. п.).

2. Пересилання даних. Про організацію блокової, символьної передачі, забезпечення надійності пересилання йтиметься при розгляді інших рівнів моделі ISO OSI. Функції канального рівня в мережі Інтернет розподілені іншими рівнями, але не вище транспортного. У цьому сенсі Інтернет не зовсім відповідає стандарту ISO. Тут канальний рівень займається тільки розбивкою бітового потоку на символи і кадри та передачею отриманих даних на наступний рівень. За надійність передачі він не відповідає.

3. Мережі комутації пакетів. Далі мова йтиме про Інтернет саме як про мережу, а не лінії зв'язку і сукупність прийомо-передавачів. Здавалося б, Інтернет цілком аналогічний телефонній мережі, і модель телефонної мережі досить адекватно відбиває її структуру і роботу. Справді, обидві вони електронні, обидві дозволяють установлювати зв'язок і передавати інформацію. Інтернет теж складається, у першу чергу, з виділених телефонних ліній. Але, на жаль, ця картина невірна і призводить до багатьох оман щодо роботи Інтернет та непорозумінь. Телефонна мережа — це так звана мережа з комутацією ліній, тобто коли ви робите виклик, установлюється зв'язок і на весь час сеансу зв'язку є фізичне з'єднання з абонентом. При цьому абоненту виділяється частина мережі, яка для інших абонентів вже не доступна. Це призводить до нераціонального використання ресурсів високої вартості — ліній мережі. Інтернет же є мережею з комутацією пакетів, що принципово відрізняється від мережі з комутацією каналів.

Інтернету більш відповідає модель *пошти*, звичайної державної поштової служби. Пошта є мережею *пакетного* зв'язку. Немає ніякої виділеної частини цієї мережі. Послання перемішується з посланнями інших користувачів, кидається в контейнер, пересилається в інше поштове відділення, де знову сортується. Хоча технології значно відрізняються, пошта є влучним і наочним прикладом мережі з комутацією *пакетів*. Модель пошти точно відбиває суть роботи і структури мережі Інтернет.

4. Протокол Internet (IP). По проводу можна переслати біти тільки з одного його кінця в інший. Інтернетом же коректно передаються дані до різних вузлів, які розкидані по всьому світу. За це відповідає мережний (міжмережний) рівень в еталонній моделі ISO OSI. Про нього і йтиметься далі.

Різні частини Інтернет — складові мережі — з'єднуються між собою за допомогою комп'ютерів, що називаються вузлами.

Мережі ці можуть бути *Ethernet, Token Ring*, мережі на телефонних лініях, *пакетні* радіомережі і тощо. Виділені лінії і локальні мережі є аналогами залізниць, літаків, пошти і поштових відділень, листонош. За їх допомогою пошта рухається з місця на місце.

Вузли — аналоги поштових відділень, де приймається рішення, як переміщувати дані (*пакети*) мережею, так само, як поштовий вузол намічає подальший шлях поштового конверта. На кожній поштовій підстанції визначається наступна підстанція, куди буде далі спрямована кореспонденція, тобто намічається подальший шлях (*маршрут*) — цей процес називається *маршрутизацією*.

Для здійснення маршрутизації кожна підстанція має таблицю, де адресі пункту призначення (чи індексу) відповідає позначення поштової підстанції, куди слід пересилати далі цей конверт. Їхні мережні аналоги називаються *таблицями маршрутизації*, які розсилаються поштовим підстанціям централізовано відповідним поштовим підрозділом. Час від часу розсилаються розпорядження зі зміною і доповненням цих таблиць.

У мережі Інтернет складання і модифікація таблиць маршрутизації (цей процес теж є частиною маршрутизації і називається так само) визначаються відповідними правилами — протоколами ICMP (*Internet Control Message Protocol*), RIP (*Routing Internet Protocol*) і OSPF (*Open Shortest Path First*). Вузли, що займаються маршрутизацією, називаються *маршрутизаторами*.

А звідки мережа знає, куди призначений пакет даних? Від користувача. Якщо користувач надсилає листа і хоче, щоб лист досяг місця призначення, він не може просто кинути аркушик паперу до скриньки. Необхідно покласти його в стандартний конверт і написати на ньому адресу одержувача в стандартній формі. Тільки тоді пошта зможе правильно обробити лист і доставити його за призначенням.

Аналогічно і в Інтернет є набір правил щодо роботи з пакетами — *протоколи*.

Протокол *IP* (англ. — *Internet Protocol*) відповідає за роботу щодо підтвердження того, що вузли розуміють, що слід робити з вашими даними шляхом їхнього подальшого проходження. Згідно з нашою аналогією, протокол IP містить правила обробки поштового конверта. У початок кожного поміщується заголовок, що несе інформацію про адресата, мережу. Щоб визначити, куди і як доставити пакет даних, цієї інформації досить.

Через деякі причини (особливо практичні, через обмеження обладнання) інформація, що пересилається мережами IP, поділяється на частини (по границях байтів), що розкладаються в окремі *пакети*. Довжина інформації всередині *пакета* звичайно складає від 1 до 1500 байт. Це захищає мережу від монополізування яким-небудь користувачем і надає всім приблизно рівні права. До того ж, якщо мережа недостатньо швидка, то чим більше користувачів її одночасно використовує, тим повільніше вона спілкуватиметься з кожним із них.

Протокол IP є *дейтаграмним протоколом*, тобто IP-пакет є *дейтаграмою*. Це зовсім не укладається в модель *ISO OSI*, у рамках якої вже мережний рівень здатний працювати за методом *віртуальних каналів*.

Одна з переваг Інтернет полягає в тому, що протоколу IP самого по собі вже цілком достатньо для роботи. Але це зовсім незручно. Як тільки дані вміщуються в оболонку IP, мережа має всю необхідну інформацію для передачі з вихідного комп'ютера одержувачу. Робота вручну з протоколом IP нагадує часи доперсональної комп'ютерної ери. Про зручність користувача і таке поняття, як «юзабіліті» ніхто і не збирався думати, тому що машинний час коштував набагато дорожче людського. Але зараз в аскетизмі потреби вже немає. Тому слід було побудувати на основі послуг, наданих IP, більш зручну систему.

Для цього спочатку розберемося з важливими проблемами, що мають місце при пересиланні інформації:

- більша частина інформації, що пересилається, за довжиною складає більше 1500 символів;
- можлива втрата пакета взагалі;
- пакети можуть надходити в послідовності, що відмінна від початкової.

Таким чином, наступний рівень Інтернет має забезпечити спосіб пересилання великих масивів інформації і подбати про «перекручування», що можуть виникати з вини мережі.

Розглянемо протокол управління передачею (TCP) і протокол користувальницьких дейтаграм (UDP).

TCP (англ. — *Transmission Control Protocol*) — це протокол, що тісно пов'язаний з IP, але використовується на більш високому рівні — транспортному рівні еталонної моделі ISO OSI. Нерідко ці протоколи через їхній тісний зв'язок іменують разом як TCP/IP.

Термін «TCP/IP» зазвичай означає все, що пов'язане з протоколами TCP і IP. Він охоплює ціле сімейство протоколів, прикладні програми і навіть саму мережу. До складу сімейства входять протоколи TCP, UDP, ICMP, telnet, FTP і багато інших.

Сам протокол TCP займається проблемою пересилання великих обсягів інформації, ґрунтуючись на можливостях протоколу IP. Як це робиться? Цілком реально можна розглянути таку ситуацію. Як можна переслати книгу поштою, якщо та приймає тільки листи і нічого більш? Дуже просто: розібрати її на сторінки і надіслати сторінки окремими конвертами. Одержувач, керуючись номерами сторінок, легко зможе книгу відновити. Цим же простим і природним методом і користується TCP.

TCP ділить інформацію, яку треба переслати, на кілька частин. Нумерує кожну частину, щоб пізніше відновити порядок. Щоб пересилати цю нумерацію разом з даними, він ніби обгортає кожен фрагмент інформації своєю обкладинкою — конвертом, що містить відповідну інформацію. Це і є TCP-конверт.

Далі він поміщається в окремий IP-конверт, і утворюється IP-пакет, з яким мережа уже вміє поводитися правильно.

Одержувач (*TCP-модуль* (процес)) розпаковує одержані IP-конверти і бачить TCP-конверти, розпаковує їх і вміщує дані з послідовності частин у відповідне місце. Якщо чогось не вистачає, то одержувач вимагає переслати цей фрагмент знову.

Зрештою інформація збирається в потрібному порядку і цілком відновлюється. Тільки після цього масив пересилається до користувача (на диск, на екран, на друк).

Це дещо спрощений погляд на TCP. У реальності пакети не тільки втрачаються, але й можуть спотворюватися під час передачі через наявність перешкод на лініях зв'язку. TCP вирішує і цю проблему. Для цього він використовує систему *кодів, що виправляють помилки*. Існує окрема дисципліна, яка займається таким кодуванням. Найпростішим прикладом такого кодування є код з додаванням до кожного пакета контрольної суми (і до кожного байта — біта перевірки на парність). При вміщенні в TCP-конверт обчислюється контрольна сума та записується в TCP-заголовок. Якщо під час прийому заново обчислена сума не збігається з тією, що зазначена на конверті, це означає, що на шляху передачі мали місце перекручування (спотворення), і необхідно переслати цей пакет знову, що і робиться.

Слід зауважити: модуль TCP розбиває потік байтів на пакети, не зберігаючи при цьому границь між записами. Тобто якщо один прикладний процес робить три записи в порт, то зовсім не обов'язково, що інший прикладний процес на іншому кінці віртуального каналу одержить зі свого порту саме три записи, причому саме таких (за розбивкою), що були передані з іншого кінця. Вся інформація буде отримана цілком і зі збереженням порядку передачі, але вона може вже бути розбита інакше і на іншу кількість частин. Не існує залежності між кількістю і розміром записуваних повідомлень з однієї сторони та кількістю і розміром повідомлень, що зчитуються, з іншої сторони. TCP вимагає, щоб усі відправлені дані були підтверджені стороною, що їх прийняла. Він використовує очікування (тайм-аути) і повторні передачі для забезпечення надійної доставки. Відправнику дозволяється передавати деяку кількість даних, не чекаючи підтвердження прийому раніше відправлених даних. Таким чином, між відправленими і підтвердженими даними існує вікно уже відправлених, але ще не підтверджених даних.

Кількість байтів, яку можна передавати без підтвердження, називається *розміром вікна*. Як правило, розмір вікна встановлюється в стартових файлах мережного програмного забезпечення. Дані можуть одночасно передаватися в обох напрямках, а підтвердження для даних, що йдуть в одному напрямку, можуть передаватися разом з даними, що йдуть у протилежному

напрямку. Приймачі на обох сторонах віртуального каналу виконують керування потоком переданих даних для того, щоб не допускати переповнення буферів.

Таким чином, протокол TCP забезпечує гарантовану доставку зі встановленням логічного з'єднання у вигляді байтових потоків. Він звільняє прикладні процеси від необхідності використовувати очікування і повторні передачі для забезпечення надійності. Найбільш типовими прикладними процесами, що використовують TCP, є *ftp* і *telnet*. Крім того, TCP використовується системою *X-Window* (стандартний багатовіконний графічний інтерфейс користувача), «*r*-командами».

Великі можливості TCP вимагають великої продуктивності процесора і великої пропускної здатності мережі. Коли прикладний процес починає використовувати TCP, то починають спілкуватися модуль TCP на машині користувача і модуль на машині сервера. Ці два кінцевих модулі TCP підтримують інформацію про стан з'єднання — віртуального каналу. Цей віртуальний канал споживає ресурси обох кінцевих модулів TCP. Канал цей, як уже зазначалося, є дуплексним. Один прикладний процес записує дані в TCP-порт, звідки вони модулями відповідних рівнів по ланцюжку передаються мережею і видаються в TCP-порт на іншому кінці каналу. Інший прикладний процес читає їх зі свого TCP-порту, емулює (створює видимість) виділену лінію зв'язку двох користувачів та гарантує незмінність переданої інформації.

Є й інший стандартний протокол транспортного рівня, що не обтяжений такими значними витратами. Цей протокол називається *UDP* (англ. *User Datagram Protocol* — протокол користувальницьких дейтаграм). Він використовується замість TCP. Тут дані містяться не в TCP-, а в UDP-конверті, що також поміщується в IP-конверт. Цей протокол реалізує дейтаграмний спосіб передачі даних.

Дейтаграма — це пакет, переданий через мережу незалежно від інших пакетів без установлення логічного з'єднання і підтвердження прийому. Дейтаграма — цілком самостійний пакет, оскільки сама містить усю необхідну інформацію для її передачі, яка відбувається без будь-якого попередження і підготовки. Дейтаграми самі по собі не містять засобів виявлення і виправлення помилок передачі, тому при передачі даних з їх допомогою слід вживати заходів щодо забезпечення надійності

пересилання інформації. Методи організації надійності можуть бути найрізноманітнішими, але зазвичай використовується метод підтвердження прийому надсиланням відлуння-відгуку при одержанні кожного пакета з дейтаграмою.

UDP простіше за TCP, оскільки він не піклується про можливу втрату даних, пакетів, про збереження правильного порядку даних і т. д. UDP використовується для клієнтів, що надсилають тільки короткі повідомлення і можуть просто заново надіслати повідомлення, якщо відгук підтвердження не прийде досить швидко.

Припустимо, що програміст створює програму, яка переглядає базу даних з телефонними номерами десь в іншому місці мережі. При цьому немає необхідності встановлювати TCP-зв'язок, щоб передати десяток символів у кожному напрямку. Достатньо просто вкласти ім'я в UDP-пакет, запакувати це в IP-пакет і надіслати. На іншому кінці прикладна програма одержить пакет, прочитає ім'я, перегляне телефонний номер, покладе його в інший UDP-пакет і відправить назад. Якщо ж пакет на шляху втратиться, програма повинна діяти таким чином: якщо вона чекає відповіді занадто довго і стає зрозуміло, що пакет загубився, вона просто повторює запит, тобто надсилає ще раз те ж послання. Так забезпечується надійність передачі при використанні протоколу UDP.

На відміну від TCP, дані, що відправляються прикладним процесом через модуль UDP, досягають місця призначення як єдине ціле. Наприклад, якщо процес-відправник робить три записи в UDP-порт, то процес-одержувач має зробити три читання. Розмір кожного записаного повідомлення збігатиметься з розміром відповідного прочитаного. Протокол UDP зберігає границі повідомлень, обумовлені прикладним процесом. Він ніколи не поєднує кілька повідомлень в одне ціле і не поділяє одне повідомлення на частини.

Дамо поради щодо використання протоколів.

Альтернатива TCP — UDP дозволяє програмісту гнучко і раціонально використовувати надані ресурси, виходячи зі своїх можливостей і потреб. Якщо потрібна надійна доставка, то кращий вибір — TCP. Якщо потрібна доставка дейтаграм, то — UDP. Якщо потрібна ефективна доставка по ненадійному каналу передачі даних, то краще використовувати TCP. Якщо потрібна

ефективність на швидких мережах з короткими з'єднаннями, найкраще буде UDP.

Акцентуємо увагу на стеку протоколів TCP/IP (табл. 2.1).

Таблиця 2.1

Стек протоколів TCP/IP

№ рівня	Назва рівня
5	Application
4	Transport
3	Internet
2	Network
1	Hardware

Hardware описує середовище передачі даних.

Network — містить апаратно залежне ПО, що реалізує поширення інформації на визначених відрізках середовища передачі даних.

Internet (міжмережний рівень) представлений протоколом IP. На цьому рівні виконується маршрутизація інформації від вузла відправника до вузла адресата, надається апаратно незалежний і єдиний інтерфейс доставки інформації рівням, що лежать вище.

Transport (транспортний рівень) представлений протоколом TLP/IPUPP. Його задача — доставка пакетів, збереження цілісності потоку пакетів, диференціація логічних об'єктів, що породжують інформацію.

Application (прикладний рівень) містить прикладні програми для розв'язання задач FTP, E-mail і т. д.

Більш докладно стек протоколів можна проілюструвати ієрархією протоколів сімейства TCP/IP, яка наведена на рис. 2.8.

TFTP — *протокол найпростішого (тривіального) пересилання файлів.* Здійснює дейтаграмне пересилання файлів. Виходить повільніше і дорожче, ніж по FTP.

DNS — *протокол доменної (регіональної) системи імен* — протокол запитів на перетворення імен з доменної форми в машинно-числову.

Служба часу — служба синхронізації рознесених у мережі годин.

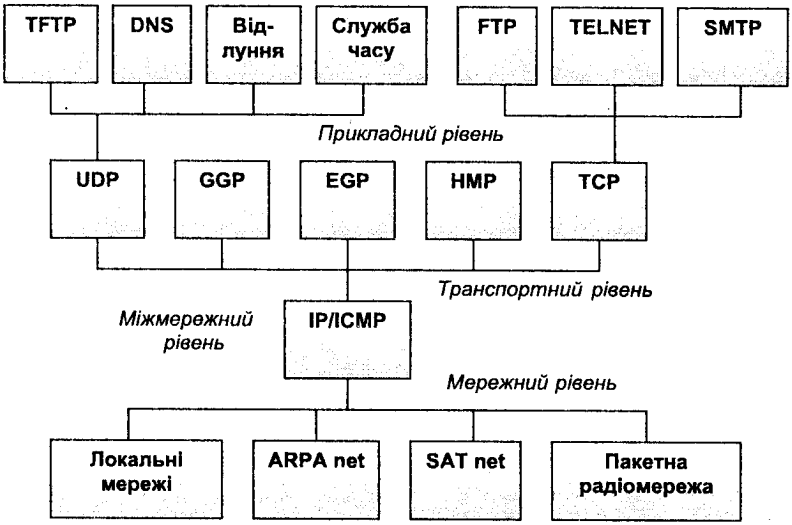


Рис. 2.8. Ієрархія протоколів та служб сімейства TCP/IP

FTP — *протокол пересилання файлів*. За допомогою цього протоколу організовується пересилання файлів мережею.

TELNET — *протокол емуляції термінала віддаленої машини*. За допомогою цього протоколу організовуються сеанси роботи на віддалених машинах мережі — емулюється термінал віддаленого комп'ютера.

SMTP — *протокол пересилання простої пошти* — протокол, що забезпечує роботу e-mail.

GGP — *протокол «шлюз — шлюз»* — протокол наскрізних мережних проводок.

EGP — *протокол зовнішнього шлюзу*.

HMP — *протокол моніторингу хосту* — протокол безупинного спостереження (контролю) за хостом.

ICMP — *протокол міжмережних керуючих повідомлень* — мережний протокол, що надає процесам у мережі можливість вибору оптимального маршруту пересилання й інших подібних речей.

Для ефективної самостійної роботи розробника Інтернет-проектів слід використовувати *RFC*-документацію, яка представлена у вигляді файлів. Вона містить опис різних стандартів, пристроїв, протоколів, пропозиції нових стандартів тощо.

RFC-документи розповсюджуються організацією *DDN NIC*. Багато вузлів мережі Інтернет пропонують набори RFC різної повноти.

2.4. Принципи адресації ресурсів мережі Інтернет

Для адресації ресурсів у мережі Інтернет існують два види адресації: абсолютна і доменна.

Розглянемо принципи абсолютної адресації.

Адреса в Інтернет складається з 4 байт. При записі байти відокремлюються один від одного крапками: 123.98.56.34 чи 5.55.55.5.

У дійсності адреса складається з декількох частин. Оскільки Інтернет є мережею мереж, то початок адреси повідомляє вузлам Інтернет, частиною якої з мереж є дана мережа. Права частина адреси повідомляє цій мережі, який вузол чи хост має одержати пакет.

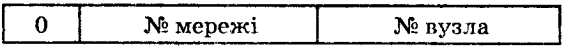
Кожен комп'ютер у мережі Інтернет має в цій схемі унікальну адресу, аналогічно звичайній поштовій адресі, а ще точніше — індексу. Обробка пакета відповідно до адреси також аналогічна. Поштова служба «знає», де знаходиться зазначене в адресі поштове відділення, а поштове відділення докладно «знає» підлеглий район. Інтернет «знає», де шукати зазначену мережу, а ця мережа «знає», де в ній знаходиться конкретний комп'ютер. Для визначення, де в локальній мережі знаходиться комп'ютер з даною числовою IP-адресою, локальні мережі використовують свої власні протоколи мережного рівня. Наприклад, *Ethernet* для відшукання *Ethernet-адреси* за IP-адресою комп'ютера, що знаходиться в даній мережі, використовує протокол ARP.

Числова адреса комп'ютера в Інтернет аналогічна поштовому індексу відділення зв'язку. Перші цифри індексу вказують на регіон, останні дві цифри — на номер поштової відділення в місті, області чи районі. Проміжні цифри можуть стосуватися як регіону, так і відділення, залежно від територіального розподілу і виду населеного пункту.

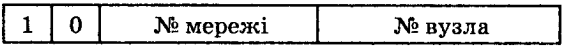
Аналогічно існує кілька типів адреси Інтернет (типи: *A, B, C, D, E*), що по-різному поділяють адреси на поля — *номери мережі* і *номер вузла*, від типу такого розподілу залежить кількість можливих різних мереж і машин у таких мережах.

Абсолютна адреса (IP-адреса) містить 4 байти інформації, кожен з яких відповідає за свій клас мереж.

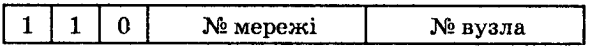
Клас A: адреса починається з нульового біта, номер мережі займає 1 байт, номер вузла — 3 байти.



Клас B: номер мережі займає 2 байти.



Клас C: під номер мережі виділяються 3 байти, <1 байта — під номер вузла, під адресу мережі — 24 біти.



Клас D: multicast (широкомовні мережі) — мережі, в яких застосовується групове розсилання.

Реально використовуються три класи адрес мережі (A, B і C).

2.5. Схеми доступу до ресурсів. Стандарт URL

Для швидкого пошуку та доступу до ресурсів, які пропонує мережа Інтернет, існує регіональна система імен. Розглянемо основні принципи її дії.

Числові адреси зручні для зв'язку машин, люди ж віддають перевагу іменам. Дуже непросто спілкуватися, використовуючи машинну адресацію (як би це звучало: «192.112.36.5 обіцяє незабаром...»?), ще важче запам'ятати ці адреси. Тому комп'ютерам у мережі Інтернет для зручності користувачів були надані власні імена. Тоді описана розмова приймає вид: «NIC обіцяє незабаром...». Усі додатки Інтернет дозволяють скористатися системними іменами замість числових адрес.

Як ми вже згадували, для розуміння корисно використовувати поштову аналогію. Мережні числові адреси цілком аналогічні поштовій індексації. Комп'ютери, які сортують кореспонденцію на поштових вузлах, орієнтуються саме за індексами, і тільки якщо з індексами виникає позаштатна ситуація, передають пошту на розгляд людям, які за адресою зможуть визначити

правильний індекс поштової відділення місця призначення. Людям же звично мати справу з географічними назвами — це аналоги доменних імен.

Звичайно, таке іменування має свої власні проблеми. Насамперед слід переконатися, що ніякі два комп'ютери, включені в мережу, не мають однакових імен. Необхідно також забезпечити перетворення імен у числові адреси, для того щоб машини (і програми) могли розуміти нас, що користуються іменами: техніка, як і раніше, спілкується мовою цифр.

На початку розвитку мережа Інтернет була невелика за розмірами, і мати справу з іменами було досить просто. NIC створив реєстратуру. Можна було надіслати запит і у відповідь одержати список імен і адрес. Цей файл (називається *«host file»*) регулярно поширювався по всій мережі — розсилався всім вузлам. Імена були простими словами, усі вони були унікальними. Після виклику імені комп'ютер переглядав цей файл і підставляв замість імені реальну числову адресу. Так само працює телефонний апарат із вбудованим списком абонентів.

Але в міру розвитку і розширення Інтернет зростала кількість користувачів, хостів, а тому збільшувався і згаданий файл. Виникали значні затримки при реєстрації й одержанні імені новим комп'ютером, стало важко вишукувати імена, що ще ніхто не використовував, занадто багато мережного часу витрачалося на розсилання цього величезного файла всім машинам, у ньому згаданим. Стало очевидно: щоб упоратися з такими темпами змін і росту мережі, потрібна розподілена оперативна система, що спирається на новий принцип.

Така система була створена, її назвали *доменною системою імен — DNS* (англ. — *Domain Name System*), а спосіб адресації — способом адресації за *доменним* принципом. *DNS* іноді ще називають регіональною системою найменувань.

Розглянемо структуру регіональної системи імен.

Доменна система імен — це метод призначення імен шляхом передачі мережним групам відповідальності за їхню підмножину імен. Кожен рівень цієї системи називається *доменом*. Домени в іменах відокремлюються один від одного крапками: comp1.group2.domain.com.ua, www.site.kharkov.ua, www.microsoft.com, www.rada.com.ua. В імені може бути різна кількість доменів, але практично їх не більше п'яти.

Першою в імені стоїть назва робочої машини — реального комп'ютера з IP-адресою. Це ім'я створене і підтримується гру-

пою (наприклад, комп'ютер comp1 у групі group2), до якої воно відноситься.

Група входить до більшого підрозділу, який у свою чергу є частиною національної мережі (наприклад, мережі України, домен *ua*). Для США найменування країни за традицією опускається, там найбільшими об'єднаннями є мережі освітніх (*edu*), комерційних (*com*), державних (*gov*), військових (*mil*) установ, а також мережі інших організацій (*org*) і мережних *ресурсів* (*net*). Але зараз, у зв'язку з великими темпами росту мережі Інтернет, з'являються нові доменні імена вищого рівня (*aero, tv* тощо).

Група може створювати чи змінювати будь-які їй підлеглі імена. Наприклад, якщо group2 вирішить поставити інший комп'ютер і назвати його main, вона ні в кого не повинна запитувати дозволу, усе, що потрібно, — це додати нове ім'я у відповідну частину відповідної всесвітньої бази даних.

Аналогічно, якщо, наприклад, у Харкові вирішать створити нову групу, наприклад *cars*, то власники домену *kharkov* зможуть це зробити, ні в кого на те не запитуючи дозволу. І тоді, якщо кожна група дотримується таких простих правил і завжди переконується, що імена, які вона створює, єдині в множині її безпосередніх підлеглих, то ніякі дві системи, де б ті не були в мережі Інтернет, не зможуть отримати однакових імен.

Ця ситуація аналогічна ситуації з присвоєнням географічних назв — організацією поштових адрес. Назви всіх країн розрізняються. Розрізняються назви всіх областей чи регіонів, і затверджуються ці назви на державному рівні (звичайно самі регіони піклуються про унікальність своїх назв, тому тут панує повна демократія: як республіка хоче, так вона і називається). В областях вирішують питання про назви районів і округів, у межах однієї республіки вони розрізняються. Аналогічно далі з містами і вулицями міст. У різних містах можуть бути вулиці з однаковими назвами: чому б не бути у всіх містах України по вулиці Шевченка? Це вулиці різних міст, і їх не переплутати. У межах одного населеного пункту вулиці неодмінно мають різні назви, причому іменування цих вулиць цілком у віданні відповідного центрального органу даного населеного пункту (мерії, сільради, міськради).

Таким чином, поштова адреса на основі географічних і адміністративних назв однозначно визначає точку призначення.

Оскільки Інтернет — мережа світова, то необхідний також спосіб передачі відповідальності за імена всередині країн їм

самим. На цей час прийняте дволітерне кодування держав, регламентоване стандартом *RFC 822*.

Так, наприклад, домен Канади називається *ca*, України — *ua*, США — *us* і т. д. Усього ж кодів країн майже 300, з яких близько 100 має комп'ютерну мережу того чи іншого роду. Єдиний каталог Інтернет знаходиться в державній організації SRI International (Каліфорнія, США).

Кожен ресурс у мережі Інтернет (web-документ, зображення, відеокліп, програма тощо) має адресу, яка може бути закодована за допомогою *універсального ідентифікатора ресурсів* (англ. — *Universal Resource Identifier*), або *URI*.

URI зазвичай складаються з трьох частин:

1) найменування способу, що використовується для доступу до ресурсу;

2) ім'я машини, на якій розташовується ресурс;

3) ім'я власне ресурсу, задане у вигляді шляху.

Для прикладу розглянемо URI специфікації HTML на сервері W3C:

http://www.w3.org/TR/PR-html4/cover.html

Наведений URI читається таким чином: «зазначений документ можна отримати за протоколом HTTP, він розміщений на вузлі-комп'ютері *www.w3.org*, локальний шлях до цього документа — */TR/PR-html4/cover.html*».

В Інтернет також можна зустріти схеми доступу до ресурсів «mailto» (для електронної пошти), «ftp» (для протоколу FTP) тощо.

Якщо необхідно адресувати фрагмент ресурсу (наприклад, частину якогось текстового документа), то використовують ідентифікатори фрагментів — вони вказують на місце всередині ресурсу. Такий URI закінчується символом «#», за яким іде власне вказівник (ідентифікатор фрагмента). Наприклад, URI

http://somesite.com/html/index.html#part_3

вказує на фрагмент файла *index.html* з іменем *part_3*.

На практиці користуються *URL* (англ. — *Uniform Resource Locator*). URL є підмножиною більш загальної схеми найменування — URI.

URL складається з доменного імені, шляху до сторінки на сайті та імені файла сторінки, як це показано на рис. 2.9.

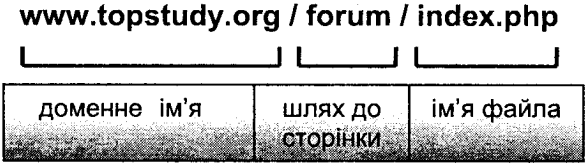


Рис. 2.9. Приклад URL

Пошук адреси за доменним іменем виконується вживанням імені, зя яким здійснюється звертання до DNS. Усі комп'ютери в Інтернет здатні користуватися доменною системою. І працюючий у мережі комп'ютер завжди знає свою власну мережну адресу. Коли користувач вводить в адресному рядку доменне ім'я, наприклад, *www.topstudy.org*, комп'ютер має перетворити його на адресу. Для цього він звертається до *DNS-серверів*. Це вузли, робочі машини, що володіють відповідною базою даних, в обов'язок яких входить обслуговування такого роду запитів.

DNS-сервер починає обробку імені з правого його кінця і рухається по ньому вліво, тобто спочатку виконує пошук адреси в найбільшій групі (домені), далі поступово звужує пошук. Але для початку опитується (на предмет наявності в нього потрібної інформації) місцевий вузол. Тут можливі три випадки:

- місцевий сервер знає адресу, оскільки ця адреса міститься в його частині всесвітньої бази даних;
- місцевий сервер знає адресу, тому що хтось недавно вже запитував ту ж адресу. Коли запитується адреса, сервер DNS утримує її в пам'яті якийсь час, саме на випадок, якщо хтонебудь ще запитає пізніше ту ж адресу — це підвищує ефективність системи;
- місцевий сервер адресу не знає, але знає як її з'ясувати.

Як місцевий сервер може дізнатися запитану адресу? Наприклад, запитується адреса *mx.kiev.ua*. У прикладному чи системному програмному забезпеченні місцевого сервера є інформація про те, як зв'язатися з *кореневим сервером*. Це сервер, що знає адреси серверів імен вищого рівня (самих правих в імені), тут це рівень держав (рангу домену *ua*). У нього запитується адреса комп'ютера, відповідального за зону *ua*. Місцевий DNS-сервер зв'язується з цим більш загальним сервером і запитує в нього адресу сервера, відповідального за домен *kiev.ua*. Тепер

уже запитується цей сервер і в нього запитується адреса робочої машини *mx*.

Насправді для підвищення ефективності пошук починається не з самого верху, а з найменшого домену, до якого входить і комп'ютер користувача, і комп'ютер, ім'я якого він запитує. Наприклад, якщо комп'ютер має ім'я *host.domain.com.ua*, то опитування почнеться (якщо ім'я не з'ясується відразу) не зі всесвітнього сервера, щоб дізнатися про адресу сервера групи *ua*, а відразу з групи *ua*, що значно скорочує пошук.

Цей пошук адреси цілком аналогічний пошуку шляху листа без надписаного поштового індексу. Як визначається цей індекс? Усі регіони пронумеровані — це перші цифри індексу. Лист пересилається на центральний поштамт цього регіону, де є довідник з нумерацією районів цього регіону — це наступні цифри індексу. Тепер лист йде на центральний поштамт відповідного району, де вже знають усі поштові відділення в підлеглому районі. У такий спосіб за географічною адресою визначається відповідний йому поштовий індекс. Таким же чином визначається й адреса комп'ютера в Інтернет, але подорожує не послання, а запит комп'ютера про цю адресу. І на відміну від випадку з поштою, інформація про адресу доходить до одержувача так, як би районний поштамт місця призначення відправляв листа, люб'язно повідомляючи на майбутнє одержувача про індекс, якого він не знав.

Наведемо зауваження щодо регіональної системи імен.

Поширеними є кілька помилок, яких припускаються початківці, маючи справу з іменами. Наведемо декілька правильних тверджень у якості опорних, щоб застерегти від типових помилок:

1) частини доменного імені вказують, хто відповідальний за підтримку цього імені, тобто в чиєму підпорядкуванні-веденні воно перебуває. Вони можуть узагалі нічого не повідомляти про власника комп'ютера, що відповідає цій IP-адресі, чи навіть (незважаючи на коди країн), де цей вузол-комп'ютер знаходиться;

2) частини доменного імені не завжди вказують локальну мережу, у якій розташований комп'ютер. Нерідко доменні імена і мережі перекриваються, і міцних зв'язків між ними немає: дві машини одного домену можуть не належати одній мережі. Наприклад, системи *mail.isp.net* і *chat.isp.net* можуть знаходитися в зовсім різних мережах.

Наголосимо ще раз: *доменні імена вказують на відповідального за домен*.

У комп'ютера (вузла) може бути багато імен. Зокрема, це вірно для машин, що надають які-небудь послуги, які у майбутньому можуть бути переміщені під опіку іншої машини. Коли ці служби будуть переміщені, то ім'я, під яким ця машина виступала як такий *сервер*, буде передано новій машині-серверу разом з послугами, — для зовнішніх користувачів нічого не зміниться. Тобто вони продовжуватимуть користуватися цією службою, запитуючи її за тим же ім'ям, незалежно від того, який комп'ютер насправді займається обслуговуванням. Імена, які за змістом відносяться до служби, називаються «канонічними іменами» чи «кіменами» (англ. — *cnames*). У мережі Інтернет вони зустрічаються досить часто.

Для зв'язку імена необов'язкові. Якщо користувачеві надходить повідомлення «адресат невідомий», то це означає, що Інтернет не може перетворити використане вами ім'я в число, — ім'я більш недієздатне в тому вигляді, у якому його «знає» ваш комп'ютер. Один раз одержавши числовий еквівалент імені, ваша система перестає використовувати для зв'язку на машинному рівні *доменну* форму адреси.

Запам'ятовувати краще імена, а не числові адреси. Деяким здається, що система імен — це «ще одна ланка в ланцюзі, що може вийти з ладу». Але адреси прив'язані до конкретних точок мережі. Якщо комп'ютер, що надає деякі послуги, переноситься з одного будинку в інший, його мережне розташування, а отже, і адреса, швидше за все зміняться. Ім'я ж змінювати не треба. Коли адміністратор одержує нову адресу, йому потрібно тільки обновити запис імені в базі даних так, щоб ім'я вказувало на нову адресу. Після цього ім'я працює, як і раніше.

Регіональна система імен, можливо, і має складний вигляд, але це одна з тих складових, які роблять спілкування з мережею більш простим і зручним. Безсумнівна перевага доменної системи полягає в тому, що вона розбиває Інтернет на набір цілком доступних для огляду і керованих частин. Хоча мережа включає мільйони комп'ютерів, усі вони пойменовані, й іменування це організовано в зручній раціональній формі, що спрощує роботу.

Із викладеного вище зробимо такі висновки:

1) доменна адресація побудована за стандартом DNS (Domen Name System);

2) DNS заснована на IP-адресі. Кожній IP-адресі відповдає певне доменне ім'я;

3) для доступу до ресурсів існують схеми доступу;

4) ознака схеми доступу: *http://, ftp://, mailto://*;

5) адресація сайтів: *ім'я_вузла.DNS-адреса/ім'я_ресурсу*;

Наприклад: *www.colordyn.com/index_eng.html*

6) адресація для електронної пошти:

ім'я_користувача @ DNS-адреса

Наприклад: *petro@adm.kharkiv.com*

2.6. Контрольні запитання

1. Що таке вузол мережі?
2. Дайте визначення мережі Інтернет.
3. Що таке Інтранет?
4. Що таке Екстранет?
5. Які існують топології мереж?
6. Що таке хост?
7. Що таке браузер?
8. Що таке WWW?
9. Наведіть приклади поширених браузерів.
10. Дайте визначення сайта.
11. Що входить до поняття «Інтернет-проект»?
12. У чому особливості технології «клієнт — сервер»?
13. Що таке протокол?
14. Що таке OSI?
15. Дайте характеристику еталонної моделі ISO OSI.
16. Охарактеризуйте рівні взаємодії в моделі OSI.
17. Як відбувається пересилання бітів у мережі Інтернет?
18. Як здійснюється пересилання даних у мережі Інтернет?
19. Що таке дейтаграма?
20. Дайте характеристику рівнів стека протоколів TCP/IP.
21. Порівняйте TCP і UDP.
22. Які типи адресації існують у мережі Інтернет?
23. Дайте характеристику IP-адресації.
24. Які основні поняття Регіональної Системи Імен?
25. Що таке DNS?
26. Що таке DNS-сервер?
27. Як здійснюється пошук за доменним іменем?
28. Як складаються імена сайтів?
29. Як складати поштові адреси?

Розділ 3
МЕТОДИ ТА ЗАСОБИ ІНТЕРНЕТ-ТЕХНОЛОГІЙ

3.1. Основні поняття web-технологій. Мова HTML

Як зазначалося вище, існує організація, що розробляє різні стандарти, протоколи, специфікації та рекомендації відносно розвитку мережі Інтернет — *World Wide Web Consortium, (W3C)* (*див.* Додаток 1). Цією організацією було розроблено й стандарт *HTML*. На основі стандарту HTML створюються вебдокументи (або web-документи).

Web-документ — це документ у форматі ASCII, що не компілюється, а опрацьовується інтерпретатором. Мова HTML до регістра нечутлива. Формати файлів, у яких *можуть* зберігатися web-документи (*див.* також Додаток 5):

.htm;
.html;
.shtml;
.phtml;
.dhtml;
.css (стильове оформлення документа);
.js (для збереження скриптів).

HTML (англ. *Hyper Text Markup Language* — мова розмітки гіпертексту) — це мова розмітки даних. Початкова її назва — *SGML*. На рисунку 3.1 подано історичну послідовність розвитку мов розмітки даних.

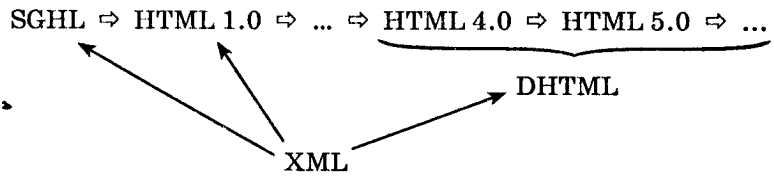


Рис. 3.1. Історична послідовність мов розмітки даних

Розглянемо структуру документа мовою HTML:

заголовок	*{ службова частина*
тіло документа	*{ корисний текст-контент*

Базовим структурним елементом документа є *тег* (від англ. *tag*), або *дескриптор*, або *контейнер*.

Дескриптори (теги) у HTML — це спеціальні елементи розмітки, що визначають, як даний документ HTML має інтерпретуватися програмою перегляду інформації — браузером. Кожний тег описує посилання на якийсь об'єкт з *об'єктної моделі браузера* (докладніше про об'єктну модель браузера *див.* Розділ 6).

Теги HTML бувають одно- і двокомандні.

Однокомандні теги зазвичай використовуються для зміни вигляду виведеної на екран інформації. Вони задаються в такому вигляді:

< команда [атрибут=значення [атрибут=значення]...]>текст ,

де *команда* — команда HTML;

атрибут=значення — задання конкретних значень параметрам (атрибутам) команди (не є обов'язковими);

текст — текст, що виводиться на екран тільки для читання у вигляді, обумовленому командою та її атрибутами.

Двокомандні теги є основним засобом реалізації HTML-документів і використовуються для задання параметрів документа (заголовок, вставка посилань, об'єктів та ін.). Вони спарені так, що за стартовим тегом має йти відповідний завершальний тег, а між ними міститься текст чи інші теги. Два теги та відокремлена ними частина документа утворюють блок, що називається *HTML-елементом*.

Двокомандні теги мають такий синтаксис:

< команда [атрибут=значення [атрибут=значення]...]>
[текст][тег] [[текст][тег]...]
</команда [атрибут=значення [атрибут=значення]...]>

де / — використовується у випадку, коли тег є останнім (закриваючим) у блоці тегів.

Якщо тег складається з двох команд, то першою має бути визначена команда без косої риски (початок виконання тегу), а другою — команда, що випереджається косою рискою (/), що інформує про закінчення дії тегу.

Наведемо приклади тегів (результат відображення в браузері *див.* рис. 3.2).

Цей текст буде виділений жирним накресленням — приклад подвійного тегу;
<hr> — приклад одиничного тегу (виводить горизонтальну лінію).

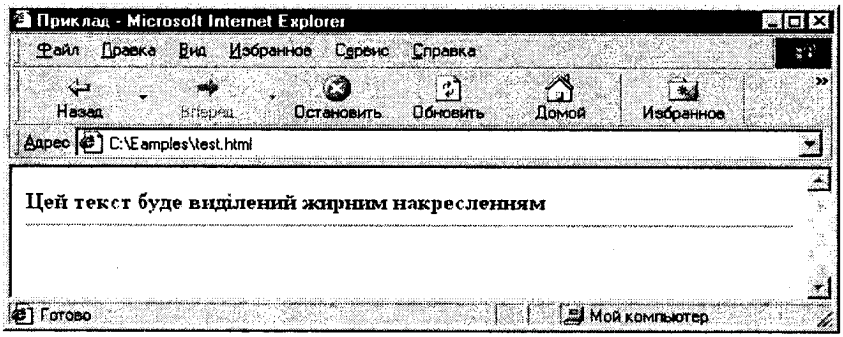


Рис. 3.2. Вигляд у браузері web-документа

Як вже було зазначено, всередині тегів можуть визначатися їхні атрибути — додаткові команди інтерпретатору браузера, що модифікують або доповнюють основні дії тегу. В одних тегів атрибутів дуже багато, а в інших немає взагалі. Якщо тег подвійний, атрибути вказуються тільки у відкриваючому тегу — у закриваючому вони будуть просто проігноровані. Кілька атрибутів пишуть через пробіл. Параметр набуває значення (числового або символьного) за замовчуванням, якщо значення не задане явно.

За специфікацією HTML значення атрибутів тегів необхідно укладати в лапки, однак більшість браузерів розуміють запис без них. Виключення складають значення, у яких є присутнім пробіл, — тоді такий атрибут обов'язково необхідно укладати в лапки.

Теги можуть бути вкладеними, тобто всередині пари тегів можна визначати інші теги, однак вкладені теги не повинні перетинатися: вкладений тег має бути закритим до того, як буде закрито зовнішній тег.

Наприклад, для задання абзацу тексту, у якому зустрічається виділений жирним накресленням фрагмент, можна використовувати таку конструкцію:

* Це абзац тексту з виділеним як болд і болд-італік фрагментом *

У той же час неправильною з погляду синтаксису була б наступна конструкція (*див.* рис. 3.3):

* Це абзац тексту з виділеним як болд і болд-італік</ b> фрагментом *

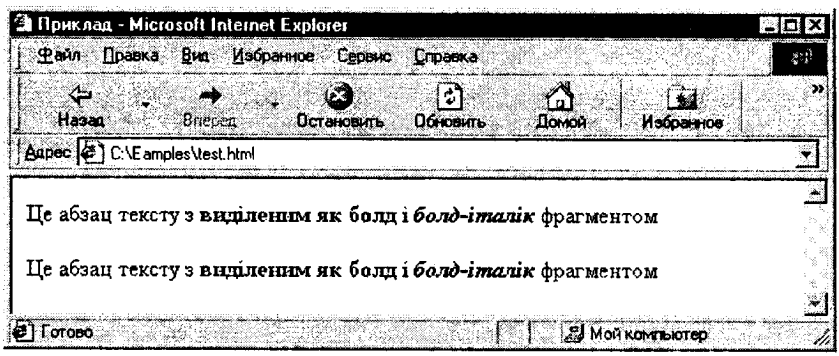


Рис. 3.3. Вигляд у браузері web-документа з використанням тегів оформлення шрифтів

Слід зазначити, що навіть такий помилковий код буде правильно інтерпретований більшістю сучасних браузерів. Навіть такі серйозні помилки, як незакриті теги таблиць, браузер може спробувати відстежити і виправити. У будь-якому випадку повідомлення про помилки в розмітці на екран не виводяться, і браузер відобразить документ. Це залежить від конкретних реалізацій браузера і допущених помилок. Може трапитися, що один браузер зможе коректно відобразити HTML, а в іншому виникнуть серйозні проблеми, тому краще не сподіватися на інтелектуальні здібності браузерів і з самого початку коректно писати код.

Мова HTML є регістронезалежною. Це означає, що можна використовувати як прописні, так і заголовні букви у визначенні тегів і атрибутів. Задавати в документі тег заголовка можна, наприклад, так:

<title></title>
<TITLE></TITLE>
<Titl></TitLe>
<tItLe></tITle>,

й усе це буде правильним з погляду синтаксису мови. Однак на практиці використовують перші два способи, коли теги пишуться тільки у верхньому чи тільки в нижньому регістрі — це є ознакою доброго стилю в програмуванні і поліпшує читаність коду (слід мати на увазі, що написане згодом швидше за все прийдеться модифікувати, причому це може бути вже інша людина).

У процесі створення документа часто виникає необхідність робити нотатки-коментарі, для того щоб згодом вільніше орієнтуватися в HTML-коді. Коментарі містяться між знаками «*<!--*» і «*-->*».

Браузер, зустрівши таку конструкцію, просто не інтерпретуватиме текст між знаками коментаря. Приклад:

```
<!-- Це коментарі в HTML-документі -->
<!--
Коментарі можна переносити
на наступний рядок. Теги усередині коментарів НЕ інтерпретуються:
<br><br><br> — нічого не відбудеться.
-->
```

Є деякі винятки щодо інтерпретації коду, поміщеного в коментарі в тегу *<script></script>*, але про це йтиметься нижче.

3.2. Класифікація тегів

Розглянемо основні теги.

1. Теги для відкриття/закриття документів і їхніх частин (структурні теги).

Перше, з чого починається гіпертекстовий документ — це тег *<html></html>*. Усі інші теги й інформаційний текст мають бути розміщені між цими тегами. Даний тег повідомляє браузеру, що документ написаний з використанням HTML.

Далі документ поділяється на дві великі частини — заголовну і безпосереднє тіло документа. Заголовна частина міститься між тегами *<head>* та *</head>*, а тіло — *<body>* та *</body>*. Жодна з цих частин не є обов'язковою.

<HTML>.....</HTML> – початок і кінець документа
<HEAD>.....</HEAD> – заголовок
<BODY>.....</BODY> – тіло документа

Заголовна частина має містити теги *<META>*, у яких утримується службова інформація для сервера (ключове слово, авторизація).

У заголовній частині можуть описуватися властивості документа, а також підключатися зовнішні файли.

Наприклад, у заголовній частині розташовується назва документа — тег *<title></title>*, у якому між відкриваючою і закриваючою частинами міститься рядок тексту, що відображатиметься у заголовку вікна.

У тегу *<body> ... </body>* розміщується вся інформаційна частина документа, його тіло. Отже, у найпростішому випадку документ із деяким текстом і заголовком матиме такий вигляд (рис. 3.4):

```
<html>
<head>
<title>Це заголовок документа</title>
</head>
<body>
Це просто текст
</body>
</html>
```

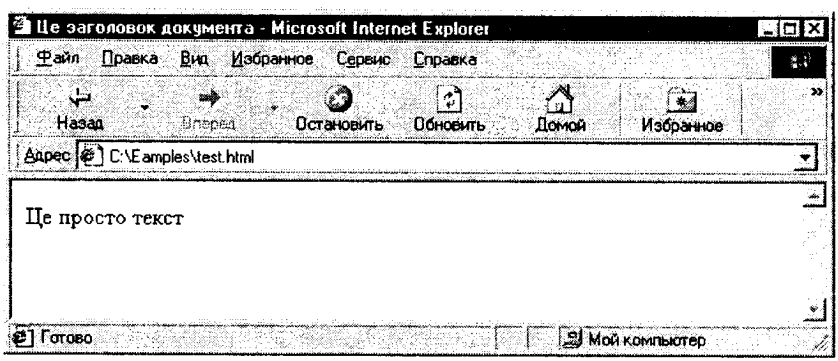


Рис. 3.4. Вигляд у браузері web-документа з простим текстом

Як уже зазначалося вище, теги можуть мати атрибути. Тег *<body></body>* може містити такі атрибути:

• *TEXT* — задає колір тексту в документі. Можна використовувати символьні або шістнадцяткові значення. Останнє задається у форматі *#rrggbb*, де *rr*, *gg*, *bb* — відповідно червоний, зелений і синій компоненти представлення кольору в моделі RGB:

<BODY TEXT="#ff0000">. Текст документа за замовчуванням буде виділений червоним кольором </BODY> (*див.* рис. 3.5);

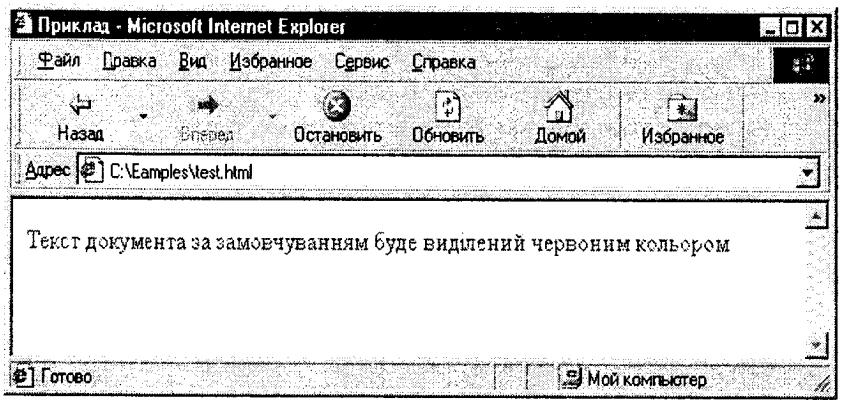


Рис. 3.5. Вигляд у браузері web-документа
із заданим атрибутом кольору тексту

• *LINK*, *VLINK*, і *ALINK* задають колір гіперпосилань у документі, відповідно для посилань, що зустрічаються в тексті, уже відвіданих і активних у даний момент часу (в момент клацання мишею). За замовчуванням ці кольори рівні *LINK=blue (#0000FF)*, *VLINK=purple (#800080)* і *ALINK=red (#FF0000)*.

• *BACKGROUND* застосовується для задання фонового рисунка. Значенням атрибута є шлях до рисунка на сервері.

```
<BODY BACKGROUND="path/imagename.gif">
...
</BODY>
```

Атрибут *BGCOLOR*, на відміну від попереднього, установлює колір фону, а не фоновий рисунок. Як і будь-який інший, атрибут кольору може задаватися числовим чи символьним значенням. Надалі ми не уточнюватимемо цей факт.

<BODY BGCOLOR="#00aa00"> — задає зелений колір фону HTML-документа (*див.* рис. 3.6).

2. Інші теги — використовуються для створення і редагування HTML-текстів. До цих тегів відносять:

— теги форматування;

— теги для задання списків;
— теги для створення таблиць;
— якірні теги (для створення гіперпосилань);
— теги для впровадження графічних зображень та карт-зображень;
— теги для впровадження об'єктів та аплетів;
— теги для задання формулярів;
— теги для створення фреймових структур;
— теги, які управляють роботою web-документа в цілому.

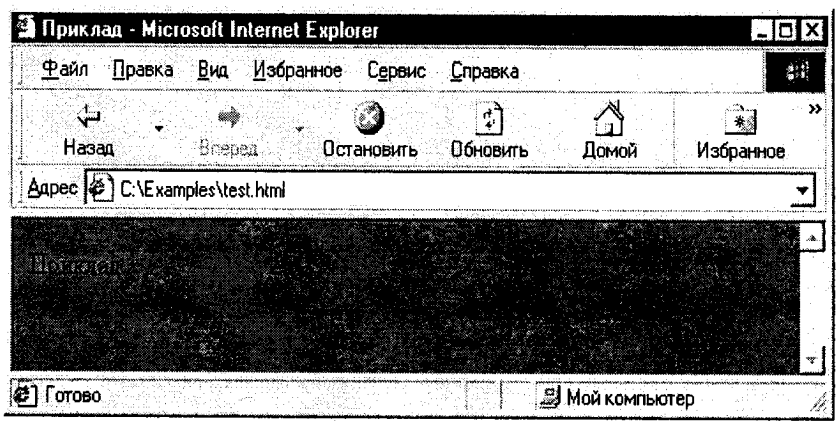


Рис. 3.6. Вигляд у браузері web-документа із заданим атрибутом кольору

За іншою класифікацією HTML-теги можуть бути умовно поділені на дві категорії:

1) теги, що визначають, яким чином відображатиметься тіло документа, його видима частина;

2) теги, що описують загальні властивості документа, такі як заголовок, автор документа або вказівки щодо пошукових робіт.

3.3. Поняття логічного та фізичного форматування

Існують два види форматування документа: *логічне* і *фізичне*. *Фізичне* форматування забезпечується буквальним виконанням браузером тегів форматування.

Логічне ж форматування залежить від змісту і структури документа.

Наприклад, <Comment>...</Comment>– форматує текст так, начебто це лістинг програми.

Які ж існують теги форматування?

Опис тегів, призначених для форматування тексту, доречно почати з опису абзаців.

Основним елементом HTML-документа є *абзац*. У тексті HTML-документа знаки переведення рядка і пробіли при обробці їх браузером *ігноруються*.

Абзац можна задати за допомогою тегів * *. Текст, розміщений між ними, буде розпочатий з нового рядка і відокремлений від попереднього. Наприклад:

```
<html>
<body>
<p>Це текст першого абзацу. Це текст першого абзацу. Це текст першого абзацу. Це текст першого абзацу. Це текст першого абзацу. Це текст першого абзацу. Це текст першого абзацу. Це текст першого абзацу. Це текст першого абзацу. Це текст першого абзацу</p>
<p>Це текст іншого абзацу. Це текст іншого абзацу. Це текст іншого абзацу. Це текст іншого абзацу. Це текст іншого абзацу. Це текст іншого абзацу. Це текст іншого абзацу. Це текст іншого абзацу. Це текст іншого абзацу. Це текст іншого абзацу. Це текст іншого </p>
</body>
</body>
</html>
```

Результат відображення в браузері наведено на рис. 3.7.

Для вирівнювання тексту в документі використовується атрибут *align*, що дозволяє вирівняти текст за лівим чи правим краєм і по центру:

align="left" — вирівнювання по лівому краю;

align="right" — вирівнювання по правому краю;

align="center" — вирівнювання по центру.

Тег * * також може мати цей атрибут.

Як уже було сказано, мова HTML не є чутливою до переносів рядків, табуляцій і ланцюжків пробілів. Будь-яку кількість пробілів браузер відобразить як один пробіл у тексті.

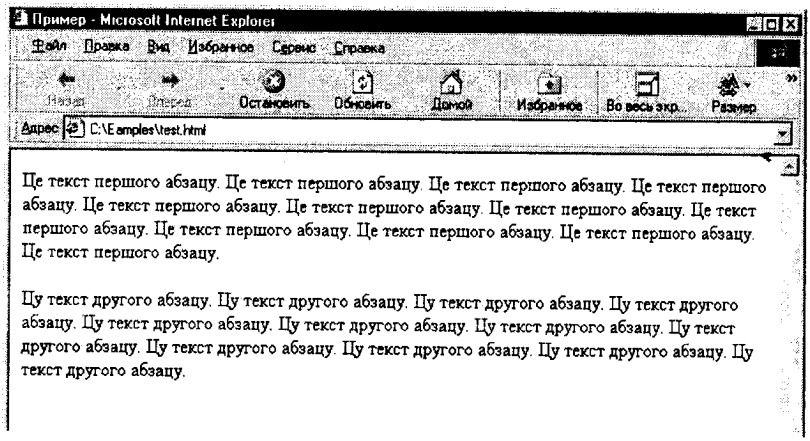


Рис. 3.7. Вигляд у браузері web-документа тексту з тегами визначення абзаців

Якщо просто скопіювати текстовий файл без тегів і помістити його в документ HTML, форматування не збережеться, якщо тільки не скористатися тегом *<pre></pre>*. Поміщений між тегами текст матиме такий самий вигляд, як і в оригіналі. Для ілюстрації розглянемо наступний приклад:

```
<html>
<body>
<pre>
Цей текст збереже своє форматування.
Можна
        робити
                відступи
табуляцією чи пропускати

рядки, а також використовувати багато (     ) пробілів.
</pre>
</body>
</html>
```

Результат відображення в браузері наведено на рис. 3.8.

А тепер поглянемо на текст, якщо не використовувати тег *<pre></pre>*:

```
<html>
<body>
А цей текст форматування не збереже.
Відступи
        табуляцією
                і пробіли (    ) не збережуться,
також, як і пропущені

рядки
</body>
</html>
```

Результат відображення в браузері наведено на рис. 3.9.

Примітка: далі в прикладах теги на початку і наприкінці документа можуть опускатися.

Частково вирішити проблему з форматуванням допомагає використання *escape-послідовності * — нерозривний пробіл (докладніше *див.* Додаток 4). Нерозривні пробіли слід використовувати замість звичайних пробілів, а не на додаток до них. Наприклад, якщо необхідно запобігти переносу рядка між *version* і *3*, то слід набрати *version 3* (але не *version 3*).

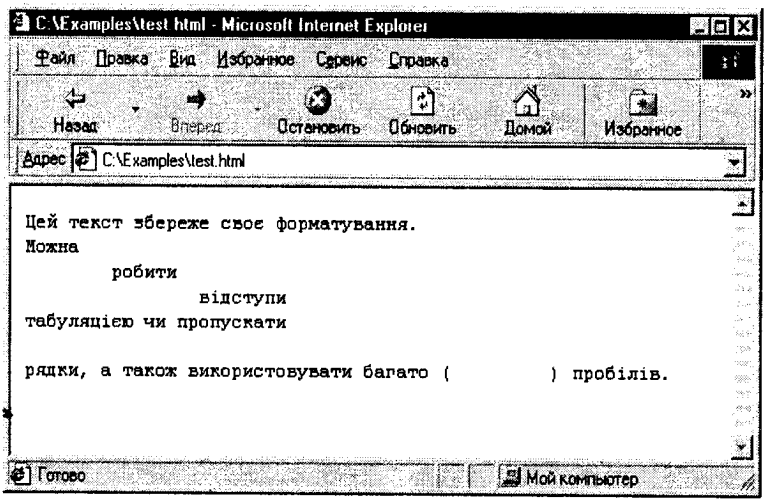


Рис. 3.8. Вигляд у браузері web-документа

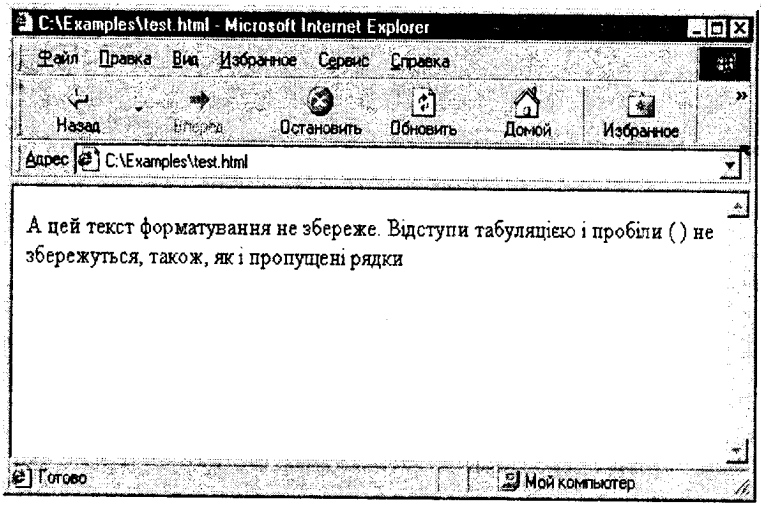


Рис. 3.9. Вигляд у браузері web-документа

Розглянемо *теги фізичного форматування*. Вони визначають зовнішній вигляд символів прямим способом. Список тегів, які часто використовуються, наведений нижче:

<B></B> — виділення жирним шрифтом (болд);

<I></I> — виділення похилим шрифтом (курсивом);

<U></U> — виділення підкресленням;

<STRIKE></STRIKE> — виділення перекреслюванням;

<TT></TT> — оформлення шрифтом з фіксованою шириною літер (моноширинним шрифтом); використовується рідко;

<BIG></BIG > — текст із великим розміром літер (використовується рідко);

<SMALL></SMALL> — текст із малим розміром літер (використовується рідко).

Результат відображення в браузері наведено на рис. 3.10.

<BLINK></BLINK> — миготливий текст (підтримується не всіма браузерами, дуже рідко використовується).

Під час верстання в HTML математичних формул, а також при написанні розмірності одиниць виникає необхідність у заданні підрядкових і надрядкових індексів. Мова дозволяє це

робити, і теги і відповідно допоможуть у вирішенні цієї проблеми.

```
S<sub>1</sub>, ..., S<sub>n</sub><br>
a<sub>1</sub>x<sup>3</sup> + a<sub>2</sub>x<sup>2</sup> +
a<sub>3</sub>x = 0
```

Результат відображення в браузері наведено на рис. 3.11.

Для відображення текстової чи графічної інформації в центрі наданого простору використовують тег .

```
<center>Цей текст буде розташовано посередині</center>
```

Результат відображення в браузері наведено на рис. 3.12.

Існує ціла група логічних елементів форматування, названа так через своє призначення — наголошувати на тому, *до чого* застосовується дане форматування. При цьому верстальник документа заздалегідь не знає, яким чином буде відображений зміст тегу — все залежить від конкретної версії браузера, у якому буде відображений документ.

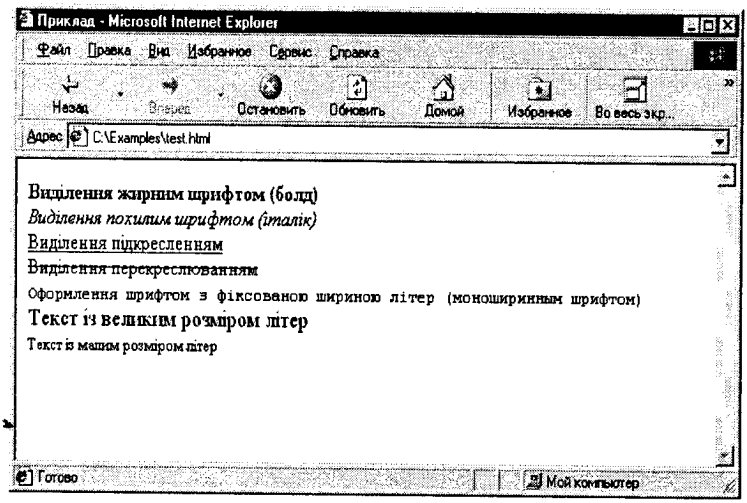


Рис. 3.10. Вигляд у браузері web-документа

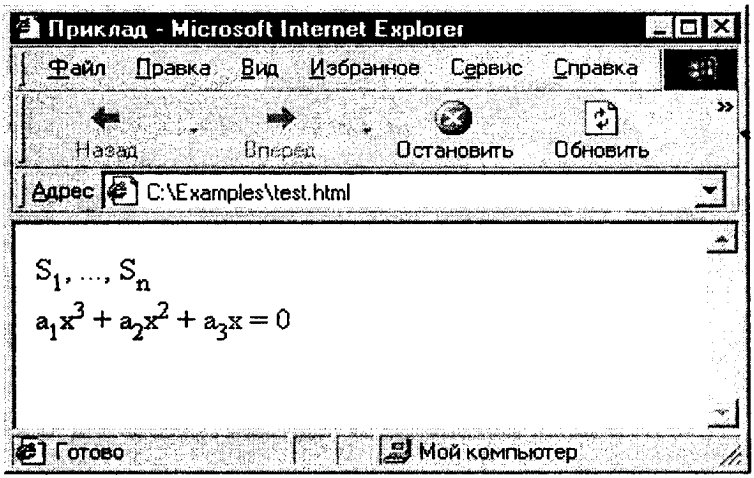


Рис. 3.11. Вигляд у браузері web-документа з формулами

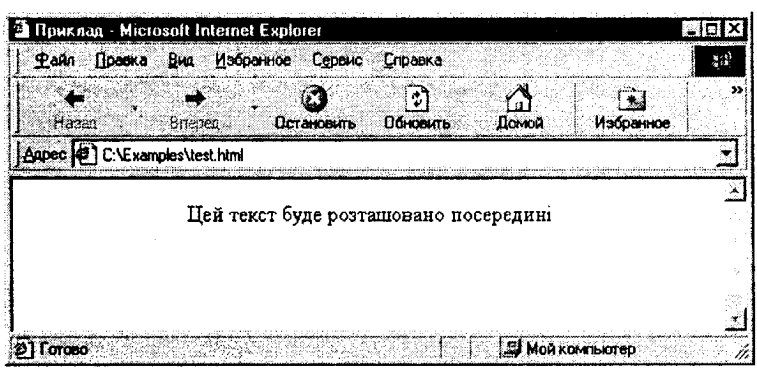


Рис. 3.12. Вигляд у браузері web-документа з центрованим текстом

Основні теги логічного форматування:

<BLOCKQUOTE></BLOCKQUOTE> — цитата; зазвичай відокремлюється пробілами зверху і знизу, а також відступом ліворуч, і відображається курсивом;

<EM></EM> — текст, що має особливе значення; зазвичай виділяється курсивом;

<STRONG></STRONG> — «посилене» виділення тексту;

<KBD></KBD> — текст, уведений користувачем (вірніше, ілюстрація цього тексту);

<CODE></CODE> — лістинг програми;

<SAMP></SAMP> — позначає текст як зразок. Текст, позначений цим тегом, зазвичай виділяється як моноширинний;

<VAR></VAR> — ім'я змінної. Зазвичай виділяється курсивом чи як моноширинний текст;

<ADDRESS></ADDRESS> — елемент визначає таку інформацію, як адреса, підпис і авторство. Зазвичай виділяється курсивом і відображається з відступами зверху та знизу;

<ACRONYM></ACRONYM> — застосовується для позначення акронімів (абревіатур), наприклад, WWW. Не відображається візуально, але необхідний для браузерів з мовним синтезом і в деяких інших випадках;

<COMMENT></COMMENT> — текст, вміщений між тегами, не відображається браузером.

Наприклад, код

```
<BLOCKQUOTE>Veni, Vidi, Vici</BLOCKQUOTE><br>
<EM>Важливий текст</EM><br>
<STRONG>Сильне виділення тексту</STRONG><br>
<KBD>Введено з клавіатури</KBD><br>
<CODE>include("adminlib/cofig.inc.php");</CODE><br>
<SAMP>Зразок</SAMP> <br>
<VAR>N</VAR><br>
<ADDRESS>м. Київ, вул. Банкова, 9</ADDRESS><br>
<ACRONYM>WWW</ACRONYM> — World Wide Web<br>
<COMMENT>Коментар до тексту</COMMENT> <br>
```

відображатиметься в браузері, як показано на рис. 3.13.

Слід зазначити, що не всі браузери підтримують той чи інший тег логічного форматування. Якщо тег не підтримується, то як і в будь-якому іншому випадку, коли інтерпретатор браузера зустрічає незнайомий тег, він просто ігнорується, тобто в даному випадку текст, відформатований таким чином, відобразиться за замовчуванням.

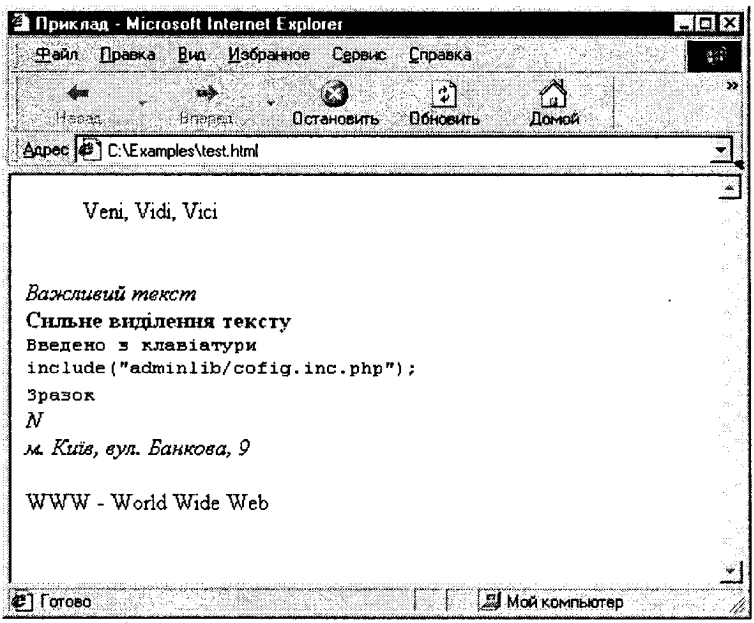


Рис. 3.13. Вигляд у браузері web-документа
з логічним форматуванням

Задання рівнів заголовків. Ці теги також можна віднести до тегів форматування. Їх використання приводить до масштабування кегля шрифту.

<Hx> текст </Hx> — де $x = 1, 2, \ldots, 6$;
<H1> — найбільший кегль;
<H3> — задається за замовчуванням;
<H6> — найменший кегль.

Приклад:

```
<h1>Заголовок 1</h1>
<h2>Заголовок 2</h2>
<h3>Заголовок 3</h3>
<h4>Заголовок 4</h4>
<h5>Заголовок 5</h5>
<h6>Заголовок 6</h6>
```

Результат відображення в браузері наведено на рис. 3.14.

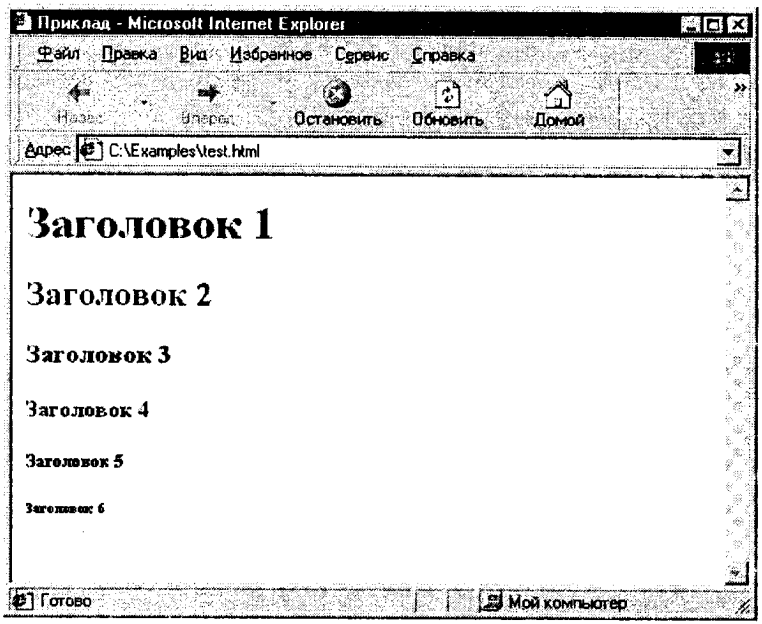


Рис. 3.14. Вигляд у браузері web-документа з форматованими заголовками

Мова гіпертекстової розмітки дозволяє задавати розміри шрифту, яким буде відображатися текст документа. За замовчуванням розмір шрифту дорівнює 3, однак, використовуючи тег *<FONT>* з атрибутом *size="..."*, його можна перевизначити. Використовуються як відносні розміри шрифтів, так і абсолютні. Наступний приклад ілюструє кодування розміру шрифту:

```
<FONT size=7>Розмір шрифту дорівнює 7.
</FONT>
<BR>
<FONT size=6>Розмір шрифту дорівнює 6.
</FONT>
<BR>
<FONT size=5>Розмір шрифту дорівнює 5.
</FONT>
<BR>
<FONT size=4>Розмір шрифту дорівнює 4.
</FONT>
```

```
<BR>
<FONT size=3>Розмір шрифту дорівнює 3.
</FONT>
<BR>
<FONT size=2>Розмір шрифту дорівнює 2.
</FONT>
<BR>
<FONT size=1>Розмір шрифту дорівнює 1.
</FONT>
```

Результат відображення в браузері наведено на рис. 3.15.

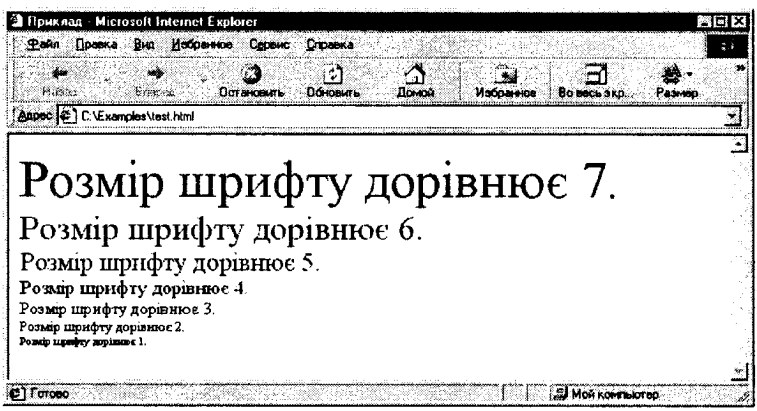


Рис. 3.15. Вигляд у браузері web-документа з кодованим розміром шрифту

Окрім абсолютних значень розмірів шрифту, в HTML можна використовувати і відносні розміри.

Якщо в тегу *<BASEFONT>* установити параметр *size* рівним деякому числу, то браузер використовуватиме зазначений розмір як базовий. За замовчуванням базовий розмір дорівнює 3. Щодо цього базового розміру можна вказувати розмір шрифту тексту, що розміщений у контейнері тегів *<FONT>*. Для цього досить присвоїти атрибуту *size* одне з наступних значень:

+1...+7 — збільшення шрифту на зазначену кількість одиниць щодо базового розміру;

−1...−7 — зменшення шрифту на зазначену кількість одиниць щодо базового розміру.

Наступний приклад демонструє управління відносним розміром шрифту:

Звичайний текст. Його абсолютний розмір дорівнює 3.
*
*
<BASEFONT SIZE=5>
Установка базового розміру 5.
*
*
<FONT size=+7>Збільшення до абсолютного розміру, рівного 12.
</FONT>
*
*
<FONT size=-2>Зменшення до абсолютного розміру, рівного 3.
</FONT>

Результат відображення в браузері наведено на рис. 3.16.

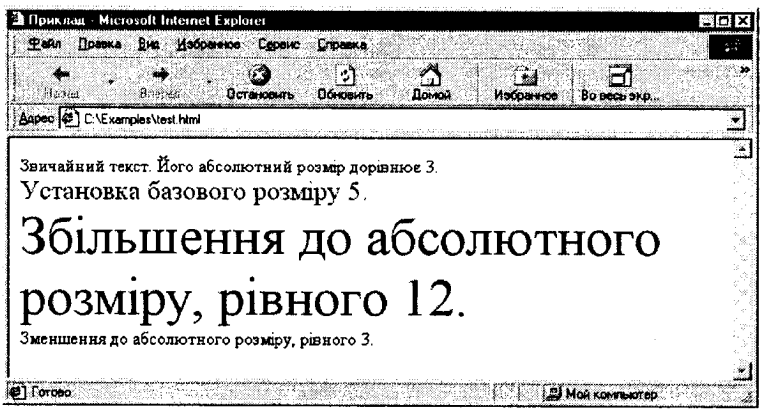


Рис. 3.16. Вигляд у браузері web-документа

Тег *<BASEFONT>* призначений для опису вихідних параметрів шрифту web-сторінки. Його дія поширюється до наступного тегу цього типу.

Цей тег має такі атрибути:

face="..." — задає ім'я шрифту чи кілька можливих шрифтів, які браузер використовує для відображення тексту. Оскільки шрифт береться з набору встановлених на комп'ютері клієнта шрифтів, рекомендується зазначити через кому декілька близьких за накресленням гарнітур, тому що немає гарантії, що обраний шрифт буде присутній на комп'ютері користувача. В іншому випадку браузер вибере встановлений за замовчуванням шрифт і, можливо, текст відображатиметься не так, як потрібно;

size="..." — задає розмір шрифту від 1 до 7. Базовий розмір шрифту дорівнює 3. Шрифти можуть бути задані щодо базового +1, −1, +2, −2, ...; +7, −7;

color="..." — задає колір шрифту. Значення даного параметра розглядалися раніше. Слід зазначити, що всі розглянуті атрибути притаманні і тегу *<FONT>*.

3.4. Списки в HTML

Теги списків — досить часто застосовувані при форматуванні елементи.

У HTML-документах можна задавати три види списків: упорядкований список (пронумерований); невпорядкований список (з графічними позначками); словник термінів.

Теги для задання списків:

<OL>...</OL> — для впорядкованого списку. Всі елементи списку перелічуються у визначеному порядку з приписуванням перед елементом тегу *<LI>*. При відображенні списку браузером його елементи автоматично нумеруються;

<UL>...</UL> — для невпорядкованого списку. При відображенні браузером цих списків використовуються спеціальні символи для виділення рядків списку;

<DL>...</DL> — для словника термінів. Це списки-визначення. Вони призначені для відображення списків термінів і їхніх визначень. Кожен елемент списку складається з двох частин: терміна і його визначення. Список у цілому має знаходитися всередині тегу *<DL>...</DL>*. Терміну передує тег *<DT>*, а визначенню — тег *<DD>*.

Приклад: *<DL>*

<DT><U>Молодь -</U>

<DD> — це надія нації

Перед кожним елементом списку необхідно ставити однокомандний тег *<LI>*.

Можна створювати вкладені списки, використовуючи різні теги списків чи повторюючи одні всередині інших.

Для цього необхідно розмістити одну пару тегів (стартовий і завершальний) усередині іншої. Чи матимуть елементи вкладеного списку ті ж маркери, що позначають елемент списку, — залежить від браузера.

Розглянемо більш докладно всі типи списків.

У нумерованому списку браузер автоматично вставляє номери елементів. Якщо видалити один чи кілька елементів нумерованого списку, то інші номери автоматично будуть перераховані.

Нумерований список починається стартовим тегом <ol> і завершується тегом . Кожен елемент списку починається з тегу <li>. Наприклад:

```
<ol>
<li> Яблука
<li> Груші
<li> Виноград
</ol>
```

Результат відображення в браузері наведено на рис. 3.17.

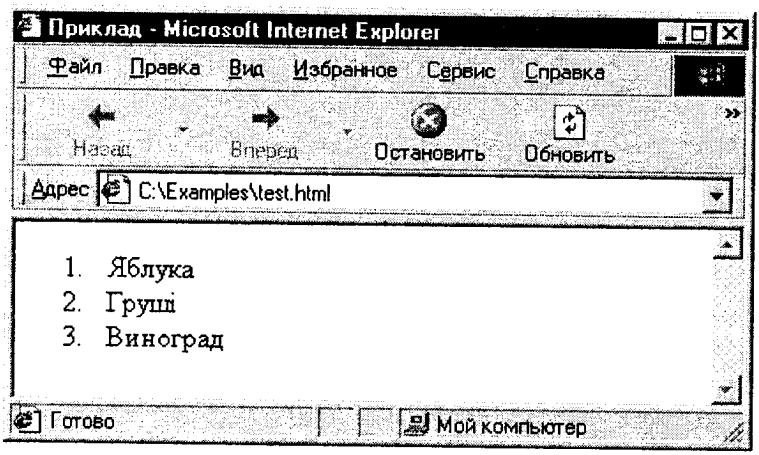


Рис. 3.17. Вигляд у браузері web-документа нумерованого (впорядкованого) списку

Тег <ol> може мати параметри:

<ol TYPE=A|a|I|i|1 START=n>

де *атрибут TYPE* задає вид нумерації:

A — великі латинські букви (*A,B,C...*);

a — маленькі латинські букви (*a,b,c...*);

I — великі римські цифри (*I,II,III...*);
i — маленькі римські цифри (*i,ii,iii...*);
1 — звичайні цифри (*1,2,3...*);
START=n — число, з якого починається відлік.

Наприклад:

```
<ol TYPE=I START=15>
<li> Яблука
<li> Груші
<li> Виноград
</ol>
```

Результат відображення в браузері наведено на рис. 3.18.

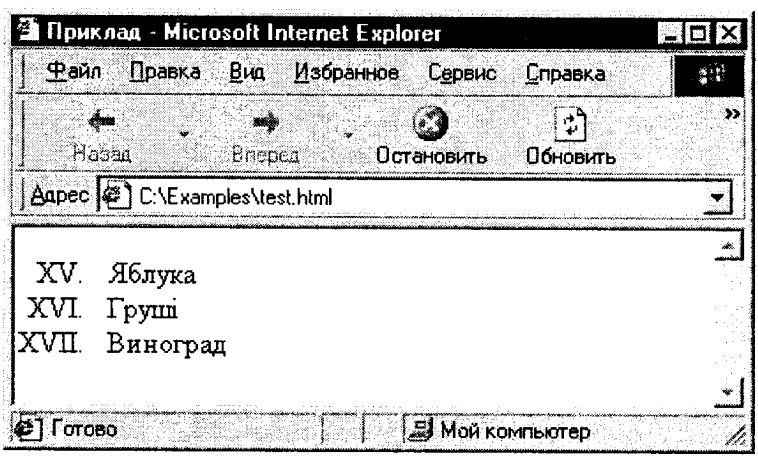


Рис. 3.18. Вигляд у браузері web-документа з нумерованим списком зі зміненим маркером

Для ненумерованих списків браузер звичайно використовує маркери для позначки елемента списку. Вигляд маркера, як правило, визначає користувач браузера в його опціях. Такий список починається стартовим тегом *<UL>* і завершується тегом *</UL>*. Так само, як і у випадку з нумерованим списком, кожен елемент списку починається з тегу *<li>*.

Наприклад:

```
<ul>
<li> Яблука
<li> Груші
<li> Виноград
</ul>
```

Результат відображення в браузері наведено на рис. 3.19.

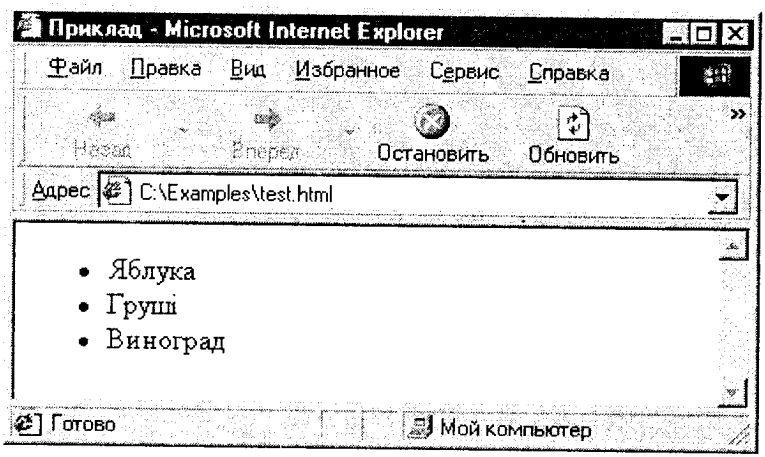


Рис. 3.19. Вигляд у браузері web-документа з ненумерованим (невпорядкованим) списком

Тег *<ul>* може мати параметр *TYPE:*

<ul TYPE=disc|circle|square>

Атрибут *TYPE* визначає зовнішній вигляд маркера за замовчуванням (*disc*), круглий (*circle*) чи квадратний (*square*). Наприклад:

```
<ul TYPE=square>
<li> Яблука
<li> Груші
<li> Виноград
</ul>

<ul TYPE=circle>
<li> Яблука
```

```
<li> Груші
<li> Виноград
</UL>

<UL TYPE=disc>
<li> Яблука
<li> Груші
<li> Виноград
</ul>
```

Результат відображення в браузері наведено на рис. 3.20.

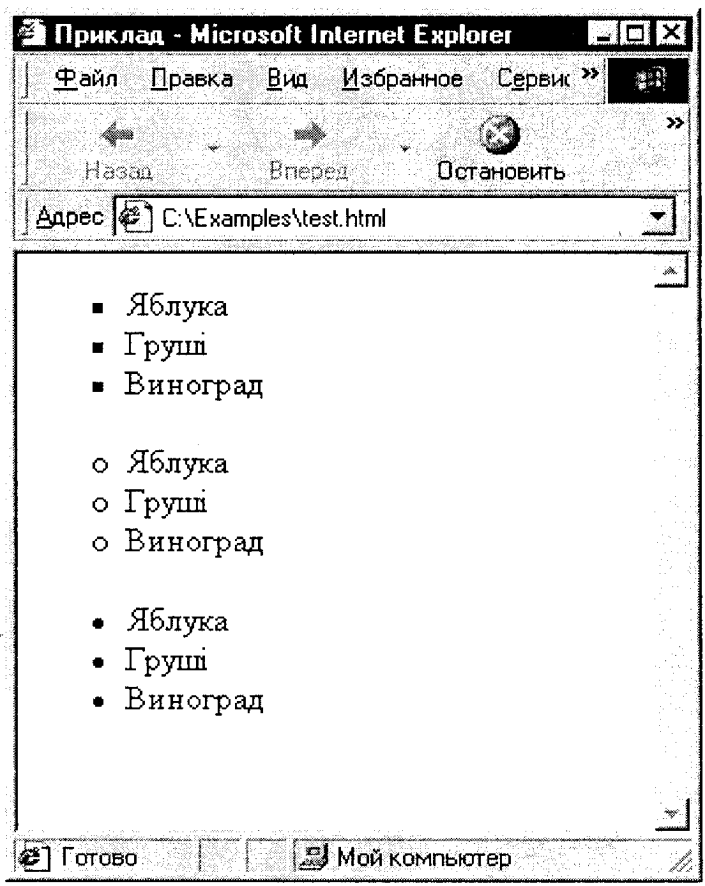


Рис. 3.20. Вигляд у браузері web-документа з різними маркерами

І, нарешті, останній тип списку — список описів, що використовується для кодування словників, переліку термінів і аркушів описів. Синтаксис його наступний:

```
<DL>
<DT>Обумовлене слово (набір слів)<DD>Опис цього слова (термінів)
...
</DL>
```

Браузер відобразить цю конструкцію в такий спосіб: спочатку вміст тегу *<DT>*, потім на новому рядку з відступом — вміст тегу *<DD>*. Теги *<DL></DL>* служать для позначення початку і кінця такого списку.

Розглянемо наступний приклад.

```
<DL>
<DT>CTR<DD>(синонім — клікабельність, від англ. click-through rate — показник клікабельності)<br>
CTR визначається як відношення числа кліків на банер до числа його показів, вимірюється у відсотках. Слово CTR вимовляється як "ci-mi-ap" чи, рідше, "це-те-ер". CTR є важливим показником ефективності роботи банера.
<DT>Rich media<DD>(річ медіа)<br>
Rich media дослівно перекладається з англійської мови як "багатий" чи "збагачений" засіб. Це нова модна технологія виготовлення рекламних матеріалів, що зазвичай використовує флеш і Java.
<DT>FAQ<DD>(англ. скор. Frequently Asked Questions — питання, що часто задаються)<br>
FAQ — розділ на сайті, присвячений розгляду типових питань користувачів. Якщо вам щось незрозуміло, перш ніж написати web-майстру, перегляньте FAQ — можливо, там уже є відповідь<br>
У більш широкому змісті FAQ — це добірка статей за тією чи іншою темою, наприклад щодо вибору телевізора чи програмування мовою PHP.
</DL>
```

Результат відображення в браузері наведено на рис. 3.21.

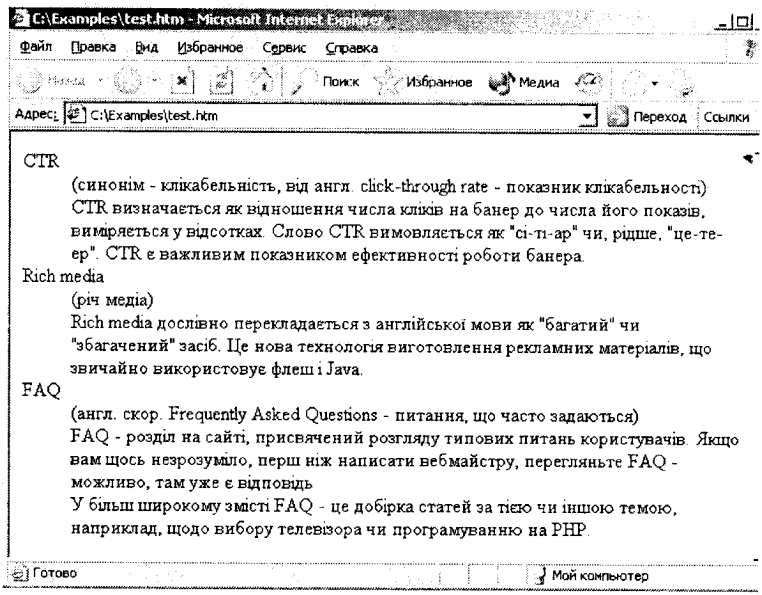


Рис. 3.21. Вигляд у браузері web-документа зі списком описів

3.5. Таблиці в HTML

Опис таблиць у HTML-документах здійснюється за допомогою контейнера *<TABLE>...</TABLE>*.

Таблиця задається двома контейнерами тегів:

<TR>...</TR> — опис рядка таблиці;

<TD>...</TD> — опис окремого елемента (клітинки) таблиці.

За замовчуванням текст усередині таблиці вирівнюється по лівому краю, а ширина стовпця таблиці визначається його найдовшим елементом.

Розглянемо приклад найпростішої таблиці (*див.* рис. 3.22):

```
<TABLE border="2"> <!-- Початок таблиці -->
 <TR>
  <TD colspan=2 align=center>
   <B>Заголовок Таблиці</B>
  </TD>
 </TR>
```

```
<TR>
 <TD align="center">
  Перша клітинка першого рядка.
 </TD>
 <TD align="center">
  Друга клітинка першого рядка.
 </TD>
</TR>
<TR>
 <td align="center">
  Перша клітинка другого рядка.
 </TD>
 <TD align="center">
  Друга клітинка другого рядка.
 </TD>
</TR>
</TABLE>      <!-- кінець таблиці -->
```

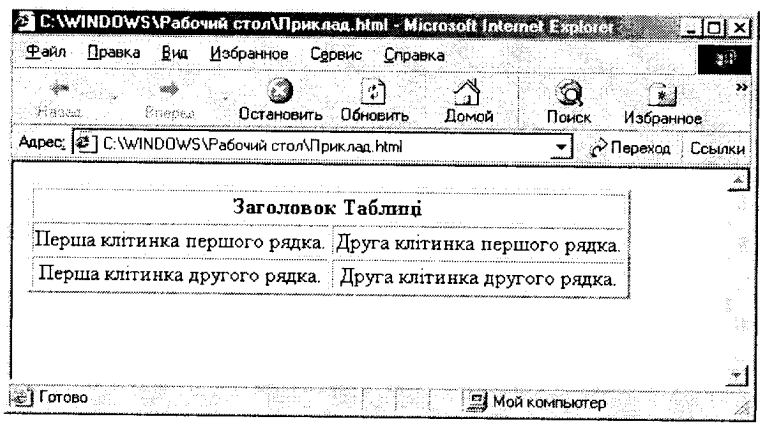


Рис. 3.22. Вигляд у браузері web-документа з таблицею

Ширина таблиці задається атрибутом *width="..."* для тегу *<TABLE>*, а значення ширини таблиці може задаватися в пікселах чи у відсотках від ширини екрана.

Вміст кожного елемента таблиці може бути вирівняний за допомогою атрибутів *align="..."* (горизонтальне розташування) і *valign="..."* (вертикальне розташування) для тегів *<TR>* чи *<TD>*.

Атрибут *valign="..."* може набувати таких значень:

top — притиснути вміст догори;

bottom — притиснути вниз;
middle — розмістити по центру.

Атрибут *align="..."* може набувати таких значень:
left — притиснути вліво;
right — притиснути праворуч;
center — розмістити по центру.

Атрибути *cellpadding="..."* і *cellspacing="..."* визначають, відповідно, відстань у пікселах між границею клітинки та її вмістом, а також між клітинками.

Допускається додавати до таблиці, рядка чи стовпчика заголовок.

Тег *<TH>*, що розташовується після тегу *<TABLE>*, задає заголовок до таблиці. Задання заголовка для рядка чи стовпця таблиці здійснюється за допомогою тегу *<TH>* після тегів *<TR>* чи *<TD>* відповідно.

Атрибут *border="..."* тегу *<TABLE>* створює рамку навколо таблиці і кожної клітинки, при цьому ширина рамки задається в пікселах.

Варто запам'ятати, що атрибути *colspan="..."* і *rowspan="..."* тегів *<TD>* і *<TR>* дозволяють об'єднувати клітинки таблиці в групи, навколо яких створюється рамка, а атрибут *nowrap* відключає розривання рядків.

Розглянемо приклад таблиці, що займає за шириною весь екран браузера:

```
<TABLE border="l" width="100%">
 <TR>
  <TD width="66%" colspan="2">
   <P align="center">
   Дві клітинки, об'єднані по горизонталі
  </TD>
 </TR>
 <TR>
  <TD width="33%" rowspan="2" valign="middle">
   Дві клітинки, об'єднані по вертикалі
  </TD>
  <TD width="33%">
   по лівому краю
  </TD>
 </TR>
```

```
<TR>
 <TD width="33%">
  <P align="right">
   по правому краю
 </TD>
</TR>
</TABLE>
```

У вікні браузера сторінка з наведеним вище кодом матиме вигляд, як показано на рис. 3.23.

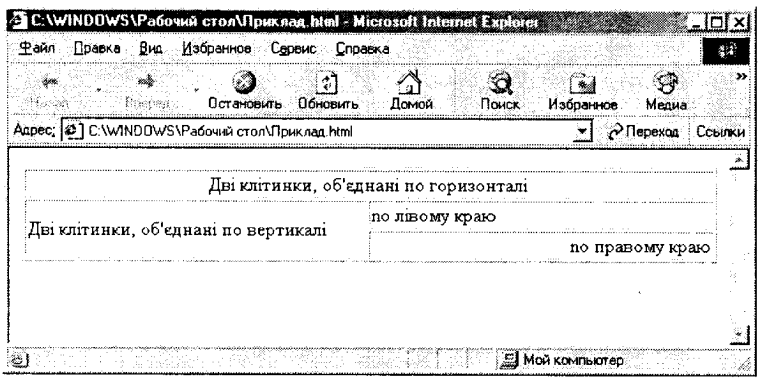


Рис. 3.23. Вигляд у браузері web-документа з таблицею, в якій наявні об'єднані клітинки

Розглянемо додаткові атрибути тегів *TABLE*, *TR* і *TD* :

bgcolor — колір фону таблиці (у форматі RGB);

background="url" — фонове графічне зображення (адреса);

bordercolor="колір" — колір рамки;

cellspacing — відстань між елементами таблиці в пікселах;

cellpadding — відстань між вмістом елемента і рамкою в пікселах;

width — ширина;

height — висота.

Розглянемо більш складну задачу з таблицями. Нехай необхідно помістити потрібну нам таблицю до клітинки іншої таблиці, при цьому колір фону цієї клітинки повинен збігатися з кольором рамки нашої таблиці.

Розв'язання задачі проілюструємо на прикладі таблиці з трьох рядків і чотирьох клітинок з рамкою червоного кольору завтовшки 1 піксел:

```
<table cellpadding="0" border="0">
<tr>
  <td bgcolor="red">
        <table cellspacing="1" border="0" width="200">
          <tr bgcolor="white">
<td> </td><td> </td><td> </td><td> </td>
    </tr>
          <tr bgcolor="white">
<td> </td><td> </td><td> </td><td> </td>
     </tr>
          <tr bgcolor="white">
<td> </td><td> </td><td> </td><td> </td>
    </tr>
          </table>
    </td>
 </tr>
</table>
```

Відображення цього коду в браузері подане на рис. 3.24.

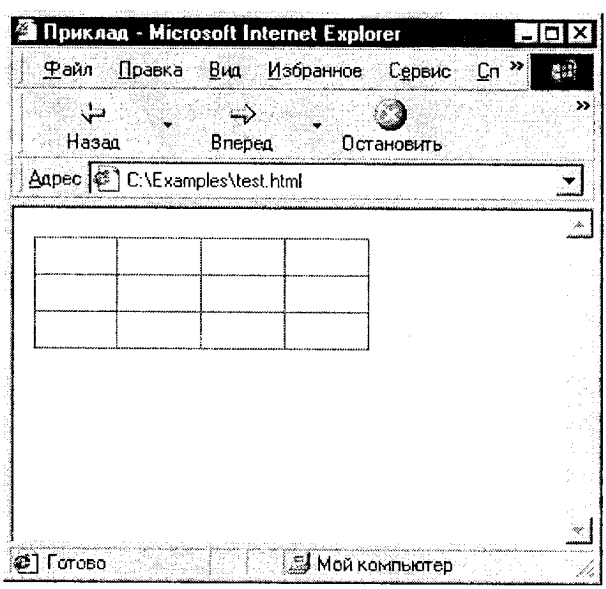


Рис. 3.24. Вигляд у браузері web-документа з таблицею 3×4

Зверніть увагу на параметр *cellspacing* у другій таблиці: змінюючи його значення, можна добиватися будь-якої товщини рамки. Якщо ж значення параметра *cellpadding* більше нуля, одержимо таблицю, зовнішня рамка якої буде товстіша за внутрішні лінії.

Однак і це ще не все. Змінюючи колір фону рядків внутрішньої таблиці, можна досягти ефекту «смугастості», що особливо зручно для подання великих таблиць з довідковою інформацією (*див.* рис. 3.25).

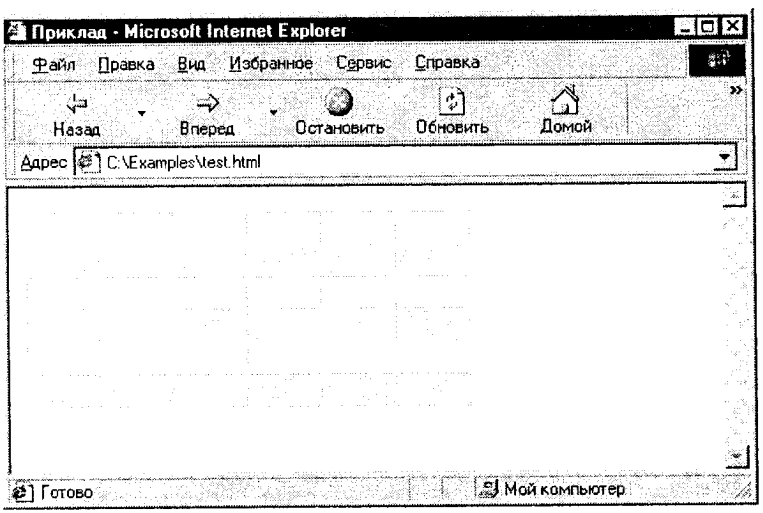


Рис. 3.25. Вигляд у браузері web-документа з таблицею з ефектом «смугастості»

Цікавий ефект можна одержати, якщо зазначити в атрибутах таблиці та її клітинок подані нижче параметри і не використовувати зовнішньої таблиці:

```
<table cellspacing="1" border="0" width="200">
<tr bgcolor="#c0c0c0" align="center">
<td>1</td><td>2</td><td>3</td><td>4</td>
</tr>
<tr bgcolor="#dadada">
  <td> </td><td> </td><td> </td><td> </td>
</tr>
<tr bgcolor="#dadada">
```

```
  <td> </td><td> </td><td> </td><td> </td>
</tr>
<tr bgcolor="#dadada">
  <td> </td><td> </td><td> </td><td> </td>
</tr>
<tr bgcolor="#dadada">
  <td> </td><td> </td><td> </td><td> </td>
</tr>
</table>
```

Відображення цього коду в браузері подане на рис. 3.26.

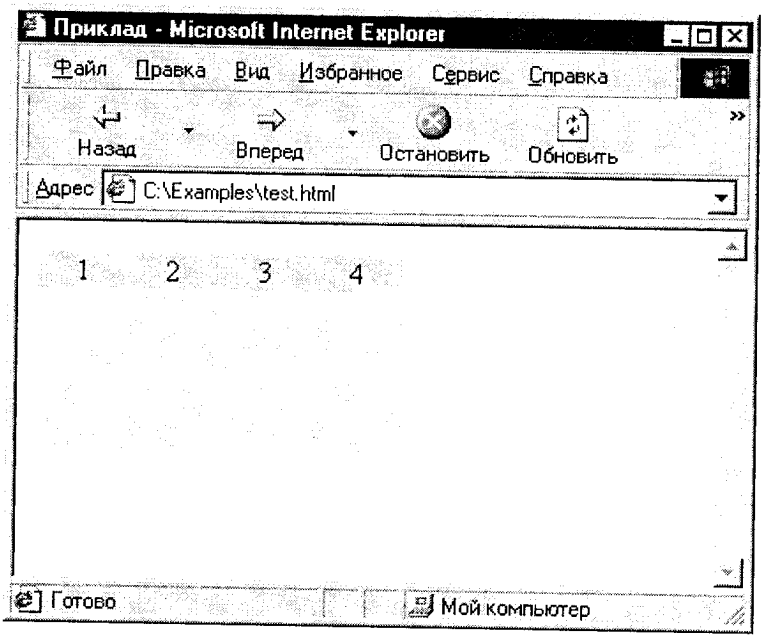


Рис. 3.26. Вигляд у браузері таблиці з різним фоном

3.6. Якірні теги. Гіперпосилання. Маршрутизатори

Один з основних тегів, без якого, мабуть, неможливо було б користуватися WWW і без якого сама технологія гіпертексту втратила би зміст, — це тег гіперпосилання.

Текст, що є гіперпосиланням, як правило, виділяється кольором і підкреслюється.

Гіпертекстові посилання (*гіперзв'язки*) — це вказівники на визначене місце в поточному чи іншому документі, чи у зовнішньому графічному, відео- або аудіофайлі.

Посилання на інформаційні ресурси мережі Інтернет здійснюються за допомогою URL (*див.* Розділ 2). Нагадаємо, що вони починаються зі схеми доступу, необхідної для визначення протоколу. Наприклад,

http://, ftp://, mailto://, file://

Після схеми доступу вказується адреса вузла (комп'ютера) в мережі та ім'я файла ресурсу. Наприклад:

file://host/directory/filename — ідентифікує конкретний файл;

ftp://user:password@host:port/directory/filename — FTP-сервер;

gopher://host:poit/gopher-path — Gopher-сервер і пункт меню;

http://host:port/directory/filename?searchpart — сервер WWW;

mailto:user@host — адреса електронної пошти;

telnet://user:password@host:port/ — TELNET-сервер.

Гіпертекстові зв'язки вводяться в HTML-документ за допомогою *якірного дескриптора* <A>..</A>.

Для вбудовування гіперзв'язків використовується загальний синтаксис:

<A HREF="адреса_ресурсу посилання">
виділений_текст_посилання-маршрутизатор </A>,

де *адреса_ресурсу* — ім'я поточного документа, шлях до іншого файла або ж Інтернет-адреса ресурсу (файла, документа тощо), з котрим встановлюється гіперзв'язок;

виділений_текст_посилання (чи маршрутизатор) — текст або інший об'єкт, при натисканні на який мишею виконується перехід за адресою, зазначеною в *адресі_ресурсу* посилання.

Рекомендується укладати значення атрибута *адреса_ресурсу* в лапки (одинарні або подвійні). Рядок у лапках не повинен містити таких самих лапок усередині себе. Так, якщо дата поміщена в подвійні лапки, то використовуються одинарні лапки для наступного виразу в лапках, і навпаки.

Можна також не укладати в лапки значення атрибутів, що містять тільки символи англійського алфавіту (A — Z, a — z), цифри (0 — 9), проміжки часу, дефіси (-).

Існують три основних типи гіперзв'язків.

1. Посилання на інше місце всередині одного документа:

```
<A HREF ="#мітка ">маршрутизатор </A>
                    ...

<A NAME="мітка"> </A>
                ...  <!- у це місце буде здійснено перехід
```

Як *мітку* можна використовувати будь-які цифри та символи латиниці, крім спеціальних.

2. Посилання на інший Інтернет-документ (файл):

```
<A>HREF="http://www.colordyn.com/>
кольородинамічні проектори
</A>
```

При натисканні мишею в тексті web-документа на виділений текст кольородинамічні проектори відбувається автоматичний перехід у мережі на сторінку, зазначену в адресі ресурсу.

3. Посилання на визначене місце іншого документа (файла):

```
<BODY>Для одержання даних про проектори клацніть
<A  HREF="="http://www.colordyn.com /video.htm#мітка">
тут </A>
   Текст документа
      ...
 </BODY>
```

Текст тут — визначає гіперзв'язок, що вказує на те, що перехід буде здійснено у визначене місце у файлі *video.htm*. При цьому у файлі *video.htm* має бути дескриптор <A NAME="мітка"></A> .

Наведемо приклад гіперпосилання:

```
<A HREF="filename"
          target="_self">
               текст посилання
</A>
```

Результат відображення в браузері наведено на рис. 3.27.

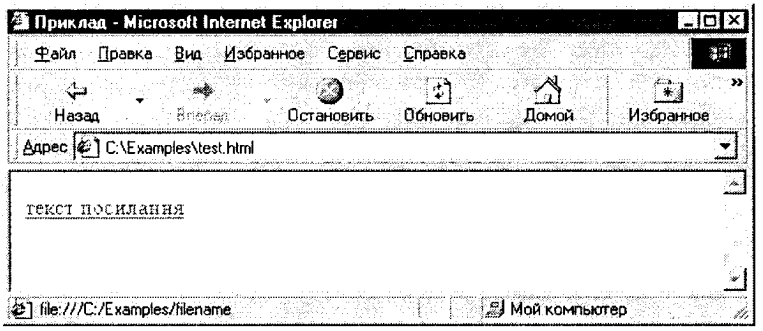


Рис. 3.27. Вигляд у браузері гіперпосилання

Як бачимо з прикладу, з'явився атрибут *TARGET*. Він задає значення вікна чи фрейму, у якому буде відкрито документ, на який вказує посилання. Атрибут *TARGET* можна використовувати там, де є будь-яка переадресація, тобто не тільки з якірними тегами.

Можливі значення атрибута:

_top — відкриття документа в поточному вікні;

_blank — відкриття документа в новому вікні;

_self — відкриття документа в поточному фреймі (про фрейми йдеться далі);

_parent — відкриття документа в батьківському фреймі.

Значення за замовчуванням: *_self* .

Наведемо інші приклади гіперпосилань:

1. *<A href="photo/my_photo.html"target="_blank">Мій фотоальбом</A>* — посилається на файл my_photo.html, розташований у каталозі photo, вкладеному в поточну папку, й утворює посилання у вигляді тексту «Мій фотоальбом». Документ відкриється в новому вікні.

2. *<A href="aboutme.html" target=MainFrame >Трохи про мене</A>* — відкриє сторінку aboutme.html, розташовану в поточній папці, й утворить посилання у вигляді тексту «Трохи про мене». Документ відкриється у фреймі «MainFrame».

3. *<A href="mailto:support@hotmail.com">Лист до технічної підтримки</A>* — виконує макрокоманду завантаження поштової програми з автозаповненням адреси одержувача.

Існує й інший тег для виконання переадресації ресурсів. Це тег *LINK*.

Тег LINK надає документові незалежний від середовища метод визначення відношення даного документа до інших документів і ресурсів мережі. Використовується з аргументами *REL* і *REV*. За допомогою тегу *LINK* можна:

- створювати в документі спеціальні навігаційні кнопки або меню;
- прив'язувати такі асоційовані ресурси, як таблиці стилів і скрипти;
- надавати альтернативні форми для даного документа.

Наприклад:

<LINK rel=help href=»http://www.name.com/help.html»>,

де *http://www.name.com/help.html* — сторінка допомоги для даного документа.

Атрибути *REL* і *REV* указують на тип зв'язку – зв'язок «вперед» або зв'язок «назад» і можуть також використовуватися з тегом *<A>*.

Теги *LINK* можуть використовуватися тільки в заголовку документа (*HEAD*).

3.7. Графічні зображення. Зображення-карти. Вставка об'єктів та аплетів

Розглянемо, які існують способи відображення графічної інформації в Інтернет та як впроваджувати до web-документів графічні об'єкти (*див.* також Додатки 2 та 3).

Існують десятки форматів подання графічної інформації, кожний з який відрізняється якістю передачі цієї інформації, орієнтацією на визначені апаратні ресурси і платформи, роздільним представленням елементів картинки, можливістю кодування додаткової інформації (наприклад, про авторство), а також наявністю специфічних можливостей.

Графічну інформацію можна представити одним із двох способів: у *растровому* чи *у векторному* вигляді.

У першому випадку у файлі у деякий спосіб задається інформація про кожну точку зображення, у другому — задається інформація про об'єкти.

У найпростішому прикладі це має такий вигляд: припустимо, у нас є зображення зеленого кола на жовтому фоні. У растровому форматі у файл буде записана інформація типу «жовта точка,

жовта точка, зелена точка...» і т.п. або у випадку з найпростішим алгоритмом стискання — «дві жовтих точки, одна зелена точка...». Таким чином будуть перераховані всі точки. Чим більший розмір зображення — тим більшим буде файл на диску.

У векторному форматі для нашого прикладу буде задано приблизно таке: «Фон — жовтий, коло з центром у точці (x, y) і радіусом R, заповнене зеленим кольором». Звичайно, на практиці усе складніше, але загальна ідея повинна бути зрозуміла.

Для відображення графіки в мережі Інтернет застосовуються растрові формати, тому докладніше зупинимося саме на них. Не перший раз згадуване обмеження на обсяг і швидкість отримання інформації, що завантажується з мережі, діє і в цьому разі. Прагнення мінімізувати розмір інформації, що зберігається і завантажується призвело до розробки алгоритмів стискання графіки.

Найбільш розповсюдженими в Інтернет форматами графічних файлів є GIF (*Graphics Interchange Format*) і JPEG (за назвою розробника — *Photographic Experts Group*), що дозволяють стискати зображення з утратою якості. Останню можна регулювати кількістю кольорів у картинці — для GIF (максимум — 256 кольорів), а також ступенем стискання — для JPEG (максимум — 100 %).

Таким чином, якщо стискається зображення з палітрою до 256 кольорів, його можна без утрати якості стиснути за допомогою алгоритму GIF. Існують також різновиди формату PNG (*Portable Network Graphics Format*), однак на цей час у мережі Інтернет він застосовується досить рідко через неповну підтримку браузерами.

Проте основна перевага форматів, що застосовують ті чи інші методи стискання (особливо з утратою якості), стає недоліком у процесі первинної обробки графічної інформації і роботи з зображеннями. Чим більшу кількість разів зберігати файл в одному з таких форматів, тим більше інформації про зображення буде загублено, отже якість погіршуватиметься з кожною ітерацією. Тому як робочі слід застосовувати формати, що забезпечують збереження якості, наприклад TIFF, BMP, PSD, і лише при остаточній підготовці графіки для web-публікації конвертувати в призначені для цього формати.

Для вбудовування посилань на графічні файли використовується тег

<IMG SRC = "ім'я.файла" ALT = "текстовий опис" ALIGN ="вирівнювання">,

де *IMG* — є ім'ям тегу визначення посилання на графічний файл, шлях пошуку якого визначається значенням опції *SRC=*.

Необов'язковий атрибут *ALT=* містить текстовий опис, що за замовчуванням відображається під картинкою. Необов'язкова опція *ALIGN=* призначена для задання місця розташування виведеного текстового опису образу і може набувати значення: *BOTTOM* (знизу), *TOP* (зверху), *MIDDLE* (посередині). Можна визначити графічний образ усередині дескриптора, що задає гіперзв'язок.

Підключити до web-сторінки нетекстові ресурси можна таким чином:

<IMG src="URL">

Атрибути:

alt="текстовий опис зображення";

align=top || middle || bottom — місце розташування зображення.

Для підключення аудіо- і відеофайлів застосовується якірний тег:

<A href="URL"> текст </A>

Зображення також можна використовувати як маршрутизатор гіперпосилання:

<A href="URL" ><IMG src="URL"></A>.

У документах HTML крім тексту можуть бути присутні графічні зображення, для вставки яких використовують тег *<IMG>*.

Допускається використання файлів у форматі GIF, PNG чи JPEG, оскільки більшість браузерів мають інтегровані модулі декодування для відтворення даних форматів, а для систем, що працюють під керуванням операційних систем сімейства Windows, допускається використання файлів формату BMP. Наступний приклад демонструє вставку в документ JPG-файла):

<IMG src=logo.jpg width="542" height="407" alt="Microsoft">

Тут атрибут *src=" "* визначає URL-адресу графічного файла.

У наведеному прикладі файл буде розміщений в області шириною 542 і висотою 407 пікселів відповідно. Можна задавати відносні розміри картинки — у відсотках. Якщо розміри, зазначені атрибутами *height=" "* (висота) і *width=" "* (ширина), не збігаються з розмірами графічного файла, то останній масштабується. Масштабування може призвести до різкого погіршення якості графічного файла, тому рекомендується задавати розміри рисунку, що відповідають розмірам графічного файла, чи не вказувати їх взагалі. Проте для великих графічних файлів рекомендується завжди задавати їхні розміри для прискорення роботи браузера. Якщо розміри не задані, то, зустрівши рисунок, браузер припиняє зчитувати сторінку і чекає завантаження всього малюнка для того, щоб визначити його розміри, а це затримує відображення сторінки в цілому.

Приклад вставки зображення:

```
<HTML>
 <HEAD>
 <TITLE>Вставка зображення </TITLE>
 </HEAD>
 <BODY>

 <IMG src=logo.jpg width="542" height="407"
   alt=»Microsoft">

 </BODY>
</HTML>
```

Атрибут *alt="..."* вказує браузеру на те, який саме текст слід підставити на місце зображення, якщо користувач відключив завантаження графічних файлів чи через розрив з'єднання файл не було завантажено.

Зображення можна так само використовувати в якості *маршрутизатора* гіперпосилання.

Наприклад:

```
<A HREF="index.html">
<IMG SRC="snail.gif" border="0" alt="На головну сторінку"
  height="84" width="92">
</A>
```

За замовчуванням браузер рисує рамку (бордер) навколо зображення, яке визначене як гіперпосилання. Для того, щоб не

показувати рамку, використовують атрибут *border="..."* у тегові *<IMG>* з нульовим значенням.

Розглянемо поняття *графічних карт*. У гіперпосиланнях як маршрутизатор можна використовувати область вбудованого в HTML-документ графічного зображення.

У HTML є можливість призначити одному рисунку кілька активних областей, кожна з яких буде зв'язана з визначеною адресою (сторінки, сайта). Для цього призначений тег *MAP*.

Можна, наприклад, створити графічне меню з однієї великої картинки таким чином, щоб кожен елемент системи меню містив визначений URL.

Зображення-карти — це графічні об'єкти у web-документі, окремі частини яких є маршрутизаторами гіперпосилання (чутливі області).

Зображення-карти задаються за допомогою тегу *MAP*, який рекомендується поміщати в розділ *HEAD*:

```
<MAP name="myname">
      <AREA   type=" "
              shape="  "
              coords=" "
              href="url / nohref">
</MAP>
```

Для того, щоб задати чутливу область, використовують тег *AREA*, що розташовується між тегами *<MAP>* і *</MAP>*.

Атрибути тегу *AREA*:

shape — форма області (*rect* — прямокутник; *circle* — коло; *poly* — багатокутник);

href — URL посилання при клацанні на області мишею;

nohref — задання неактивної області (при клацанні на ній мишею переходу за посиланням не відбувається);

coords — координати області. Можуть набувати таких значень:

"x1, y1, x2, y2" — для прямокутника (*rect*),

"x, y, r" — для кола (*circle*),

"x1, y1, x2, y2, ..., xn, yn, x1, y1" — для багатокутника (*poly*).

Розподіл посилань по картинці описується в тегові IMG (у тому місці, де поміщається карта) параметром

<IMG SRC="url" USEMAP="url#myname">,

де *USEMAP* задає розташування *myname*.

Якщо URL не задано, то пошук *myname* ведеться в поточному документі. Наприклад:

```
<MAP NAME="myname">
<AREA SHAPE=RECT COORDS="0,0,20,20" HREF="my1.htm">
  <AREA SHAPE=RECT COORDS="40,0,60,20" HREF="my2.htm">
 </MAP>
```

У даному випадку одна з активних областей має форму прямокутника і посилається на файл *my1.htm*.

Атрибут *COORDS* задає координати області в пікселах. Відлік починається з нуля. Коло має три координати, прямокутник — чотири, а для багатокутника необхідно описати кожен його кут у двох координатах.

Якщо область має форму прямокутника, наводять дві пари координат (верхніх лівих і нижнього правого кутів) прямокутника по осях *x* і *y* у вигляді *"x,y,x,y"*.

Якщо область має форму окружності, наводять координати вигляду *"x,y,r"*, де *x*, *y* — координати центра окружності, а *r* — її радіус.

Якщо область має вигляд багатокутника, то її координати задаються рядком типу *"x,y,x1,y1,x...,y..."*, тобто кожна вершина багатокутника описується своєю координатою.

Остання пара координат автоматично замикається на першій. Для зв'язку карти з рисунком карта вміщується в розділ HEAD, а для рисунка вказується:

```
<timg src="pic.gif" border=0 usemap=#myname> .
```

Тепер при клацанні мишею в координатах першого квадрата завантажиться сторінка *my1.htm*, а в межах другого квадрата — *my2.htm*. Всі інші області рисунка виявляться незадіяними.

Над активною областю рисунка курсор набуває форми руки, як над звичайним посиланням.

Приклад карти:

```
<BODY>
<IMG SRC="view.jpg" ALT="Клацніть тут" USEMAP="#map">
...
<MAP NAME="map">
<AREA SHARE="RECT" COORDS=0,0,64,64
                HREF="http://www.kture.kharkov.ua/">
 </MAP>
</BODY>
```

Окрім графічних об'єктів до web-документа можна впроваджувати інші об'єкти та аплети мовою Java.

Для впровадження *об'єктів* застосовується контейнер *<OBJECT>*. Він призначений для впровадження аплетів на JAVA, Active і об'єктів типу *MIME* (*див.* Додаток 1).

Синтаксис:

```
<object параметри>
        <param атрибути...>
                ...
</object>
```

Усередині *<object>* знаходяться теги

```
<param  name=ім'я властивості об'єкта
        value=значення, присвоєне властивості об'єкта>.
```

Вони дозволяють передати об'єкту деякі значення.

Приклад: `<param name="pole1" value="12345">`

Припустимі параметри тегу *object*:

id — ім'я об'єкта в документі;

data — адреса ресурсу, звідкіля об'єкт повинен одержати дані;

type — визначає MIME-тип об'єкта;

align — задає вирівнювання;

classid="CLSID: 1234-xxxx-xx...". Клас *ID* визначає унікальний ідентифікатор, що присвоюється об'єкту при створенні;

codebase — визначає адресу, звідкіля був отриманий об'єкт (де знаходиться код об'єкта);

codetype — визначає тип носія Інтернет-коду (необов'язковий);

height та *width* — висота та ширина в пікселах вікна для заданого об'єкта відповідно;

name — визначає ім'я для об'єкта, якщо він є частиною формуляра;

border — ширина рамки в пікселах.

Розглянемо питання застосування аплетів мовою Java у web-сторінках. Вставка аплета забезпечується тегом *APPLET*:

```
<APPLET>...</APPLET>
```

Код аплета завантажується окремо від сторінки, тобто це автономно відкомпільований код. Аплет перебуває у файлі з розширенням *.class*. До аплета можна передавати параметри й одержувати параметри від нього.

Атрибути тегу *Applet:*

```
<APPLET code="ім'я файла-аплета"
   codebase='url аплета'
   alt='альтернативний_текст' // якщо аплет не може завантажитися
   name='ім'я'
   width=
   height=
   align= // вирівнювання
   vspace= // вертикальні рамки навколо вікна аплета
   hspace= // горизонтальні рамки навколо вікна аплета
   object= // задає ім'я файла, що містить серійний аплет —
//аплет, що ініціалізований, але не запущений на виконання
   archive= // список архівних файлів .jar, що потрібно попередньо
//завантажити до браузера. У цих архівах можуть зберігатися
//будь-які класи, аплети, що можуть використовуватися аплетом.>
</APPLET>
```

Аплет може бути порожнім, або в ньому можуть бути списки параметрів, що передаються до аплета:

```
<PARAM name=ім'я_змінної' value='значення'>
```

Наприклад:

```
<applet code='1.class'>
<param name='myfio' value='Petrenko'>
</applet'>
```

3.8. Формуляри

Форма (формуляр) — це інструмент, за допомогою якого HTML-документ може надіслати деяку інформацію в заздалегідь визначену точку мережі, де інформація буде оброблена. Програми, що обробляють дані, передані формами, називають *CGI-скриптами* (*див.* Розділ 2).

Форми передають інформацію програмам-оброблювачам у вигляді пар:

[ім'я змінної]=[значення змінної].

Імена змінних необхідно задавати латинськими літерами. Значення змінних сприймаються оброблювачами як рядки, навіть якщо вони містять тільки цифри.

Форма відкривається тегом <*FORM*> і закінчується </*FORM*>. web-документ може містити в собі кілька форм, однак форми не повинні знаходитися одна всередині іншої. HTML-текст, включаючи мітки, може розміщатися всередині форм без обмежень.

У тега *FORM* є такі атрибути:

ACTION — обов'язковий атрибут. Визначає, де знаходиться оброблювач форми. Наприклад,

```
<FORM METHOD="POST" ACTION="/cgi-bin/data">;
```

METHOD — визначає, яким чином (за допомогою якого методу протоколу передачі гіпертексту) дані з форми будуть передані до оброблювача. Допустимі значення: *METHOD=POST* і *METHOD=GET*. Якщо значення атрибута не встановлено, за замовчуванням передбачається *METHOD=GET*;

ENCTYPE — визначає, яким чином дані з форми будуть закодовані для передачі до оброблювача. Якщо значення атрибута не встановлено, за замовчуванням передбачається *ENCTYPE=application/x-www-form-urlencoded*.

Форма складається з одного або декількох *елементів управління* — елементів інтерфейсу, що дозволяють користувачеві взаємодіяти з HTML-документами.

Тег, пов'язаний з елементами управління, — *INPUT*.

Розглянемо різновиди елементів управління *INPUT*. Вони визначаються значенням атрибута *TYPE*.

Розглянемо елемент управління «*Текстове поле*».

```
<INPUT
        TYPE="text" NAME="var-name"
        VALUE="рядок за замовчуванням"
        SIZE="розмір"
        MAXLENGTH="макс_довжина">.
```

Опція *TYPE* дозволяє вказувати, який саме тип керуючого елемента описується. Кожен тип має своє унікальне ім'я.

Опція *NAME* вказує ім'я змінної, у якій система зберігатиме отримане значення. Ім'я не має містити пробілів.

Опція *VALUE* (необов'язкова) дозволяє визначити текст, що поміщається в рядок введення при відображенні форми.

Опція *SIZE* визначає максимальне число символів, які може побачити при введенні користувач. Ця опція працює разом

з опцією *MAXLENGTH*, яка визначає кількість символів, що вводяться користувачем. Якщо значення *MAXLENGTH* більше за значення *SIZE*, і якщо користувач уведе більше символів, ніж зазначено в *SIZE*, то текст прокручуватиметься по горизонталі.

Наведемо приклад форми з одним рядком вводу:

```
<HTML>
<HEAD> <TITLE> Це приклад </TITLE> </HEAD>
<BODY>
<FORM
      METHOD="POST"
      ACTION="/cgi-bin/data">
Введіть ваші дані:
<INPUT
        TYPE="text"
      NAME="username"
      VALUE="Петренко О.В."
      SIZE="25"
        MAXLENGTH="45">
</FORM> </BODY> </HTML>
```

Результат наведено на рис. 3.28.

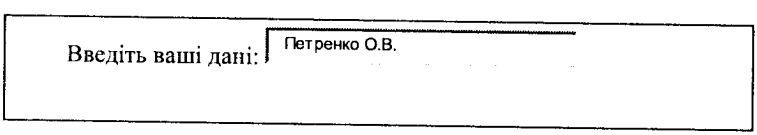


Рис. 3.28. Відображення в браузері рядка вводу

Уведення пароля. Рядок пароля забезпечує визначений захист даних, оскільки він заміняє символи, що вводяться до нього, зірочками. Визначення рядка введення пароля дуже схоже на визначення рядка введення тексту, за винятком значення опції *TYPE*.

Стандартне визначення рядка введення пароля має такий вигляд:

```
<INPUT TYPE="password"
      NAME="var-name"
      VALUE="рядок за замовчуванням"
      SIZE="25"
       MAXLENGTH="15">
```

Опція *NAME* вказує ім'я змінної, у якій зберігатимуться введені дані.

Опція *SIZE* визначає максимальне число символів, які може побачити користувач при введенні. Ця опція працює разом з опцією *MAXLENGTH*, яка визначає кількість символів, що вводяться користувачем.

Перемикачі-радіокнопки дуже схожі на перемикачі, з тією різницею, що одночасно може бути обрана тільки одна кнопка з групи. Для опису кнопок використовується тег *<INPUT>* з опцією *TYPE="radio"*.

Опція *VALUE* застосовується для того, щоб вказати значення, що повертається до програми серверної сторони. Кожен тег *<INPUT>* визначає окрему кнопку. Елементи, що мають однакове значення опції *NAME*, поєднуються в групу. Опція *VALUE* є обов'язковою. Вона описує значення, що повертається. За замовчуванням жодна з кнопок не є позначеною. Можна вказати опцію *CHECKED* для однієї з кнопок, щоб зробити її позначеною за замовчуванням:

```
<FORM METHOD="POST"
    ACTION="/cgi-bin/data">
Виберіть колір:
<P> <INPUT TYPE="radio"
           NAME="var-name"
           VALUE="red" CHECKED> Червоний
    <INPUT TYPE="radio"
           NAME="var-name"
           VALUE="grn"> Зелений
    <INPUT TYPE="radio"
           NAME="var-name"
           VALUE="blu"> Синій
</FORM>
```

У результаті радіокнопки у вікні браузера матимуть такий вигляд (рис. 3.29):

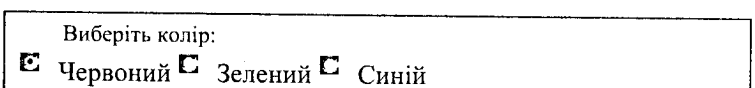


Рис. 3.29. Радіокнопки у браузері

Опцію *CHECKED* можна вказувати тільки для однієї кнопки в кожній групі.

У таблиці 3.1 наведені *елементи управління*, які можна використовувати на формі.

Таблиця 3.1

Елементи управління формою

Тип	Опис
checkbox	Керуючий елемент — перемикач (прапорець перевірки)
hidden	Прихований текст, використовують для обміну даними між формами
password	Рядок вводу пароля
reset	Кнопка «Reset»
submit	Кнопка «Submit»
text	Рядок для введення тексту

Уведення декількох рядків тексту. Якщо потрібно ввести великий текст, то використовується область для введення тексту. Вона визначається тегом

```
<TEXTAREA
      NAME="var-name"
      ROWS="4"
      COLS="80">
Текст, за замовчуванням, що відображається в області введення
</TEXTAREA>
```

Опція *NAME* вказує ім'я змінної, у якій зберігатимуться отримані дані, *ROWS* і *COLS* визначають розмір області в рядках і в стовпцях відповідно.

Текст, що відображатиметься в області введення за замовчуванням, повинний знаходитися між початковим тегом *<TEXTAREA>* і завершальним *</TEXTAREA>*.

Приклад форми, що запитує в користувача кілька рядків тексту:

```
<TEXTAREA NAME="MyText" ROWS=8 COLS=80>
Текст, відображуваний за замовчуванням
</TEXTAREA>
```

Результат наведено на рис. 3.30: на екрані відобразиться поле вводу, що містить текст, заданий за замовчуванням, і яке має лінійки вертикального та горизонтального скролінгу.

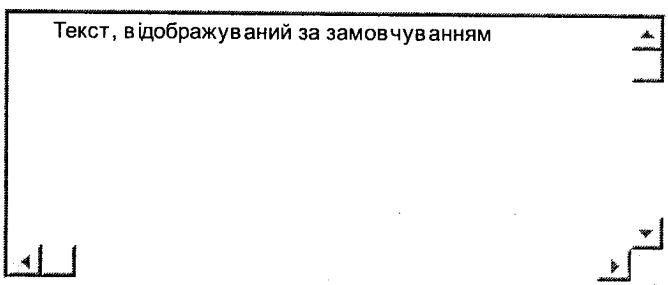


Рис. 3.30. Відображення багаторядкового вікна

Окрім елементів управління на формі можна розмістити *спливаючі меню*. Воно дає можливість користувачеві вибрати з меню один або кілька елементів. Меню визначається всередині тегу *<SELECT>...</SELECT>*.

Кожен пункт меню починається тегом <OPTION>. Зазвичай перший у списку елемент є елементом за замовчуванням, однак можна змінити елемент за замовчуванням, вказавши опцію *SELECTED*.

```
<FORM
      METHOD="POST"
      ACTION="/cgi-bin/data">
 Виберіть елемент:
<SELECT
NAME="var-name">
      <OPTION>Перший
      <OPTION>Перший
      <OPTION>Третій
      <OPTION SELECTED>Четвертий
      <OPTION>П'ятий
      </SELECT>
</FORM>
```

3.9. Фрейми

При використанні фреймової структури вікно браузера розділяється на підвікна, у кожному з яких можна відображати довільні документи мовою HTML, організувати незалежне прокручування їхнього вмісту і змінювати розміри фреймів у межах вікна браузера.

Гіпертекстові посилання, що присутні в документі, розташованому в одному з підвікон, можуть бути використані для відновлення вмісту інших фреймах.

У фреймутримуючих документах містяться дані, що визначають структуру поділу вікна браузера на підвікна. У цих документах немає інформації, яку необхідно помістити в конкретні фрейми, а тільки посилання на звичайні гіпертекстові документи мовою HTML.

У загальному випадку структура фреймутримуючого документа може мати такий вигляд:

```
<HTML>
<HEAD> ... </HEAD>
<FRAMESET  COLS="30%, 70%">
        <FRAME SRC=frame1.html>
        <FRAME SRC=frame2.html>
 <NOFRAMES>
Задання інформації для браузера, що не підтримує фрейми.
 </NOFRAMES>
 </FRAMESET>
</HTML>
```

Це структура файла-фреймоутримувача. Фрейм-документ є специфічним HTML-файлом; не має тегу *body*.

Тег *frameset* вказує, на скільки рядків або стовпців-фреймів має розділяти вікно браузера.

У даному прикладі вікно браузера буде розділено на два фрейми, розташованих один під іншим, при цьому висота першого складе 30 % висоти вікна, а другого — 70 %.

ftame1.html і *frame2.html* є гіпертекстовими документами. Тег *frameset* описує або вертикальне розташування фреймів, або ж горизонтальне:

- *<FRAMESET rows=[рядки]>* — при горизонтальному розташуванні фреймів;

<FRAMESET cols=[стовпці]> — при вертикальному розташуванні фреймів.

Існує декілька способів задання розмірів фрейма: попіксельний; із використанням символів * та % .

*Приклад: <frameset cols=»100, *»>* — поділяє екран на два фрейми: перший — шириною 100 px (пікселів), а другий займає решту вікна браузера.

Атрибути тегу *frameset*:

- *border* — товщина ліній границі;
- *frameborder* — товщина розмежувальної лінії.

Якщо з якихось причин браузер не підтримує фрейми, то використовується контейнер: *<noframes> </noframes>.*

Усередині тегу *frameset* можна розмістити потрібну кількість тегів *frame*:

```
<frame
src="url web-документа, що повинен завантажуватися в цей фрейм"
name='MyFrame'>
```

<frame> може бути однокомандним.

Атрибут *name* може містити власне ім'я фрейму, щоб можна було визначити, якому фрейму використовувати посилання.

Тег *frame* може містити необов'язкові параметри:

- *SCROLLING=* — наявність смуг прокручування;
- *SCROLLING=YES* — смуги прокручування присутні;
- *SCROLLING=NO* — смуги прокручування відсутні;
- *SCROLLING=AUTO* — смуги прокручування присутні тільки в тому випадку, якщо документ не міститься у фреймі;
- *NORESIZE* — користувач не може змінювати розмір фрейму;
- *NAME* — задає ім'я фрейму, використовується в основному для того, щоб можна було оновлювати інформацію в одному фреймі, активізуючи гіпертекстове посилання в іншому;
- *MARGINWIDTH* — ширина бічних розділювальних смуг між фреймами;
- *MARGINHEIGHT* — ширина верхніх і нижніх розділювальних смуг.

Приклад задання фрейму:

```
<frameset cols="*, 50%">
   <frame
                src="1.htm"
                name="my1">
<frameset rows="20%, 20%, *">
   <frame
                src="2.htm"
                name="my2">
   <frame
                src="3.htm"
                name="my3">
   <frame
                src="4.htm"
                name="my4">
 </frameset>
</frameset>
```

Результат відображення в браузері наведено на рис. 3.31.

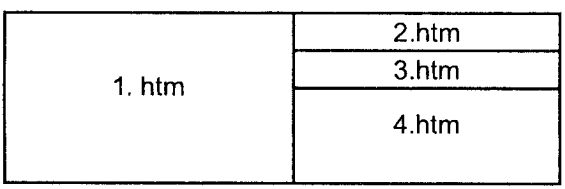


Рис. 3.31. Вигляд в браузері фреймової структури

Особливості застосування атрибута *target* розглянемо на прикладі, зображеному на рис. 3.32. Нехай перший фрейм — головний і кожне посилання має містити атрибут, що дозволяє повернутися до головної сторінки.

Якщо власні імена фреймів починаються із символу «_» — це цільові фрейми. Їх необов'язково указувати у фреймовій структурі. Нагадаємо знову їх значення:

• *_blank* — якщо деякий атрибут *target* посилається на нього, то документ з'являтиметься в новому порожньому вікні;

• *_self* — нова сторінка з'являтиметься в тому ж вікні;

• *_parent* — усі документи, що викликаються за цим посиланню, з'являтимуться в батьківській фреймовой структурі;

• *_top* — документи з'являються в окремому нефреймовому вікні, для перегляду буде згенероване нове вікно.

Недоліком фреймових структур є те, що вкладеність фреймів ускладнює навігацію.

Таким чином, сформулюємо загальні властивості фреймів:

1. Кожний фрейм має власний URL.
2. Кожний фрейм має власне ім'я (name), що дозволяє переходити з одного фрейму в інший.
3. Розмір фрейму може змінюватися користувачем, якщо це не заборонено.
4. Фрейми дозволяють створювати складні інтерфейсні рішення (наприклад, розміщувати статичну інформацію (кнопки, пункти меню) в одному фреймі, а динамічну — в іншому); створювати вікна запитів (в одному вікні — запит, в іншому — його результати); створювати форми для звертання до баз даних.

3.10. Метатеги

Пошукові сервери при реєстрації web-сервера, сайта або сторінки використовують інформацію, що зберігається в спеціальних тегах *<meta>*.

Метатеги — це службові теги, що не опрацьовують корисний текст документа, а призначені для управління документом у цілому. Метатеги завжди перебувають у заголовній частині, їх може бути будь-яка кількість, вони завжди однокомандні і завжди з атрибутами.

Метатеги мають два можливих атрибути:

```
<META
        name =ім'я об'єкта, яким хочемо управляти
        content = значення >
```

або

```
<META
        HTTP-EQUIV = ім'я http-заголовка
        content = значення>
```

Приклад:

```
<Meta name="description" content="This is my document">
```

Метатеги мають знаходитися в заголовку HTML-документа між *<HEAD>* і *</HEAD>* (особливо це важливо для документів, що використовують фрейми).

Розглянемо варіанти складання метатегів, які найбільш часто використовуються.

1) Для того, щоб при завантаженні документа браузер брав його останню версію, а не версію з кеша:

```
<META
    HTTP-EQUIV="expires"
    content="Mon, 13 Nov 2004 00:00:01 GMT">
    Якщо content="0", то обновити потрібно зараз же
```

2) Для визначення браузером мови, якою написано сторінку:

```
<META
    HTTP-EQUIV="content-Type"
    content="text/html; charset=Windows-1251">
```

3) Для примусового перезавантаження або автоматичного завантаження будь-якого документа:

```
<META
    HTTP-EQUIV="refresh"
    content="t"; url=адреса>
```

Час завантаження в секундах, після закінчення якого браузер почне завантаження за зазначеною адресою;

4) Для управління кешуванням:

```
<META
    HTTP-EQUIV="cashe-control"
    content="public/private/no-cashe/no-store">
```

5) Для контролю рівня «дорослості» сайта (платформонезалежна схема рейтингу):

```
<META
    HTTP-EQUIV="pics-label"
    content="violence">
```

6) Для визначення вікна поточної сторінки:

```
<META
    HTTP-EQUIV="window-target"
    content="top/bottom">
```

7) Для зазначення імені автора і додаткової інформації про нього:

```
<META
    name="autor"
    content="Ваші П.І.Б., E-Mail...">
```

8) Для зазначення відомостей щодо авторських прав:

```
<META
    name="copyright"
    content="Ваша_фірма...">
```

9) Для зазначення ключових слів і термінів:

```
<META
    name="keywords"
    content="Ключові слова">
```

Призначений для пошукових систем (індексування). Максимальна довжина — 1000 символів;

10) Для короткого опису сторінки:

```
<META
    name="description"
    content="Опис_Сторінки">
```

Рекомендації з використання метатегів:

1) пошукові машини, як правило, працюють з метатегами і тільки за їх відсутності — із вмістом сторінки;

2) від коректності складання метатегів залежить швидкість відшуку сайта за допомогою пошукових систем;

3) метатеги корисні, якщо в документі мало тексту; головна сторінка зроблена у вигляді скрипту; головна сторінка є фреймсетом;

4) у ключових словах і описах слід використовувати якомога більшу кількість синонімів. Ключові слова мають повторюватися не менше трьох разів;

5) довжина вмісту метатегів *descriptions* не повинна перевищувати 200 символів, а *keywords* — 1000 символів. Це пов'язано з тим, що пошукові сервери, як правило, використовують саме

такі величини при індексуванні інформації про сайт. Пошукові машини відкинуть частину опису, що перевищує встановлені норми, у результаті чого в каталог потрапить не вся бажана інформація, або сервер просто відкине реєстрацію сайта.

3.11. Контрольні запитання

1. Як задаються і відображаються списки?
2. Як відображаються гіперпосилання?
3. Назвіть види гіперзв'язків.
4. Як відображаються графічні об'єкти?
5. Як використовувати графічний образ як маршрутизатор гіперпосилання?
6. Опишіть етапи створення web-документа.
7. Як відредагувати існуючий HTML-документ?
8. Як відображається текст HTML-документа в текстовому редакторі?
9. Як відображається HTML-документ у вікні браузера?
10. Опишіть синтаксис мови HTML.
11. Дайте загальну характеристику HTML-стандарту.
12. У чому суть гіпертекстової форми представлення web-документів?
13. Що таке гіперзв'язок?
14. Що таке маршрутизатор гіперпосилання?
15. Як відображаються гіперпосилання?
16. Назвіть види гіперзв'язків.
17. Як відображаються графічні об'єкти?
18. Як використовувати графічний образ як маршрутизатор гіперпосилання?
19. Назвіть основні групи тегів HTML.
20. Як задаються і відображаються заголовки?
21. Як відображаються шрифти?
22. Як задається і відображається форматований текст?
23. Як відображаються фрейми, діалогові форми, меню, графічні об'єкти?
24. Для чого використовуються фрейми?
25. Як створюються перемикачі-радіокнопки?
26. Як створюються меню і списки, що розкриваються?
27. Які особливості використання метатегів?

3.12. Завдання для самостійного виконання

Розробити web-сайт з елементами HTML (різні шрифти, заголовки, різні кольори, абзаци, списки, картинки, гіперпосилання і мітки, таблиці, метатеги). Використати різноманітні засоби форматування тексту, фрейми.

Для розробки пропонуються такі теми сайтів:

1. Електронна версія газети.
2. Електронна версія журналу.
3. Сайт-візитівка про себе.
4. Сайт-візитівка фірми.
5. Сайт спортивної команди.
6. Сайт про тварин.
7. Географічний довідник країн світу.
8. Зоологічний довідник.
9. Ботанічний довідник.
10. Електронний довідник за будь-якою вивченою дисципліною.
11. Сайт, присвячений власному хобі.

Розділ 4
МЕТОДИ ТА ЗАСОБИ СТИЛЬОВОГО ОФОРМЛЕННЯ ДОКУМЕНТІВ

4.1. Використання палітри кольорів

Розглянемо способи кодування кольорів у web-технологіях.

Перший спосіб — *текстове кодування* (завдання кольору за допомогою ключових слів *red, green, blue, silver, gray, black, white* тощо). Повний перелік доступних кольорів наведений у Додатку 2.

Другий спосіб — *шістнадцяткове подання* кольору. Колір кодується за такими шаблонами:

1) *#XX XX XX*
R G B

Наприклад: *#00cc00*;

2) *rgb(x,x,x)*,
де *x* — число від 0 до 255.

Наприклад: *rgb(0,204,0)*;

3) *rgb (x%,x%,x%)*,
де *x%* — число від 0 до 100.

Наприклад: *rgb(0%,80%,0%)*.

Примітка: усі приклади відображають один і той же колір.

Розглянемо теги й атрибути для задання колірної гами документа.

Кольорова гама всього документа задається синтаксисом:

<body
атрибут=значення>.

Доступні такі атрибути:

bgcolor — визначає колір фону, задається числом;

background="url" — визначає фоновий рисунок *url*-адреси файла-рисунка;

link — колір гіперпосилання;

vlink — колір переглянутого гіперпосилання;

alink — колір гіперпосилання в момент одержання фокуса.

4.2. Каскадні таблиці стилів CSS

CSS — це мова розмітки, яка містить набір властивостей для визначення зовнішнього вигляду документа. Вона також дозволяє динамічно переформатувати документ.

У рекомендаціях W3C таблиці стилів називаються «каскадними таблицями стилів», тому що для верстки web-сторінки можна застосовувати не одну, а декілька таблиць. При цьому програма перегляду (браузер) сама визначає послідовність використання таблиць і вирішує конфлікти між ними за принципом «каскадування» (каскадного виконання).

Наприклад, таблиця стилів для сторінки може бути визначена не тільки її автором, але й читачем, і тоді правила каскадування визначають, яка з таблиць стилів матиме силу.

Кожному правилу браузер приписує ваговий коефіцієнт. При інтерпретації кожного тегу браузер переглядає всі правила цього тегу і сортує їх за ваговими коефіцієнтами. Виграє найбільш «вагоме» правило.

Як і у випадку з тегами форматування, існують два стилі оформлення документів — фізичний і логічний.

Фізичний стиль передбачає надання прямої вказівки браузеру на модифікацію оформлення.

Логічний стиль означає, що автор документа не може знати заздалегідь, що побачить користувач.

І фізичними, і логічними стилями можна управляти за допомогою таблиць стильового оформлення — каскадних таблиць стилів CSS і CSS1.

Кожне визначення стилю в CSS називається *правилом* (rule). Правило містить *селектор* (тег HTML), за яким далі вказується *декларація* (визначення стилю), наприклад:

```
H1 {color: blue}.
```

Декларацію укладають у фігурні дужки. Кожна декларація складається з двох частин, розділених двокрапкою: назви властивості й значення, що присвоюється йому. У CSS існує багато властивостей (наприклад, *font-size*, *font-style*, *color*, *margin-right* і т. д.). Кожна властивість може набувати кілька значень, одне з яких приписується йому за замовчуванням.

Коментування таблиці стилів. У міру ускладнення таблиці стилів знадобиться вмістити в неї додаткові відомості про

призначення того чи іншого правила. Коментарі розташовуються між символами /* та */ й ігноруються програмами перегляду (браузерами).

4.3. Засоби визначення таблиці стилів

Розглянемо основні способи визначення (опису) CSS:

1. *Визначення стилю за допомогою спеціального тегу безпосередньо в документі:*

<STYLE> текст опису стилю </STYLE>

Всередині контейнера *<style>* вміщаються описи типу

ім'я_контейнера{властивість: значення; властивість: значення; ...}

Наприклад: *BODY{font-size: 14pt}*

2. *Розміщення опису стилю в самому тегу за допомогою атрибута style*:

<ім'я_тегу style="властивість: значення; властивість: значення... ">

Наприклад: <h1 style="color: red">

3. Імпорт опису стилю в документ із зовнішнього файла.

Опис стилю може зберігатися в зовнішньому текстовому файлі, що є набором рядків з описувачами відповідного формату:

ім'я_контейнера {опис стилю}

Файл можна називати будь-як, наприклад *css.htm*, але частіше дають таке ім'я:

ім'я_стилю.css

При цьому приєднання стильового опису, розташованого за межами документа, може здійснюватися такими способами:

- за допомогою тегу *<link>*. Контейнер *<link>* призначений для встановлення відносних зв'язків між документами (тобто

для «лінкування» документів). Має два обов'язкових атрибути *rel* (вказує на зв'язок «назад», тобто «до себе») або ж *rev* (вказує на зв'язок «уперед»):

<link rel="URL">

або

<link rev="URL">

Наприклад, правильним є таке підключення зовнішньої таблиці стилів до поточного документа:

<link type="text/CSS" rel="url" або "stylesheet" href="url">;

• безпосередній імпорт стилю за допомогою оператора *@import*. Імпорт стилів можна здійснити або всередині тегу *<style>*, або всередині зовнішнього файла, що описує стиль:

@import: url (http: // . . . /1.css)

Стиль, що імпортується, можна також визначити через атрибут *style*.

Розглянемо докладніше використання тегу *<style>*. Він може використовуватися як самостійний контейнер або бути атрибутом іншого контейнера. Якщо тег *<style>* використовується як самостійний, то він має такий синтаксис:

<style>
selector { властивість: значення; властивість: значення...}
</style>

В описі стилю зазвичай задають:

- стилі для тегів;
- стилі для класів об'єктів (задаються атрибутом *class*);
- стилі для об'єктів із заданим унікальним ідентифікатором (для цього використовується атрибут *id*, який має унікальне значення);
- стилі для класів тегу *<A>*.

Розглянемо вказані способи описів докладніше.

Стилі для тегів задаються вказівкою імені тегу на місці селектора.

Наприклад:

p {color: green}.

Якщо необхідно задавати однакові стилі неоднорідним тегам розмітки, їх перераховують через кому, наприклад: *p, b, h1{color: green}.*

У наведеному прикладі стильове оформлення шрифту зеленим кольором буде застосоване до всіх тегів <*p*>, <*b*> та <*h1*>. Таким чином, за допомогою класів можна задавати стильове визначення без прив'язування до тегу.

Якщо необхідно застосувати вкладені (каскадні) визначення стилів, тоді ім'я селектора задається таким синтаксисом:

ім'я_контейнера1 ім'я_контейнера2...ім'я_контейнераN {...},

де *N* — показник ступеня вкладеності.

Наприклад, для вкладеного тегу, що визначає список

```
<ul>
<li><A href...><img src=...></img></A>
<li>
</ul> ,
```

визначення стилю має вигляд

ul li A img {src: url}.

Стилі контейнерів можуть успадковуватися вкладеними об'єктами, при цьому об'єкти успадковують стилі тільки в тому випадку, якщо вони не мають стилю за замовчуванням і якщо для них не перевизначені властивості батьківського контейнера. Наприклад:

```
<style>
P{color: green}
I{font-size: 16pt}
</style>
```

Розміщений усередині параграфа контейнер <*I*> відображатиметься зеленим кольором.

- І ще приклад:

```
<style>
P{color: green}
I{font-size: 16pt; color: red}
</style>
```

У даному ж випадку контейнер </> відображатиметься червоним кольором, оскільки відбулося перевизначення.

Атрибут *class* дозволяє застосовувати однакове оформлення до різнотипних об'єктів, тобто створює псевдоклас, від якого начебто «успадковані» усі об'єкти-контейнери, які позначені вказаним атрибутом. Наприклад:

A.myclass1 {color: black}
.myclass2 {font-size: 12pt}.

У прикладі перший рядок задає чорний колір тільки для тих гіперпосилань, які позначені атрибутом *class="myclass1"*, а другий — розмір шрифту для всіх об'єктів, у яких один з атрибутів *class="myclass2"*.

Іноді використовують визначення стилю для об'єктів із заданим унікальним ідентифікатором за допомогою атрибута *id:*

* текст .*

Тоді стильове визначення може мати вигляд:

#myid {back-ground-color: orange}.

Воно задає оранжевий колір фону для об'єкта з атрибутом *id="myid"*.

Залишився останній спосіб опису — це стилі для класів тегу *<A>*.

Стилі гіперпосилань задають за допомогою псевдокласів тегу *<A>*:

link — лінк;
active — активне гіперпосилання;
visited — гіперпосилання, яке вже відвідане;
hover — гіперпосилання в момент піднесення курсора.

Псевдокласи при визначенні відокремлюються знаком *":"*. Наприклад:

```
<style>
a:link {color:black}
a:visited {color:silver}
a:hover {color:red}
</style>
```

Існують такі загальні *принципи розв'язання конфліктів* між таблицями стилів:

1. Таблиця стилів автора сторінки є більш «вагомою», ніж таблиці стилів читача, які у свою чергу є більш «вагомими», аніж установки браузера за замовчуванням.

2. При посиланні на зовнішній опис можливі різноманітні комбінації опису стилів. У цьому випадку діють правила старшинства стилів:

1) спочатку застосовуються стилі за замовчуванням (стилі установки браузера);
2) стилі за замовчуванням перевизначаються приєднаними стилями (тег *<link>* у розділі *<head>*);
3) приєднані стилі перевизначаються описувачами в тегу *<style>*;
4) стилі тегу *<style>* перевизначаються атрибутом *style* у будь-якому тегу.

4.4. Блокові та рядкові елементи

В описі розмітки мови HTML існують два типи елементів: рядковий елемент *in-line* і блоковий — *block*. То ж розглянемо їх докладніше.

Блоковий елемент розмітки відображається всередині прямокутного фрагмента, що знаходиться на окремому рядку і відділений від попереднього і наступного фрагментів.

Рядковий елемент не займає окремого рядка і довкола нього знаходяться інші елементи. Заголовок — це блоковий елемент, а курсив — рядковий. Блокові елементи можна вкладати, а рядкові вкладати і перетинати.

Для стильового узагальнення рядкових і блокових елементів виступають теги * * і **.

Контейнер * * є універсальним блоковим елементом. Його властивості:

- завжди відокремлюється від інших елементів порожнім рядком;
- не несе ніякого навантаження за значенням, а є розділь-ником сторінок. Якщо не задано опис *css*, то контейнер * * просто створює порожній рядок тексту;
- контейнер * * дозволяє застосовувати атрибути стилю, пов'язані з границею блоків (відступ блоку від границь старшого елемента).

Якщо браузер не підтримує *css*, то застосовувати * * не рекомендується.

Контейнер ** є узагальненим рядковим контейнером розмітки. Застосовується для заміни тегів *</>*, *<B>*, *<U>* й інших тегів, зв'язаних з форматом шрифту.

Наприклад, синтаксис

<I>текст</I> замінюється на

текст ,

<U>текст</U> замінюється на

текст

або на

**,

а в таблиці стилів необхідно помістити

span.decor{ font-decoration: underline }.

Застосування тегу ** також обмежується браузерами, що підтримують *css*.

Блоковий елемент розмітки відображається всередині прямокутного фрагмента, що знаходиться на окремому рядку і відділений від попередніх таких фрагментів. Рядковий елемент не займає окремого рядка і довкола нього знаходяться інші елементи. Тобто, наприклад, заголовок — це блоковий елемент, а курсив — рядковий. Блокові елементи можна вкладати, а рядкові не тільки вкладати, а ще й перетинати.

У зв'язку з появою тегів * * та ** організація W3C не рекомендує вживати такі теги:

<u>, *<strike>*, *<font>*, *<menu>*, * *, *<plaintext>*

та атрибути:

align (застосовувати для *<table>*), *<hr>*, *alink*, *background*, *bgcolor*, *pspace*, *hspace* та ін.

4.5. Властивості CSS

CSS дозволяє визначити широкий спектр властивостей таблиць стилів. Імена властивостей складаються з одного, а частіше — з двох або трьох слів, розділених дефісом. У складних назвах перше слово зазвичай представляє категорію й одночасно є скороченим варіантом імені властивості для всієї категорії. Наприклад, категорія «background» відповідає за всі властивості, пов'язані з фоном елемента та може містити підкатегорії «color» (тобто відповідає за колір фону), «image» (відповідає за фоновий рисунок) тощо.

Усі змінювані властивості поділяються на п'ять груп. Розглянемо їх.

1. Властивості для визначення кольору і фону (див. табл. 4.1).

Таблиця 4.1

Властивості кольору і фону

Назва властивості (категорія-підкатегорія)	Опис властивості	Допустимі значення властивості
color	Визначає колір	color: колір/ transparent (прозорий)
background-color	Визначає колір фону	background-color: колір / transparent
background-image	Завантажує картинку	background-image: URL / none Наприклад: background-image: url(fon1.gif)
background-repeat	Задає повторення фонового зображення	background-repeat: repeat / repeat_x / repeat_y / no_repeat
background-aftachment	Дозволяє або забороняє прокручування фонового рисунка разом зі скролінгом сторінки (фіксує фоновий рисунок)	background-aftachment: fixed / scroll
background-position	Визначає положення фонової картинки	background-position: bottom / center / top / left / right / % від ширини екрану

2. Властивості шрифтів і тексту наведені в табл. 4.2.

Таблиця 4.2

Властивості шрифтів і тексту

Назва властивості (категорія-підкатегорія)	Опис властивості	Допустимі значення властивості
1	2	3
font-family	Назва шрифту	*font-family: Arial, TimesNewRoman та ін.*
font-style	Вид шрифту	*normal / italic*
font-variant	Управляє регістром шрифту	*normal / caps / small*

Продовження табл. 4.2

1	2	3
font-size	Розмір шрифту	наприклад: 12pt або 12px
font-weight	Жирність шрифту	font-weight: *bold*
height	Встановлює загальну висоту елемента — блоку тексту чи зображення	будь-який спосіб зазначення розміру (у тому числі %); *auto* — браузер автоматично встановить оптимальні розміри елемента
line-height	Відстань між базовими лініями рядків тексту	числове значення у відсотках від аналогічної властивості батьківського елемента або ж значення *normal*
text-decoration	Використовується для встановлення оформлення шрифту	*none* — властивість відключено; *underline* — підкреслення; *overline* — лінія над рядком; *line-through* — закреслення; *blink* — миготіння тексту
text-indent	Вказує на відступ у першому рядку	числове значення; можна створити «висячий» відступ, якщо призначити *цій* властивості від'ємне значення, а властивості *margin-left* — додатне значення
text-transform	Вказує, як трансформуватиметься текст	*capitalize* — перша літера кожного слова буде прописною; *uppercase* — усі літери тексту стануть прописними; *lowercase* — усі літери тексту стануть рядковими; *none* — трансформація заборонена
vertical-align	Вирівнювання рядкових елементів	*baseline* — вирівнює базову лінію елемента за базовою лінією батьківського елемента; *middle* — вирівнює середину елемента за серединою батьківського елемента; *sub* — опускає елемент на підрядковий рівень; *super* — піднімає елемент на надрядковий рівень;

Закінчення табл. 4.2

1	2	3
		text-top — вирівнює вершину елемента за верхом тексту батьківського елемента; *text-bottom* — вирівнює низ елемента за низом тексту батьківського елемента; *top* — вирівнює верхівку елемента за найвищим елементом рядка; *bottom* — вирівнює низ елемента за найнижчим елементом рядка
word-spacing	Відстань між словами тексту	*normal* — рішення приймає браузер; *будь-яке числове значення*, наприклад, 1 px
white-space	Визначає, як браузер інтерпретуватиме вільне місце всередині елемента	якщо властивість не визначити, то браузер стисне вільний простір; *normal* — вільний простір стискається; *pre* — вільний простір буде інтерпретовано так, як це виконується за допомогою тегу <pre> (*див.* Розділ 3); *nowrap* — дозволяється переведення рядка тільки за вказівкою
letter-spacing	Встановлює відстань між символами	*normal* або довжина в пікселах

3. Властивості списків наведені в табл. 4.3.

Таблиця 4.3

Властивості списків

Назва властивості (категорія-підкатегорія)	Опис властивості	Допустимі значення властивості
1	2	3
list-style-type	Зображення маркера	*disc* – диск; *circle* — коло; *square* — квадрат *decimal* — арабські цифри (1, 2, 3, 4, ...);

Закінчення табл. 4.3

1	2	3
list-style-type	Зображення маркера	*lower-roman* — маленькі римські цифри (i, ii, iii, iv, ...); *upper-roman* — великі римські цифри (I, II, III, IV, ...); *lower-alpha* — рядкові літери (a, b, c, d, ...); *upper-alpha* — прописні літери (A, B, C, D, ...); *none* — маркер відсутній
list-style-image	Зображення маркера списку	*url ( )* — адреса зображення
list-style-position	Зміна позиції маркера	*inside* — текст пункту при переводі рядка починається під маркером; *outside* — під текстом попереднього пункту («висячий» відступ)

4. Властивості блокових елементів (див. табл. 4.4). Вони задають різноманітні зовнішні та внутрішні відступи контенту від границь блоку, а також властивості самих границь (бордеру).

Таблиця 4.4

Властивості блокових елементів

Назва властивості (категорія-підкатегорія)	Опис властивості
1	2
margin-top *-right* *-left* *-bottom*	Зовнішні відступи блокового елемента
padding-top *-right* *-left* *-bottom*	Внутрішні відступи блокового елемента
border-top-width *border-right-width* *border-left-width*	Для визначення ширини відповідних меж блокового елемента
border-width	Для визначення товщини всіх меж

Закінчення табл. 4.4

1	2
border-style	Визначає стиль рамки: *none* — рамка відсутня (за замовчанням); *dotted* — лінія з точок; *dashed* — штрихова лінія; *solid* — звичайна лінія; *double* — подвійна лінія, ширина ліній і відстань між ними відповідає значенню властивості *border-width*; *groove* — тривимірна вдавлена лінія того кольору, який визначено властивістю *color*; *ridge* — тривимірна випукла лінія того кольору, який визначено властивістю *color*; *inset* або *outset* — тривимірна лінія того кольору, який визначено властивістю *color*

5. Властивості позиціонування об'єктів (див. табл. 4.5).

За допомогої властивостей з цієї групи можна управляти розміщенням та появою об'єктів у вікні браузера, наприклад встановлювати абсолютне позиціонування елементів. Наприклад, за допомогою властивості *position* можна відображати елементи у вікні браузера, використовуючи дійсні координати (в пікселах), які відраховуються, починаючи з лівого верхнього кута вікна.

Наведемо приклад:

.mypic {position: absolute; top:20; left:25 } — усі об'єкти, які відносяться до псевдокласу *.mypic*, розташовуватимуться, починаючи з позиції 20×25 пікселів від верхнього лівого кута вікна браузера.

Таблиця 4.5

Властивості позиціонування об'єктів

Назва властивості (категорія-підкатегорія)	Опис властивості	Допустимі значення властивості
1	2	3
display	Визначає, чи буде показаний елемент	За замовчанням елемент показується; *none* — елемент не «видно» і місце під нього на сторінці не відводиться

Закінчення табл. 4.5

1	2	3
visibility	Визначає видимість елемента	*visible* — елемент є видимим; *hidden* — елемент є невидимим, але він займає своє місце на сторінці, тобто він є «прозорим»; *inherit* — елемент є видимим за умови, що батьківський елемент теж є видимим
clear	Забороняє або дозволяє виведення елементів, що «плавають»	*none* — елементи, що «плавають», дозволяються з обох сторін; *left* — елементи, що «плавають», заборонені ліворуч; *right* — елементи, що «плавають», заборонені праворуч; *both* — елементи, що «плавають», заборонені з обох сторін
float	Вказує на те, що елемент розташовується ліворуч або праворуч, при цьому інші елементи його обтікають	*none* — виводить елемент без обтікання; *left* — розташовує елемент ліворуч, при цьому текст обтікає елемент; *right* — розташовує елемент праворуч, при цьому текст обтікає елемент
overflow	Визначення видимості елемента, якщо вміст більше розмірів виводу	може мати значення *scroll*, *hidden*, *visible*, *auto*.
z-index	Вказує, в якій послідовності елементи перекриватимуть один одного	Ціле число (елементи з більшим значенням *z-index* розташовуватимуться над елементами з меншим *z-index*)
height, left, top,width	Властивості, які відповідають за висоту, ліву сторону, верхню сторону та ширину елемента відповідно	число або %

6. Додаткові властивості. До них відносять:

1) описувачі стилів псевдокласів тегу <A>: *active*, *hover*, *link*, *visited*, які описують стиль якірного елемента при активному гіперпосиланні, під час наведення мишею, за замовчуванням та гіперпосилання, яке вже відвідане;

2) властивості атрибута cursor — визначають вид курсора для елемента при наведенні на нього миші. Допустимими є значення *auto*, *crosshair*, *default*, *hand*, *move*, *e-resize* тощо.

Одиниці виміру для використання з властивостями в CSS є такими:

- *em* — висота використовуваного елементом шрифту;
- *ex* — *x-height*, ширина літери *x* використовуваного елементом шрифту;
- *px* – пікселі;
- *in* – дюйми;
- *cm* — сантиметри;
- *mm* — міліметри;
- *pt* — пункти (1 pt = 1/72 дюйма);
- *pc* — піки (1 pc = 12pt);
- % розміри: "+" або "-" потім число та "%" без пропусків. Наприклад: -566% .

Існує ще й такий спосіб визначення розмірів:

"+" або "-" далі число та одиниця виміру без пропусків. Наприклад: -566pt

Групування властивостей для спрощення визначення стилю

Більшість з описаних вище властивостей можуть групуватися один з одним.

Так, замість опису

h1 {font-weight: bold; font-style: normal; font-size: 12pt; font-family: serif}

можна записати більш короткий

h1 {font: bold normal 12pt serif}.

Але такий синтаксис потребує обов'язкового тестування в різних браузерах. Наприклад, у браузері Mozilla скорочений синтаксис CSS не підтримується.

4.6. Контрольні запитання

1. Як скласти і підключити таблицю стилів?
2. Які існують способи використання стилів у документі?
3. Які можливості CSS?
4. Які способи обробки шрифтів за допомогою CSS?
5. Опишіть способи задання стильових описів.
6. Який колір задається описом: *rgb(0,0,255)*?
7. Як задавати посилання на файл зі стилевим описом?

4.7. Завдання для самостійного виконання

У розробленому web-проекті (згідно з завданням підрозділу 3.12) використати такі елементи CSS:

— селектори;

— псевдокласи;

— *id*-селектори.

Відпрацювати різні способи впровадження CSS у сторінку, підпорядкованість стилів, зміну шрифтів, кольорів, фону, вирівнювання, зміну стилів списків, границь і рамок, курсорів, використати різні одиниці виміру.

Розділ 5
ОСНОВИ МОВ ПРОГРАМУВАННЯ СЦЕНАРІЇВ

5.1. Мови програмування сценаріїв

JavaScript — мова для складання сценаріїв (скриптів), розроблена фірмою Netscape. За допомогою JavaScript можна легко створювати інтерактивні web-сторінки. JavaScript базується на синтаксисі мови Java, але це не Java! JavaScript є компактною об'єктно-орієнтованою мовою для складання скриптів. У таблиці 5.1 подані відмінності мов JavaScript і Java.

Таблиця 5.1

Порівняння JavaScript та Java

JavaScript	Java
Інтерпретується і виконується браузером (тобто у клієнта)	Компілюється на сервері, завантажується браузером, потім виконується віртуальною машиною Java
Використовує вбудовану мову й об'єктну модель браузера. Класи можуть створюватися, але не можуть успадковувати функції інших класів	Повністю об'єктно-орієнтована, здатна визначати класи і підтримувати спадкування
Змінні не вимагають оголошення і задання типу	Усі змінні мають бути оголошені і мати чітко визначений тип
Не має засобів прямого доступу до клієнтських ресурсів (диска та пам'яті)	Не має засобів прямого доступу до диска клієнта без наявності цифрового підпису

Типи даних

На відміну від Java або C++, JavaScript є мовою з вільно обумовленими типами. Змінним формально не присвоюється жоден тип. Однак змінні можуть зберігати значення кожного з п'яти внутрішніх типів даних, короткий опис і приклади значень яких наведені в таблиці 5.2.

Значення, що зберігається в змінній, за необхідності автоматично перетворюється інтерпретатором. Число може бути виведене як рядок, а рядок може бути інтерпретований як ціле число без виконання операцій приведення типів або перетворення.

Оголошення змінних

У мові JavaScript змінні не вимагають оголошення. Коли змінна зустрічається перший раз, її тип визначається за контекстом і додається в список змінних інтерпретатора.

Таблиця 5.2

Типи даних JavaScript

Тип даних	Опис
Числовий	У мові JavaScript не існує відмінностей між цілими і раціональними числами, а є загальний тип — числовий
Логічний	Подібний булевому в мовах програмування. Припустимими логічними значеннями є *true* і *false*
Рядковий	Рядком називається послідовність, що складається з нуля або більше символів, укладених у подвійні (") або одинарні (') лапки. Рядок має обмежуватися лапками одного типу
Null	Спеціальний символ, який вказує, що значення змінної не визначене
Об'єкт	Як і в будь-якій об'єктно-орієнтованій мові програмування, об'єкт є сукупністю даних, а також функцій, що оперують цими даними

Використання змінних обмежується областю їхньої видимості. У JavaScript є дві таких області: *глобальна* і *локальна*.

Змінна, що має *глобальну* область видимості, доступна будь-яким сценаріям або функціям, що використовуються в документі.

Змінна, що має *локальну* область, може бути використана тільки сценарієм або функцією, у якій вона визначена.

За замовчуванням усі змінні є глобальними. Для оголошення глобальної змінної їй просто потрібно присвоїти значення. Щоб оголосити локальну змінну, необхідно при першому її використанні поставити перед нею ключове слово *var*.

Синтаксис операцій JavaScript подібний синтаксу операцій у мовах Java або C++.

Принцип розташування коду JavaScript на web-сторінці

Код скрипту JavaScript розміщується безпосередньо на web-сторінці.

Розглянемо такий приклад:

```
<html>
<body>
<br>
Це звичайний web-документ.
<br>
 <script language="JavaScript">
  document.write("Це JavaScript")
 </script>
<br>
Знову web-документ
</body>
</html>
```

Результат відображення в браузері наведено на рис. 5.1.

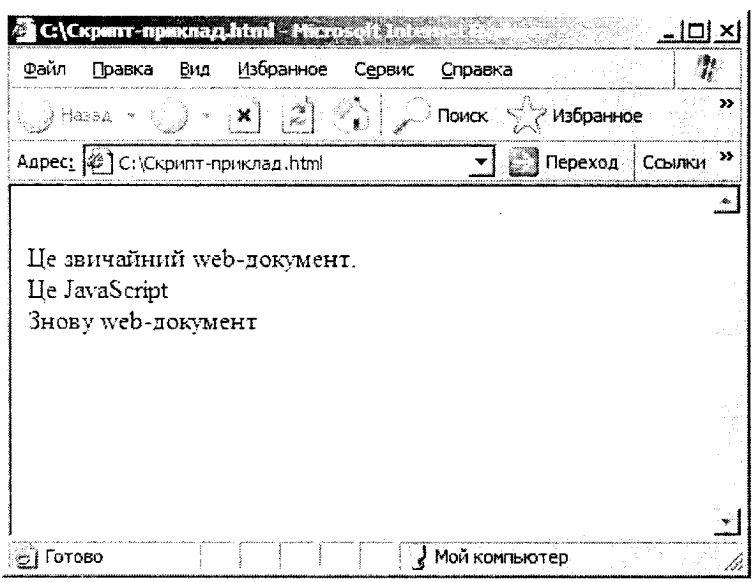


Рис. 5.1. Приклад простого сценарію

З першого погляду приклад нагадує звичайний файл HTML. Єдине нововведення — це конструкція:

```
<script language="JavaScript">
  document.write("А це JavaScript!")
</script>
```

Даний скрипт не настільки корисний — те ж саме і навіть більш просто можна було б написати «чистою» мовою HTML. Цей приклад необхідний лише для демонстрації тегу *<script>*.

Таким чином, все, що стоїть між тегами *<script>* і *</script>*, інтерпретується як код мовою JavaScript.

У прикладі також можна побачити використання інструкції *document.write()* — однієї з найбільш важливих команд, яка використовується при програмуванні мовою JavaScript.

Команда *document.write()* використовується, коли необхідно що-небудь написати в поточному документі (у даному випадку таким є наш web-документ).

5.2. Засоби підключення та запуску сценарних програм

Елемент Script

Сценарії на сторінці можуть бути зв'язані з документом з використанням одного з трьох методів. Найбільш загальний метод полягає в розміщенні програми в елементі *<script>* (два інших методи поміщають програму в окремий файл і посилаються на нього за допомогою тегу *<script>* або поміщають програму в атрибут події в іншому тегові).

Елемент *<script>* є контейнером для скриптової програми. Елемент *<script>* може містити програму, що знаходиться в документі, або посилатися на зовнішній файл. Сценарії, що утримуються усередині елемента *<script>*, можуть бути зв'язані з елементом за допомогою програми, спеціальних атрибутів елемента *<script>* або за допомогою залежного від мови механізму. Індивідуальні елементи можуть мати сценарії, що зв'язані з ними безпосередньо атрибутами подій, представлених у самому елементі.

Синтаксис елемента *<script>*:

```
<SCRIPT LANGUAGE= "JavaScript">
тіло скрипту
</SCRIPT>
```

Місце розташування сценаріїв у документі

Допускається будь-яка кількість <*script*> у документі. Їх можна розміщувати як у <*head*>, так і в <*body*>. Місце розташування <*script*> не відбивається на дизайні сторінки.

Рекомендується розміщувати коди функцій в окремих контейнерах <*script*> і розміщати їх у заголовній частині документа. Якщо зі скрипту код буде записаний у потік або працюватиме з іншим контейнером документа, то сценарії мають розташовуватися з урахуванням виклику цих функцій.

Якщо зі сценарію відбувається доступ до контейнера заголовка документа, то сценарій має вміщуватися першим, і тільки в заголовку документа.

<h1 title="Це заголовок"> — спливаюча підказка.

Звертання до функцій з використанням механізму подій:

<h1 onMouseOver="MyToolTip()">.

Найпростіший засіб виконання сценарного коду (крім задання коду в тегу <*script*>):

<A href="javascript: 'рядок коду;' ">.

Бібліотеки сценаріїв

Сценарії можуть знаходитися в зовнішньому файлі і бути пов'язані з будь-якою кількістю web-документів. Посилання на зовнішній сценарій здійснюються за допомогою атрибута *src:*

```
<SCRIPT
      LANGUAGE= "JavaScript"
      SRC= "File-name.js">
</SCRIPT>
```

Оскільки сценарій – це текстовий файл, то можна об'єднувати сценарії і створювати бібліотеки. Зазвичай вони мають розширення *.js*.

Підключення бібліотечного сценарію:

```
<script language= src="url"> код
</script>
```

Поряд із сценаріями існують скриплети (сценарії, які виконуються хоча б частково на стороні сервера). Код скриплета може бути написаний у клієнта, а виконуватися на сервері.

5.3. Основи мови JavaScript

Усі операції, які можна виконувати в програмі мовою JavaScript, описують дії над добре відомими і зрозумілими об'єктами, якими є елементи робочої області браузера і контейнери мови HTML. Є об'єкти з набором властивостей і набір функцій над об'єктами. Останні називаються *методами*. У JavaScript є *події* — аналог програмних переривань. Ці події також орієнтовані на роботу в World Wide Web, наприклад, завантаження сторінки в робочу область браузера або вибір гіпертекстового посилання.

До основних понять мови відносяться: *об'єкт, властивість, метод, подія і колекція.*

Об'єкт — сукупність властивостей, методів, подій і колекцій, надана браузером у рамках об'єктної моделі. Всі об'єкти створюються самим браузером, і доступ до них здійснюється через *екземпляр об'єкта* за схемою:

- об'єкт.метод
- об'єкт.властивість
- об'єкт.подія
- об'єкт.колекція

Властивість — змінна в рамках об'єкта, що може використовуватися для одержання якихось значень або установки нових. Деякі доступні тільки для читання. У HTML доступ до властивостей забезпечується за допомогою атрибутів тегів.

Наприклад:

a[0].href="url".

Метод — процедура або функція, надана об'єктом для виконання яких-небудь дій або управління властивостями об'єкта.

Наприклад:

window. alert (Ви впевнені?).

Подія — яка-небудь дія користувача або момент роботи браузера. Для реакції на події створюються оброблювачи подій.

Наприклад:

forms[0].onMouseOver="MyTest()".

Колекція — упорядкований набір властивостей, схожий на масив, доступ до якого здійснюється спеціальними засобами.

Наприклад:
document. all. h1[0].

Стандартні об'єкти JavaScript — це такі об'єкти, що забезпечуються не об'єктною моделлю браузера, а інтерпретатором мови.

Розглянемо оператори JavaScript.

Оголошення змінних здійснюється в такий спосіб:
var x=7 або x=7.
При конфлікті типів відбувається перетворення на тип *string*. Цей тип є основним.
Основні оператори:
1) присвоювання: =, *= , +=, -=, /=, %=;
2) математичні вираження: += ; %= ;
3) вираження порівняння: = ;
Оператори порівняння: == ; < > ; || ; ! ;
===, ! == — те ж саме, що і ==, ! =, але не відбувається перетворення типів.

Порядок виконання:
1. ++, - -
2. *, / — множення та частка з плаваючою точкою
3. +,-
4. <<, >>
5. Операторй порівняння.
6. Побітове «ТА».
7. Побітове «АБО».
8. Логічне «ТА».
9. Логічне «АБО».
10. Умовний оператор.
11. Оператор присвоювання.

Зарезервовані елементи:
// — однорядковий коментар;
/*...*/ — багаторядковий коментар.

Елементи мови:
break, continue, for, for.. in, function, if.. else, new, return, var, while, with.

break — вказує на те, що виконання коду в даному блоці повинно бути завершено.

continue — усі такі за ним у тілі циклу оператори пропускаються, але цикл виконується далі.

Цикл for

for ... in використовується для одержання назв і властивостей об'єктів:

```
for (var in object) .
```

Наприклад:

```
for (i in history {window. document. write(i)}) .
```

Змінні можуть бути локальними і глобальними. Наприклад:

```
<html>
<head>
<script>function My() {}</script>
<body>
<script> var x; </script>
        function my2 {};
        my2 може викликати My, але не навпаки.
with(ім'я об'єкта) {}
switch (змінна або вираз){
        case значення1: {дія1}
        case значення2: {дія2}
        .....
        default: {дія}
}
```

Дія1, дія2 виконуються, якщо змінна або вираз дорівнюють відповідним значенням.

5.4. Вбудовані об'єкти мови JavaScript

На відміну від об'єктів, створюваних користувачами, і об'єктів, що складають об'єктну модель браузера, вбудовані об'єкти забезпечуються безпосередньо інтерпретатором мови — будь то інтерпретатор, що входить до складу браузера Internet Explorer чи до складу Netscape Navigator.

Об'єкт Array

Застосовується для роботи з масивом. Вважається глобальним методом, тому що викликається звідусіль.

Властивості:
array. length — довжина масиву.

Методи:

- *array. join()* — об'єднує всі елементи масиву в рядок;
  - *.sort()* — сортування елементів масиву;
  - *.reverse()* — переписує масив у зворотному порядку.

Засоби задання масиву:
1) *var MyMas=new Array(N);*
Заповнення: *MyMas[0]="значення"*
2) *var MyMas=new Array('str1',..., 'strN');*

Виведення інформації:

```
window. alert("string");
window. alert(MyVar);
```

Наприклад:

```
window. alert("Результат:"+x);
document. write("<h1>"+'текст'+"</h1>") .
```

Наведемо приклад застосування масиву: створимо найпростішу анімацію.

```
<html>
<head>
<script language="JavaScript">
function MyAnim(){
var MyArray=newArray()
MyArray[0]='1.gif'
.....
MyArray[100]='100.gif'
i=0;
while (i<100){
document.images[0].src=MyArray[i].src;
i++}
}
</script>
<body
        onload='MyAnim()'>
<img src=1.gif>
</body>
</html>
```

Створення багатомірних масивів

Використовується, коли масив складається з сукупності інших масивів. Якщо потрібно створити масив $m \times n$, то створюється масив із n елементів, кожний з елементів якого — масив із m елементів. Кількість вкладень не обмежено.

Звертання виконується за наступним синтаксисом: *a[i][j]*

Об'єкт Boolean

Об'єкт Boolean використовується для перетворення небулівських значень у булівські. Для роботи з об'єктом за допомогою конструктора *new* створюється екземпляр об'єкта:

```
myfalse=newBoolean (false);
mytrue=newBoolean (true)
```

Об'єкт *Boolean* має такі методи:

- *valueOf()* — повертає значення булівської змінної у вигляді булівського значення;
- *toString()* — повертає значення булівської змінної у вигляді рядка.

Об'єкт Date

Вбудований об'єкт *Date* призначений для роботи з датою і часом. Не має властивостей, має поля і методи. Дата подається у вигляді кількості мілісекунд, який пройшли з 1.01.1970.

```
mydate=new Date ([параметри]);
```

Види параметрів:

1. Відсутність параметрів. У цьому випадку створений екземпляр об'єкта *Date* міститиме поточну дату і час:

```
var todayDate=new Date();
```

2. Параметром може бути рядок, у якому записана дата в такому форматі:

```
var todayDate=new Date([місяць, день, рік, години:хвилини:секунди]);
var myDateOldDate("Sep 29, 2001, 11:40:01");
```

3. Набір цілочислових значень у форматі: рік, місяць, день.

4. Набір цілих чисел, що позначають рік, місяць, день, год.: хв.:сек.

Наприклад:

```
<script>
todayDate=new Date(2001, 09, 15, 10:25:01)
document. write('Сьогодні+todayDate)
</script>
```

Засоби задання Date

Для задання застосовують різноманітні методи об'єкта *Date*. Ось їхня класифікація:

1. Методи для опрацювання *Date*.
2. Методи для установки дати і часу.
3. Методи для визначення дати і часу.
4. Методи для перетворення дати і часу.

Робота з об'єктом Date

Об'єкт *Date* і його методи використовуються для роботи з датою і часом у скриптових програмах. Цей об'єкт має великий набір методів для установки дати, одержання її значення і виконання різних перетворень. Об'єкт *Date* не має властивостей. Дата в мові JavaScript є числом мілісекунд, що пройшли з 1 січня 1970 року.

Для створення екземпляра об'єкта *Date* використовується конструктор *new*:

MyDate = new Date([параметри]).

Можливі такі параметри:

• жодних параметрів — екземпляр міститиме поточну дату і час. Наприклад, *today = new Date()*;

• рядок, що представляє дату в наступному форматі:
«місяць, день, рік, години:хвилини:секунди».
Наприклад:
someDate = new Date(«May 15, 1999»).

Якщо число годин, хвилин або секунд не вказано, то їхні значення рівні 0;

• набір цілочислових значень для року, місяця і дня.
Наприклад:
otherDay = new Date(99, 5, 15);

• набір цілочислових значень для року, місяця, дня, годин, хвилин і секунд.
Наприклад:
someDay = new Date (96, 4, 15, 15, 30, 0).

Методи об'єкта *Date* наведені в табл. 5.3.

Таблиця 5.3

Методи об'єкта *Date*

Метод	Опис
getDate	Повертає день місяця як ціле число від 1 до 31
getDay	Повертає день тижня як ціле число від 0 (неділя) до 6 (субота)
getHours	Повертає число годин як ціле від 0 до 23
getMinutes	Повертає число хвилин як ціле від 0 до 59
getMonth	Повертає номер місяця як ціле від 0 (січень) до 11 (грудень)
getSeconds	Повертає число секунд як цілу від 0 до 59
getTime	Повертає число мілісекунд між 1 січня 1970 року, 00 00 00 і датою, заданою об'єктом Date
gettimeZoneOffset	Повертає число хвилин, що складають різницю між локальним часом і часом за Гринвічем
getYear	Повертає дві останні цифри року
parse	Повертає число мілісекунд між 1 січня 1970 року, 00 00 00 і датою, заданою у вигляді рядка
setDate	Встановлює день місяця
setHours	Встановлює число годин
setMinutes	Встановлює число хвилин
setMonth	Встановлює номер місяця
setSeconds	Встановлює число секунд
setTime	Встановлює час
setYear	Встановлює рік
toGMTString	Перетворює локальний час у час за Гринвічем і повертає його у вигляді рядка
toLocaleString	Перетворює час за Гринвічем у локальний час і повертає його у вигляді рядка
UTC	Повертає число мілісекунд між 1 січня 1970 року, 00 00 00 і датою, заданою у вигляді параметра

Усі методи використовують наступний формат даних (табл. 5.4).

Таблиця 5.4

Формат даних для об'єкта *Date*

Значення	Діапазон
Число секунд і хвилин	0...59
Число годин	0...23
День тижня	0...6
Дата	1...31
Місяць	0...11 (Січень...Грудень)
Рік	Число років з 1990

Об'єкт function

Вбудований об'єкт *function* дозволяє задати рядок коду мовою JavaScript, що виконується як функція:

```
var myFunc=new Function("i", "document. bgColor=i")
```

Наприклад: перевірити правильність заповнення формуляра перед посиланням.

```
<html>
<body>
<script language="JavaScript">
function CheckForm() {
        var myDoc=window.document;
        if (mydoc.forms[0]. elements[0] value="студент")
          {alert("Thet's OK");
          return true}
        else return false }
<formName="MyForm" action="url" method=post>
<input type="text" name=mytext value="студент">
<input type="submit" value="Відправити дані"
onclick="CheckForm()">
</form>
</body>
</html>
```

Об'єкт Math

Надає властивості і методи для одержання математичних констант і виконання математичних функцій. Розглянемо властивості об'єкта (табл. 5.5).

Таблиця 5.5

Властивості об'єкта *Math*

Найменування	Опис
abs	Модуль
sin	Тригонометричні функції
cos	
tan	
asin	Зворотні тригонометричні функції
acos	
atan	
exp	Експонента
log	Логарифм
round	Округлення
floor	Ціле число, яке менше або дорівнює аргументу
pow	Степінь
sqrt	Корінь квадратний
min	Менше або більше з двох аргументів
max	

Об'єкт Number

Має властивості, використовувані для одержання значень різноманітних числових констант:

- *MAX_VALUE* — максимальне значення;
- *MIN_VALUE* — мінімальне значення;
- *NaN* (Not a Number);
- *NEGATIVE_INFINITY* — містить $-\infty$;
- *POSITIVE_INFINITY* — містить $+\infty$.

Об'єкт String

Містить методи для управління рядками і властивість *length*. Наприклад:

```
var myStr=newString("студент").
```

Методи об'єкта мають два типи: для виконання перетворень над рядками і ті, що повертають HTML-версії рядків (табл. 5.6).

Таблиця 5.6

Методи об'єкта *String*

Метод	Опис
substring()	Повертає зазначений підрядок з рядка
split()	Розділяє екземпляр об'єктів String на масив рядків
Link()	Створює HTML-посилання
indexOf()	Повертає позицію підрядка в рядку
lastIndexOf()	Повертає позицію останнього підрядка в рядку
toLowerCase()	Переведення до нижнього регістру
toUpperCase()	Переведення до верхнього регістру
ChartAt()	Символ із зазначеної позиції рядка

Об'єкт Navigator

Забезпечує інформацію про браузер.

Властивості:

- *appCodeName* — кодове ім'я браузера;
- *appVersion* — версія браузера;
- *appName* — назва браузера;
- *JavaEnabled* — визначає, чи включена підтримка Java;
- *CookieEnabled* — визначає, чи включена підтримка *Cookie*.

Методи:

- *plugins* — визначення встановлених плагінів;
- *mimeType* — для визначення підтримуваних типів даних; У кожного елемента масиву *MimeType* є три властивості:
  - *type* — назва;
  - *description* — докладний текстовий опис;
  - *suffixes* — розширення імен файлів.

Об'єкт Object

Призначений для забезпечення загальної функціональності всіх об'єктів JavaScript.

Методи:

- *toString*
- *valueOf* — повертає або самий об'єкт, або його значення.

Властивості:

- *constructor* — задає конструктор для об'єкта (застосовується для створення екземплярів об'єктів);
- *prototype* — задає прототип об'єкта.

Ці властивості є у всіх об'єктів, крім *Math* і *Global*. Властивість *constructor* використовується для перевірки засобу створення даного об'єкта. Властивість *prototype* забезпечує доступ до прототипів методів об'єктів. За допомогою нього можна створювати нові методи об'єктів.

Об'єкт Global

Цей об'єкт використовується для об'єднання в один об'єкт декількох глобальних методів і властивостей. Не має конструктора. При звертанні до методів і властивостей ім'я Global не вказується.

Властивості:

- *NaN* — містить значення Not a Number;
- *infinity* — містить $+\infty$.

Методи:

- *escape()* — перетворює рядки таким чином, щоб вони читалися на всіх платформах (докладно про escape-послідовності *див.* у Додатку 4);
- *eval()* — виконує рядок таким чином, якби це було вираз мовою JavaScript;
- *isNa()* — визначає, чи є числом аргумент;
- *isFinite()* — визначає, чи є аргумент скінченним;
- *parseFloat()* — перетворює рядок у тип *Float*;
- *parseInt()* — перетворює рядок у тип *Integer*.

Зазначені функції є вбудованими функціями мови JavaScript.

Приклад:

```
var mystr="2001*3%5";
alert("Результат"+mystr+"="+eval(mystr)).
```

Якщо рядок, переданий функції, містить цифри, +, -, то функція повертає *float*. Якщо в рядку є неприпустимі символи, то перетворення проводиться до першого неприпустимого символу. У випадку, якщо перший символ неприпустимий, функція поверне 0.

Метод *escape()* використовується при передачі з форми. Весь набір символів розділяється на три діапазони:

1) символи латинського алфавіту, цифри, арифметичні знаки залишаються без змін;

2) символи з кодом менше *0xFF* перетворюються до такого формату:

%xy,

де *xy* — шістнадцятковий код даного символу;

3) символи з кодом більше *0xFF* перетворюються таким чином:

%uxyzw,

де *u* — UNICODE, *xyzw* — шістнадцятковий код.

Наприклад:

Petrenko I. I. → *Petrenko%20I. I.*

Функція *unescape()* призначена для зворотного читання.

5.5. Засоби задання власних функцій у JavaScript

Елементарною одиницею взаємодії з DOM є функція.

Елемен*т function* використовується для створення власних функцій і об'єктів. Функції є підпрограмами, що можуть викликатися з коду мовою JavaScript:

function NameOfFunc() {код функції} — оголошення функції.

Функція може викликатися з іншого документа, із поточного, за подією.

Функцію можна оголосити й іншим способом:

var MyFuncName=new Function('x1',..., 'xn', 'дія'),

де *x1*, ..., *xn* — аргументи функції, а *дія* — дія з цими аргументами.

Наприклад:

var Minus=new Function('x', 'y', 'z', 'x-y-z').

Перевага такого способу: виконуваний код є останнім елементом конструктора. Недолік: елементарні дії простіше виконувати безпосередньо.

Такі функції називаються *динамічними* (тимчасове створення функції).

Наприклад, можна створити функцію для перевірки правильності заповнення полів форми перед її відсиланням на сервер:

```
<script language="JavaScript">
function SendForm()
{
if(Check())window.document.forms[0].submit;
}
function Check()
{
var doc = window.document;
if (doc.forms[0].elements[0].value == ""||
doc.forms[0].elements[1] value == ""||
doc.forms[0].elements[2].value == "")
{
alert('Поля не можуть бути порожніми');
return false;
}
else
return true;
}
</script>
```

У прикладі функція *Check* виконує перевірку перших трьох полів форми і, якщо хоча б одне з них не містить інформації, виводить попереджуюче повідомлення.

Функція *SendForm* (у якій викликається функція перевірки *Check*) «прив'язується» до форми в такий спосіб:

```
<form name="rqform" action="request.dll/doform" method="post">
  <b> введіть інформацію:</b>
<input type="Submit" value="Надіслати"
          onClick="return SendForm()">
<input type="Reset" value="Відмінити"
           onClick="history.back(-1)"
</form>
```

В оброблювачі натискання кнопки «Надіслати» вказано, що повинна викликатися функція *SendForm*:

```
onClick='return SendForm()'.
```

5.6. Базові події JavaScript. Оброблювачі подій

Події й оброблювачі подій є дуже важливою частиною для програмування мовою JavaScript. Події, головним чином, ініціюються тими чи іншими діями користувача. Якщо він клацає по деякій кнопці, відбувається подія *Click*. Якщо покажчик миші перетинає яке-небудь посилання гіпертексту — відбувається подія *MouseOver*. Існує кілька різних типів подій.

Ми можемо змусити JavaScript-програму реагувати на деякі з них. І це може бути виконане за допомогою спеціальних програм обробки подій. Так, у результаті клацання по кнопці може створюватися вікно, що випадає. Це означає, що створення вікна має бути реакцією на подію *Click*. Програма-оброблювач подій, яку ми повинні використовувати в даному випадку, називається *onClick*. І вона повідомляє комп'ютеру, що потрібно робити, якщо відбудеться дана подія.

Наведений нижче код представляє простий приклад програми обробки події *onClick*:

```
<form>
<input
        type="button"
         value="Click me"
         onClick="alert('Попередження!')">
</form>
```

Наведений приклад має кілька нових особливостей — розглянемо їх один за одним. Можна побачити, що створюється деяка форма з кнопкою. Перша нова особливість — це *onClick="alert('Попередження!')"* у тегу *<input>*. Цей атрибут визначає, що відбувається, коли натискають на кнопку. Таким чином, якщо має місце подія *Click*, комп'ютер має виконати команду *alert('Попередження!')*.

Функція *alert()* дозволяє створювати вікна повідомлень. Під час її виклику необхідно в дужках задати деякий рядок. У нашому випадку це *'Попередження!'*. Це той текст, що з'явиться у вікні, що випадає (рис. 5.2).

Нагадаємо особливості використання лапок, одинарних та подвійних: немає значення, у якому порядку використовувати лапки — спершу подвійні, а потім одинарні чи навпаки.

Тобто можна точно так само написати і *onClick='alert("Попередження!")'*.

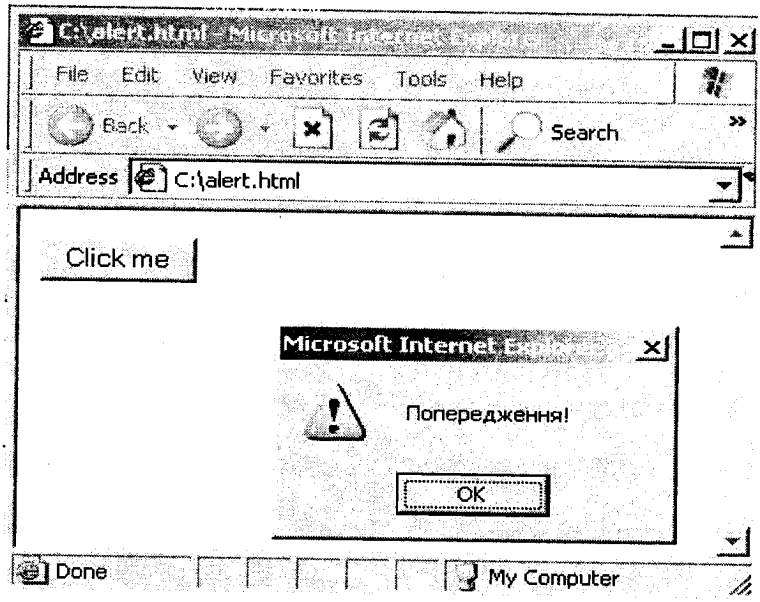


Рис. 5.2. Демонстрація роботи функції *alert*
(після натискання кнопки *Click me*)

Наведемо загальні оброблювачі подій у табл. 5.7.

Таблиця 5.7

Оброблювачі подій JavaScript

Оброблювач події	Значення
onBlur	Подія втрати «фокуса» якимось об'єктом
onChange	Подія зміни якогось значення
onClick	Подія клацання мишею на об'єкті
onFocus	Подія одержання «фокуса» якимось об'єктом
onLoad	Подія завантаження об'єкта
onMouseOver	Покриття мишею зони якогось контейнера
onSelect	Подія виділення якогось пункту меню
onSubmit	Подія відсилки даних з форми
onUnload	Подія вивантаження об'єкта

Існують такі схеми роботи з подіями:

1) визначення події як атрибут тегу

<object onLoad="ім'я функції-оброблювача">;

2) створення процедур, призначених для глобальних подій;

3) призначення конкретних сценаріїв для конкретних подій за допомогою тегу *<script>*. Для цього використовуються його додаткові атрибути *event* та *for*.

Атрибут *event* призначений для оголошення подій, а атрибут *for* визначає, для якого об'єкта на сторінці використовуватиметься сценарій при виникненні події. Окрім цих атрибутів додатково можна використати атрибут *eventname*, який задає псевдонім події.

Наприклад:

```
<script  event="MouseOver"
  for="mylabel"
      language="JavaScript"
      src="URL"
      eventname="ім'я методу">;
```

4) будь-яка подія для будь-якого елемента управління може використовуватися як атрибут тегів *<a>*, *<body>*, *<input>*, *<object>*. Тоді значенням події є процедура.

Наведемо чотири практичні приклади роботи з подіями.

Приклад 1. При наведенні мишею на текст «Завершити роботу браузера», браузер закриється.

```
      <a      href="url"
              onMouseOver="window.close('')">
Завершити роботу браузера
      </a>
```

Приклад 2. Створення банера.

Банер — це «висяча» на сторінці реклама у вигляді графічного об'єкта, що є маршрутизатором гіперпосилання. Наведемо приклад скрипту, що змінює картинку банера при наведенні на неї мишою.

Скрипт складається з двох частин. Перша частина вставляється в документ тільки один раз. А другу необхідно вставити стільки разів, скільки на сторінці буде рисунків, що змінюються.

Перша частина:

```
<SCRIPT LANGUAGE="JavaScript">
browser_name = navigator.appName;
browser_version = parseFloat(navigator.appVersion);
if (browser_name == "Netscape" && browser_version >= 3.0) { roll =
'true'; }
```

```
else if (browser_name == "Microsoft Internet Explorer" &&
browser_version >= 3.0) { roll = 'true'; }
else { roll = 'false'; }
function over(img,ref) { if (roll == 'true')
{ document.images[img].src = ref; } }
function out(img,ref)  { if (roll == 'true') { document.images[img].src =
ref; }
}
if (roll == 'true')
{
a1=new Image;a1.src="image1.gif";
a2=new Image;a2.src="image2.gif";
...
aX=new Image;aX.src="imageX.gif";
}
</SCRIPT>
```

Наприкінці першої частини скрипту здійснюється підвантаження зображень натиснутих кнопок. Необхідно довантажити картинки натиснутої і ненатиснутої кнопок.

```
...
a=new Image;a.src="image.gif";
```

Ці два рядки показують те, що необхідно прописати підвантаження усіх рисунків, замість *X* повинне бути число.

Друга частина:

```
<A HREF="page.htm"
      onMouseOver="over('image_name','image2.gif');"
      onMouseOut="out('image_name','image1.gif');">
   <img
            name="image_name"
            alt="текст"
            src="image1.gif"
            width=x
            height=y
            border="0">
</A>
```

Приклад 3. Забезпечення динамічної сумісності сторінки з браузером.

Наведений нижче скрипт здійснює пересилання сторінки на іншу, залежно від типу браузера у користувача:

```
<script language="JavaScript">
 if (navigator.appName == "Netscape")
```

```
    window.location.href = "index1.htm";
    else
  if (navigator.appName == "Microsoft Internet Explorer")
window.location.href = "index2.htm";
    else window.location.href = "index3.htm";
</script>
```

Приклад 4. Перевірка правильності заповнення формуляра.

Перевірити, чи правильно користувач вказав у полі введення свою електронну адресу. У найпростішому випадку це вирішується перевіркою рядка на наявність символу @.

```
<script language="JavaScript">
function check()
{
 if(document.myform.mytext.value.indexOf('@')==-1)
 {
 alert ("Неправильний формат e-mail адреси!");
return;
 }}
</script>
```

Для роботи скрипту створюємо форму *myform:*

```
<form name=myform>
 <input
      type=text
      size=20
      name=mytext>
 <input
       type=button
      value='Перевірити!'
      onClick='check();'>
</form>
```

Якщо необхідно перевіряти кілька форм, то можна написати функцію один раз і викликати її з бібліотеки:

```
<script language="JavaScript">
 function check(form_name, i, checked_text, error_message)
{
var myf=window.document;
if(myf.myform.elements[i].value.indexOf(checked_text)==-1)
{
alert (error_message);
return;
}
}
</script>
```

Приклад виклику у формі:

```
<FORM bgcolor="green" name="myform">
    < input
type ="text" size=20 name="mytext">
    < input
type ="text" NAME="mytext2">
    < input
type ="text" NAME="mytext3">
    < input
type ="text" NAME="mytext4">
    <input
type=button
value="Перевіритиі!"
onClick="check('myform1', '0','@','Неправильна
адреса!');check('myform1',    '1','***','Неправильні дані!');">
</FORM>
```

Результат роботи наведено на рис. 5.3.

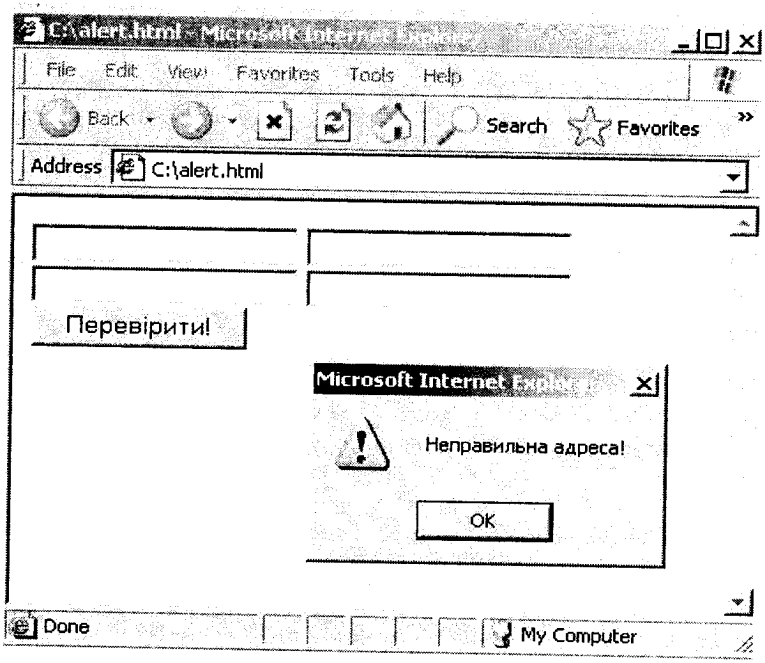


Рис. 5.3. Демонстрація роботи скрипту,
який перевіряє правильність заповнення усіх полів форми

5.7. Особливості використання мови VBScript

VBScript — це мова програмування сценаріїв, заснована на синтаксисі мови Basic.

Розглянемо основні концепції мови:

1. Основний функціональний блок — процедура. Є два типи процедур:

а) *sub* — це послідовність операторів, укладених між *sub* і *endSub*. Процедури *sub* можуть приймати параметри, що передаються під час виклику процедури, але не повертають значення;

б) *function* — це послідовність операторів, укладених між *function* і *endfunction*. На відміну від *sub* можуть повертати значення.

2. Оператор присвоєння є основним оператором мови.

3. Встановлено наступний порядок вставки сценарію до web-документа:

```
<head>
<script language="VBScript"> код мовою VBScript </script>
</head>
```

Якщо потрібно підключити сценарій із зовнішнього джерела, застосовують атрибут src:

```
<script language="VBScript" src=url/*.vs>.
```

Об'єктна модель мови VBScript

Об'єктна модель заснована на синтаксисі:

ім'я_об'єкта. метод | властивість | подія.

При використанні об'єктної моделі браузера синтаксис не відрізняється від синтаксису JavaScript.

Наприклад, для виклику вікна повідомлення:

```
window. alert(' ').
```

При використанні подій у *VBScript* використовують такі ж прийоми, як і у *JavaScript*.

5.8. Контрольні запитання

1. З якою метою розроблена мова JavaScript?
2. Порівняйте JavaScript та Java.
3. Які особливості роботи скриптових програм?
4. Як підключати сценарії до web-документа?
5. Які об'єкти входять до об'єктної моделі мови JavaScript?
6. Які типи даних характерні для JavaScript?
7. Як оголошуються змінні в JavaScript?
8. Що таке глобальна та локальна область видимості змінних?
9. Як задавати дату в JavaScript?
10. Які існують способи задання функцій?
11. Що таке динамічна функція?
12. Що таке вбудовані функцій JavaScript?
13. Які базові події підтримуються в JavaScript?
14. Які способи створення оброблювачів подій?
15. Як підключити бібліотеку скриптів?
16. Що таке банер і як його створювати?
17. Як забезпечується сумісність з різними типами браузера?
18. Які види процедур є в VBScript?
19. Які базові події підтримуються в JavaScript та VBScript?

5.9. Завдання для самостійного виконання

Необхідно розробити і помістити до web-сторінки такі функції мови JavaScript:

1) функцію для підтвердження переходу за гіперпосиланням;

2) функції для динамічного управління фреймами;

3) функція для управління завантаженням документа залежно від властивостей об'єкта *screen*;

4) функція для роботи з об'єктом *Date* (на вибір: годинник, дата, календар, лічильник часу і т. п.);

5) сценарій для перевірки типу браузера і динамічного завантаження сторінки;

6) сценарій для перевірки правильності заповнення форми;

7) сценарій для динамічного управління фреймами;

8) сценарії для використання стандартних подій;

9) сценарій для динамічного управління графічними об'єктами.

Розділ 6
ОБ'ЄКТНА МОДЕЛЬ БРАУЗЕРА ТА DHTML

6.1. Об'єктні моделі браузерів Internet Explorer, Netscape Navigator та динамічний HTML

Динамічний HTML — це прикладний програмний інтерфейс (англ. — *API*) для управління поведінкою тегів і елементів HTML-документа.

Існує багато способів виведення даних — наприклад, табличний. Але при цьому документ залишається статичним. Для надання web-контенту динамічності можна написати, наприклад, скрипт мовою JavaScript, що виводить повідомлення в рядок стану браузера.

Іншим способом забезпечення динамічності є використання форми, яка обробляється за допомогою CGI (*Common Gateway Interface*). Можна навіть написати скрипт, застосувавши технологію Active. Але для дійсно інтерактивного документа необхідно мати можливість управляти кожним елементом HTML.

Використовуючи динамічний HTML, розробник одержує можливість оперативно перевизначати властивості індивідуальних елементів. Обмеження, що накладаються на елементи теговою моделлю, зникають. Тепер розробник сам визначає, що і як теги робитимуть з документом.

За допомогою динамічного HTML можна створювати скрипти, що обробляють дії користувача й інші поточні події. Подія буде запускати скрипт, що практично миттєво змінює вигляд елемента і не потребує перезавантаження всієї сторінки.

Робота DHTML. Якщо для створення документа використовується динамічний HTML, всі елементи документа вважаються *об'єктами.*

Будь-який елемент можна назвати *підоб'єктом* головного або супероб'єкта, яким є документ. У документ можуть вміщуватися частини з інших файлів HTML. Вони також вважаються підоб'єктами головного об'єкта.

Прикладом такого зовнішнього елемента-підоб'єкта може служити таблиця стилів. Кожний елемент CSS вважається породженим елементом об'єкта CSS.

Кожний тег HTML також є об'єктом. До цих об'єктів можна звертатися і маніпулювати ними декількома методами. Динамічний HTML дозволяє безпосередньо змінювати властивості тегів.

Створений за допомогою динамічного HTML документ є об'єктом, який можна використовувати для маніпуляції документом, наприклад, для включення нового елемента.

Скрипти також є об'єктами. Вони можуть містити власні підоб'єкти.

Використовуючи об'єктно-орієнтоване програмування, можна значно розширити можливості створення HTML-документів. За сигналом таймера або визначеною дією користувача (наприклад, при наведенні миші на якийсь елемент документа) запускається скрипт, який обробляє цю подію.

Об'єктна модель динамічного HTML. Браузер для динамічної зміни вигляду документа повинен мати якийсь відправний пункт. Оскільки документ має постійно оновлюватися без зупинок, то між браузером і фізичним компонуванням документа повинен існувати спеціальний інтерфейс.

Єдиним способом, при якому програма перегляду може управляти виглядом документа, є збереження його структури, яка може модифікуватися за допомогою скрипту, що запускається у відповідь на яку-небудь подію.

Колекція, що може редагувати скрипт, автоматично створюється під час прийому документа. На початку прийому документа браузер переглядає контейнер *<head>* у пошуках інструкцій, де містяться звертання до документа (такі, як колір тексту і фону, таблиць стилів, зв'язків тощо).

Потім програма перегляду звертається до тіла документа і збирає відомості про всі елементи HTML, скрипти і таке інше. З цих даних будується масив, що містить інформацію про позицію і порядок проходження кожного елемента в документі. І, нарешті, браузер виводить документ на екран.

Щоб реалізувати технологію DHTML на практиці, розробник повинен добре знатися з *об'єктною моделлю браузера*.

На сьогоднішній день найбільш поширеними браузерами в світі є *Microsoft Internet Explorer* та *Netscape Navigator*. Досить

популярними є також браузери *Opera* та *Mozilla*. Існує також багато так званих псевдобраузерів, які використовують об'єктну модель перелічених вище браузерів, надаючи тільки новий інтерфейс користувача. Розглянемо більш докладно об'єктні моделі поширених браузерів.

Браузер Microsoft Internet Explorer на сьогодні існує у таких версіях:

- Internet Explorer 3.0 (3.1) — для операційної системи Windows 95. Цей браузер ще не реалізовує технологію DHTML;
- Internet Explorer 3.0 (3.1) — це перший DHTML-браузер;
- Internet Explorer 5.0, 5.5, 6.0 — це DHTML-браузери, які вже більш повно відповідають вимогам технології DHTML.

Усі вбудовані об'єкти DHTML-браузера Internet Explorer, починаючи з версії 4, можна представити у вигляді ієрархії (рис. 6.1). Більшість програм використовують цю систему класів і не створюють нових.

Як видно з рисунка, об'єктами, які надають можливості для DHTML, є об'єкти *document, event, screen*.

Розглянемо дочірні об'єкти об'єкта *document:*

- *all* — колекція всіх тегів;
- *anchors* — масив гіперпосилань;
- *applets* — масив об'єктів-аплетів;
- *body* — утримує тіло сторінки;
- *forms* — відповідає за всі форми;
- *frames* — відповідає за всі фрейми;
- *images* — відповідає за об'єкти-зображення;
- *links* — відповідає за всі зв'язки документа;
- *selection* — реалізує поля вибору;
- *scripts* — відповідає за контейнери *script*;
- *stylesheets* — працює з таблицями стилів.

Для *Netscape* об'єктна модель має вигляд, показаний на рис. 6.2.

Як видно з рис. 6.1, з'явився незвичний об'єкт — колекція *all*. Розглянемо більш докладно його особливості з точки зору DHTML.

Колекція дозволяє інтерпретувати різні аспекти документа більш організовано порівняно зі звичайним статичним HTML-документом. Ось як це працює:

1. Браузер створює колекцію, схожу на індексований список.
2. Елементи, теги і класи сортуються.

3. Відсортовані дані індексуються в порядку їхньої появи в документі.

4. Відсортовані дані також індексуються за виставленими ключами (об'єкти забезпечують методи доступу до батьківських і підлеглих елементів).

Колекція складається з елементів, тегів і класів. Класифікація методів збору колекції описана в таблиці 6.1.

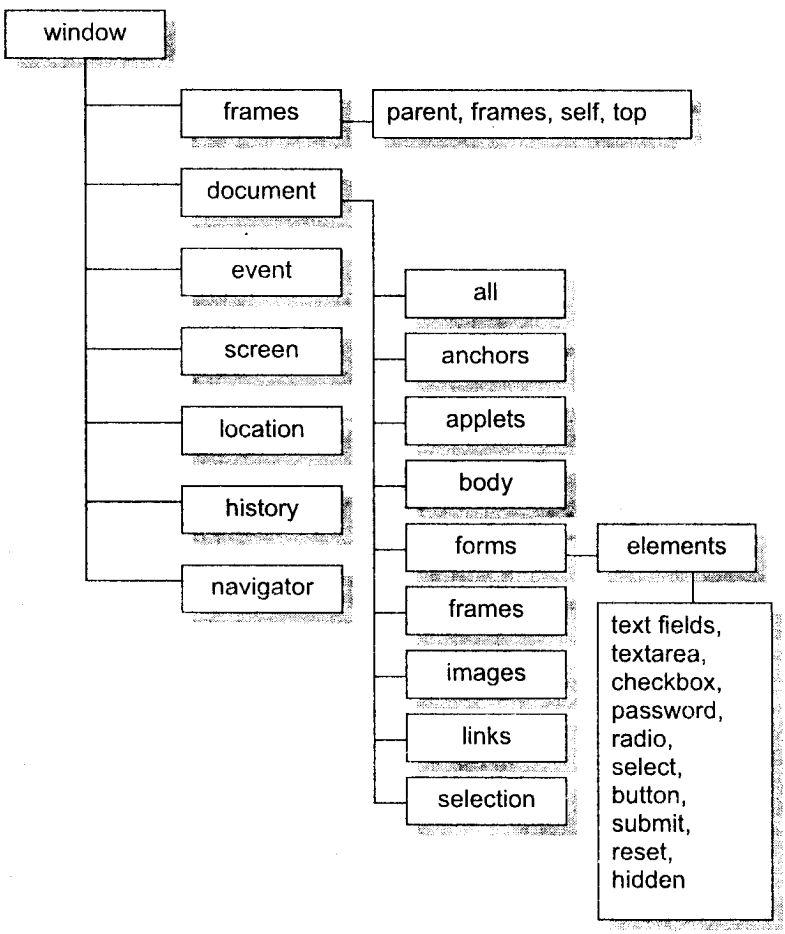


Рис. 6.1. Об'єктна модель
браузера Microsoft Internet Explorer 4.0

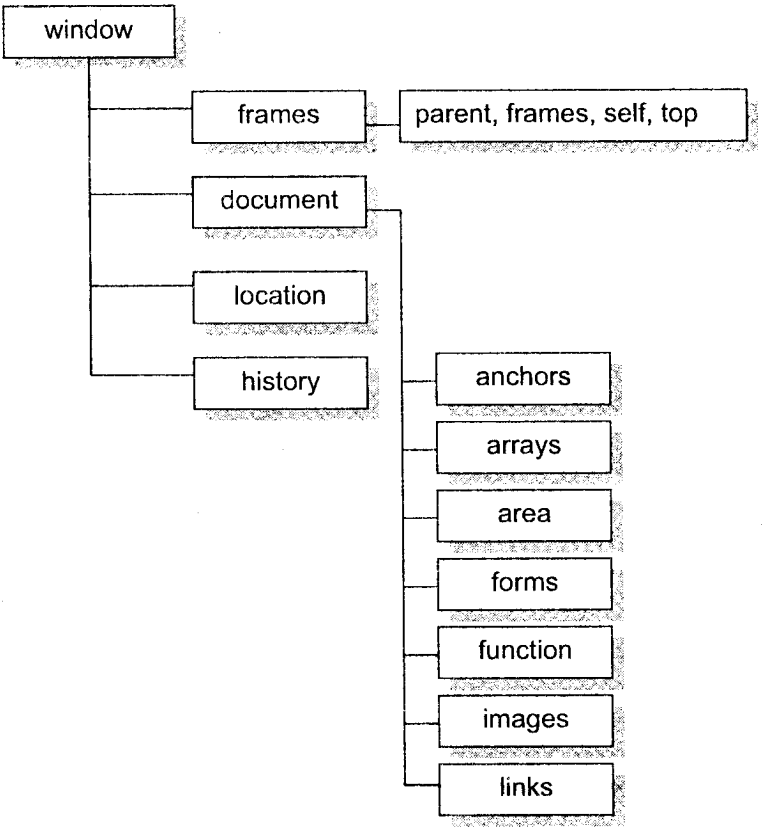


Рис. 6.2. Об'єктна модель браузера Netscape

Таблиця 6.1

Класифікація методів збору колекції

Методи	Опис
Елемент	Використовується для пошуку окремого елемента в документі. Якщо елемент входить у документ багаторазово, то він повертає колекцію, якщо один раз, то повертає елемент
Тег	Використовується для створення колекції тегів і повертає колекцію, що складається із зазначених тегів. Цей метод нечутливий до регістра клавіатури
Класи	Повертає колекцію класів, що складається із зазначених класів. Цей метод нечутливий до регістра клавіатури

Колекції завжди представляють поточний стан документа. Колекція, складена при завантаженні документа, автоматично оновлюється при будь-якій маніпуляції користувача.

6.2. Методи і властивості об'єкта window

Об'єкт *window* завжди є об'єктом верхнього рівня. Це легко зрозуміти, якщо співвіднести модель з тим, що реально бачить користувач на екрані.

Насамперед, він бачить вікно браузера, потім документ, що розташовується у вікні, форми всередині документа і т.д.

Об'єкт *window* має цілий набір властивостей, методів і подій.

Властивості цього об'єкта представлені в таблиці 6.2.

Таблиця 6.2

Властивості об'єкта *window*

Властивість	Опис
defaultStatus	Повертає текст, заданий у рядку статусу
status	Задає текст, що відображається в панелі стану
name	Задає ім'я вікна (ім'я, задане в тегу *frameset*)
length	Повертає кількість фреймів у батьківському вікні
top	Повертає посилання на верхнє за ієрархією вікно
parent	Повертає посилання на батьківське вікно
self	Повертає посилання на поточне вікно
closed	Вказує на те, що вікно було закрито
opener	Повертає посилання на вікно, яке створило поточне вікно
clientInformation	Посилання на об'єкт *Navigator*
returnValue	Дозволяє події або діалоговій панелі повертати значення

Об'єкти *screen, location, history, document, frames* також є властивостями об'єкта *window*.

Методи об'єкта window:

1) *alert('message')* — виводить вікно повідомлення з рядком message і кнопкою *OK* для закриття; *message* — це або рядок, або властивість одного з існуючих об'єктів;

2) *confirm(message)* — виводить вікно повідомлення з рядком message і кнопками *OK* і *Cancel* для закриття. Повертає «істинне значення» для *OK* та «хибне» — для *Cancel*;

3) *prompt("message","DefaultStr")* — виводить діалогове вікно, що містить рядок *message*, кнопки *OK* чи *Cancel* і поле для введення рядка, у якому відображається текст, заданий у *DefaultStr*. Якщо параметр *DefaultStr* не задано, то рядок міститиме значення за замовчуванням (*undefine*). Повертає рядок, що введено, або *NULL*, якщо рядок не був введений і не задане значення за замовчуванням;

4) *open("URL", "windowName", ["windowFeatures", replace])* — відкриває нове вікно браузера і завантажує в нього зазначений у URL документ. Параметри цього методу:

- *windowName* — ім'я, що присвоюється вікну, яке відкривається;
- *windowFeatures* — параметри вікна, що відкривається, (необов'язкові):
  - *toolbar[=yes|no]|[=1|0]* — чи показувати у вікні панель інструментів;
  - *location[=yes|no]|[=1|0]* — чи показувати рядок адреси відкритого документа;
  - *directories[=yes|no]|[=1|0]* — чи виводити список каталогів;
  - *status[=yes|no]|[=1|0]* — рядок стану;
  - *menubar[=yes|no]|[=1|0]* — панель меню;
  - *scrollbars[=yes|no]|[=1|0]* — смуги прокручування;
  - *resizable[=yes|no]|[=1|0]* — чи можливо змінювати розміри вікна за допомогою миші;
  - *channelmode [=yes|no]|[=1|0]* — рядок каналів;
  - *width* та *height* — ширина та висота вікна;
- *replace[=yes|no]|[=1|0]* — показує, чи заміщує документ, що відкривається, попередній документ у списку history.

Зауваження

Якщо вказано хоча б один з параметрів, то всі інші незазначені параметри вважаються заданими зі значеннями *'0'*;

5) *ID.close() або windowName.close()* — закриває поточне вікно браузера або вікно, що було до цього відкрите методом *open()*;

6) *setTimeout("actions or function", time, language [for actions only])* — запускає таймер після закінчення *time*, заданого у *мілісекундах*, викликає функцію *function*;

7) *clearTimeout([ID])* — зупиняє таймер, створений методом *ID=setTimeout()$*

8) *setInterval("actions or function", interval, language [for actions only])* — запускає таймер і повторює зазначені дії з визначеним інтервалом;

9) *clearInterval(ID)* — зупиняє таймер, створений методом *setInterval()*;

10) *showHelp('URL')* — викликає файл довідки **.hlp* за адресою URL;

11) *scroll(x,y)* — запускає програмний скролінг на задану в *x* та *y* кількість пікселів;

12) *showModalDialog(URL,argums,reatures)* — викликає модальне вікно діалогу;

13) *navigate('URLl')* — завантажує нову сторінку;

14) *focus* — перекладає фокус на дану сторінку;

15) *blur* — викликає втрату сторінки і фокуса.

Додаткові методи об'єкта window. Розміщення в заданих координатах і з заданим розміром:

- *moveTo(x, y)*;
- *resizeTo(x,y)*.

До основних ***подій об'єкта window*** відносять *onLoad* та *onUnload*.

Наведемо приклад, який ілюструє роботу методів і властивостей *window*. Створимо два файли зі сценаріями.

Файл *opener.html*:

```
<script language="JavaScript">
open("closer.htm","myWindow","resizeable=no, width=200,heiht=50" );
</script>
```

Файл *closer.html*:

```
<script Langauge="JavaScript">
        setTimeout("opener.close()",5000,"JavaScript")
</script>
```

У цьому прикладі перший файл відкриває вікно 100*200, яке у свою чергу відкриває інше вікно з ім'ям *myWindow* і завантажує в нього файл *closer.htm*. Вікно *myWindow* закриває попередній файл через п'ять секунд після свого відкриття.

6.3. Властивості, методи, події і колекції об'єкта *document*

Об'єкт *document* являє собою web-сторінку, яку користувач буде бачити у вікні браузера, разом із текстом, маршрутизаторами посилань, формами та іншими елементами, які є на сторінці. Об'єкт *document* складається з властивостей, методів і колекцій.

Властивості об'єкта *document* поділяють на три групи:

1. Загальні властивості (інформація про сам документ):
   - *last/modified* — дата останньої модифікації документа;
   - *domain* — використовується для управління фреймами, якщо в кожному фреймі відображаються документи з різних джерел;
   - інші властивості (*див.* табл. 6.3).
2. Властивості кольору:
   - *bgColor* — колір фону;
   - *fgColor* — колір шрифту;
   - *linkColor* — колір гіперпосилання;
   - *alinkColor* — колір переглянутого гіперпосилання;
   - *vlinkColor* — колір посилання в момент одержання фокуса і натискання кнопки.
3. Додаткові властивості, які дозволяють одержати інформацію про активний елемент документа, що посилається:
   - *activeElement* — повертає елемент сторінки, що має фокус у даний момент. Елемент може одержати фокус за допомогою миші або за допомогою методу *focus*;
   - *referrer* — повертає url сторінки, що посилається на поточну сторінку;
   - *readyState* — властивість, що сповіщає про готовність. Набуває значення *true* або *false*.

Методи об'єкта *document* поділяють на такі групи:

1. Методи загального призначення:
   - *write()*;
   - *writeln()*;
   - *open*;

- *close*;
- *createElement*;
- *elementFromPoint*;
- *clear*.

2. Командні методи — методи, що дозволяють дізнатися, чи доступна команда, та інші; призначені для управління діапазонами.

Події об'єкта *document* класифікують таким чином:

1. Події, пов'язані з діями миші:
   - *onClick*;
   - *onMouseDown*;
   - *onMouseMove*;
   - *onMouseOut*;
   - *onMouseUp*;
   - *onDragStart*;
   - *onSelectStart*.
2. Події, пов'язані з клавіатурою:
   - *onKeyDown*;
   - *onKeyUp*;
   - *onKeyPress* — повертає ASCII-код з урахуванням клавіш *Ctrl*, *Alt*, *Shift*;
   - *onHelp*.
3. Зміна значень, завантаження, вивантаження:
   - *onReadyStateChange* — змінює властивості *ReadyState*;
   - *onBeforeUpdate* — відбувається попереднє відновлення компонента;
   - *onAfterUpdate* — відбувається послідовне відновлення компонента;
   - *onLoad* — відбувається по завершенні повного завантаження документа.

Колекції

Колекція — це масив різнотипних об'єктів. Існують 12 колекцій для доступу до інформації:

- *all* — усі теги тіла документа;
- *anchors* — деякі теги документа;
- *applets* — усі аплети;
- *embeds* — усі об'єкти, вмонтовані за допомогою тегу *<embed>*;
- *frames* — колекція усіх фреймів;

- *forms* — колекція усіх форм;
- *filters* — усі фільтри;
- *images* — колекція графічних елементів;
- *lnks* — колекція гіперпосилання і карт-зображень;
- *scripts* — скрипти поточного документа;
- *styleSheets* — таблиці стилів.

Особливості колекцій:

1. Колекції — набори елементів у HTML-документі.

2. Колекції підтримують методи для створення підмножини на основі тегів, класів або ідентифікаторів.

3. Колекції дозволяють звертатися до кожного елемента за іменем або ідентифікатором.

Властивість для роботи з колекціями:

length — кількість елементів у колекції.

Методи для роботи з колекціями:

- *tags()* — повертає колекцію, що містить визначені теги мови HTML. Наприклад:

  tags('h3').

- *item()* — всі об'єкти, що відповідають даному індексу або ідентифікатору в колекції. Наприклад:

  var x=document.all[5].

Колекцію *all* використовують у таких випадках:

1. Для одержання повного списку тегів, що складають даний документ. Список слід виводити в циклі і друкувати імена тегів. Їх можна одержати за допомогою властивості *tagname*, а виводити краще у відкрите вікно.

Наприклад:

```
var newwind=window.open('about:blank');
newwind.document.write(...).
```

2. Для кожного тегу з глобальної колекції доступні всі його атрибути, тому можна одержати доступ до атрибутів тегів, наприклад, дізнатися значення атрибута *align*:

```
h1=document.all.tags('h1');
newwind=window.open('about.blank');
newwind.document.write(h1[i].align).
```

Властивості і методи об'єкта *document*, які часто використовуються, зведені до таблиці 6.3.

Таблиця 6.3

Властивості та методи об'єкта *document*

Найменування		Опис
Властивості	*linkColor*	Колір неактивного посилання
	alinkColor	Колір активного посилання
	vlinkColor	Колір переглянутого посилання
	bgColor	Колір фону
	fgColor	Колір шрифту
	all	Колекція всіх елементів об'єкта *document*
	anchors	Масив усіх якорів (міток) документа
	links	Масив об'єктів типу *link*, доступ до елементів якого дозволений тільки для читання. Наприклад, звертання до третього посилання в документі: *document.links(2)*
	forms	Масив усіх форм документа
	location	URL-адреса документа
	lastModified	Дата останньої зміни документа
	title	Рядок-заголовок документа
	cookie	Кукіс-документ (*див.* підрозділ 6.5)
	referrer	URL-адреса документа, з якого був викликаний поточний документ при переході за посиланням
	activeElement	Вказівка на активний у даний момент об'єкт документа
	readyState	Повертає поточний стан документа, що завантажується
Методи	*write (вираз, [вираз...])*	Записує в тіло документа значення довільного числа виразів або констант
	writeln ()	Те ж, що і *write*, але наприкінці виведеного тексту додає перехід на новий рядок
	close	Закриває потік виводу, відкритий методом *open()*
	clear	Цілком очищає весь вміст документа
	open([mime-type])	Відкриває потік вводу, знищуючи весь колишній вміст документа, якщо він існував до цього. На відміну від однойменного методу об'єкта *window*, має один параметр
	createElement	Створює екземпляр об'єкта для зазначеного тегу
	elementFrom-Point(x,y)	Повертає елемент, що знаходиться в координатах *x* та *y*

Робота з об'єктом *document*

Наведемо приклад скрипту, що виводить дату оновлення документа при кожній його зміні:

```
<SCRIPT LANGUAGE="JavaScript">
document.writeln(document.lastModified)
</SCRIPT>
```

Як ви вже знаєте, доступ до одного з декількох однотипних об'єктів виконується за допомогою індексів масиву. Можна визначити кількість елементів у такому масиві шляхом використання властивості *length*. Наприклад:

document.forms.length.

Кожен елемент такого масиву є окремим об'єктом даного типу. Наприклад, для того щоб одержати доступ до першого посилання в документі, слід записати:

document.links(0).

Оскільки для скриптів власником є об'єкт *window*, а не *document*, то посилання на властивості і методи повинні виконуватися в такий спосіб:

document.name.

Не можна опускати частину рядка, що містить *document.**, так, як це можна робити для *window*.

Розглянемо окремо підоб'єкти об'єкта *document*.

Об'єкт links — це масив об'єктів типу *link*, відсортованих у порядку їх появи. Властивості об'єкта збігаються з властивостями *location* тільки щодо посилань.

Об'єкт anchor. Властивість *anchor* об'єкта *document* є масивом об'єктів типу *anchor*, доступ до елементів якого дозволено тільки для читання. Одержати доступ можна так само, як і до будь-якого іншого об'єкта, окрім звертання за іменем.

Об'єкт forms. Об'єкт *document* може містити кілька об'єктів типу *form*. Усе залежить від того, чи використовуються теги *<form>* у web-сторінці. Якщо на сторінці використовуються форми, то їхня кількість зберігатиметься в такій властивості:

document. forms.length.

Властивості об'єкта *forms* наведені в таблиці 6.4.

Таблиця 6.4

Властивості об'єкта *forms*

Властивість	Опис
action	Значення однойменного атрибута тегу *form*
elements	Масив всіх елементів форми (*input* і т. д.)
encoding	Значення однойменного атрибута тегу *form*
length	Містить кількість елементів у відповідній формі
method	Значення однойменного атрибута тегу *form*
target	Значення однойменного атрибута тегу *form*

button, checkbox, hidden, password, radio, reset, select, submit, text, textarea також є властивостями об'єкта *form*.

Об'єкт *forms* має єдиний метод — *submit*, який відсилає дані програмі-оброблювачу, і єдину подію *onSubmit()*, що виникає при відправленні форми.

Доступ до об'єктів-елементів форми здійснюється через масив *elements*. Для доступу до них необхідно використовувати повну форму запису:

document. form,

тому що об'єкти *document* і *forms* мають різні області видимості власних об'єктів. Тільки якщо розташувати скрипти всередині форми, то не буде необхідності використовувати повну форму запису для доступу до елементів форми.

Властивості, методи і події об'єкта *element* наведено в таблиці 6.5.

Таблиця 6.5

Властивості, методи і події об'єкта *element*

Властивості	Методи	Події
form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex	*click, focus, blur, select, removeItem, addItem, clear*	*onClick, onFocus, onBlur, onChange onSelect*

Крім зазначених вище об'єктів, браузер має дуже потужний об'єкт *Global*. Він містить глобальні методи (таблиця 6.6)

і властивості. Сам об'єкт при їхньому використанні ніколи не вказується.

Таблиця 6.6

Методи об'єкта *Global*

Метод	Опис
parseInt('string',[base])	Перетворює рядок *string* у ціле число тієї системи числення, яка задана в *base*. Якщо число не може бути перетворене, то метод повертає нуль
parseFloat('string')	Перетворює рядок *string* у число з плаваючою точкою. Якщо число не може бути перетворене, то повертає нуль
escape('string')	Перетворює рядок *string* у *escape*-послідовність
eval('string')	Перетворює рядок *string* у число. Якщо рядок є арифметичним виразом, то обчислює його

6.4. Методи і властивості об'єктів History, Navigator, Location, Event, Screen

Об'єкт history

Містить інформацію про адреси сторінки у форматі URL. Кожне вікно містить об'єкт *history*, у якому зберігається список адрес web-сторінок, переглянутих у даному вікні.

Має властивість *length* (довжина масиву *history*) і такі методи:

- *back* — завантажує попередню сторінку зі списку history;
- *forward* — завантажує наступну сторінку зі списку *history*;
- *go*(n) — завантажує *n*-ту сторінку.

Наприклад:

```
n=history.length;
<a href='JavaScript: history.go(-1)'> // попередня сторінка
</a>
<a href='JavaScript: history. go(-2)'> // передпопередня сторінка
</a>
```

Об'єкт navigator

Кожне вікно містить об'єкт *navigator*, у якому знаходиться інформація про використовуваний web-браузер. Властивості цього об'єкт наведені в таблиці 6.7.

Таблиця 6.7

Властивості об'єкта *navigator*

Властивість	Опис
appCodeName	Кодове ім'я браузера
appName	Ім'я браузера. Наприклад: Microsoft Internet Explorer
appVersion	Версія браузера у форматі: *releaseNumber (platform; country).* Наприклад, для IE 4.0 ця властивість містить рядок 4.0 (compatible; MSIE 4.01; Windows 98)
userAgent	Рядок, що містить інформацію про браузер, яка відсилається за HTTP-протоколом від клієнта серверу
javaEnable	Увімкнена чи ні підтримка Java
cookieEnable	Увімкнена чи ні підтримка кукісів

Методів об'єкт не має.

Об'єкт location

Містить інформацію про URL поточної сторінки. Забезпечує методи для її перезавантаження.

Має такі властивості:

- *href* — містить повний URL сторінки;
- *hash* — містить ту частину адреси, яка йде за символом #;
- *host* — містить частину адреси *"hostname: port"*;
- *hostname* — містить ім'я хосту (IP — адреса вузла);
- *pathname* — містить ім'я файла і шлях до нього;
- *port* — містить ім'я порту;
- *protocol* — містить ім'я протоколу;
- *search* — містить рядок запиту (всі символи після '?').

Для перезавантаження або завантаження іншої сторінки досить змінити значення *href*:

```
window. location. href="url"
```

Методи об'єкта *location:*

- *assign(url)* — завантажує нову сторінку;
- *reload()* — перезавантажує поточну сторінку;
- *replace(url)* — завантажує нову сторінку, при цьому відбувається заміщення поточної сторінки новою сторінкою в списку *history*.

Кожне вікно містить об'єкт *location*, що зберігає URL-адресу завантаженої web-сторінки. Формат адреси:

protocol//hostname:port pathname search hash

Об'єкт script

Скрипти існують усередині об'єкта *window*. Кожен об'єкт такого типу являє собою процедуру або функцію.

Якщо розробник працює зі скриптами всередині фрейму, то можна одержати швидкий і легкий доступ до процедур вікна, використовуючи ім'я вікна в об'єднанні з ім'ям процедури. Наприклад, якщо написаний скрипт використовується у фреймі з великим ступенем вкладеності, то доступ до скрипту вікна найвищого рівня може бути здійснений так:

top.MyProcedure().

Одержати доступ до скриптів інших фреймів можна, використовуючи ім'я фрейму або зарезервоване слово *parent*. Щоб одержати доступ до скрипту другого фрейму поточного вікна, необхідно записати:

frames(1).MyProcedure().

Здійснення доступу до батьківського фрейму виконується так:

parent.MyProcedure().

Об'єкт event

Цей об'єкт дозволяє скриптовій програмі одержувати докладну інформацію про подію, що відбувається.

Об'єкт *event* має такі особливості:

1. Доступний тільки під час самої події.
2. Звертатися до *event* можна тільки з оброблювачів подій або функцій.

Властивості об'єкта *event* наведені в таблиці 6.8.

Методи об'єкта event:

- *returnValue* — дозволяє запобігти дії елемента за замовчуванням;
- *cancelBubble* — булівське значення, яке повідомляє, чи «спливає» подія за ієрархією. Метод можна використовувати для виключення помилкового спрацьовування скриптів.

Таблиця 6.8

Властивості об'єкта *event*

Властивість	Опис
type	Містить тип події
srcElement	Дозволяє визначити джерело події
srcElement. tageName	Повертає ім'я тегу-джерела події
client X	Повертає координати події в клієнтських координатах
client Y	
screenX	Вказують вертикальну і горизонтальну координати події щодо вікна
screenY	
offsetX	Вказують координати події щодо контейнера
offsetY	
x	Горизонтальна і вертикальна координати події
y	
button	Вказує натиснуту кнопку миші
keyCode	Вказує код натиснутої клавіші
altKey	Зберігаються булівські значення, що відповідають натисканню клавіш Alt, Ctrl і Shift
ctrlKey	
shiftKey	
cancelBubble	Булівське значення, вказує, чи відбувається подія в ієрархії об'єкта
fromElement	Зберігається джерело події
toElement	Зберігається приймальник події

Приклад. Необхідно одержати інформацію про подію для всього документа. Наведемо код цього скрипту:

```
<script language='JavaScript'>
 function InfoEvent(){
   var ev=window.event;
   var st=new String;
   st="window.event="+ev.type;
   st+="event.srcElement.tagName"
   alert(st) }
```

Об'єкт screen

Використовується для встановлення екранних властивостей клієнтського браузера. Доступні такі властивості:

- *availHeight* і *availWidth* — повертають відповідно висоту і ширину доступної області екрана користувача в пікселах;
- *colorDepth* — глибина кольору (максимальна кількість кольорів, які підтримуються системою);
- *pixelDepth* — повертає кількість біт на один піксел;
- *bufferDepth* — глибина буфера. Якщо значення *bufferDepth* дорівнює нулю або –1, то значення *colorDepth* дорівнює кількості біт на піксел. Якщо значення *bufferDepth* більше нуля, то *colorDepth* дорівнює цьому числу;
- *updateInterval* — зберігає інтервал відновлення екрана;
- *height* і *width* — повертають відповідно висоту і ширину екрана в пікселах.

Рекомендації з використання властивостей об'єкта screen: їх можна використовувати для адаптації кольорової гами і роздільної здатності на моніторах різних користувачів. Оптимальне рішення — створити декілька таблиць стилів *.css* для різноманітних параметрів.

Приклад для роботи з екранами. Встановити в клієнта свої розміри вікна. Для цього визначити поточну роздільну здатність екрана. Наведемо код скрипту:

```
<script language='JavaScript'>
max=800;
max=600;
ce=screen.availHeight;
ce=screen.availWidth;
```

```
window.moveTo(max/2 — cl/2, 0;
window. resizeTo(max, max)
</script>
```

6.5. Засоби маніпулювання об'єктною моделлю DHTML

Розглянемо класифікацію методів та засобів доступу до об'єктної моделі документа в сучасних Інтернет-проектах:

1. За допомогою спеціальних об'єктів DHTML.
2. За допомогою:

а) сценаріїв, що знаходяться всередині сторінки;

б) сценаріїв, що зберігаються у вигляді віддалених ресурсів (зовнішні сценарії).

3. За допомогою впроваджених об'єктів:

а) аплетів мовою Java;

б) елементів керування (Active-об'єктів).

Нагадаємо, що *сценарій* — це програмний код, поміщений у web-документ, який виконується інтерпретатором браузера.

Технологія cookies

Cookies — механізм, що дозволяє серверу зберігати інформацію на клієнтському комп'ютері і за необхідності використовувати її. Звичайно інформація зберігається у вигляді текстового файла.

У *cookies* звичайно зберігається така інформація: ім'я користувача, параметри настроювань, службова інформація тощо. Частіше за все *cookies* використовується для зберігання інформації про користувача. У файлі *cookies.txt* записуються результати зв'язку браузера з різноманітними серверами.

Для підтримання механізму використовують властивість *document. cookie.*

Властивість *cookie* має такі атрибути:

- *name='value'* — застосовується для збереження інформаційного елемента;
- *expires='date'* — термін придатності інформаційного елемента. Якщо не використовується, то термін придатності закінчується при закритті браузера. Якщо дата більша за поточну, то відбувається зберігання елемента, а якщо менша — то видалення елемента;

- *domain='domain-name'* — задає ім'я домена, із якого завантажується інформаційний елемент;
- *path='path'* — задає шлях до вмісту інформаційного елемента.

Приклад. Подати запит про ім'я користувача, зберегти його у вигляді інформаційного елемента та при наступних відвідинах сайта відображати його у привітанні. Наведемо код цього скрипту:

```
<script language='JavaScript'>
   function MyCook(){
     var myname="Myname";
     if (document.cookies.indexOf(MyName)!=-1)
        start=document.cookie.indexOf(MyName);
        end= document.cookie.indexOf(';');
        myvalue=document.cookie.subString(start, end);
        alert('Вітаю Вас'+unescape(myvalue));
     else
     myname=prompt('Ваше ім'я, 'Студент')
     document.cookie.=ecsape(myname);
     }
</script>
```

Для виклику функції на головній сторінці сайта розміщуємо:

```
<body onLoad='MyCook()'>
```

Cookie зберігають пари *«ім'я — значення»* в одному рядку. Кожна пара завершується ';'.

Для перетворення на пару запишемо:

```
document.cookie="MyName"+escape(MyName)+';'
```

Для коректної роботи пари необхідно в методі *substring* зазначити початок і кінець:

```
substring(start+myname. length+end)
```

Існують такі засоби встановлення значень *cookie*:
- за допомогою скриптів мовою JavaScript;
- за допомогою метатегів;
- за допомогою CGI-скриптів.

Типові задачі, розв'язувані за допомогою *cookies:*

- передача імені користувача, пароля;
- налагодження індивідуального профілю для кожного зареєстрованого клієнта;
- оформлення замовлень в online-магазині;
- використання *cookies* у рекламному бізнесі (для визначення цільової аудиторії тощо).

Недоліки технології *cookies*:

- можливість одержання третіми особами відомостей про клієнта;
- необхідність використання *cookies*-менеджера (для контролю за записами *cookies*).

6.6. Модель подій DHTML

Подія — це дія, виконувана користувачем у межах сторінки. Концепція подій є базовою в DHTML.

Основними подіями є:

- *onClick* — натискання кнопки миші;
- *onFocus* — одержання елементом фокуса;
- *onBlur* — втрата фокуса;
- *onChange* — зміна значень текстового поля;
- *onSelect* — вибір тексту в текстовому полі;
- *onMouseOver* — переміщення курсора в область об'єкта;
- *onMouseOut* — вихід курсора з області об'єкта;
- *onReset* — натискання кнопки типу *Reset* (скидання елементів форми);
- *onSubmit* — для кнопки типу *Submit*;
- *onLoad* — завершення завантаження сторінки або об'єкта;
- *onUnload* — перехід на іншу сторінку або завершення роботи з браузером;
- *onError* — помилка завантаження документа або об'єкта.

Способи визначення подій

Для роботи з подіями потрібно створювати оброблювачі подій. Для їхнього створення використовуються два способи:

1) визначення події за допомогою атрибутів контейнерів HTML. Наприклад,

```
<table onMouseOver='MyFunc()'>
...
</table>
```

2) визначення події через функцію. Наприклад,

```
<script>
    function window.onLoad(){
        код }
</script>
```

Особливістю оброблювача є те, що він одержує управління тільки на час перебігу подій.

Наприклад:

```
<script>
   function PicError() {
   alert("Перевірте з'єднання з мережею")}
</script>
<img src="URL" onError="PicError"()>
```

У DHTML для <script> передбачено нові атрибути: *FOR* і *EVENT*. Вони призначені для пов'язування функцій із подіями:

```
<SCRIPT FOR="ім'я_ID" language='JavaScript' EVENT='ім'я_події'>
```

Властивість *event.type* повертає ім'я події без приставки *on*:

```
event. type='mouseover'
```

Наведемо приклад її використання:

```
switch (event. type)
        {case 'click': код;break}
```

Нагадаємо, що для роботи з мишею існує властивість *event. button*, яка може мати такі значення:

- 0 — кнопки не були натиснуті;
- 1 — натиснута ліва кнопка;
- 2 — натиснута права кнопка;
- 3 — натиснуті обидві кнопки;
- 4 — натиснута середня кнопка.

6.7. Динамічні стилі та візуальні фільтри DHTML

Працюючи в DHTML, можна використовувати такі властивості (атрибути):

- *filters* — сукупність фільтрів, застосовуваних до визначеного документа. Атрибут *filters* повною мірою реалізований, починаючи з 5-го покоління браузерів;
- *innerHTML* та *innerText* — властивості, які дозволяють вставляти до будь-якого елемента із заздалегідь присвоєним *id* новий HTML-код або текст, цілком замінивши попередній;
- *style* — реалізований у вигляді властивості, що визначає стиль для конкретного елемента;
- *title* — відображає спливаючу підказку. За замовчуванням там записано нульовий рядок. Синтаксис:

  <ім'я_елемента title='string'>

Розглянемо практичний приклад, який показує можливості динамічної зміни об'єктів.

Необхідно задати шари, появою яких слід динамічно управляти.

Нагадаємо, що *шари* у Web — це спеціально створені об'єкти, за допомогою яких можна динамічно змінювати контент, який показується користувачеві в браузері.

Зауважимо, що шари в браузерних платформах на основі Internet Explorer слід створювати за допомогою контейнерів * *. Шари в браузерах Netscape задаються за допомогою спеціальних тегів *<layer>* та *<ilayer>* і не є сумісними з браузером Internet Explorer.

Для керування відображенням шарів скористаємось атрибутом *visibility* в такій JavaScript-функції:

```
function change()
{
next=current+1; if(next>1) next=0;
window.document.all.item("myname",current).visibility="hidden";
window.document.all.item("myname",next).visibility="visible";
current=next; }
```

У прикладі *myname* — це значення атрибута *id*.

Виклик функції розмістимо до гіпертекстового посилання за схемою *javascript:*

```
<a href="javascript:change(); void(0);">
...
</a>
```

Задамо код самого шару:

```
<div id=myname  style="position:absolute;
 top:230px;left:55px;width:550px;
visibility:hidden;">...
</div>
```

Використання фільтрів CSS (у Internet Explorer)

Починаючи з 5-го покоління, доступні DHTML-фільтри. Розглянемо деякі з них:

1) фільтр *Flip* — перевертає об'єкт горизонтально.

Синтаксис:

STYLE="filter:Flip"

2) фільтр *Flip* — перевертає об'єкт вертикально.

3) фільтр *Glow* — об'єкт сяє.

Синтаксис:

style="filter:Glow(Strength=strength, Color=color;)",

де *color* — колір, яким засяє текст;

strength — сила, з якої він засяє (0–100).

4) фільтр *Wave* робить об'єкт хвилястим.

Синтаксис:

filter: Wave(Freq=freq, Add=add, LightStrength=strength, Phase=phase, Strength=strength),

де *Freq* — число хвиль;

LightStrength — сила хвилі;

Phase — кут хвилі;

Strength — інтенсивність хвилі.

Засоби доступу до фільтра

Синтаксис для застосування фільтра:

style='filter: ім'я_фільтра (пар1=значення, пар2=значення,...) '

У таблиці 6.9 наведено параметри фільтрів.

Таблиця 6.9

Параметри фільтрів

Фільтр	Призначення фільтра	Параметр	Призначення параметра
1	2	3	4
Alpha	Встановлення прозорості об'єкта	*pacity=0..100*	Рівень прозорості
		finishOpacity=0..100	Граничний рівень прозорості

Продовження табл 6.9

1	2	3	4
Alpha	Встановлення прозорості об'єкта	*style=0..3*	Встановлює вид градієнту (1 — лінія, 2 — коло, 3 — прямокутник)
		startX= *startY=*	Встановлюють координати початку градієнта
		finishX= *finishY=*	Встановлюють координати кінця градієнта
Blur	Створює розмивання об'єкта	*add=0,1,2,3,...*	Додає розмивний об'єкт до розмитого (якщо 0 — додавання не відбувається)
		direction=0..315	Визначає напрямок розмиття з кроком 45
		strength=	Глибина зображення об'єкта в пікселах
Chroma	Робить прозорим визначений колір рисунка	*color=#xxxxxx*	Задає потрібний колір
Drop-Shadow	Додає силует об'єкта, зсунутий у визначеному напрямку	*color=*	Задає колір тіні
		offX= *offY=*	Встановлюють горизонтальний і вертикальний зсув тіні в пікселах
		positive=1/0	Задає тінь для видимих/невидимих пікселів об'єкта
FlipH	Перевертає об'єкт горизонтально	—	—
FlipV	Перевертає об'єкт вертикально	—	—
Glow		*color=колір*	Задає колір сяйва
		strength=0..100	Задає глибину сяйва
Gray	Робить об'єкт чорно-білим	—	—

Закінчення табл 6.9

1	2	3	4
Invert	Реверсує яскравість, насиченість і колірний відтінок	—	—
Light	Підсвічує об'єкт	—	—
Mask	Виділяє об'єкт	*color=колір*	Задає колір виділення
Shadow	Задає тінь об'єкта	*color=колір step=45*	Задає колір тіні
Wave	Робить об'єкт хвилястим	*add=true | false*	Вказує, чи доданий початковий об'єкт до фільтрованого
		freq=	Визначає кількість хвиль (частоту)
		lightStrength=	Визначає силу хвилі в %
		phase=	Визначає фазу хвилі в % (50%=180°)
		strength=	Визначає інтенсивність хвилі
X-ray	Визначає рентгенівське світіння об'єкта	—	—
Reveal-Trans	Показує і ховає об'єкт	*duration=$t_1.t_2$, де t_1 — час, с; t_2 — час, мс*	Час переходу з одного стану до іншого
		transition=0 – 23	Задає вид переходу (якщо значення >23, перехід не визначений)
Blend-Trans	Показує об'єкт через визначений проміжок часу	*duration=$t_1.t_2$, де t_1 — час, с; t_2 — час, мс*	Час, через який об'єкт буде проявлено

Методи фільтрів

Фільтр *RevealTrans( )* має такі методи:

- *apply ( )* — застосовує фільтр до зазначеного об'єкта;
- *play ( )* — запускає дію фільтра;
- *stop ( )* — зупиняє дію фільтра.

Для застосування цих методів об'єкту слід присвоїти власний *ID*.

Фільтр *BlendTrans* має атрибут *visibility:*

```
BlendTrans (duration=45.30); visibility: hidden
```

Приклад. Необхідно забезпечити поступову появу і зникнення об'єкта, зробити прив'язку до подій. Наведемо відповідний код:

```
<html>
  <head>
    <script language='JavaScript'>
      function MyStart ( ) {
        mypic.filter.item (0).apply ( );
        mypic.style.visibility=' ';
        mypic.filter.item ( ).play ( );
        mypic2.filter.item (0).apply ( );
        mypic2.style.visibility='hidden ';
        mypic.filter.item (0 ).play ( ); }
      function MyStop ( ) {
        mypic.filter.item (0).stop ( );
        mypic.style.visibility='hidden ';
        mypic2.filter.item (0).stop ( );
        mypic2.style.visibility=' ';
        }
    <body>
      <img src='1.jpg' id='mypic' stile="filter: revealTrans (Duration=5.0,
Transition=11);
visibility: hidden">
<img src='1.jpg' id='mypic2' stile="filter: blendTrans (Duration=5.0) ">
      <form>
<input type='button' value='Проявити рисунок' onClick='MyStart ( )'>
<input type='button' value='Відмінити фільтр' onClick='MyStop ( )'>
```

У прикладі при натисканні на кнопку «Проявити рисунок» у вікні поступово зникає перший рисунок і з'являється другий.

Приклад застосування властивості *innerHTML*. Необхідно замінити текстовий вміст контейнера на картинку з фільтром. Зміну вмісту прив'язати до дії кнопки.

```
<p innerHTML>
<h1 id='myh1'>текст</h1>
<form>
<input  type='button'
        value='Зміна вмісту'
        onClick='MyInner ( )'>
<head>
<script language='JavaScript'>
   function MyInner ( ){
   myh1. innerHTML="<img src='1. jpg'  id='mypic'  style='filter: ... '>" }
</script>
</form>
```

6.8. Контрольні запитання

1. Надайте характеристику об'єктної моделі Internet Explorer.
2. Надайте характеристику об'єктної моделі Netscape Navigator.
3. Які особливості реалізації динамічного HTML?
4. Надайте характеристику об'єкта *window*.
5. Надайте характеристику об'єкта *document*.
6. Надайте характеристику об'єкта *event*.
7. Надайте характеристику об'єкта *screen*.
8. Надайте характеристику об'єкта *navigator*.
9. Які існують засоби маніпулювання об'єктною моделлю DHTML?
10. У чому сутність *coocies*-технології?
11. Як використовувати вбудовані фільтри DHTML?

6.9. Завдання для самостійного виконання

1. Використовуючи матеріал розділів 4, 5 та 6, виконати такі завдання:

1) реалізувати випадне меню (з налагодженням кольорової гами);

2) створити тести (у вигляді анкети) з реєстрацією та аналізом результатів;

3) розробити меню, що розгортається (дерево), з налагодженням колірної гами;

4) створити гру в «хрестики-нолики» з реєстрацією та веденням рахунку гри;

5) створити сценарій «снігопад» з налагодженням швидкості та інтенсивності появи сніжинок;

6) розробити графічний годинник, який рухатиметься за курсором або реагуватиме на прокрутку;

7) створити власну графічну прокрутку у вікні браузера;

8) розробити адаптивний сайт — для кожного користувача запам'ятовується набір із трьох найбільш відвідуваних сторінок і виводиться у вікні «швидкий перехід» для спрощення навігації;

9) створити гру «п'ятнадцять» (реєстрація і ведення рахунку).

2. Реалізувати фільтри:

1) ліхтар маяка (фільтр «світіння»);

2) миготливий текст (фільтр «альфа»);

3) календарик, що переливається (фільтр «альфа» на двох рисунках);

4) «хвилястий» рисунок (фільтр «хвиля»);

5) мультфільм (птах, що летить, — фільтр «альфа» на масиві рисунків);

6) презентація (ефект розкриття чи проявлення слайдів — *revealtrans*);

7) градієнтне розфарбування (фільтр «альфа» для окремих частин градієнтно зарисованої фігури).

Розділ 7
ВИКОРИСТАННЯ СЕРВЕРНИХ МОВ СЦЕНАРІЇВ

7.1. Технології серверної сторони

Для розробки додатків серверної сторони необхідно обрати базову технологію, на якій ці додатки будуть засновані.

На даний момент існують і успішно застосовуються різні види технологій побудови web-додатків серверної сторони. Усі такі додатки мають спільну мету — реалізацію бізнес-логіки на стороні сервера і генерацію коду для клієнта. Також у цих додатків однакова архітектура взаємодії сервера і клієнта, а також спільний протокол взаємодії — HTTP.

Загальна логіка роботи додатка серверної сторони представлена на рис. 7.1.

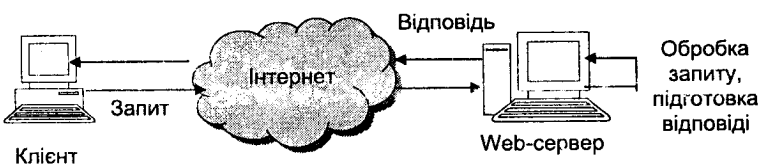


Рис. 7.1. Робота додатків серверної сторони

Як видно з рис. 7.1, робота серверних додатків відбувається в три основних етапи:

1. Запит. Клієнт, використовуючи web-браузер, ініціює запит до сервера.

2. Обробка запиту, підготовка відповіді. Після одержання запиту web-сервер проводить обробку запитуваного ресурсу. У випадку, якщо запитується статичний ресурс (HTML-сторінка, рисунок, документ), ця інформація форматується для протоколу HTTP і передається клієнтові як відповідь.

Якщо ж запитується динамічний ресурс, запит передається на обробку відповідному контейнеру web-додатків, де і відбувається подальша робота.

3. Після формування дані передаються клієнтові за допомогою протоколу HTTP як відповідь. Відповідь містить дані (зазвичай, HTML-код або двійкові дані), а також додаткові параметри, передані в заголовках HTTP відповіді.

Робота додатків серверної сторони завжди відбувається за описаним вище сценарієм. Очевидно, що такий підхід створює труднощі у створенні web-додатків, основною з яких є відсутність постійного стану web-додатка (так звана stateless programming). Це означає, що додаток працює виключно в режимі «запит–відповідь», не маючи даних про попередні кроки користувача або будь-якої іншої постійної інформації. Для вирішення цієї проблеми застосовується поняття користувальницької сесії, яка дозволяє зберігати дані на сервері протягом сеансу роботи користувача.

Однак наявністю сесій проблеми цілком не усуваються. Чим більше можливостей надає платформа реалізації для додатків серверної сторони в подоланні цих складностей, тим швидше й ефективніше може здійснюватися розробка. Далі будуть розглянуті різні підходи до створення додатків серверної сторони, їхні переваги і недоліки, а також конкретні платформи.

7.2. Вимоги до додатків сторони сервера

Під час створення додатків серверної сторони необхідно виділити два основних підходи:

- безпосередня обробка запитів і формування відповідей;
- вбудовування програмного коду в шаблони web-сторінок.

Перший підхід надає найбільші можливості щодо управління обробкою і підвищення продуктивності. Він передбачає передачу всіх даних про запит коду, який безпосередньо виконується, що може як сформувати відповідь зі сторінкою для користувача, так і відкрити на передачу потік двійкових даних, наприклад для передачі зображення. Однак при такому підході всі дані для передачі формуються програмним шляхом, що сповільнює розробку простих сторінок і ускладнює взаємодію між розробником дизайну та програмістом. Прикладами такого підходу є технології CGI та Java Servlets.

Другий підхід використовує шаблони сторінок користувача, оформлені певним чином, що дозволяє вставляти в них ділянки програмного коду. Цей підхід особливо ефективний при створенні простих додатків, основна інформація в яких статична, а динамічна інформація може бути згенерована простими програмними конструкціями. У процесі розробки складних програмних систем цей варіант ускладнює взаємодію між компонентами й реалізацію складної архітектури. Також він менш ефективний за продуктивністю й обмежує можливості з реалізації складних сторінок. Прикладами цього підходу є технології PHP, ASP, JSP.

Крім різних підходів до генерації сторінок, платформи розробки по-різному задовольняють сучасні вимоги до створення складних web-систем. Наведемо найбільш важливі з цих вимог:

- незалежність від платформи;
- продуктивність, можливість масштабування;
- можливості розширення й інтеграції;
- простота використання, наявність засобів розробки;
- наявність необхідних програмних бібліотек.

Отже, ми визначили низку вимог, яким має відповідати сучасна платформа розробки. Нижче розглянуто найбільш популярні на даний момент платформи та їхні особливості з точки зору наведених критеріїв.

7.3. Технологія CGI

Технологія *CGI* (*Common Gateway Interface*) відрізняється від інших тим, що є найбільш низькорівневою і є стандартом інтерфейсу, що служить для зв'язку зовнішньої програми з web-сервером. Сам протокол розроблено таким чином, щоб можна було використовувати будь-яку мову програмування, яка може працювати зі стандартними пристроями вводу/виводу. Оскільки така можливість є на рівні операційної системи, то, якщо не потрібно складного скрипту, його можна оформити у вигляді командного файла.

Розглянемо основні особливості технології CGI:

- CGI не висуває особливих вимог до платформи і web-сервера, тому працює на всіх популярних платформах і web-серверах.

Також технологія не прив'язана до конкретної мови програмування і може використовуватись будь-якою мовою, що працює зі стандартними потоками вводу/виводу;

• продуктивність CGI-додатків не є високою. Основною причиною цього є те, що при черговому звертанні до сервера для роботи CGI-програми створюється окремий процес, що вимагає великої кількості системних ресурсів;

• технологія не передбачає вбудованих засобів масштабування, про це розробникам необхідно піклуватися окремо;

• CGI-програма є готовим до виконання файлом, що перешкоджає розширенню системи.

На цей час розробники віддають перевагу більш розвиненим та зручним платформам, які відрізняються більшою продуктивністю. Однак велика маса вже працюючих додатків вимагає брати до уваги технологію CGI.

Далі розглянемо найбільш популярні технології — PHP, JSP, Java Servlets, ASP.NET.

7.4. Технологія сервлетів

Технологія *Java Servlets* (*сервлети*) була розроблена компанією Sun Microsystems, щоб використовувати переваги платформи Java для вирішення проблем технології CGI- та API-розширень сервера.

Технологія вирішує проблему продуктивності, виконуючи всі запити в одному процесі. Сервлети також можуть легко розділяти ресурси і не залежать від платформи, оскільки виконуються всередині JVM (*Java Virtual Machine* — віртуальна машина Java).

Технологія відрізняється широкими функціональними можливостями. Велика кількість бібліотек надає різноманітні засоби, необхідні в розробці програм.

Модель безпеки Java надає можливість точного управління рівнем доступу, наприклад дозволяючи доступ тільки до певної частини файлової системи. Обробка виключень Java робить сервлети більш надійним засобом, ніж розширення серверів мови C/C++.

Сервлет є класом Java і тому має виконуватися за допомогою JVM так званим *сервлет-контейнером* (servlet container,

servlet engine). Сервлет-контейнер завантажує клас сервлета при першому звертанні до нього або відразу при запуску сервера за спеціальною вказівкою. Далі сервлет залишається завантаженим для обробки запитів, поки він не буде вивантажений, або до зупинки контейнера.

Технологія є поширеною і може бути використана з усіма популярними web-серверами (Enterprise Server від Netscape, Microsoft Internet Information Server (IIS), Apache, Java Web Server від Sun тощо).

Програмний інтерфейс дозволяє сервлетам обробляти запити на будь-якому рівні, за необхідності використовувати будь-які низькорівневі дані, такі як заголовки запитів, їхній тип тощо. Це забезпечує велику гнучкість для розробки нестандартних оброблювачів, наприклад під час роботи з двійковим або мультимедійним змістом.

Оскільки сервлети обробляються в одному процесі за допомогою створення потоків усередині нього, програмний код сервлетів має бути безпечним для потоку. Це накладає певну відповідальність на програміста, але за допомогою стандартних прийомів, таких як відмова від використання полів у класах сервлетів і збереження необхідних даних у контексті або зовнішньому сховищі, такі властивості коду легко досягаються. До того ж сервлети здобувають таку перевагу, як можливість масштабування.

Отже, сервлети забезпечують компонентний, платформонезалежний метод для побудови web-додатків без обмежень продуктивності CGI-програм. Вони мають широкий діапазон доступних прикладних API, дозволяють використовувати всі переваги Java, легко розширюються і масштабуються, підтримуються всіма популярними web-серверами. З огляду на це для розробки великих web-систем рекомендується технологія сервлетів.

7.5. Технологія JSP

Технологія *Java Server Pages* (*JSP*) від компанії Sun Microsystems з'явилась як надбудова над технологією Java Servlets. Вона забезпечує більш швидку і просту розробку web-додатків за допомогою застосування *шаблонного підходу*.

Для розуміння архітектури і переваг JSP необхідно знати технологію Java Servlets, оскільки вони тісно пов'язані. Сторінки

Java Server Pages є шаблонами сторінок HTML та схожі із шаблонами PHP і ASP. Основною відмінністю від інших подібних технологій є те, що код, який перебуває всередині спеціальних тегів, не інтерпретується при звертанні до сторінки, а попередньо компілюється в Java Servlet. Статичні ділянки шаблона перетворюються у виклики функцій для їхнього розміщення в потік виводу. Код компілюється так, ніби він перебуває всередині сервлета. Компіляція JSP-сторінок у сервлети є трудомісткою, але проводиться один раз: або при першому звертанні до сторінки, або при запуску сервлет-контейнера.

Технологія JSP вдало поєднує шаблонний підхід до побудови сайтів і всі переваги Java-платформи. Завдяки цьому технологія набула поширення як серед професійних комерційних розробників, так і при створенні відкритих некомерційних проектів.

Важливим кроком до розширення шаблонного підходу стали так звані *бібліотеки тегів* (*tag libraries*). Вони забезпечують можливість інтегрувати стандартні, сторонні або власні програмні компоненти в сторінки. Простота створення і використання бібліотек тегів стали причиною їх великої популярності. Завдяки роботі на основі Java, технологія JSP не прив'язана до конкретної апаратної або програмної платформи. Тому JSP є вдалим рішенням для використання в гетерогенних середовищах.

Проте продуктивність технології обмежена об'єктивними особливостями архітектури. По-перше, сторінки мають бути відкомпільовані в сервлети, що забирає значний час. По-друге сервлети виконуються в середовищі Java, тобто в режимі інтерпретації. Однак ці обмеження компенсуються додатковими можливостями. Сучасні контейнери підтримують кластеризацію серверів, що перекладає навантаження на апаратне забезпечення. Це є економічно виправданим рішенням. Задача ж компіляції в сервлети є разовою і виконується або при першому звертанні, або при запуску сервлет-контейнера. У такий спосіб це не позначається на загальній продуктивності системи за достатній період часу.

Основними перевагами JSP є:

- простота розробки, характерна для шаблонного підходу;
- наявність великої кількості сторонніх бібліотек, легкість їхнього використання;
- потужні та різноманітні середовища розробки.

З огляду на зазначене, JSP є перспективною технологією розробки для створення web-сайтів. Однак при створенні складних web-систем даються взнаки обмеження, що накладаються шаблонним підходом.

7.6. Технологія .NET

Технологія .NET є новою розробкою компанії Microsoft і являє собою новий етап у розвитку засобів взаємодії між додатками. Вона доступна як доповнення .NET Framework до сімейства операційних систем Microsoft Windows, а також у продукті Windows Server 2003. Також ведуться роботи зі створення .NET Framework на інших операційних системах. Платформа .NET спрощує розробку додатків і підвищує надійність коду. Зокрема, вона забезпечує автоматичне керування часом життя об'єктів, доступ до нейтральних до мов бібліотек класів, обробку виключень і налагодження.

Основа .NET — Common Language Runtime (загальне середовище виконання мов) — спирається на системні служби операційної системи і керує виконанням коду, написаного будь-якою сучасною мовою програмування. Набір базових класів дає доступ до сервісів платформи, які розробники можуть використовувати з будь-якою мовою програмування. Common Language Runtime і базові класи разом складають основу .NET платформи.

NET пропонує високорівневі сервіси:

- ADO.NET — покоління ADO, що використовує XML і SOAP для обміну даними;
- ASP.NET — версія ASP, що дозволяє використовувати .NET-сумісну мову для програмування web-сторінок;
- Windows Forms і Web Forms — набір класів для побудови користувальницького інтерфейсу локальних і web-орієнтованих додатків.

Розгортання систем на платформі .NET здійснюється таким чином. Вихідні коди компілюються не в команди процесора ×86 або інші машинні коди. Замість цього компілятор створює код «проміжною мовою Microsoft» (Microsoft Intermediate Language — MSIL). Файл, що містить MSIL, може виконуватися на платформі будь-якого процесора, якщо операційна система надає .NET CLR.

Важливою складовою частиною платформи .NET є середовище ASP.NET (раніше використовувалася назва ASP+). Можливості ASP.NET досить великі, так що її складно назвати наступною версією ASP. В її основі лежить інша платформа, і основними мовами програмування для неї обрані C# і Visual Basic, замість колишніх скриптових мов. Водночас, технологія дозволяє створювати ASP-сторінки будь-якою мовою.

У ASP.NET закладено все для того, щоб зробити весь цикл розробки web-додатка більш швидким, а підтримку — простішою.

Наведемо основні можливості та принципи роботи ASP.NET:

- компілювання коду при першому звертанні;
- широкий вибір бібліотек компонентів, що поставляються з .NET;
- підтримка потужного засобу розробки — Visual Studio. NET;
- мовна незалежність у межах платформ, для яких реалізоване спільне мовне середовище виконання CLR;
- можливість розширення за допомогою багатопроцесорних та кластерних рішень;
- нові можливості з обробки помилок;
- об'єктно-орієнтовані мови розробки (мова C#);
- розширені можливості повторного використання компонентів.

Очевидно, що платформи .NET і ASP.NET надали нові можливості для розробки web-систем. Вони відповідають всім сучасним вимогам і дозволяють значно прискорити і спростити розробку складних додатків. Проте на даний момент .NET у повному обсязі існує тільки для платформи Windows. Розробки з її переносу на інші системи ще не завершено. Що стосується розробки сайтів, то ASP.NET прив'язана до сервера IIS, і хоч архітектура .NET дозволяє перенести додатки ASP.NET на іншу платформу, але на даний момент реальна можливість відсутня. Тому багатоплатформеність поки ще не забезпечується платформою .NET. Слід відзначити, що така система повинна мати можливості інтеграції з платформою .NET (особливо web-сервіси), оскільки її майбутнє широке використання не викликає сумнівів.

7.7. Технологія PHP

Технологія PHP (*Personal Home Page*) набула значного поширення завдяки своїй безкоштовності і підтримці найпопулярніших платформ. Вона базується на принципі побудови сторінок із шаблонів, що вперше з'явились у Active Server Pages, але розвиває і доповнює його.

Сторінки PHP мають вигляд звичайних HTML-сторінок, у яких можуть використовуватися спеціальні теги вигляду *<?php* і *?>*. Між тегами вставлюються рядки програмного коду спеціальною мовою сценаріїв PHP.

Принцип шаблонів дозволив розроблювачам створювати програми набагато швидше і без помилок, властивих традиційним CGI-програмам, що видають HTML-вміст у потік виводу. На сьогоднішній день діапазон систем, побудованих на шаблонах, вельми значний — від простих сторінок з вибірками з бази даних до великих додатків електронної комерції, заснованих на XML.

Шаблонні системи користуються великою популярністю серед розробників, оскільки найбільше підходять для типових сайтів.

Розглянемо основні переваги і недоліки платформи:

- застосовувана у PHP мова — проста і зручна, однак не є в повному обсязі об'єктно-орієнтованою;
- для PHP розроблено великі бібліотеки, а також значна кількість вбудованих функцій для розв'язання найрізноманітніших задач;
- при використанні PHP із web-сервером Apache є можливість ефективного виконання ядра як розширення сервера. В інших випадках продуктивність платформи невисока;
- власних засобів масштабування PHP не має, усі функції з кластеризації цілком лягають на web-сервер і розробників;
- можливості інтеграції обмежені включенням модулів і використанням зовнішніх функцій, що не відповідає сучасним вимогам.

Шаблонний підхід PHP, при усіх великих можливостях, містить і серйозні недоліки. Із загальних недоліків цього підходу, що стосуються як до PHP, так і ASP та JSP, необхідно виділити такі:

• файл-сторінку може підтримувати тільки спеціаліст високої кваліфікації, який добре володіє як програмуванням, так і HTML;

• один файл в конкретний момент часу може редагувати тільки одна людина. Це означає, що працює або програміст, або дизайнер. Тобто не можна забезпечити поділ праці там, де він потенційно можливий;

• збереження бізнес-логіки у файлах-сторінках у розподіленому за керуючими елементами вигляді призводить до утруднення її винесення в об'єкти другого рівня.

Як загальний підсумок розгляду платформи можна зауважити, що завдяки простоті використання, наявності великого числа функцій і бібліотек, поширеності і підтримці більшості існуючих web-серверів і платформ, PHP є дуже зручним засобом розробки невеликих систем. У той же час обмеження щодо продуктивності, масштабування, за мовою програмування, а також щодо розширення й інтеграції перешкоджають використанню платформи при розробці масштабних систем.

7.8. Порівняльний аналіз базових технологій серверної сторони

У попередньому матеріалі були розглянуті найбільш популярні базові технології побудови додатків серверної сторони. З розглянутого можна виділити основні підходи до архітектури серверних додатків.

1. *Окреме виконання запитів.* При кожному запиті динамічного вмісту запускається окрема програма для обробки запитів. Програма генерує контент, який передається клієнтові. Цей підхід використовується в класичних CGI-скриптах.

2. *Нагромадження процесів, що виконуються.* Підхід аналогічний попередньому, але якщо запит виконується повторно, то нового запуску програми не відбувається, а обробка передається існуючому процесу. Даний підхід застосовується в технологіях Java Servlets, Fast CGI.

3. *Шаблони сторінок.* При запиті шаблони заповнюються динамічним контентом, який зазвичай (але необов'язково), створюється інтерпретаційною мовою сценаріїв. Підхід застосовується в технологіях ASP, JSP, PHP.

4. *Розширення web-сервера.* Web-сервер звертається до особливих розширень для обробки динамічного змісту. Розширення специфічні для web-сервера. Цей підхід використовується в IS API, NSAPI, *mod_perl.*

Кожний із розглянутих підходів має свої можливості й обмеження, і, відповідно, свою сферу застосування. Модель окремого виконання запитів істотно обмежує продуктивність. Варіант нагромадження процесів є розвитком цієї технології, підвищує продуктивність, зберігаючи при цьому максимальну гнучкість розробки.

Шаблонний підхід є зручним для розробки невеликих систем, однак у разі збільшення складності він починає гальмувати процес розробки і не є придатним для великих систем.

Він також відрізняється невисокою продуктивністю, хоча дослідження показують, що у визначених умовах може демонструвати досить високі показники і конкурувати з підходом 2.

Розширення web-сервера не є найбільш зручним засобом розробки, оскільки жорстко прив'язують систему до певного web-сервера, але демонструють максимальну продуктивність і дають найбільшу гнучкість у розробці.

Розглянемо платформи згідно з вимогами, визначеними раніше.

CGI не входить в огляд, оскільки є незручним у використанні і має низьку ефективність, а розширення серверів занадто сильно прив'язані до конкретних програмних продуктів.

За схемою обробки запитів платформи розподіляються в такий спосіб:

- PHP-шаблони. У разі виконання на web-сервері Apache інтерпретатор може бути розширенням сервера (в експериментальному режимі IIS);
- Java Servlets. Віддрізняються нагромадженням процесів для кожного сервлета;
- JSP-шаблони. При їх обробці виконується попередня компіляція в Java Servlets, що дозволяє використовувати схему нагромадження процесів;
- ASP.NET-шаблони. Використовується схема попередньої компіляції, а не інтерпретації коду. В результаті використовується розширення web-сервера IIS. Можуть використовуватися і низькорівневі оброблювачі.

Основні характеристики платформ для оцінки порівняємо в зведеній таблиці 7.1, де символ «-» означає повну відсутність

підтримки, «-/+» — недостатня підтримка, «+/-» — підтримка не в повному обсязі, і «+» — повна підтримка. Для порівняльних характеристик, таких як мова реалізації або продуктивність, оцінки відповідають ступеню переваги технології.

Таблиця 7.1

Порівняння технологій

Характеристики для порівняння	PHP	Java Servlets	JSP	ASP .NET
Багатоплатформовість	+/-	+	+	-/+
Продуктивність	-/+	+/-	+/-	+
Масштабованість	-	+	+	+
Мова реалізації	+/-	+	+	+
Можливості розширення й інтеграції	-	+	+/-	+
Простота використання, наявність засобів розробки	+/-	+/-	+	+
Наявність необхідних програмних бібліотек	+	+	+	+
Розподіл дизайну і логіки	+/-	-/+	+/-	+
Засоби візуальної розробки	-/+	+/-	+	+
Можливість побудови компонентної архітектури	-	+	+/-	+

З наведеного порівняння можна зробити висновок, що платформи найбільш популярного типу — шаблонні, — не підходять для розробки великих web-систем, оскільки схема їхньої роботи ускладнює побудову багатокомпонентної архітектури. При використанні систем нешаблонного типу розробка ускладнюється відсутністю можливості швидко і зручно модифікувати дизайн сайта, оскільки він визначається вже сформованим програмним кодом.

Що стосується візуалізації, то вона присутня тільки при використанні шаблонних платформ, причому винятково при розробці системи. Це призводить до того, що велика web-система, яка вимагає частого поновлення, не повинна будуватися винятково на існуючих базових платформах, а тому є необхідною система управління сайтом, що поєднує різні підходи.

7.9. Вступ до PHP

PHP — це мова серверних скриптів (*server scripting language*), що вбудовується в HTML, інтерпретується і виконується на сервері.

PHP є препроцесором HTML. Його робота побудована за схемою, зображеною на рис. 7.2.

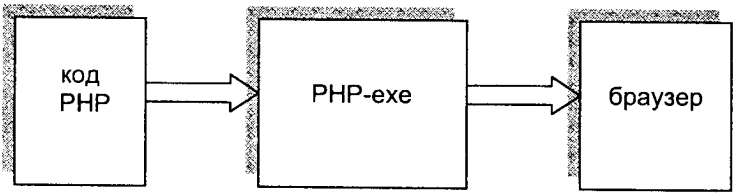


Рис. 7.2. Схема роботи PHP

До того, як сервер «віддасть» файл браузеру, його переглядає препроцесор-інтерпретатор. Для того, щоб це відбувалося, файли, які підлягають обробці препроцесором, повинні мати певне розширення (зазвичай це *.phtml* або *.php*, але ці значення можна змінити) і містити (хоча і необов'язково) код для препроцесора.

Перед відправленням сторінки PHP-код «програється» на сервері і брозеру видається результат у вигляді HTML-сторінки, яка може дуже відрізнятися від тієї, що зберігається на сервері. Звичайні ж сторінки, що мають розширення *.html* чи *.htm*, web-сервер відправлятиме браузеру без будь-якої обробки.

Основна відмінність від CGI-скриптів, написаних іншими мовами (наприклад, Perl або C) — це те, що в CGI-програмах програміст сам пише HTML-код і, використовуючи PHP, він вбудовує свою програму-скрипт у готову HTML-сторінку, використовуючи теги (наприклад, *<? php i? >*).

PHP називається мовою *серверних скриптів* — на відміну від JavaScript/Jscript/VBScript, які є мовами *клієнтських скриптів*. Це означає, що PHP-скрипт виконується на сервері, а клієнту передається результат його роботи, тоді як в JavaScript код повністю передається на клієнтську машину і тільки там виконується браузером.

Простіше за все роботу *php* показати на прикладі. Такий вигляд має web-сторінка з елементами *php*:

```
<html>
 <head>
        <title>Приклад</title>
 </head>
        <body>
                <?php echo "Привіт, я PHP-програма!";?>
        </body>
 </html>
```

Після виконання цього скрипту отримаємо сторінку, в якій буде написано:

Привіт, я PHP-програма!

Відкривши вихідний текст даної сторінки, ми побачимо таке:

```
<html>
<head>
        <title>Example</title>
</head>
<body>
        Привіт, я PHP-програма!
</body>
</html>
```

Можливості PHP

Коротко кажучи, мовою PHP можна зробити все, що можна зробити за допомогою CGI-програм. Наприклад, обробляти дані з форм, генерувати динамічні сторінки, одержувати і посилати кукіси (cookies). До того ж у PHP передбачена підтримка багатьох баз даних (БД), що робить написання web-додатків з використанням БД дуже простим.

На додаток до всього *php* «розуміє» протоколи IMAP, SNMP, NNTP, POP3 і HTTP, а також має можливість працювати з сокетами (*sockets*) і спілкується за іншими протоколами.

Стисла історія PHP

Початком *php* можна вважати осінь 1994 року, коли *Rasmus Lerdorf* вирішив поширити можливості своєї домашньої сторінки і написати невеликий движок для виконання найпростіших задач. Такий движок був готовий до початку 1995 року й отри-

мав назву Personal Home Page Tools (звідси і скорочення PHP). Його можливості були досить обмежені.

До середини 1995 року з'явилася друга версія, що називалася PHP/FI Version 2. Приставка FI приєдналася з іншого пакета Rasmusa, що обробляв форми (Form Interpreter HP/FI компілювався всередину Apache і використовував стандартний API Apache). Виявилося, що PHP-скрипти працюють швидше за аналогічні CGI-скрипти, бо серверу не було необхідності породжувати новий процес. Мова PHP за можливостями наблизилася до Perl, найбільш популярної мови для написання CGI-програм. Була додана підтримка багатьох відомих баз даних (наприклад, MySQL і Oracle). Інтерфейс до GD-бібліотеки дозволяв генерувати gif-зображення «на льоту». З цього моменту почалося широке розповсюдження PHP/FI.

Наприкінці 1997 Zeev Suraski і Andi Gutmans вирішили переписати внутрішній движок з метою виправити помилки інтерпретатора і підвищити швидкість виконання скриптів. У 1998 році вийшла нова версія, що була названа *php3*, яка здобула дуже широку популярність серед розробників. На цей час поточна версія — *php5*.

Обґрунтування вибору php

Розробникам web-додатків немає необхідності говорити, що web-сторінки — це не тільки текст і картинки. Гідний уваги сайт повинен підтримувати деякий рівень інтерактивності з користувачем: пошук інформації, продаж продуктів, конференції тощо. До недавніх часі все це традиційно реалізувалося CGI-скриптами, написаними мовою Perl. Але виявилося, що CGI-скрипти дуже погано масштабуються. Кожний новий виклик CGI-скрипту вимагає від ядра породження нового процесу, а він займає процесорний час і витрачає оперативну пам'ять.

PHP пропонує інший варіант — він працює як частина web-сервера, і тим самим схожий на ASP від Microsoft або ColdFusion від Allaire.

Синтаксис *php* дуже схожий на синтаксис C або Perl. Спеціалісти, які знайомі з програмуванням, дуже швидко зможуть почати писати програми мовою *php*. У цій мові немає точної типізації даних і немає необхідності в діях щодо виділення чи вивільнення пам'яті.

Програми, написані мовою PHP, на відміну від Perl-програм читаються досить легко.

Пара *PHP — MySQL* є крос-платформенною. Це означає, що можна, працюючи в Windows, розробляти програми, призначені для роботи під Unix. Крім того, PHP може працювати як зовнішній CGI-процес або як звичайний інтерпретатор скриптів, або як модуль, що приєднується до web-сервера Apache або IIS.

І нарешті, оскільки даний продукт розробляється спільними зусиллями багатьох компаній, існує велика кількість документації і списків розсилки, до яких можна звернутися у випадку виникнення будь-яких питань. Знайдені помилки виправляються досить швидко, усі пропозиції і зауваження розглядаються і, якщо вони виявляються цінними, реалізуються в новій версії.

Використання PHP

Перша PHP-сторінка. Створимо текстовий файл під ім'ям *test.php* (вихідний код першої php-сторінки):

```
<html>
<body>
<? php
        $myvar="Hello, World";
        echo $myvar;
? >
</body>
</html>
```

Тепер відкриємо браузер і наберемо в ньому URL створеної сторінки, наприклад *http://localhost/test. php*. На екрані в браузері побачимо таке:

Hello, World

Відкривши вихідний код сторінки в браузері, побачимо таке:

```
<html>
<body>
        Hello, World
</body>
</html>
```

Це відбулося тому, що PHP-движок на сервері продивився сторінку, знайшов в ній PHP-код, обробив його і видав результат, що web-сервером був відправлений до браузера.

Це вже не статична HTML-сторінка з фіксованим текстом, а справжня програма, що залежно від поданих до неї даних може видавати різноманітні результати. Ця можливість цілком змінює підхід до публікації документів у Інтернеті, перетворюючи їх зі статичних в динамічні, що змінюються в залежності від дій користувача. В нашій першій сторінці-програмі ці можливості звичайно не такі вже й потужні, адже єдине, що вона робить — це виводить текст, який було б простіше написати вручну. Тим не менше вже через декілька сторінок стає зрозуміло, наскільки гнучкий і потужний інструмент з'явився в руках у розробника. Розглянемо детальніше нашу першу PHP-сторінку.

Перше, на що треба звернути увагу в наведеному вище коді, це обмежувачі. Знайдіть рядок, що починається з *<?php*.

Для PHP-движка цей код означає початок блоку команд, які треба обробити і виконати. Закінчується блок обмежувачем *?>*. Іншими словами, символи *<?php і ?>* виконують роль дужок. Все, що знаходиться поза ними, PHP-движок пропускає і відправляє у Web без будь-якої обробки, виконуючи лише тільки те, що перебуває всередині цих дужок.

Потужність PHP полягає в тому, що PHP-код можна вставляти в будь-яке місце HTML-сторінки. Пізніше ми розглянемо деякі з корисних прийомів.

Замість дужок *<? php ...? >* можна використати і скорочену нотацію *<? ...? >*.

Деяким програмістам, що працюють також з ASP, зручніше писати дужки, використовуючи комбінацію *<% ... %>*. PHP можна налагодити на використання й таких дужок.

Можливий ще один з варіантів вставки PHP-сценарію, показаний нижче:

```
<SCRIPT LANGUAGE='PHP'>
        Інструкції
<SCRIPT>
```

Ще одна деталь, на яку слід звернути увагу, — це крапка з комою в кінці кожного рядка коду. Це роздільник, призначений для відокремлення одного набору команд від іншого. Взагалі весь PHP-код можна писати в одному рядку, поділяючи команди крапкою з комою. Але читати такий код буде незручно, тому в наших прикладах після кожної крапки з комою ми ставили два порожніх рядки, щоб ясніше виділити групи команд. Не забувайте

про крапку з комою в кінці рядка (це найчастіша помилка у програмістів-початківців).

Нарешті, можна помітити, що перед словом *myvar* є символ *$* (долар). Цей символ повідомляє PHP, що перед ним — змінна. Ми присвоїли (використовуючи символ «=») рядок *"Hello, World"* змінній *$myvar*. Змінні, окрім рядків, можуть містити числа і масиви. В будь-якому випадку будь-яка змінна завжди позначається символом *$*.

Функції

Широкі можливості мови PHP містяться в його функціях. Взагалі, *функція* — це блок команд, що виконує яку-небудь операцію. Якщо скомпілювати PHP з усіма наявними для нього доповненнями, то можна отримати доступ до більш ніж 700 функцій.

Кожна функція має свою назву і синтаксис. У першому прикладі була використана функція *echo*, яка виводить рядок, укладений у лапки, або змінну, що іде слідом.

Розглянемо уважніше першу PHP-сторінку. В принципі, вона не відрізняється від звичайної HTML-сторінки. Тільки замість розширення *.html* (або *.htm*) ми присвоїли їй розширення *.php*. Для web-сервера це розширення стало сигналом, що дану сторінку перед відправленням треба пропустити через PHP-движок. Рядки *<html><body> ... </body></html>* будуть проігноровані PHP-движком. Він зверне увагу тільки на те, що написане всередині дужок *<?php ...?>*. У результаті ми отримаємо: *Hello, World*. У принципі, всю HTML-сторінку ми могли б згенерувати з допомогою PHP-команд, наприклад:

```
<? php
        echo "<html>";
        echo "<body>";
        $myvar="Hello World";
        echo $myvar;
        echo "</body>";
        echo "</html>";
? >
```

Результат був би той же. Однак з точки зору програмування взагалі, цей код — неякісний, адже PHP-движку доведеться обробити вже шість рядків коду замість колишніх двох. Навіщо

утрудняти PHP-движок виводом тегів *<html>*, *<body>*, *</body>*, *</html>*, якщо вони й так виводяться в сторінці? При написанні коду слід пам'ятати про продуктивність. Завжди необхідно намагатися покращити або змінити код так, щоб PHP-движок витрачав якомога менше часу на його обробку. Деякі поради з оптимизації коду наведемо декілька пізніше.

7.10. Використання PHP

Використання будь-яких сценарних (скриптових) мов програмування зазвичай зводиться до використання вбудованих та створення власних функцій.

Розглянемо, як ефективно використовувати PHP.

Однією з дуже корисних функцій мови PHP є *phpinfo()*. Вона виводить на екран всю інформацію про PHP-движок, встановлений на сервері, причому в зручній формі таблиці.

Створимо сторінку з наведеним нижче кодом і викличемо її з браузера:

```
<html>
        <body>
        <? php
            phpinfo ();
        ? >
        </body>
</html>
```

У результаті одержимо сторінку, яка містить корисні відомості не тільки про сам PHP-движок, але й про web-сервер, його розширення і можливості.

Виведення тексту в HTML-сторінку

Найпростіший спосіб спілкування з користувачем через web-сторінку — надіслати йому в сторінці необхідний текст. Це можна зробити за допомогою функції *print* або *echo:*

```
<? php
print "Hello, world.";
? >

<? php
echo "Hello, world.";
? >
```

Print — це функція, що відправляє браузеру текст. Між словом «print» і символом «;» буде розміщений рядок, що обмежується лапками. Все, що знаходиться всередині лапок, буде відправлене браузеру.

Отже, текст «Hello, World» можна відправити декількома способами. Створимо файл *print.php* з таким текстом:

```
<html>
<body>
<? php
 print "Тут використовується функція print.";
 print "<p>";
 echo "А тут використана функція echo.",
"",
"P. S. Можна додати другий текстовий рядок.",
"",
"Текстові рядки поділяються комами.";
print "<p>";
printf ("Тут використовується функція printf. ");
print "<p>";
printf ("Функція printf звичайно використовується для форматованого виводу чисел і рядків.");
print "<p>";
printf ("Не забувайте про дужки, коли користуєтеся функцією printf. ");
? >
</html>
</body>
```

У результаті виконання коду в браузері буде відображений такий результат:

Тут використовується функція print.

А тут використана функція echo. P. S. Можна додати другий текстовий рядок. Текстові рядки поділяються комами.

Тут використовується функція printf.

Функція printf звичайно використовується для форматованого виводу чисел і рядків.

Не забувайте про дужки, коли користуєтеся функцією printf.

Таким чином, функція *print* — найпростіший засіб відправлення тексту в браузер.

Функція *echo* працює так же, як і *print*, однак дозволяє додавати до першого текстового рядка інші рядки, поділяючи їх комами.

Функція *printf* відображає числа в певному форматі, наприклад, виводить дробове число з певною кількістю нулей після коми, тому в функції *printf* використання дужок обов'язкове.

Працюючи з дужками, слід узяти до уваги таке:

- *echo* ніколи не використовується з дужками;
- *printf* завжди використовується з дужками;
- *print* використовується з дужками і без них.

Робота з формами

Покажемо, як у PHP обробляти дані, отримані від HTML-форм. Від читача вимагаються міцні знання мови HTML і принципів роботи HTML-форм, а також розуміння різниці між двома засобами передачі даних у них (*див.* методи GET та POST).

Найчастіше серверні скрипти використовуються для обробки результатів заповнення форм. Наприклад, у гостьовій книзі відвідувач вводить дані до форми, яка після цього обробляється на сервері. Відповідаючи на будь-яке опитування, користувач встановлює значення певних полів форми.

Нагадаємо, які теги й атрибути повинна містити форма:

```
<FORM NAME='ім'я_форми'
      ACTION='шлях_до_оброблювача_форми'
      METHOD='метод_передачі_змінних'
Поля введення...
</FORM>
```

Передусім розберемося, що таке «оброблювач». Це скрипт на сервері, до якого будуть передані значення полів вводу.

Кожне поле вводу має атрибут *NAME*, що буде переданий до оброблювача разом зі своїм значенням. Існує два методи передачі даних: GET і POST. Їхня відмінність полягає в тому, що при використанні методу GET значення полів приєднуються до URL, зазначеного в атрибуті *ACTION*. Відбувається це таким чином:

http://site.domain/action.php?Ім'я=значення&... Ім'я=значення

Пари *«Ім'я=значення»* створюються для кожного елемента вводу, ім'я якого вказане атрибутом *NAME*.

У випадку використання методу POST значення полів передаються в заголовку запиту до сервера. Формат передачі при цьому способі нам взагалі не цікавий. Просто візьмемо до уваги, що значення передаються «непомітно» для звичайного користувача.

У процесі виконання скрипту мовою PHP створюються змінні з іменами, які відповідають іменам полів.

Припустимо, що ми створили форму такого вигляду:

```
<FORM ACTION='mult.php' METHOD='GET'>
<INPUT TYPE='text' NAME='first' SIZE='4' MAXLENGTH='4'>
<INPUT TYPE='text' NAME='second' SIZE='4'
    MAXLENGTH='4'>
<INPUT TYPE='submit' VALUE='Помножити'>
</FORM>
```

Скрипт, що міститься в файлі *mult.php*, може мати такий вигляд:

```
<? php
header ('Content-'type: text/html");
echo "$first помножити на $second дорівнює", $first*$second;
? >
```

7.11. Контрольні запитання

1. Дайте порівняння сучасним технологіям серверної сторони.
2. Які особливості технології CGI?
3. Які особливості технології сервлетів?
4. Які особливості технології JSP?
5. Які особливості технології .NET?
6. Які особливості технології PHP?
7. У чому переваги PHP?

7.12. Завдання для самостійного виконання

1. Перевірте роботу функції *phpinfo()*.

Для цього створіть сторінку з кодом (с. 189) та перевірте її роботу в браузері. Знайдіть рядки, що починаються на MySQL. Вони містять інформацію про MySQL, який працює на сервері. Якщо ці рядки відсутні, то з якихось причин PHP не підтримує роботу з MySQL.

2. Перевірте роботу форми та оброблювача форми, які представлені на с. 192.

Розділ 8
ОСНОВИ XML

8.1. Вступ до XML. Структура XML-документа

Розширювана мова розмітки XML (*eXtensible Markup Language*) — це мова опису документів, схожа на мову розмітки гіпертексту (*HyperText Markup Language — HTML*), а точніше — це стандарт для створення мов розміток.

У XML розширюваною є не сама XML, а мова документа, складена на XML. HTML — це мова, що використовується для створення web-сторінок і заснована на певному наборі тегів, які показують програмному забезпеченню (браузеру), як представляти вміст сторінки. Подібно до HTML, XML подається системою тегів, які описують компоненти документа. У дійсності XML і HTML є підмножинами стандартної узагальненої мови розмітки SGML (*Standard Generalized Markup Language*).

На відміну від HTML, XML забезпечує користувачам можливість визначати теги, надаючи потужні засоби для опису структури і природи інформації, поданої в документі. Це означає, що стандартні браузери не зможуть нічого зробити з цими документами. Тому створення програмного забезпечення для XML є набагато складнішою справою. Якщо в HTML використовується визначений набір правил мови, то в XML можна застосовувати власні набори правил. Для цього може використовуватися посилання або на набір правил в окремому файлі, або на набір правил власне у документі.

Наприклад, у XML для відповідного визначення правил можна застосовувати такі дескриптори, як *<ГРУПА> ПОАС-2003 </ГРУПА>*. У цьому разі браузер одержує інформацію про те, що *ПОАС-2003* є *ГРУПА*. Що таке є *ГРУПА*, браузер довідається зі спеціального файла правил.

За допомогою розширеної мови XML можна використовувати інформацію для вибору визначеного стилю.

Розглянемо структуру XML-документа.

Наприклад, ієрархічна база даних відділу кадрів заводу на XML описується так:

```
<кадри>
        <робочий id='1'>
                <прізвище>Петренко</прізвище>
                <посада>слюсар</посада>
        </робочий>
        <робочий id='2'>
                <прізвище>Барабанов</прізвище>
                <посада>слюсар</посада>
        </робочий>
                ...
</кадри>
```

На рис. 8.1 подано ієрархію цього документа.

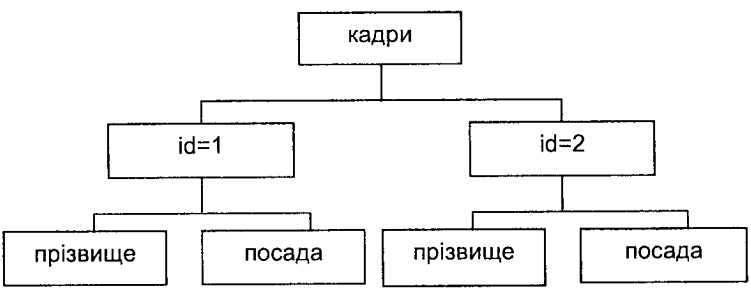


Рис. 8.1. Ієрархічна структура XML-документа

Другий приклад: електронна таблиця має за допомогою XML обмінюватися даними з базою даних. У цьому випадку таблиця передавалася б у такому вигляді:

```
<ПЕРСОНАЛ>
<СПІВРОБІТНИКИ id="Студенти">
<ПРІЗВИЩЕ> Барабанов </ПРІЗВИЩЕ>
<ОСОБИСТИЙ НОМЕР> ПЗ03-1-23</ОСОБИСТИЙ НОМЕР>
<СТИПЕНДІЯ> 33 </СТИПЕНДІЯ>
<ХОБІ> Музика </ХОБІ>
<IMG
        SRC="1.jpg"> </СПІВРОБІТНИКИ >
<ПРІЗВИЩЕ >Довгаль </ПРІЗВИЩЕ >
<ОСОБИСТИЙ НОМЕР>ПЗ03-1-24</ОСОБИСТИЙ НОМЕР>
```

<СТИПЕНДІЯ> 33 </СТИПЕНДІЯ>
<IMG
SRC="2.jpg">
</СПІВРОБІТНИКИ >
</ПЕРСОНАЛ>.

Поля бази даних стають дескрипторами, які, як правило, «підігнані» до бази даних. У базі даних *ПЕРСОНАЛ* є поля *Співробітники*, що містять нові дані, наприклад *ІМ'Я, ОСОБИСТИЙ НОМЕР, СТИПЕНДІЯ*, а також необов'язкові поля типу *ХОБІ*. Є й фотографія співробітника — у вигляді гіперзв'язку. Для роботи з подібною структурою даних має існувати файл правил, які визначають порядок роботи.

Для цього в початок документа стандарту XML додаються рядки з посиланням на файл правил DTD:

<?XML VERSION="1.0"?>
<!doctype publisher SYSTEM "hnure.dtd">.

Як і в HTML, кожен тег укладається в кутові дужки (< >), і за ним зазвичай подають текст. За текстом має бути тег </...>. Тег може не мати змісту, і тоді за ним має бути кінцевий тег (<СТУДЕНТ></СТУДЕНТ>, чи тег може сам закінчуватися прямим слешем (<СТУДЕНТ/>). Однак, на відміну від HTML, кінцевий тег потрібен завжди.

Коментарі надаються у формі <!- ... ->. XML дозволяє використовувати коментар для оточення тих рядків коду, які бажано зробити недіючими.

XML-документ існує в трьох рівнях коректності:

1. Правильно побудований XML-документ — це такий, у якому елементи структуровані у вигляді дерева з коректно розставленими відкриваючими і закриваючими тегами.

2. Діючий XML-документ — правильно побудований і містить теги, що відповідають оголошенню типу документа. Він містить тільки елементи і значення атрибутів, які відповідають DTD. Хоча XML-документ може підготовлюватися і читатися без DTD, останній важливий для встановлення дієздатності.

3. Синтаксично коректний XML-документ знаходиться поза контролем XML. За логічну структуризацію такого документа відповідає розробник.

Розглянемо *основні конструкції в XML.*

1. Елемент даних — структурна одиниця XML. Вмістом елемента можуть виступати текст, інші елементи, коментар,

CDATA (секції), інструкції. Структура документа визначається набором елементів.

У процесі наступного пошуку в документі клієнтська програма спиратиметься на інформацію, що утримується в структурах документа.

Наприклад:

```
<vus><university id='KNURE'>
        <faculty id='KN'>
                <group id='POAS'>
                  <student>
                  </student>
                </group>
        </faculty>
    </university>
    <university id='KPI'>
</vus>
```

У кожному XML-документі визначається кореневий елемент, із якого починається аналіз (розбір).

Якщо елемент не має вмісту (тобто порожній), для нього потрібно передбачити закриваючий тег:

```
<empty/>, <img/>, <br/>
```

2. Коментар: `<! --  вміст коментаря  -->`.

3. *Атрибути* являють собою пари «*назва — значення*».

4. Escape-послідовності: & ім'я, &# шістнадцяткове подання, строкове позначення символів за допомогою Entity (сутності).

5. Директиви аналізаторів:

```
<? директива? >
```

для управління аналізатором під час розбору документа.

6. Секції CDATA задають область документа, що при розборі розглядається як звичайний текст. На відміну від коментарів, секції можна використовувати у додатках. Усередині секції поміщається будь-яка інформація, що може знадобитися програмі клієнта.

Таким чином, сформулюємо сім правил створення XML-документа:

1) у XML теги поділяють на такі, що відкриваються, такі, що закриваються, і порожні;

2) тіло документа XML складається з елементів розмітки (*markup*) і вмісту документа (*content*);

3) XML-теги призначені для визначення елементів та атрибутів документа;

4) перший тег XML-документа задає номер версії мови, номер кодової сторінки;

5) у XML враховується регістр символів;

6) значення атрибутів укладають у лапки;

7) інформацію, що розташовано в контейнері, розглядають як дані, тобто враховуються всі символи форматування.

Документ, що задовольняє зазначеним правилам, називається *формально правильним*. Аналізатори, що виконують розбір документа, працюватимуть із ним коректно. Перевірку коректності XML-документа проводять спеціальні аналізатори, які називаються що *верифікуючими* (або ж *верифікаторами*).

Взагалі існують два способи контролю правильності структури документа: застосування *dtd* та схеми даних (Semantic Schema).

8.2. Поняття DTD

Зміст тегу визначається в оголошенні типу документа (*document type declaration — DTD*). Це тіло коду, що визначає теги через набір елементів. Наприклад:

```
<!DOCTYPE product [
 <!ELEMENT PRODUCT (product_id,
product_name, unit_of_measure?, specification*)>
 <!ELEMENT product_id (#PCDATA)>
 <!ELEMENT product_name (#PCDATA)>
 <!ELEMENT unit_of_measure (#PCDATA)>
 <!ELEMENT specification (variable, value)>
 <!ELEMENT variable (#PCDATA)>
 <!ELEMENT value (#PCDATA)>
                    ]
```

DTD для XML-документа може бути або частиною документа, або зовнішнім файлом. Якщо це зовнішній файл, то все одно оператор *DOCTYPE* має бути присутнім у документі з аргументом

SYSTEM filename,

де *filename* є іменем файла, що утримує DTD.

Наприклад, якби наведене вище DTD містилося в зовнішньому файлі з ім'ям *xxx.dtd*, то оператор *DOCTYPE* мав би вигляд

<!DOCTYPE product SYSTEM xxx.dtd> .

Такий же рядок має бути першим рядком файла *xxx.dtd*.

Визначення елемента *product* містить список інших елементів, у даному випадку — *product_id, product_name, unit_of_measure* і *specification*.

Знак після *unit_of_measure* означає, що може бути одне або декілька входжень; це необов'язковий елемент.

Знак «*» після *specification* означає, що елемент є необов'язковим, але допускається одне чи більше число входжень. Якщо після якогось елемента списку присутній знак «+», то це вказує на обов'язковість елемента, а також на те, що допускається більше одного входження.

У свою чергу, кожний з елементів списку має бути визначений в одному з наступних рядків.

#PCDATA означає, що тег міститиме вкладений тег, який може бути розібраний браузером.

У XML розрізняють символи верхнього і нижнього регістрів. Усі ключові слова XML складаються із символів верхнього регістра. Символи в іменах тегів набирають у тому ж регістрі, що й у визначенні DTD. У попередньому прикладі імена сутностей (таблиць) складаються із символів верхнього регістра, а імена атрибутів (стовпців) — із символів нижнього регістра.

Атрибути

У тегів можуть бути атрибути. Наприклад, замість перерахування відповідних тегів у визначенні *<!ELEMENT specification (variable, value)>* можна було б додати в DTD такі рядки:

<!ATTLIST specification variable CDATA #required>
<!ATTLIST specification value CDATA #required>

Ці рядки створюють *variable* і *value* як два атрибути *specification*, так що вони не зобов'язані з'являтися у вигляді окремих елементів.

8.3. Засоби розбору XML-документа

Аналізатор XML має містити такі аналізатори: лексичний, синтаксичний, семантичний, прагматичний. Виходячи з цього,

XML-документи можуть бути некоректними, коректними, але недійсними, і дійсними.

Розглянемо різні підходи до аналізу XML-документа.

Для аналізу XML-документа можна використовувати об'єкт MSXML у браузері. Є дві його реалізації: одна — мовою Java (платформонезалежна), інша мовою C++.

Для використання об'єктної моделі XML необхідно застосовувати об'єкт *ActiveXObject ('msxml')*:

var xdoc=new ActiveXObject('msxml'),

де *xdoc* — документ.

У результаті цієї операції змінній *xdoc* присвоюється об'єкт, що має тип MSXML, а його властивості і методи використовуватимуться для доступу до XML-структури.

Об'єктна модель XML-аналізатора має такий набір внутрішніх об'єктів:

1) XML Element — робота з кожним із документів;

2) XMLDocument — робота з документом у цілому;

3) XMLCollection — доступ до елементів за іменем або індексом.

Розглянемо складові частини об'єктної моделі аналізатора.

XMLDocument:

- *root* — повертає кореневий елемент XML-документа;
- *url* — повертає або задає адресу документа;
- *charset* — задає ім'я кодувальної таблиці (за стандартом ISO);
- *version* — повертає версію XML;
- *doctype* — повертає вміст елемента *doctype*;
- *filesize* — розмір файла документа;
- *fileModifiedDate* — дата останньої зміни документа;
- *fileUpdateDate* — дата останнього відновлення документа;
- *mimeType* — повертає вміст елемента *mimeType*;
- *createElement(тип_елемента, 'ім'я_елемента')* — створює елемент для поточного документа.

XMLElement:

- *type* — повертає тип елемента (0 — елемент, 1 — текст, 2 — коментар, 3 — документ, 4 — dtd);
- *tagName='ім'я_ТЕГУ'* — назва тегу;
- *text* — повертає текстовий вміст;

- *AddChild*() — додає новий елемент і всіх його нащадків у поточну гілку. Параметри:

об'єкт типу Element; індекс нового елемента в списку;

- *RemoveChild()* — видаляє елемент і його нащадків (елементи при цьому залишаються в пам'яті);
- *parent* — повертає покажчик на поточний батьківський елемент;
- *children* — повертає колекцію успадкованих елементів. Доступ до елементів може здійснюватися за номером у колекції;
- *SetAttribute()* — установлює зазначений атрибут і його значення. Параметри: *ім'я_атрибута*, *значення*;
- *SetAttribute('color', 'green')*;
- *GetAttribute()* — повертає значення зазначеного атрибута;
- *removeAttribute()* — видаляє зазначений атрибут.

Приклад. Необхідно розробити найпростіший аналізатор на XML.

```
<?xml version="1.0"?>
<univer>
  <student><name id="1">Ivanov</name>
       <age id="a">20</age>
  </student>
      ...
  </univer>
```

Документ із JavaScript-аналізатором:

```
<head>
       <script language="JavaScript">
       var myxmldoc=new ActiveXObject ("msxml");
        myxmldoc.url="url ";
       function MyParse(){
this.document.writeln("Місце розташування документа"+
myxmldoc.url+"<BR>");
       this.document.writeln("Коренева"+myxmldoc.root.tagName);
       this.document.writeln("Розкладка"+myxmldoc.charset);
       this.document.writeln("Кількість"+myxmldoc.root.children.length);
       }
      for (i=0; i< myxmldoc.root.children.length; i++){
       elem=msxml.root.children.Item(i);
       this.document.writeln(i+"-й елемент:"+elem.tagName);
```

```
        this.document.writeln(elem.text);
...
}
MyParse();
</script>
```

Доступ до об'єктів XML можна забезпечити також за допомогою тегу <xml>. Розглянемо його синтаксис.

```
<xml
        id="ім'я_псевдокласу"
        src="url_xml-документа">
xml-контент
</xml>
```

Однак на практиці працювати з тегом <xml> можна тільки за допомогою сценаріїв:

```
<script language="JavaScript">
var mynode=ім'я_xml-документа.document.all.XMLDOMDocument.
node;
</script>
```

Які ж межі застосування XML?

1. За допомогою XML можна розробляти web-проекти, що відображають дані з баз даних найбільш простим способом.

2. Для розбору XML-документів застосовують програмне забезпечення як на стороні сервера, так і на стороні клієнта.

3. Для розбору XML можна використовувати Java-аналізатори в аплетах.

4. Точність інтепретації обмежується *DTD*.

Доступ до об'єктної моделі XML-документа забезпечується після завантаження HTML-документа за допомогою посилання на елемент даних HTML-сторінки.

Існують декілька способів підтримки XML у сучасних HTML-браузерах:

- завантаження XML-документа як звичайної HTML-сторінки;
- форматування XML-документів за допомогою XSL. При цьому XSL застосовується для управління процесом відображення елементів на екрані браузерів, змінюючи залежно від місця розташування документа форматуючі теги. XSL виконує пошук потрібних фрагментів усередині документів і виводить окремі

частини незалежно від іншого вмісту, фільтрує й опрацьовує XML-документи з рекурсивно вкладеними і складнопідрядними об'єктами;

• за допомогою сценаріїв, що одержують доступ до будь-якого елемента документа через відповідний об'єкт. При цьому дані XML-документа завантажуються в сторінку за допомогою тегів (секцій) *CDATA: <xml>, <script>, <object>* або їхніх комбінацій.

Наприклад, XML-документ може завантажуватися в сценарії через методи об'єкта *DOM*, опрацьовуватися за допомогою стильових таблиць у тегу *<xml>* або за допомогою XSL-стилів прямо зі сценаріїв. Певні дії можуть виконуватися в сценарії за допомогою секції CDATA.

Розглянемо *стандартні об'єкти DOM для роботи з XML:*

• *XMLDOMDocument* — реалізує базовий інтерфейс *DOM*;

• *XMLDOMNode* — здійснює маніпулювання окремим вузлом дерева *document*;

• *XMLDOMNodeList* — реалізує список вузлів, містить методи для обходу дерева;

• *XMLDOMParseError* — призначений для одержання інформації про помилки в процесі розбору документа.

Властивості об'єкта XMLDOMNode:

• *nodeName* — повна назва поточного вузла;

• *baseName* — назва вузла;

• *dataType* — тип поточного вузла (*NODE_ELEMENT*, *NODE_ATTRIBUTE*, *NODE_TEXT*, *NODE_CDATA*, *NODE_COMMENT*, *NODE_NOTATION*, *NODE_DOCUMENT* — для кореневого вузла);

• *NodeTypeString* — тип вузла в текстовому вигляді;

• *attribute* — список атрибутів вузла у вигляді колекції;

• *definition* — DTD-визначення для поточного вузла;

• text — повертає текстовий вміст вузла (доступний для запису і для читання) і всіх дочірніх елементів;

• *xml* — поточне піддерево в XML-поданні;

• *childNodes* — колекція дочірніх елементів;

• *firstChild* — перший дочірній елемент;

• *nextSibling* — наступний дочірній елемент;

• *lastChild* — останній дочірній елемент;

• *parentNode* — посилання на батьківський вузол;

• *ownerDocument* — посилання на документ, у якому утримується поточний вузол.

Методи об'єкта XMLDOMNode:

• *appendChild (ім'я)* — додає до поточного вузла новий дочірній елемент;

• *insertBefore (ім'я, r)* — додає дочірній елемент (*r* — ім'я, ліворуч якого буде вставлятися дочірній елемент);

• *cloneNode()* — створює копію поточного елемента;

• *replaceChild (n, r)* — заміняє елемент на новий (*n* — ім'я нового вузла, *r* — ім'я старого вузла);

• *removeChild ()* — видаляє вузол із списку дочірніх елементів;

• *transformNode (StyleSheetName)* — опрацьовує вузол за допомогою зазначеної таблиці стилів, *StyleSheetName* — посилання на об'єкт, що містить XSL;

• *transformNodeToObject()* — опрацьовує вузол, результат передається в інше дерево. Параметри: *SSName* — ім'я таблиці стилів, *nameOblect* — ім'я об'єкта XML, куди передаватиметься вузол дерева.

Об'єкт XMLDOMDocument

Перебуває у верхній частині ієрархії і працює з усім документом. Будь-який документ розглядається як кореневий вузол із вкладеними елементами. В таблицях 8.1 та 8.2 подано описи властивостей та методів об'єкта *XMLDOMDocument*.

Таблиця 8.1

Властивості *XMLDOMDocument*

Властивість	Опис властивості
1	2
parseError	Повертає посилання на об'єкт *XMLDOMParseError*
readyState	Повертає посилання на поточний стан аналізатора: 1 — процес завантаження документа; 2 — завантаження відбулося, але аналіз не розпочато, 3 — інтерактив (задано об'єктну модель, що доступна тільки для читання), 4 — документ розібрано
onreadyStateChange	Посилання на оброблювач подій, викликається кожного разу, коли змінюється властивість *readyState*
ontransformnede	Викликається з кожною зміною вузла стильових таблиць

Закінчення табл. 8.1

1	2
doctype	Повертає тип документа
url	Повертає адресу
documentElement	Посилання на кореневий елемент документа

Таблиця 8.2

Методи *XMLDOMDocument*

Метод	Опис методу
loadXML(stringXML)	Завантажує об'єкт XML
save(url)	Зберігає документ у файлі
abort	Перериває процеси завантаження і зберігання
CreateNode(t, n, s)	Створює вузол типу *t* з ім'ям *n* і префіксом *namespace-s*
CreateCDATASection()	Створює секцію, параметр — значення області *CDATA*
CreateAttribute()	Створює новий атрибут для поточного елемента
CreateElement()	Створює новий елемент
CreateComment()	Створює новий елемент *Comment*
CreateTextNode()	Створює текст усередині документа
getElementByTagName()	Повертає посилання на колекцію елементів документа за іменем тегу
nodeFromID()	Шукає елемент за його ідентифікатором
hasChieldNode()	Повертає істину, якщо поточний вузол містить піддерево

Об'єкт XMLDOMNodeList

Має глобальну властивість *length* — повертає кількість елементів у списку вузлів.

Методи:

- *item(n)* — повертає об'єкт *XMLDOMNode*, *n* — номер елемента в списку;
- *nextNode()* — повертає такий елемент у списку. При першому виклику встановлюється на перший елемент списку;
- *reset()* — скидає покажчик на елемент.

Об'єкт XMLDOMParserError

Має властивості:

- *errorCode* — повертає код помилки і 0 — якщо помилки немає;
- *url* — містить url оброблюваного документа;
- *filepos* — повертає зсув щодо початку файла того фрагмента, де відбулася помилка;
- *line* — повертає номер рядка з помилкою;
- *reason* — опис помилки;
- *srcText* — текст рядка, у якому відбулася помилка.

Створення об'єктної моделі XML-документа

Першим кроком у роботі з XML-документом є створення об'єкта, що реалізує клас документа.

Наприклад:

```
<script language="JavaScript">
        var myxmldoc=newActiveXObject("Microsoft. XMLDOM")
```

Далі виконують такі дії:

1. Завантаження XML-документа. Для цього можна використовувати метод *load*:

myxmldoc. loadXML(url або рядок, що містить фрагмент документа)

2. Аналіз документа. Проводиться безпосередньо після навантаження. Для управління аналізом використовуються такі властивості об'єкта *XMLDOMObject*:

- *async* — зміна режиму опрацювання (синхронний або асинхронний);
- *preserveWhiteSpace* — визначає, чи ігноруватимуться у процесі набору символи поділу;
- *resolveEnternals* — визначає, чи розбиратимуться у процесі аналізу *dtd*;
- *validateOnParse* — вмикає або вимикає верифікацію документа.

3. Опрацювання помилок. Використовується властивість *errorCode* для визначення, чи були помилки.

4. Зберігання документа. Використовується метод *save*:

myxmldoc.save("url").

XML-документ можна зберегти в іншому XML-документі, якщо як параметр передати посилання на цей документ.

У процесі розбору документа обхід дерева елементів проводять у такому порядку: обхід піддерев і перегляд потрібного піддерева (може бути рекурсивним); навігація по документу; при цьому використовується або перебір масиву елементів у циклі, або метод *NextNode*.

Пошук елемента здійснюється за схемою: *selectNode()*. Параметр: рядок XSL-запиту.

Далі докладніше розглянемо застосування XSL.

8.4. Основи XSL

Каскадні таблиці стилів CSS дозволяють змінювати форматування відомих тегів HTML і визначати нові теги. В принципі CSS можуть служити і для форматування документів XML.

Головна перевага XML полягає в тому, що він подає формат документа для можливих маніпуляцій у вигляді деревоподібної структури. На жаль, CSS не здатні взаємодіяти з деревом і можуть тільки форматувати документи XML «як вони є». Можна вивести документ на екран у будь-якому форматі, але не можна здійснити яке-небудь вибіркове подання його даних без застосування мови сценаріїв. Більш того, для використання CSS доведеться вивчити ще один синтаксис. Зазначені обмеження привели до створення XSL.

XSL — це додаток XML із власною семантикою (фіксованим набором елементів). Він може бути використаний для створення таблиць стилів (шаблонів документів), зрозумілих будь-якій програмі розбору XML. Шаблони XSL підпорядковуються специфікації *XSLT*.

Таблиці стилів XSL описують, яким чином документи XML мають перетворюватися в інші формати, такі як HTML або RTF. Однак таблиці стилів XML — це дещо більше, ніж просто перетворювачі форматів; вони також надають механізм для маніпулювання даними. Наприклад, дані можна сортувати, робити за ними пошук, видаляти або додавати прямо з браузера.

XSL-запит є засобом визначення шляху до ресурсу. Допускаються спеціальні символи:

- '..' — посилання на батьківський елемент;

- '*' — виділення всіх дочірніх елементів;
- '.' — посилання на поточний елемент.

Для стилів XSL використовуються *MIME*-типи *text/xml* і *application/xml*. Для звертання до елементів із простору імен елементів використовується префікс *XSL:im.*

Шаблон містить елементи, що визначають фіксовану структуру кінцевого елемента.

Стиль містить набір правил і шаблонів. Структура шаблону є такою:

```
<?xml version="1/0"?>
<XSL: stylesheet xmlns: XSL="url">
// наприклад:"W3C.org/tr/WD — XSL"
<XSL: template match="/">
//вказує, що шаблон застосовуватиметься до всього вихідного
//документа XML
...
<XSL: template>
</XSL: stylesheet>
```

xmlns (ns — простір імен) — декларація, що вказує на те, що документ підпорядковується специфікації XSL.

Для застосування в XML-документі після першого рядка необхідно вказати:

```
<?xml. stylesheet type="text/XSL" href="1. XSL"? >
```

Розглянемо таку таблицю стилів:

```
<?xml version = "1.0" ?>
  <xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<!–декларація, що документ є таблицею стилів і що він пов'язаний з xsl: namespace –>

<xsl:template match="/">

<!–застосувати шаблон до усього у вихідному документі XML –>

<HTML>
    <BODY>
<H1> Contacts
```

```
<xsl:for-each select=" contacts/author">
<H2>Name:
<xsl:value-of select="last_name"/>
<P>Title:
<P>Publication:
<P>Street:
<P>City:
<P>State:
<P>Zip-code:
<P>E-Mail:
</xsl:for-each>
        </BODY>
    </HTML>
</xsl:template>
</xsl:stylesheet>
```

За умови збереження на диску цей шаблон буде застосовано до документа *contacts.xml* у разі додавання до нього наступного рядка після першого:

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

Однак XSL може діяти як функція текстового процесора *merge-print*. Визначений як невід'ємна частина простору імен XSL, елемент *xsl:for-each* повідомляє процесору про те, що він має циклічно обробляти всі вузли у вихідному файлі XML. Атрибут *xsl:value-of* вставляє значення вузла XML в елемент HTML.

Отже, за необхідності повернутися до *contacts.xml* і вставити десятки або сотні контактних адрес, то вони без будь-яких змін будуть відображені в таблиці стилів. Завдяки тому, що інформацію про форматування потрібно передати тільки один раз, XML і XSL збільшують пропускну здатність.

Таблиці стилів XSL імітують функцію *merge-print* ще й в тому, що вони дозволяють вибірково опускати поля даних при відображенні документа. Крім того, дані, що виводяться, можуть бути відсортовані за будь-яким конкретним полем даних. Для сортування бази даних контактних адрес в прямому алфавітному порядку елемент *xsl:for-each* слід змінити в такий спосіб:

```
<xsl:for-each
        select="contacts/author"
        order-by="+last_name">
```

Технологія XSL здатна також здійснювати умовну трансформацію висновку в залежності від значень різних елементів або атрибутів. Більш того, він дозволяє запитувати дані з використанням різноманітних операторів шаблонів, символів підстановки, фільтрів, булевих операторів. Технології XML і XSL не призначені для заміни SQL, до того ж навряд чи знайдеться багато бажаючих зберігати свої бази даних безпосередньо у форматі XML. Технологія XSL відкриває можливість різноманітного пошуку за даними після їхнього завантаження в браузер.

8.5. Контрольні запитання

1. Назвіть властивості мови XML.
2. У чому відмінності XML від інших мов розмітки?
3. Опишіть структуру XML-документа.
4. Назвіть рівні коректності XML-документа.
5. Для яких задач використовують XSL?

8.6. Завдання для самостійного виконання

1. Створити XML-документ «Кадровий облік університету».

Перевірити обробку створеного документа в браузері, що підтримує XML.

2. Створити XML-проект за одною із тем:
   1) навігаційне меню — перелік книжок для магазину/бібліотеки за розділами (пошук за тематикою, автором, ключовими словами);
   2) навігаційне меню — перелік товарів у магазині за відділами — вибір кошика покупця;
   3) розклад руху поїздів/літаків/автобусів на вокзалі/аеропорті/автовокзалі — пошук маршруту;
   4) дошка об'яв (у кожного учасника свій стиль) — пошук об'яви за необхідною тематикою;
   5) меню в кафе (формування замовлення);
   6) електронний словник (Simple Lingvo);
   7) тематичний класифікатор тексту (словник ключових слів за тематикою).

Розділ 9
ОСНОВИ РОЗРОБКИ ІНТЕРНЕТ-ПРОЕКТІВ

9.1. Етапи створення Інтернет-проекту

Після визначення, який Інтернет-проект ми хочемо реалізувати і чи зможемо ми втілити поставлені задачі в життя, починається етап безпосередньо проектування і розробки. На рис. 9.1 наведено основні етапи створення Інтернет-проекту.

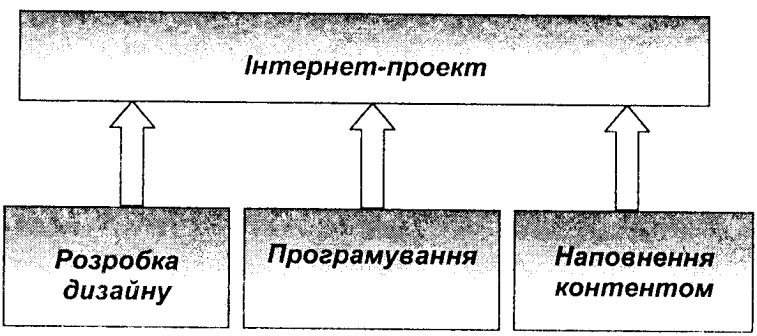


Рис. 9.1. Етапи створення Інтернет-проекту

Проектування починається з розробки *організаційної структури* сайта. Вона залежить від поставлених цілей і тих пріоритетів, що були визначені на початковому етапі. Найкраще відображати організаційну структуру за допомогою схеми чи діаграми, тоді з першого погляду видно, які розділи присутні на сайті і як вони взаємозалежні.

Далі визначають пункти і підпункти головного меню, їхній логічний поділ. Найбільш значимі пункти, як правило, виносяться до початку списку, щоб відвідувач сайта звернув на них увагу.

Схема організаційної структури є, разом з *технічним завданням* і *специфікацією*, одним з основних документів, якими керуються в процесі розробки.

Під час проектування уточнюються потреби в ресурсах, визначаються графіки виконання робіт — це ітеративний процес, що повторюється в міру додавання нових вимог, уведення нових параметрів і залежить від вибору того чи іншого підходу до програмної реалізації. Етап проектування закінчується написанням *специфікації* — посібника з реалізації з описом технічних подробиць і застосовуваних методів. Лише після досконального проробляння структури сайта починається втілення проекту в графіку і програмному коді.

Дизайнер розробляє концепцію графічного виконання сайта з огляду на фірмовий стиль компанії чи торгової марки (за необхідності). Зазвичай варіантів оформлення багато, і затверджується найбільш прийнятний та адекватний ідеології сайта. Обраний варіант оформлення приймається як робочий і допрацьовується: у рамках затвердженої концепції незначно змінюються елементи оформлення, формуються дрібні деталі тощо.

Програміст у цей час проектує структуру бази даних, якщо це необхідно, а також аналізує можливість використання й адаптації типових модулів. Каркас сайта поступово «обростає» новими розділами, що наповнюються в міру обробки отриманої від замовника інформації.

Після того, як дизайнер завершує основну роботу, сайт інтегрується з так званим *движком сайта* — програмною частиною, що забезпечує логіку його функціонування і взаємодії всіх модулів. Модульна структура дозволяє швидко вносити зміни в уже існуючий код і дизайн, тому будь-які удосконалення здійснюються оперативно і «безболісно» для інших частин.

У міру виконання плану робіт з розробки сайта розробляють технічну документацію з описом його характеристик, функціональних модулів і їхніх взаємозв'язків, файлів із програмним кодом і таблиць бази даних. Там же описують логіку функціонування сайта і варіанти дій у випадку, якщо в процесі експлуатації знадобиться незначною мірою розширити деякі можливості сайта без залучення фахівців, силами адміністратора сайта. За потреби розробляють посібник з адміністрування і підтримки сайта, хоча, як правило, цього не роблять, оскільки інтерфейс має бути інтуїтивно зрозумілим.

Останні дні перед завершенням проекту в календарному плані робіт приділяють для фінального тестування розробленого продукту і розміщення його на *хостингу*.

9.2. Usability Інтернет-проектів

Усім відома приказка: «за одежинкою зустрічають», і коли ця одежинка — це дизайн сайта, то до оформлення проекту та зручності навігації необхідно ставитися професійно. Однак не слід ставити дизайн до вершини кута: яким би красивим не був сайт, він так і залишиться «у гордій самітності» без відвідувачів, якщо він нецікавий з погляду надання користувачам інформації і не відповідає їх чеканням і вимогам.

Під час розробки графічного оформлення в першу чергу необхідно спроектувати інтерфейс користувача.

Інтерфейс — це сукупність наданих програмним продуктом засобів для взаємодії користувача з програмою. Інтерфейс сайта можна порівняти з кнопками на пульті дистанційного керування до телевізора: якщо вони незручно розташовані, це викликатиме роздратування щораз при його використанні. Те ж саме і з сайтом: недостатньо продумана система навігації ускладнює перегляд ресурсу і в остаточному підсумку викликає незадоволеність відвідувачів. Однак на відміну від пульта дистанційного керування, для заміни якого потрібні деякі ресурси і час, у випадку з web-сайтом користувач просто може перейти на аналогічний за тематикою і наповненням, але більш зручний сайт.

Зручність інтерфейсу, логічність навігації, можливість оцінити поточне місце розташування в ієрархії документів, простота освоєння інтерфейсу, а також суб'єктивне задоволення користувача від роботи із системою — ось далеко не повний список параметрів, обумовлених поняттям *usability* (*«юзабіліті»*).

Якоб Нільсен, один із найвідоміших фахівців у галузі експлуатаційних характеристик web-вузлів і питань usability, доктор фізичних наук, керівник компанії Nielsen Norman Group, визначає usability як «міру якості роботи користувача в деякому інтерактивному середовищі, будь це web-сайт, традиційне програмне забезпечення чи будь-який інший пристрій, з яким може так чи інакше працювати користувач».

Визнаний російський дизайнер Артемій Лебєдєв пише: «задача *usability* — зробити так, щоб було зручно і зрозуміло. Це ергономіка в проектуванні способів взаємодії людини і будь-якого предмета — від домкрата до курсора миші. *Usability* — така ж широка галузь діяльності, як і психологія. Не можна виписати на аркуш усі випадки її застосування».

Міжнародна організація стандартизації (ISO) у документі «Draft International ISO DIS 9241-11 Standard» надає власне визначення *usability*. Відповідно до нього, це «ступінь ефективності, зручності, з якими продукт може бути використаний конкретними користувачами для досягнення конкретних цілей у визначеному контексті».

Проблеми *usability* вивчають багато наук — ергономіка, інженерна психологія, психологія праці, естетика праці, і зараз відзначаються тенденції до поглибленого вивчання цього питання.

Чи настільки важливо акцентувати увагу на зручності експлуатації сайта? Так, безумовно. За даними дослідження компанії Forrester Research, інтернет-магазини втрачають біля 50 % покупців, які просто не можуть знайти потрібний товар, і близько 40 % користувачів не повертаються на сайт, з яким мали негативний досвід роботи. Цифри говорять самі за себе: скупий платить двічі, і недостатня увага до проектування інтерфейсу на етапі розробки сайта може вилитися в слабку відвідуваність ресурсу згодом.

У загальному випадку сайт повинен:

1) бути адекватним чеканням користувачів;

2) бути простим у використанні, з інтуїтивно зрозумілим інтерфейсом;

3) мати адресу, легку для запам'ятовування при наступних відвідуваннях.

Тут варто торкнутися поняття «інтуїтивно зрозумілий інтерфейс», яке досить часто зустрічається в тематичній літературі та матеріалах, що стосуються розробок в цій галузі. Інтуїція — це вже з галузі психології, і кожна людина володіє цією здібністю в різній мірі. Тому в даному випадку вірніше було б мати на увазі звичність інтерфейсу для користувача, передбачуваність поведінки системи при використанні наданого інтерфейсу.

У наведеному вище прикладі з пультом дистанційного керування не все так просто. Візьмемо гіпотетично «поганий» пульт. Чим він може не сподобатися? По-перше, занадто великими чи занадто маленькими кнопками. Паралелі з сайтом напрошуються самі собою: необхідно намагатися, щоб користувачу не доводилося напружено поцілювати курсором миші в маленький пункт меню або відволікатися від читання тексту через кнопки «Відвідайте наш форум» на півекрана.

Користувач завжди має знати, куди він потрапить, натиснувши на елемент навігації чи просто на посилання. Тому назви для них повинні бути однозначними і не допускати розбіжностей. При проектуванні піктограм і графічних зображень, що виступають як елементи навігації, необхідно бути уважними. Невдалі навігаційні піктограми без пояснень сприйматимуться як загадкові малюнки, що ведуть «у невідомість». Рекомендується використовувати звичні загальновживані піктограми:

- «Будиночок» — посилання на головну сторінку сайта. Краще використовувати схематичне зображення, аналогічне кнопці «Додому» («Home») у браузері Internet Explorer;
- «Конверт» — посилання на сторінку зворотного зв'язку на сайті або посилання для відправлення листа на зазначену адресу за допомогою поштової програми, встановленої в системі (наприклад, ** маршрутизатор **). Іноді для позначення таких посилань використовують зображення поштової скриньки, однак, на думку авторів, це більш поширено на англомовних ресурсах;
- «Лу́па» — посилання на сторінку, що надає можливість пошуку по сайту. Ця піктограма також може застосовуватися поруч з полем для введення ключових слів для пошуку. Рідше застосовується зображення бінокля.

У більшості випадків застосування власних позначень замість загальновживаних призводять до заплутування користувачів, яким доводиться вгадувати їхнє призначення. Це все одно, що для позначення кнопки зміни яскравості використовувати не сонечко, а зірочку: так, зовні всі логічно, сонце — це теж зірка, а інші зірки теж світять. Але користувач не повинен робити таких розумових пошуків і гаяти час на висновки в перший раз, а при повторному використанні пульта ще і згадувати, що ж відбулося при натисканні цієї кнопки минулого разу.

Звісно, при розробці великих комерційних проектів проектуванням інтерфейсів займаються фахівці в цій галузі, а якщо мова йде про сайти з десятками або навіть сотнями тисяч відвідувань у день, то проводяться дослідження за участю різних груп користувачів, різними характеристиками їхніх комп'ютерів і способів доступу в Інтернет. Чим зручнішим буде інтерфейс і чим менше відвідувачі плутатимуться в навігації, тим швидше

вони одержать потрібну їм інформацію, що при таких обсягах відвідувачів може значно скоротити навантаження на сервер. Тобто кошти, вкладені в поліпшення usability, гарантовано окупляться.

На жаль, далеко не кожна організація може дозволити собі такі дорогі дослідження, тим більше, якщо проект некомерційний. Найпростіший спосіб перевірити якість usability — попросити людину, не знайому з розробленим щойно сайтом, знайти на ньому яку-небудь інформацію, і проаналізувати його дії, тобто те, наскільки швидко вона зорієнтувалася в інтерфейсі. Навіть такий простий спосіб дозволить виявити найбільш серйозні недоліки.

Структура розділів сайта, закладена на етапі проектування, є відправною точкою для розробки засобів навігації по сайту, які, в свою чергу, є складовою частиною дизайну. Центральним елементом навігації, як правило, є головне меню.

Проектуючи головне меню, необхідно виходити з пріоритетності інформації, яку необхідно донести до відвідувачів сайта. Верхній рівень меню повинен містити посилання на найбільш значимі розділи сайта, при цьому посилання зазвичай розташовуються в порядку убування значимості.

При використанні графічних меню обов'язковим є застосування атрибута *alt* тегу *<img>* для картинок-елементів меню. Це викликано тим, що багато користувачів переглядають сторінки, відключивши завантаження картинок у опціях свого браузера. Звичайно, можна здогадатися, що посилання *about.html* веде на сторінку «Про компанію», однак за правилами гарного тону під час верстання сторінок все-таки необхідно вказувати заміщуючий текст у картинках, тим більше що не всі знають англійську мову і не всі посилання настільки інформативні. Ось який вигляд може мати, наприклад, код одного елемента меню:

```
<a href="about.html">
<img
src="images/menu_about.jpg"
width="120"
 height="15"
border="0"
 alt="Про компанію">
</a>
```

Іноді доцільно дублювати основне меню в нижній частині сторінки, особливо якщо вона досить «довга» по вертикалі, — тоді користувачам не доведеться прокручувати документ до початку, щоб скористатися меню. Можна також дублювати не всі елементи меню, а лише найбільш важливі його пункти, або просто створити посилання на початок поточної сторінки, наприклад:

```
<a href="#">На початок</a>
```

Якщо приймається рішення дублювати основне меню в нижній частині, то доцільно зробити це за допомогою звичайних текстових посилань, а не графіки: пам'ятайте, що це меню в першу чергу має бути функціональним, а завантаження додаткової графіки — це ще один іспит для користувачів із низькою пропускною здатністю каналу. До того ж, знову спливає проблема відключеної графіки в браузері користувача.

Повернемося до прикладу з пультом керування. Як користувач довідується, на який канал він щойно переключився? Правильно, за номером каналу, що відображається на екрані, або за допомогою логотипу (позначки) телекомпанії, що розміщується десь у куточку. Іншими словами, користувач завжди знає, який канал він переглядає в даний момент, і навіть не замислюється, чому це так, приймаючи цю зручність як щось саме собою зрозуміле. Зручно? Так, безумовно. Так чому ж багато web-сайтів не дотримуються правила «Користувач завжди може довідатися, де він знаходиться»? Тому рекомендується не повторювати цю типову помилку.

Для цього завжди необхідно задавати назву документа в тегу *<title></title>*. Це дозволить користувачу швидко оцінювати назву поточного документа, навіть якщо він не бачить його заголовка усередині тексту. У загальному випадку можна виділити чотири основних способи задання назви в *<title></title>*. Розглянемо їх на прикладі.

Припустимо, у нас є сайт «Товариства захисту тварин», у розділі «Документи» якого є посилання на всілякі статутні положення Товариства, серед яких є документ «Про охорону рідкісних видів». Заголовок документа можна задати в такий спосіб (роздільник у вигляді двокрапок обраний довільно):

1) *Товариство захисту тварин :: Документи :: Про охорону рідкісних видів.*

Як бачимо, заголовок відображає «логічний шлях» до документа на сервері. Плюсом є те, що відразу видно, де конкретно в ієрархії документів перебуває користувач; такий спосіб дозволяє більш чітко відображати структуровану інформацію. Мінусом є більша довжина рядка заголовка, що може не відобразитися повністю, особливо якщо користувач переглядає сторінку з низькою роздільною здатністю;

2) *Про охорону рідкісних видів :: Документи :: Товариство захисту тварин.*

Практично те ж саме, тільки навпаки. Але навіщо це зроблено? Все дуже просто. Такий спосіб гарантує, що користувач повністю побачить назву самого документа, при цьому будуть збережені деякі плюси попереднього способу. Ще однією перевагою є те, що при збереженні відвідувачем сайта декількох сторінок на диску йому буде простіше орієнтуватися в них: око перш за все «схоплює» початок рядка, а не його кінець. А оскільки сторінки збережені з одного сайта, то й починалися б вони однаково;

3) *Товариство захисту тварин :: Про охорону рідкісних видів.*

Мабуть, це найпоширеніший спосіб, коли спочатку вказується назва сайта, а потім назва конкретного документа, без проміжних розділів. Аналогічно можна спочатку вказувати назву документа, а потім ім'я сайта;

4) *Про охорону рідкісних видів.*

Цей спосіб найкраще підходить для «знеособлених» документів, документів загального характеру, наприклад, Закон України «Про охорону праці». Існує думка, що через обмеженість довжини заголовка взагалі не варто його захаращувати його нічим іншим, крім назви конкретного документа.

У межах одного сайта найкраще дотримуватся одного способу іменування документів.

Крім того, якщо сайт має чітку ієрархічну структуру, у тексті документа (зазвичай у верхній частині) доцільно розміщати смугу, аналогічну заголовку з 1-го способу, тільки назви розділів подані посиланнями на відповідні розділи. Це привнесе додаткові зручності в інтерфейс. Наприклад:

- *Головна >> Новини >> Архів >> 27.07.2005*

На сайтах з більш-менш складною структурою правилом гарного тону є створення *карти сайта* — деревоподібної структури основних розділів аж до окремих документів на сайті.

9.3. Розробка дизайну Інтернет-проектів

Краще обійтися зовсім без дизайну, ніж примушувати користувача довго чекати на завантаження картинок і споглядати псевдомистецтво. Іншими словами, якщо неможливо доручити розробку дизайну професіоналові, потрібно реально оцінити свої дизайнерські здібності і вміти вчасно зупинитися. Витриманий сайт без зайвих модних «наворотів» і практично без графіки виглядає набагато краще, ніж сайт з абсолютно непрофесійною графікою, перевантажений анімованими картинками і величезними фоновими малюнками.

Найголовніші запитання, які потрібно задати собі, приступаючи до проектування дизайну, такі: яка тематика сайта? які функції виконує сайт? для кого призначений сайт? Саме з розуміння відповідей на ці запитання і починається проектування дизайну.

У першому розділі ми вже розглядали приблизну класифікацію web-сайтів. Дизайн сайта повинен бути адекватним його тематиці. Погодьтеся, сайт компанії з продажу антикварних меблів, що має дизайн у «космічному» стилі, мав би досить дивний вигляд.

Дизайнеру треба чітко уявляти, які функції покладаються на сайт, розуміти специфіку його експлуатації. Так, якщо це — Інтернет-магазин, особливу увагу варто приділити зручності перегляду товару (оптимальний розмір картинки товару, зрозумілі позначення для можливості «покрутити» сторінку). Якщо це віртуальна галерея — слід акцентувати увагу саме на картинах, а не на оздоблювальних деталях сайта, що просто можуть «переважувати» картини і привертати до себе забагато уваги. Сайт новин має містити якомога більшу кількість інформації на одиницю площі, але так, щоб це було прийнятним.

Розуміння функціональних задач, покладених на web-ресурс, що розробляється, допоможе вибрати оптимальний обсяг графічного оформлення. Дуже часто доводиться спостерігати «мистецтво заради мистецтва», коли web-сайт, безперечно, професійно оформлений, але професійно не з погляду web-дизайну, а з погляду володіння інструментарієм для створення такої графіки. У результаті маємо красивий, але абсолютно непридатний до публікації в мережі дизайн.

Уявіть, наприклад, що пошуковий сервер вирішив раптом «порадувати» вас футуристичною картиною «Пошук у Мережі»,

яка б мала розмір 100 Кб і завантажувалася б при кожному звертанні до сервера. Начебто і картинка красива, але щось не так... А все дуже просто: пошуковий сервер НЕ призначений для демонстрації зразків комп'ютерного мистецтва, його функціональне призначення — надання користувачам зручного інтерфейсу для пошуку в мережі. Таким чином, дизайн без функціональності нікому не потрібний.

І, нарешті, відповідь на запитання «для кого призначений сайт?» дозволить більш точно визначити середній обсяг графіки на одну сторінку, а також чіткіше позиціонувати дизайн. Найпростіший приклад: на web-сайт, присвячений бізнесу, як правило, заходять люди з рівнем статку, вищим за середній, отже, канали зв'язку в них імовірно кращі, ніж у середньостатистичних користувачів. Тобто за необхідності збільшення обсягу графіки на сайті не буде критичним.

Перш ніж проектувати власний дизайн, необхідно проаналізувати і порівняти дизайни сайтів подібної тематики. Для цього бажано переглянути як сайти — переможці конкурсів, так і рядові, знайдені у web-каталозі. Рекомендується визначити позитивні і негативні сторони цих сайтів, виділити вдалі і невдалі рішення, оскільки аналіз чужих помилок дозволить менше припускатися своїх.

На початковому етапі розробки рекомендується визначити колірну гаму, у якій буде витриманий сайт, а також гарнітури шрифтів, що використовуватимуться. Дизайнерам-початківцям краще не використовувати гами більш ніж з трьох-чотирьох кольорів, а також використовувати «кричущі» кольори чи навпаки — занадто невиразні відтінки. Як правило, не варто використовувати більше двох-трьох шрифтів на одній сторінці.

Уже з самого початку процесу розробки дизайну розробнику, імовірно, знадобиться додатковий графічний матеріал, якого немає і не може бути в популярних фотокліпартах — логотипи, фото продукції, співробітників чи підприємства організації, подій і таке інше. Тому варто заздалегідь потурбуватися про такі речі, як фотозйомка, сканування плівок чи фотографій (якщо немає можливості використовувати цифрову апаратуру) тощо.

Проектування дизайну взагалі — процес творчий, і давати якісь рекомендації на зразок «Алгоритм створення красивих сайтів» навряд чи можливо. Проте web-дизайн має свою специфіку.

Як уже згадувалося вище, web-дизайнер набагато більше обмежений у використанні великих повнокольорових зображень, ніж дизайнер для поліграфії. Однак іноді є можливість зекономити на розмірі картинок, що завантажуються, використовуючи повторювані елементи — у цьому випадку браузер користувача одержить всього одну картинку, яку потім підставлятиме в потрібних місцях необхідну кількість разів.

У процесі розробки дизайнер повинен мати уявлення, які з елементів дизайну відображатимуться на сторінці картинками, а які — засобами HTML.

У web-дизайні існує поняття безпечної палітри. В часи, коли більшість моніторів могли відображати усього лише 256 кольорів, дизайнери були змушені використовувати тільки ці визначені кольори, щоб сайт відображався однаково на різних комп'ютерах у різних браузерах. З розвитком апаратного (відеосистема комп'ютера, пристрої відображення інформації) і програмного забезпечення комп'ютерів (нові браузери) необхідність використання безпечної палітри поступово послабла.

Однак останнім часом, з урахуванням появи на ринку мобільних пристроїв доступу в Інтернет, багато з яких мають обмеження за кількістю відображуваних на дисплеї відтінків, проблема знову стала актуальною. Для визначення безпечної палітри можна скористатися наведеною в Додатку 2 таблицею Д1. Нагадаємо, що кожний колір задається за допомогою трьох чисел або шістнадцяткового числа вигляду *#RRGGBB*, де RR, GG і BB — відповідно червоний, зелений і синій компоненти кольору (згідно з моделлю подання кольору RGB).

Слід зазначити, що при заданні кольору в атрибуті *color* тегів у деяких випадках можна використовувати символьне позначення кольору, а не числове. Наприклад, замість запису

```
<td bgcolor="#000080">
```

можна записати

```
<td bgcolor="navy">,
```

і результат буде однаковим — браузер однаково сприймає такі позначення. У Додатку 2 наведені назви таких кольорів і їхні значення. Деякі кольори в різних браузерах можуть відображатися по-різному, це слід враховувати, розроблюючи сайт.

Як правило, при проектуванні дизайну web-сайта варто передбачати на сторінці місце для розміщення стандартних блоків банерної реклами і можливість розширення меню за рахунок нових розділів.

І, нарешті, питання вибору інструментарію. У якому редакторі проектувати дизайн? Вибір визначається перевагами розробника і фінансовими можливостями. Звичайно, складно створити шедевр за допомогою стандартного графічного редактора Paint, що поставляється разом з ОС Windows — потрібні більш серйозні інструменти. Загальновизнаними лідерами серед виробників професійних продуктів для дизайнерів є компанії Adobe і Corel. З огляду на вартість цих продуктів, можна порекомендувати пакети з меншими можливостями, але з відповідною ціною чи взагалі безкоштовні.

Завершуючи поради щодо дизайну, наведемо приклади вдалих дизайнів (рис. 9.2 та 9.3).

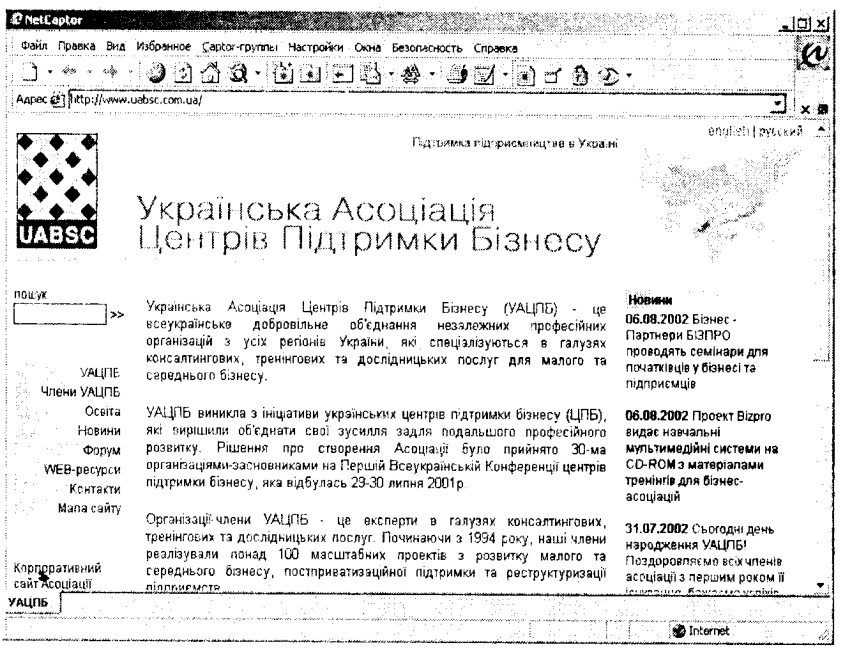


Рис. 9.2. Сайт Української асоціації центрів підтримки бізнесу

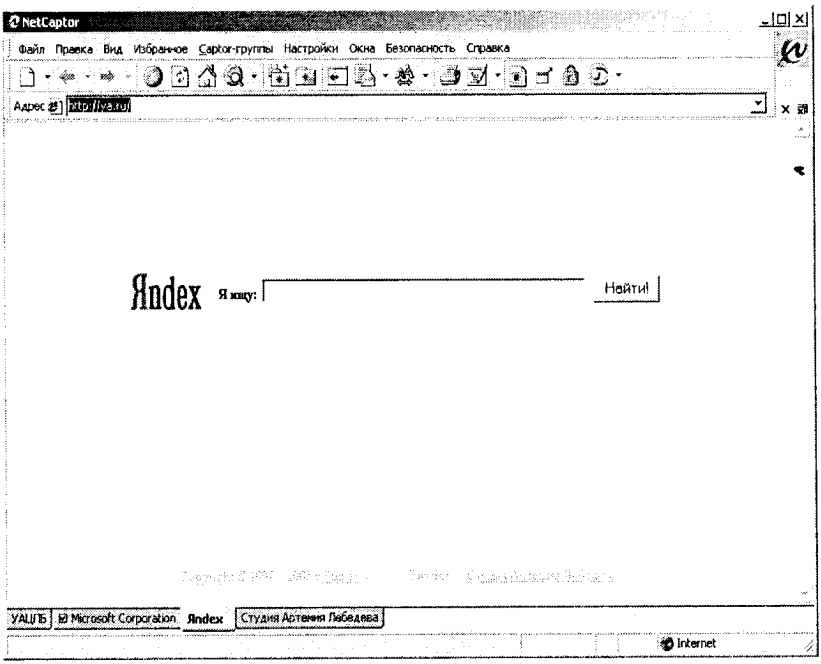


Рис. 9.3. Дизайн інтерфейсу пошукової системи Yandex

9.4. Шрифти у web-дизайні

Шрифт в оформленні web-сайта має не меншу значимість, ніж графіка. Тому правильне використання шрифтів — неодмінний атрибут грамотного дизайну.

Відомо, що в поліграфії і дизайні існують десятки і, можливо, навіть сотні тисяч шрифтів. До того ж за допомогою спеціальних програм можна створювати власні шрифти. Залежно від галузі застосування, розрізняють *видільні, титульні* й *акцидентні* (*декоративні*) текстові шрифти. Крім цього, можна виділити групу шрифтів, призначених для набору спеціальних знаків.

Назва групи текстових шрифтів говорить саме за себе: такі шрифти застосовуються для відображення основного інформаційного тексту. Дизайнер має подбати про те, щоб використовуваний для цих цілей шрифт був максимально читабельним і легко сприймався.

Варто бути обережним при використанні *акцидентних* шрифтів. Ці шрифти відрізняються нестандартним накресленням символів і часто є рисованими. Як правило, сфера застосування таких шрифтів досить обмежена внаслідок їхньої специфічності, і той факт, що шрифт має гарний вигляд в одному дизайні, зовсім не обов'язково приведе до такого ж результату в іншому.

Шрифти можна розділити на *рублені* та *із засічками*.

Рублені шрифти, на відміну від шрифтів із засічками, не мають на кінцях елементів букв додаткових штрихів — засічок. Відмінність першого виду від другого прекрасно видно при порівнянні шрифтів Arial і Times New Roman, які дуже часто використовуються.

Існують кілька загальновідомих класифікацій шрифтів за їх накресленням, включаючи стандарти України, історичну класифікацію і класифікації, розроблені відомими компаніями, що працюють у сфері комп'ютерних технологій і дизайну.

Розглянемо класифікацію, прийняту в операційній системі Microsoft Windows:

- Roman — визначає шрифти з засічками, наприклад Times;
- Swiss — визначає рублені шрифти, що мають змінну товщину штрихів, наприклад Helvetica;
- Modern — визначає шрифти, що мають постійну товщину штрихів, і всі моноширинні шрифти, наприклад Courier;
- Script — визначає шрифти, що імітують рукописні, наприклад Decor;
- Decorative — визначає декоративні шрифти, наприклад готичні;
- Dont know — визначає загальний тип шрифтів і використовується в тому випадку, коли інформація про шрифт недоступна.

Шрифти можуть бути моноширинними або зі змінною шириною символів. У першому випадку ширина будь-якого символу однакова для заданого розміру шрифту. Відповідно, у другому випадку ширина символів різна. Шрифти зі змінною шириною символів мають більше поширення внаслідок своєї більш високої читабельності. Для підвищення останньої інтервали між різними парами символів можуть розрізнятися. Файли метрик шрифту зберігають інформацію про величину інтервалів між парами символів, що називаються *парами кернінгу*.

Деякі шрифти мають кілька різновидів накреслення і можуть бути **жирними** (Bold), *похилими* — виділеними курсивом (Italic) або ***комбінованими***. Відповідні визначення, як правило, присутні в назві шрифту.

Група шрифтів, що складається з основного шрифту і його варіантів, називається сімейством шрифтів або *гарнітурою*. Наприклад, гарнітура Garamond складається з Garamond, Garamond-italic, Garamond-bold, Garamond bold-italic, Garamond demi-bold і Garamond demi-bold-italic.

Якщо обирається шрифт, що відображатиметься в браузері користувача текстом HTML, а не у вигляді рисунка, варто мати на увазі, що використовуваний шрифт може не бути встановленим на комп'ютері користувача. У цьому випадку браузер спробує відобразити текст найбільш близьким за накресленням шрифтом, що є в наявності.

Сучасні графічні редактори дозволяють змінювати базове накреслення шрифту — розтягувати чи стискати символи тощо. Незмінними повинні бути шрифти, що стали класичними, створені видатними майстрами шрифтового мистецтва. Досвідчені дизайнери знають, який шрифт можна використовувати без змін, а який необхідно пристосувати для своїх потреб шляхом трансформації, тому новачкам краще користуватися незмінними гарнітурами шрифтів.

Anti-aliasing (також називається згладжуванням шрифту — font smoothing) — це набір прийомів, використовуваних для відтворення символів на пристроях з низькою роздільною здатністю (таких як монітор). Проблема з відображенням символів шрифту полягає в тому, що символи складаються з контурів, а більшість пристроїв виводять їх окремими точками.

Очевидний спосіб вирішення цього протиріччя полягає в тому, що всі точки (пікселі) всередині контуру розфарбовуються чорним, а поза контуром — залишаються білими (не зафарбовуються). Але тут виникає нова проблема — із точками, розташованими на границі контуру, які не можна однозначно віднести до чорних чи білих. Оптимальний алгоритм полягає в тому, щоби вивести ці крапки сірим кольором. Антиелайзинг (anti-aliasing) і служить для рішення цієї задачі. Особливо помітний вплив антиелайзингу для великих і маленьких кеглів (розмірів) шрифту.

9.5. Захист інформації в Інтернет-проектах

Однією з характерних рис нашого часу є верховенство інформації, цінність якої може перевищувати будь-які матеріальні ресурси. Але, як і в усі часи, існують люди, які намагаються незаконно заволодіти чужими цінностями, в даному випадку — конфіденціальною інформацією з обмеженим доступом. Усім відомі гучні історії про великі акції *хакерів*, однак багато хто, напевно, подумав: «Мій ресурс не настільки значний, отже жодного хакера не зацікавить». Але ніхто не може бути застрахований, чи не так?

Саме тому з'явилася наука, що займається розробкою методів перетворення (шифрування) інформації з метою її захисту від протиправних користувачів — *криптографія*. Безумовно, ніхто не збирається відкидати аксіому «Все, що було зашифровано однією людиною, може бути розшифроване іншою», однак зробити так, аби вартість зусиль, витрачених зловмисником на злом, перевищувала вартість отриманої таким чином інформації, — це необхідна умова зберігання та обробки конфіденційної інформації.

Перше, що можна порадити, — це не винаходити власні методи шифрування. Як влучно зауважив автор відомого шифру PGP Філ Зімерман, «кожен, хто вважає, що начебто винайшов непробивну схему шифрування, — або надзвичайно рідкісний геній, або ж просто наївний та неосвічений...». Тому при трансляції секретної інформації незахищеними каналами, використовуючи при цьому шифрування, реалізуйте тільки перевірені алгоритми або використовуйте вже існуючі надійні модулі.

Розглянемо найбільш поширені підходи до захисту інформації.

Технологія SSL

Будь-якому користувачеві Інтернет добре відома абревіатура HTTP. Найчастіше вона потрапляє на очі в каталогах з посиланнями або в адресному рядку браузера. Нагадаємо, що мова йде про один з основних протоколів, що використовується для обміну інформації. Але абревіатура HTTPS чомусь значно менш відома. Саме цей протокол перетворює звичайний, не захищений канал передачі даних в Інтернеті, в засекречений або захищений; і при більш пильному розгляді ресурсів, які відвідуються, зустрічається не так вже і рідко.

Найчастіше цей протокол використовується в складі будь-якого Інтернет-ресурсу, який здійснює маніпуляції з особистими або фінансовими даними його користувачів, що відвідують Інтернет. Найчастіше це банки, Інтернет-магазини або будь-які інші віртуальні місця, в яких користувачі, що приходять, вимушені передавати свої особисті, а іноді секретні дані. Цього може вимагати і проста реєстрація, і процедура оплати товару, або будь-яка інша процедура, при якій користувачі примушені видавати свої паспортні дані, PIN-коди і паролі. Використовуючи звичайний HTTP-протокол, користувач передає й одержує інформацію в чистому, незашифрованому вигляді. Отже, інформація, що передається користувачем, може бути легко перехоплена і використана сторонньою людиною.

Окрім цього, існує й інша, на перший погляд незначна, загроза. Уявіть, що банк користувача розміщений за адресою *http://MyHomeBank.com*. Тепер уявімо, що деякий зловмисник реєструє іншу адресу, скажімо *http://MyH0meBank.com*, і створює з нею сайт, зовні схожий на сайт банку користувача. Ці адреси так схожі, що рано чи пізно користувач напевно помилиться і випадково потрапить не в банк, а на сайт зловмисника.

Із зазначеного випливають два досить важливих висновки:

- перший — інформацію, що передається, треба шифрувати;
- другий — ми повинні бути впевнені, що передаємо інформацію саме туди, куди потрібно.

Саме для вирішення цих двох питань і використовується технологія *SSL*. Розглянемо принцип дії безпечних протоколів.

Протоколи HTTP і HTTPS

Спроби розробити універсальний мережний протокол, здатний забезпечити належний рівень безпеки для роботи в Інтернет, здійснювалися давно і досить великою кількістю різноманітних фірм і організацій.

HTTP-протокол пропонував достатньо простий, парольний спосіб ідентифікації того або іншого користувача. В момент сполучення з сервером користувач вводив пароль, який передавався серверу у незашифрованому вигляді, і далі, перевіривши відповідність пароля й імені користувача, сервер відкривав або не відкривав з'єднання.

У подальшому, з розвитком мережі Інтернет, було створено декілька різноманітних безпечних протоколів. Офіційний протокол, розробку якого спонсорувала організація *IETF*, називався *Secure HTTP* (*SHTTP*).

Окрім нього, розроблялися і були створені ще декілька неофіційних проектів, один з яких — під назвою *SSL* (*Secure Sockets Layer*), створений Netscape, — отримав більшу популярність і широке розповсюдження. Щоправда, незважаючи на свою популярність, SSL не є офіційним Інтернет-стандартом.

Головним призначенням SSL-протоколу є забезпечення приватного і надійного засобу обміну інформацією між двома віддалено взаємодіючими програмами. Протокол реалізується у вигляді двошарового (багатошарового) середовища, спеціально призначеного для безпечного переносу секретної інформації, через незасекречені канали зв'язку.

Як перший шар у такому середовищі використовують деякий надійний транспортний протокол, наприклад TCP. За словом «транспортний» не важко здогадатися, що TCP бере на себе функції того, «що несе», тобто стає «візником» для всіх шарів (протоколів), що лежать вище.

Другим шаром, що накладається на TCP, є *SSL Record Protocol*.

Разом ці два шари, TCP і SSL Record Protocol, формують своєрідне *ядро SSL*. У подальшому це ядро стає первинною герметичною оболонкою для всіх наступних, більш складних протокольних інфраструктур.

Як одну з таких структур використовують *SSL Handshake Protocol*, який дозволяє серверу і клієнту ідентифікувати один одного і погоджувати криптографічні алгоритми і ключі перед тим, як програми, що працюють на серверній і клієнтський сторонах, зможуть почати передачу або прийом інформаційних байтів у захищеному режимі.

Однією з важливих переваг SSL є його повна програмно-платформенна незалежність. Протокол розроблений на принципах «переносимості», й ідеологія його побудови не залежить від тих програм, у складі яких він використовується. Окрім цього, важливо і те, що поверх протоколу SSL можуть накладатися й інші протоколи: або для подальшого збільшення ступеня захисту цільових інформаційних потоків, або для адаптації криптографічних можливостей SSL під яку-небудь іншу, цілком визначену задачу.

Використання SSL починається в той момент, коли користувач в адресному рядку свого браузера вводить URL, що починається з абревіатури HTTPS. У результаті він підключається до порту за номером 443, що для SSL зазвичай використовується за замовчанням (для стандартного HTTP-сполучення найчастіше використовується порт 80).

У процесі підключення браузер користувача (в подальшому — клієнт) надсилає серверу вітальне повідомлення (hello message). У свою чергу сервер також повинен надіслати клієнту своє вітальне повідомлення.

Ці первинні ініціалізуючі повідомлення містять інформацію, яка буде використана при подальшому налагодженні каналу, що відкривається секретно. В загальному випадку вітальне повідомлення встановлює чотири основних параметри: версію протоколу, ідентифікатор сесії, спосіб шифрування, спосіб компресії, а також два випадкових числа, що спеціально згенерувалися; і сервер, і клієнт, генерують такі числа незалежно один від одного, а після цього просто обмінюються ними.

Після отримання вітального повідомлення від клієнта сервер відсилає свій сертифікат, якщо такий у нього є. Також, за необхідності, сервер може надіслати і деяке ключове повідомлення, наприклад у випадку відсутності сертифіката.

Якщо сервер авторизований (тобто має відповідний сертифікат), він може запитати і клієнтський сертифікат, якщо того вимагає обраний спосіб шифрування даних. Після цього виконується ще низка проміжних обмінних операцій, у процесі яких остаточно уточнюється обраний алгоритм шифрування, ключі і секрети. Далі сервер надсилає клієнту деяке фінальне повідомлення, після чого обидві сторони починають обмін зашифрованою інформацією.

На практиці процес обміну ключами і сертифікатами інколи може займати відносно багато часу. З цією метою часто передбачається можливість повторного використання одних і тих же ідентифікаційних даних. Виникають ситуації, коли після встановлення сполучення з SSL-сервером у користувача з'являється бажання відкрити ще одне вікно браузера і через нього здійснити ще одне підключення до того же SSL-сервера. В цьому випадку, щоб не повторювати весь цикл попередніх обмінних операцій, браузер може відправити серверу ідентифікатор сесії попереднього з'єднання. Якщо сервер прийме цей ідентифіка-

тор, то весь набір шифрувальних і компресійних параметрів буде взятий від попереднього сполучення.

Браузери Netscape також можуть здійснювати так званий «keep alive» запит. При цьому після завершення передачі шифрованих даних встановлене SSL-з'єднання закривається не відразу, а лише впродовж деякого часу.

SSL теоретично може забезпечити практично *повний захист* будь-якого Інтернет-з'єднання. Проте для успішного функціонування SSL, окрім нього самого, необхідні також і суто програмні засоби, що втілять технологію SSL в життя. Програми, які використовують SSL-протокол, як не дивно, є часом найбільш вразливим місцем цієї технології. Саме через помилки в цих програмах можлива майже повна втрата всіх досягнутих після використання SSL рівнів захисту. До таких програмних інструментів передусім відносять Інтернет-браузери.

Одним із найбільш показових критеріїв *рівня захисту* є розмір ключів, що використовуються, або, інакше кажучи, *стійкість шифру*, що використовується. Чим більший цей розмір, тим відповідно надійніше захист. Браузери, в основному, використовують три розміри: 40, 56 і 128 біт. Причому 40-бітовий варіант ключа досить не надійний, його цілком можна розкрити менш ніж за добу. Отже, перевагу слід віддавати саме 128-бітовим ключам.

Стосовно Internet Explorer це означає завантаження додаткового пакета (*security pack*), оскільки інтернаціональні версії цього браузера завжди постачаються з 40- або 56-бітовим захистом. Для визначення, який саме розмір ключа використовується в браузері, в Netscape Navigator достатньо відкрити підменю «Options/Security Preferences», а в Internet Explorer — підменю «Help/About».

Але *розмір ключа* не гратиме вирішальної ролі, якщо в захисті браузера є внутрішні вади системи — проломи. Повідомлення про відкриття таких проломів у тих чи інших браузерах з'являються регулярно. Такий пролом нагадує відкриту кватирку в кімнаті, що протоплюється, все тепло миттєво вивітрюється. З цього приводу доречно згадати випадок, що відбувся з Netscape Navigator у травні 2000 року. Тоді один корейський студент виявив досить неприємну особливість цього браузера. При спробі з'єднання з сервером, що володіє непридатним сертифікатом, з подальшою відмовою від продовження такого сполучення

відбувалося таке. Netscape помилково поміщав цей сертифікат до списку придатних, при наступному підключенні вже не видавав користувачу ніяких повідомлень і спокійно підключався до цього, не цілком надійного, сервера.

Але всі ці та до них подібні проломи не йдуть у жодне порівняння з тією загрозою, яку можуть являти для користувача *сертифікати, які вчасно не були відкликані.*

Справа в тому, що браузери зазвичай поставляються з деяким цілком певним набором дійсних сертифікатів, але автоматичного механізму перевірки їх придатності по завершенню деякого часу — не існує. Таким чином, можлива ситуація, що той або інший сертифікат, який використовується браузером, вже давно втратив чинність: міг скінчитися термін придатності, міг бути втрачений контроль над особистим ключем, відповідним цьому сертифікату, тощо.

У будь-якому з цих випадків сертифікат автоматично відкликається і поміщається в так званий *revocation list*, або список непридатних сертифікатів, що створюється і оновлюється тим або іншим сертифікаційним співтовариством. Якщо не усунути такий сертифікат з браузера, він, як і раніше, вважатиметься придатним, з усіма наслідками, що з цього випливають.

Таким чином, протокол SSL можна назвати одним із найбільш вдалих рішень проблеми захисту даних користувача при їх передачі «відкритим» каналом.

9.6. Основні поняття хостингу

Отже, у вас є розроблений Інтернет-сайт. Що далі? Насамперед потрібно визначитися з тим, хто займатиметься підтримкою нового ресурсу. Найчастіше на невеликих фірмах складно виділити робоче місце новому співробітнику та й найняти його не завжди виявляється можливим. У такому випадку функції адміністратора виконує співробітник, який володіє комп'ютерними технологіями на рівні користувача офісних програм. У такій ситуації вирішальну роль зіграє якість розробки сайта. Професійно розроблений проект — це не тільки те, що бачить відвідувач сайта у вікні свого браузера, це ще й набір інструментів з його супроводу і «розкрутки», так би мовити, «засоби виробництва» адміністратора. Ці засоби виробництва мають бути

простими у використанні і надавати адміністраторові максимум зручностей.

Перерахуємо функції, які має виконувати адміністратор.

По-перше, це доступ до інформаційної частини сайта, до його наповнення. Більшість людей шукає в мережі Інтернет цілком визначену інформацію, і адміністратор зобов'язаний стежити за її регулярним поновленням і наданням відвідувачам того, що їм потрібно.

По-друге, одним з найважливіших показників роботи Інтернет-ресурсу є наявність зворотного зв'язку, тобто відгуків з боку відвідувачів. Адміністратор — це та людина, яка невидима для відвідувачів, але є представником компанії в спілкуванні з ними. Від того, яке враження складеться у відвідувача від листування з адміністратором, багато в чому залежить його зацікавленість у подальшому співробітництві.

По-третє, звичайно ж, адміністратор буде контактною особою у переговорах із провайдером у випадку виникнення потреби в наданні додаткового дискового простору на хостингу, одержанні звітів про розміри трафіка чи для оперативного вирішення проблем, що виникли з вини провайдера. Тому адміністратор має знати координати служби технічної підтримки провайдера — телефон, e-mail, адресу в Інтернеті. Адміністратор повинний бути ознайомлений з умовами договору про надання послуг хостингу і знати права та зобов'язання сторін, а також умови усунення неполадок, що викликані форс-мажорними обставинами. Крім того, дуже бажано, щоб адміністратор ознайомився з інформацією, представленою на сайті провайдера: дуже часто там можна знайти список можливих проблем і шляхи їх вирішення, додаткові послуги. Він цілком відповідає за функціонування сайта.

У разі виявлення помилок або прийняття рішення щодо розширення функціональності сайта адміністратору доведеться контактувати з його розроблювачами.

Після того, як роботи над програмною частиною Інтернет-проекту завершені або добігають кінця, якраз час вирішити питання, пов'язані з *хостингом*.

Хостинг — це безпосереднє розміщення web-ресурсу в Інтернеті, або назва цієї послуги, що пропонують спеціалізовані компанії — хостинг-провайдери. Вони надають місце на диску свого сервера та комунікації для доступу до ресурсів. Звісно,

за певну платню, залежну від пропускної спроможності каналів провайдера, обсягу наданого місця на диску, додаткових сервісів, підтримки різноманітних баз даних та мов програмування. Одне з найбільш повних визначень хостингу було запропоноване консалтинговою компанією PricewaterhouseCoopers: «Web-хостинг включає забезпечення апаратними засобами, підтримку їхньої роботи та керування ними; системні програми, недоторканість контенту, безпеку, високошвидкісне підключення web-сайта до мережі Інтернет. В обмін за надання таких послуг хостинг-провайдер отримує винагороду від власника сайта у вигляді платні при підключенні та абонентської плати».

Може статися, що існує можливість розмістити сайт не в провайдера, а на сервері власника сайта. В цьому випадку необхідно мати виділений канал, відповідні характеристики сервера та людські ресурси — хтось повинен підтримувати роботу апаратної та програмної частини сервера. Додайте до цього цілодобове функціонування та підтримку сайта. Тому такий підхід не завжди виправданий з економічної точки зору (оплата трафіка і кваліфікованого адміністратора сервера коштують недешево), і, скоріш за все, доведеться скористатися послугами хостинг-провайдера.

Тісно пов'язане з хостингом поняття *colocation («колокейшн»)*. Цей термін у перекладі з англійської означає «сумісне розміщення». Це послуга фізичного розміщення комп'ютера замовника на технічній площадці провайдера та підключення його до високошвидкісного магістрального каналу мережі Інтернет.

На відміну від звичайної послуги хостингу, при використанні *colocation* відсутні будь-які обмеження в апаратному та програмному забезпеченні Інтернет-сервера, тобто можна змінювати їх для своїх потреб. Управління сервером здійснюється або віддалено, або безпосередньо на площадці провайдера. Також зазвичай пропонується підтримка роботи сервера спеціалістами провайдера.

Обирати хостинг-провайдера та тип розміщення треба в залежності від необхідних вимог та від наявних матеріальних ресурсів, тобто коштів. Можна скористатися безкоштовними хостингами, але натомість доведеться розмістити на сайті запропоновану рекламу, наприклад, банер або спливаючі вікна

з рекламою. До того ж, такі хостинги не дають жодних гарантій безперервного функціонування сайта, та, як правило, за функціональністю помітно поступаються платним (підтримка БД, докладна статистика і таке інше).

9.7. Контрольні запитання

1. Назвіть етапи створення Інтернет-проекту.
2. Якими документами керуються в процесі розробки Інтернет-проекту?
3. Що таке usability Інтернет-проектів?
4. Які вимоги висуваються до зручності сайтів?
5. Які загальні вимоги висуваються до дизайну Інтернет-проектів?
6. Які шрифти використовуються у web-дизайні?
7. Як організувати захист інформації в Інтернет-проектах?
8. Які завдання адміністратора Інтернет-проекту?
9. Що таке хостинг?
10. Що таке *colocation*?

ДОДАТКИ

Додаток 1
Глосарій англійських термінів

Active Channel
вузол Web, що поставляється автоматично на робочий стіл користувача.

Active Desktop
інтерфейс, інтегрований з робочим столом Windows і броузером Microsoft Internet Explorer.

ActiveMovie
технологія цифрового відео, що дозволяє через Web переглядати файли *avi, QuickTime* або *mpeg.*

Active
термін програмного інтерфейсу технології Microsoft, що дозволяє розробникам створювати інтерактивний контент для Web, а також для компонентів програмного забезпечення, створених різними мовами. Основні елементи технології Active — COM і DCOM.

Address
унікальний код з інформацією, що міститься у файлі конкретної мережі.

Anchor
якір, що утворює гіперпосилання.

ANSI
кодування або таблиця кодування символів — American National Standards Institute.

Applet
аплет, невелика програма (додаток).

API
прикладний інтерфейс програмування — Application Programming Interface.

ASCI
кодування або таблиця кодування символів — American Standard Code for Information Interchange. Один із найстаріших стандартів кодування.

Authorization
право, що дається користувачеві на той або інший ресурс комп'ютерної системи.

Batch-file
пакетний файл (містить сценарій запуску декількох команд або програм).

BBS
Bulletin Board System. Тип комп'ютерного сервісу, призначений для читання і публікації різних повідомлень, передачі або скачування файлів.

Bookmark
закладка — файл Gopher або WWW. Інформація в цьому файлі розміщена так, що отримати доступ до сторінки можна без додаткового серфінгу.

Browser
браузер (або броузер) — програма клієнтської сторони, яка надає графічний інтерфейс для перегляду інформації з мережі Інтернет.

Bullet
маркер, елемент оформлення тексту або списку (коло, квадрат, зірочка тощо).

CGI
сценарій шлюзу (Common Gateaway Interface), який виконується в середовищі Unix аналогічно *batch*-файлам.

Character Set
символи, об'єднані у певну групу, тобто набір символів, наприклад ASCII, ANSI, UNICODE.

Character
символ, введений з клавіатури до комп'ютера.

Chart
діаграма — рисунок, що показує взаємодію даних.

Check Box
прапорець у вигляді хрестика. Один з елементів графічної операційної системи. Дозволяє вмикати або вимикати ту чи іншу дію.

Combo Box
поле зі списком, що розкривається.

CompuServe
комерційна комп'ютерна мережа США.

Cookies
технологія, що дозволяє зберігати індивідуальну інформацію про користувача мережі на комп'ютері користувача.

Cracker
хакер, який зламує складні програмні коди.

CSS
Cascading Style Sheets — каскадні таблиці стилів.

Cybermall
електронний магазин.

DNS
Domain Name System — доменна система адресації. База даних, яка конвертує доменне (складене з літер) ім'я у набір цифр. Цей набір цифр є IP-адресою.

Domain Name Server
ім'я сервера домену. Кожен підрозділ Інтернету має два домени. Основний DNS зазвичай розташовується на мережній машині.

Будь-який хост може отримати відповідний DNS у найближчого інформаційного сервера DNS через мережний протокол DNS. Хост надсилає запит на відому IP-адресу DNS-сервера: свою IP-адресу й ім'я сервера. Сервер DNS знаходить у БД IP-адресу і відправляє на хост. Якщо ж сервер DNS не знаходить потрібну комбінацію, то він відсилає запит на так званий кореневий сервер, який, у свою чергу, звіряє інформацію з файлом настроювань root.cache. Так відбувається, поки ім'я хоста не буде знайдене в мережі. .

Domain
підрозділ мережі Інтернет. У кожного домену є своя мітка. Наприклад, мітки *.com, .net, .org* означають, що це відповідно домени комерційної, мережної і громадської організацій.

Download
закачування програмного забезпечення з іншого комп'ютера на власний вінчестер.

DTD
Document Type Definition — визначення типу документа.

Ethernet
мережна технологія, в якій використовується шинна топологія; працює на швидкості 10 Мбіт/с. Її версії: Fast Ethernet (100 Мбіт/с), Gigabit Ethernet (1 Гбіт/с).

Event Model
модель подій.

FAQ
Frequently Asked Questions — збірники відповідей на запитання, які часто виникають.

Firewall
мережний бар'єр, який створюється з метою, щоб до внутрішньої мережі несанкціоновано не заходили зовнішні користувачі.

Font
шрифт або сімейство шрифтів з індивідуальним стилем.

Freeware
безкоштовне програмне забезпечення.

FTP
File Transfer Protocol — метод пересилання файлів на комп'ютер клієнта та відповідний йому протокол мережі.

FTP-client
програмне забезпечення, яке дозволяє під'єднатися до сервера FTP.

FTP-команди
команди з синтаксисом FTP.

FTP-mail
засіб для отримання файлів від серверів FTP електронною поштою.

FTP-server
вузол-комп'ютер, який надає інформацію і підтримує протокол FTP.

Gateway
шлюз, інтерфейс між двома різними за структурою чи протоколами мережними системами.

GIF
Graphic Interchange Format — один зі стандартних графічних файлів Інтернет.

Gopher
система, яка забезпечує можливість звертання до ресурсів мережі (файлів, баз даних, архівів). Ресурси відображаються у вигляді ієрархічних меню. Є попередником WWW.

Hacker
особа, яка зламує комп'ютерні системи. Може проникати в мережу, наприклад, через порт термінала або порт електронної пошти. Хакер насамперед визначає, чи є на сервері легкі паролі, погано побудоване програмне забезпечення або вразлива операційна система, а потім обирає метод власної безпеки.

Homepage
одна або декілька web-сторінок, які представляють у мережі Інтернет фізичну або юридичну особу.

HTML
HyperText Markup Language — мова гіпертекстової розмітки, що лежить в основі створення документів World Wide Web.

HTTP
HyperText Transport Protocol — протокол, за допомогою якого передаються документи HTML мережею.

Hyperlink
гіперзв'язок — фрагмент тексту або елемент графіки, який посилається на інше місце поточного документа, на інший документ або на інше місце іншого документа.

Hypertext
гіпертекст — це технологія зберігання й обробки текстових документів, що дозволяє встановлювати і підтримувати зв'язки між документами та/або окремими їх фрагментами, забезпечуючи для користувача можливості навігації в такій структурі для переходу від одного її компонента до іншого. Оскільки такого роду організація інформації звільняє користувача від необхідності тільки послідовного перегляду документів, гіпертекст називають також нелінійним текстом. Гіпертекстові технології знаходять широке застосування для побудови різних електронних підручників, енциклопедій і т. д. Особливого поширення гіпертекст отримав у зв'язку з розвитком мережі Інтернет як базова технологія представлення інформаційних ресурсів WWW. Термін «гіпертекст» був введений 1965 року Т. Нельсоном.

Hypermedia
гіпермедіа (гіпермедія) — це узагальнення концепції гіпертексту, що передбачає можливість використання в якості вузлів у структурі гіпертексту не тільки розмічених тестових документів, але й графічних об'єктів, а також інформаційних ресурсів у середовищі аудіо та відео.

Internet Information Server, IIS
інформаційний сервер мережі Інтернет, сервер IIS — мережний сервер, що підтримує численні протоколи; передає інформацію

переважно мовою HTML, використовуючи протокол передачі гіпертексту (HTTP).

Informatics
інформатика — комплексний науковий міждисциплінарний напрямок, що вивчає моделі, методи і засоби збору, збереження, обробки і передачі інформації. Теоретичною інформатикою називають науку про структури, що ґрунтуються на математиці та логіці. Практична інформатика є інженерною дисципліною, що спирається на мережі і системи. До кола її питань входять бази даних і знань, інформаційно-пошукові системи, гіперсередовище, питання мов, комп'ютерного перекладу. Вона спирається на теорію інформації, штучний інтелект, електроніку, семіотику та ін.

Information Society
інформаційне суспільство — концепція постіндустріального суспільства; нова історична фаза розвитку цивілізації, у якій головними продуктами виробництва є інформація і знання.
Характерними рисами інформаційного суспільства є:
— збільшення ролі інформації і знань у житті суспільства;
— зростання частки інформаційних комунікацій, продуктів і послуг у валовому внутрішньому продукті;
— створення глобального інформаційного простору, що забезпечує ефективну інформаційну взаємодію людей, їхній доступ до світових інформаційних ресурсів і задоволення їхніх потреб в інформаційних продуктах і послугах.

Informatization
інформатизація — організаційний соціально-економічний і науково-технічний процес створення оптимальних умов для задоволення інформаційних потреб і реалізації прав громадян, органів державної влади, органів місцевого самоврядування, організацій, громадських об'єднань на основі формування і використання інформаційних ресурсів.

Information system
інформаційна система — частина структури, відповідальна за розробку, експлуатацію й обслуговування комп'ютерних систем. Інша назва — інформаційна технологія, технологія інформаційних систем або обробка даних.

Internet
Інтернет — світова вільна конфедерація комп'ютерних мереж, що поєднує безліч локальних мереж, безліч комп'ютерів і користувачів.

IP Address
4-байтова адреса протоколу Інтернет, що містить номери вузла і мережі.

IP
міжмережний протокол, який дозволяє файлу при пересиланні проходити через різні мережі.

IRC
Internet Relay Chat — спосіб, за допомогою якого користувачі можуть спілкуватися між собою на одному з IRC-серверів.

ISP
Internet service provider — Інтернет-провайдер, компанія, яка надає користувачам доступ до мережі Інтернет.

ISDN
цифрова телефонна мережа, яка забезпечує високу швидкість передачі даних за допомогою спеціального модема.

Java
розроблена фірмою Sun незалежна від платформи мова програмування, мета якої — зробити Інтернет інтерактивним.

Java-applet
додаток, створений мовою Java, який завантажується браузером і виконується у вікні, але у спеціальному віртуальному середовищі — віртуальній машині Java.

JavaScript
на відміну від Java, це мова для створення сценаріїв (скриптів), які виконуються безпосередньо в документі.

JPEG
Joint Photographic Experts Group — графічний формат, прийнятий за стандартний формат при створенні web-контенту.

Lamer
недосвідчений користувач.

Login
реєстраційне ім'я користувача.

Mail server
поштовий сервер — комп'ютер, що обробляє електронну пошту.

Mailing list
список розсилання електронних поштових повідомлень.

Marquee
рядок, що біжить (у документі HTML).

MIME
Multipurpose Internet Mail Extensions — протокол передачі звуку, графіки й інших двійкових даних.

Mosaic
найдавніший графічний користувальницький інтерфейс, що дозволяє переглядати WWW.

MPEG
Moving Pictures Expert Group — протокол, за яким упаковуються відеозаписи.

Netscape Communicator
один із найвідоміших браузерів.

Newsgroup
область повідомлень у конференціях Usenet.

NNTP
мережний протокол, за допомогою якого користуються можливостями конференцій Usenet.

Object Model
об'єктна модель.

Offline
автономний режим роботи комп'ютера.

Online
інтерактивний режим роботи мережного комп'ютера.

Prompt
поле запиту для віддаленого термінала.

Protocol
метод, за допомогою якого передається інформація від хост-комп'ютера до клієнта.

Provider
постачальник послуг, у тому числі й доступу до мережі Інтернет.

SMTP
Simple Mail Transfer Protocol — протокол передачі даних для електронної пошти.

Tag
тег, дескриптор, контейнер.

TCP/IP
Transmission Control Protocol/Internet Protocol — сімейство протоколів, на основі яких передаються дані в мережі Інтернет.

Telnet
протокол емуляції термінала і програма, яка дозволяє одержувати доступ до віддаленої комп'ютерної системи.

URL
адреса в мережі Інтернет, яка однозначно ідентифікує сторінку з даними.

Winsock
Windows-інтерфейс, який забезпечує доступ додатків операційної системи Windows до служб мережі Інтернет.

Додаток 2
Кодування кольору

Наведемо для зручності таблицю Д1, в якій подаються назви кольорів та їх шістнадцяткові й RGB-коди.

Таблиця Д1

Стандартні кольори

Назва англійською	Hex	Red	Green	Blue
1	2	3	4	5
aliceblue	#F0F8FF	240	248	255
antiquewhite	#FAEBD7	250	235	215
aqua	#00FFFF	00	255	255
aquamarine	#7FFFD4	127	255	212
azure	#F0FFFF	240	255	255
beige	#F5F5DC	245	245	220
bisque	#FFE4C4	255	228	196
black	#000000	00	00	00
blanchedalmond	#FFEBCD	255	235	205
blue	#00FFFF	00	255	255
blueviolet	#8A2BE2	138	43	226
brown	#A52A2A	165	42	42
burlywood	#DE8887	222	136	135
cadetblue	#5F9EA0	95	158	160
chocolate	#D2691E	210	105	30
coral	#FF7F50	255	127	80
cornflowerblue	#6495ED	100	149	237
cornsilk	#FFF8DC	255	248	220
crimson	#DC143C	220	20	60
cyan	#00FFFF	00	255	255
darkblue	#00008B	00	00	139
darkcyan	#008B8B	00	139	139
darkgoldenrod	#B8860B	184	134	11
darkgray	#A9A9A9	169	169	169
darkgreen	#006400	00	100	00

Продовження табл. Д1

1	2	3	4	5
darkkhaki	#BDB76D	189	183	109
darkmagenta	#8B008B	139	00	139
darkolivegreen	#556B2F	85	107	47
darkorange	#FF8C00	255	140	00
darkorchid	#9932CC	153	50	204
darkred	#8B0000	139	00	00
darksalmon	#E9967A	233	150	122
darkseagreen	#8FBC8F	143	188	143
darkslateblue	#483D8B	72	61	139
darkslategray	#2F4F4F	47	79	79
darkturquoise	#00CED1	00	206	209
darkviolet	#9400D3	148	00	211
deeppink	#FF1493	255	20	147
deepskyblue	#00BFFF	00	191	255
dimgray	#696969	105	105	105
dodgerblue	#1E90FF	30	144	255
firebrick	#B22222	178	34	34
floralwhite	#FFFAF0	255	250	240
forestgreen	#228B22	34	139	34
fuchsia	#FF00FF	255	00	255
gainsboro	#DCDCDC	220	220	220
ghostwhite	#F8F8FF	248	248	255
gold	#FFD700	255	215	00
goldenrod	#DAA520	218	165	32
gray	#808080	128	128	128
green	#008000	00	128	00
greenyellow	#ADFF2F	173	255	47
honeydew	#F0FFF0	240	255	240
hotpink	#FF69B4	255	105	180
indianred	#CD5C5C	205	92	92
indigo	#4B0082	75	00	130

Продовження табл. Д1

1	2	3	4	5
ivory	#FFFFF0	255	255	240
khaki	#F0E68C	240	230	140
lavender	#E6E6FA	230	230	250
lavenderblush	FFF0F5	255	240	245
lemonchiffon	#FFFACD	255	250	205
lightblue	#ADD8E6	173	216	230
lightcoral	#F08080	240	128	128
lightcyan	#E0FFFF	224	255	255
lightgoldenrodyellow	#FAFAD2	250	250	210
lightgreen	#90EE90	144	238	144
lightpink	#FFB6C1	255	182	193
lightsalmon	#FFA07A	255	160	122
lightseagreen	#20B2AA	32	178	170
lightskyblue	#87CEFA	135	206	250
lightslategray	#778899	119	136	153
lightsteelblue	#B0C4DE	176	196	222
lightyellow	#FFFFE0	255	255	224
lime	#00FF00	00	255	00
limegreen	#32CD32	50	205	50
linen	#FAF0F6	250	240	246
magenta	#FF00FF	255	00	255
maroon	#800000	128	00	00
mediumaquamarine	#66CDAA	102	205	170
mediumblue	#0000CD	00	00	205
mediumorchid	#BA55D3	186	85	211
mediumpurple	#9370DB	147	112	219
mediumseagreen	#3CB371	60	179	113
mediumslateblue	#7B68EE	123	104	238
mediumspringgreen	#00FA9A	00	250	154
mediumturquoise	#48D1CC	72	209	204
mediumvioletred	#C71585	199	21	133

Продовження табл. Д1

1	2	3	4	5
midnightblue	#191970	25	25	112
mintcream	#F5FFFA	245	255	250
mistyrose	#FFE4E1	255	228	225
moccasin	#FFE4B5	255	228	181
navajowhite	#FFDEAD	255	222	173
navy	#000080	00	00	128
oldlace	#FDF5E6	253	245	230
olive	#808000	128	128	00
olivedrab	#6B8E23	107	142	35
orange	#FFA500	255	165	00
orangered	#FF4500	255	69	00
orchid	#DA70D6	218	112	214
palegoldenrod	#EEE8AA	238	232	170
palegreen	#98FB98	152	251	152
paleturquoise	#AFEEEE	175	238	238
palevioletred	#DB7093	219	112	147
papayawhip	#FFEFD5	255	239	213
peachpuff	#FFDAB9	255	218	185
peru	#CD853F	205	133	63
pink	#FFC0CB	255	192	203
plum	#DDA0DD	221	160	221
powderblue	#B0E0E6	176	224	230
purple	#800080	128	00	128
red	#FF0000	255	00	00
rosybrown	#BC8F8F	188	143	143
royalblue	#4169E1	65	105	225
salmon	#FA8072	250	128	114
sandybrown	#F4A460	244	164	96
seagreen	#2E2B57	43	46	87
seashell	#FFE5EE	255	229	238
sienna	#A0522D	160	82	45

Закінчення табл. Д1

1	2	3	4	5
silver	#C0C0C0	192	192	192
skyblue	#87CEEB	135	206	235
slateblue	#6A5ACD	106	90	205
slategray	#708090	112	128	144
snow	#FFFAFA	255	250	250
springgreen	#00FF7F	00	255	127
steelblue	#4682B4	70	130	180
tan	#D2B48C	210	180	140
teal	#008080	00	128	128
thistle	#D8BFD8	216	191	216
tomato	#FF6347	255	99	71
turquoise	#40E0D0	64	224	208
violet	#EE82EE	238	130	238
wheat	#F5DEB3	245	222	179
white	#FFFFFF	255	255	255
whitesmoke	#F5F5F5	245	245	245
yellow	#FFFF00	255	255	00
yellowgreen	#9ACD32	154	205	50

Додаток 3
Особливості застосування графічних форматів

Існують десятки форматів подання графічної інформації, кожний з який відрізняється якістю передачі цієї інформації, орієнтацією на певні апаратні ресурси і платформи, наявністю роздільного подання елементів картинки, можливістю кодування додаткової інформації (наприклад, про авторство), а також наявністю специфічних можливостей.

Графічну інформацію можна представити одним із двох способів: у растровому чи векторному вигляді. В першому випадку у файлі у будь-який спосіб задається інформація про кожну точку зображення, у другому — задається інформація про об'єкти.

У найпростішому прикладі це має такий вигляд: припустимо, у нас є зображення зеленого кола на жовтому фоні. У растровому форматі до файлу буде записана інформація типу «жовта точка, жовта точка, зелена точка...» і т. п. або у випадку з найпростішим алгоритмом стиску «дві жовтих точки, одна зелена точка...». Тобто будуть перераховані всі точки. Чим більший розмір зображення — тим більшим буде розмір файла на диску.

У векторному форматі в нашому прикладі буде задане приблизно наступне: «Фон — жовтий, коло з центром у точці (x, y) і радіусом R, заповнене зеленим кольором».

Для відображення графіки в Інтернеті застосовуються растрові формати, тому докладніше зупинимося саме на них. Не перший раз згадуване обмеження на обсяг і швидкість отримання інформації, що завантажується з мережі, накладає свій відбиток і на цей раз.

Прагнення мінімізувати розмір інформації, що зберігається і завантажується, призвело до розробки алгоритмів стиснення графіки.

Найбільш розповсюджені в Інтернеті формати графічних файлів — GIF (*Graphics Interchange Format*) і JPEG (за назвою авторів — *Joint Photographic Experts Group*), що дозволяють стискати зображення із втратою якості. Останню можна регулювати кількістю кольорів у картинці для GIF (максимум — 256 кольорів), а також ступенем стиску для JPEG (максимум — 100 %).

Таким чином, якщо стискається зображення з палітрою до 256 кольорів, його можна без втрати якості стиснути за допомогою алгоритму GIF.

Існують також різновиди формату PNG (*Portable Network Graphics Format*), однак на цей час у мережі він застосовується досить рідко через неповну підтримку браузерами.

Особливості застосування кожного з форматів наведені в таблиці Д2.

Таблиця Д2

Основні графічні формати мережі Інтернет

Формат	Опис	Застосування
1	2	3
GIF	GIF використовує 8-бітовий колір і ефективно стискає суцільні кольорові ділянки, при цьому зберігаючи деталі зображення. Можна також використовувати формат GIF, щоб створити анімовані рисунки. GIF використовує вільний від втрат метод стиснення. Проте при збереженні 24-бітового рисунка як 8-бітовий GIF зазвичай спотворює якість. GIF також підтримує прозорість фону	Текст, логотипи, ілюстрації з чіткими краями, анімовані рисунки, зображення з прозорими ділянками
JPEG	JPEG підтримує 24-бітовий колір і зберігає яскравість і відтінки кольорів у фотографіях. JPEG стискає файл, вибірково відкидаючи дані, тому стиснення JPEG називається стисненням із втратами. JPEG-метод може спотворити деталь зображення, особливо текст чи зображення з чіткими краями. Можна створити прогресивний файл JPEG, у якому версія рисунка з низькою роздільною здатністю з'являється у вікні перегляду перед його повним завантаженням. Формат JPEG не підтримує прозорість. Коли зберігається образ як JPEG, прозорі піксели заповнюються визначеним кольором	Фотографії. Не використовується для рисунків з прозорими ділянками, дрібними деталями чи просто текстом
PNG-8	Аналогічний GIF, однак підтримується не всіма програмами. Використовує поліпшений формат стиснення даних	*Див.* GIF

Закінчення табл. Д2

1	2	3
PNG-24	Формат PNG-24 підтримує 24-бітовий колір. Подібно до формату JPEG, PNG-24 зберігає яскравість і відтінки кольорів у фотографіях. Подібно до GIF і форматів PNG-8, PNG-24 зберігає деталі зображення, наприклад, у лінійних рисунках, логотипах, чи ілюстраціях з текстом	Те ж саме
	Формат PNG-24 використовує той же вільний від втрат метод стиснення, що і формат PNG-8. З цієї причини файли PNG-24 зазвичай більші за файли JPEG того ж самого рисунка. Подібно до формату PNG-8, формат PNG-24 підтримує прозорість фону. Крім того, формат PNG-24 підтримує багаторівневу прозорість, у якій можна зберегти до 256 рівнів прозорості. Однак багаторівнева прозорість підтримується не всіма програмами перегляду	Фотографії, рисунки, що містять прозорі ділянки, рисунки з великою кількістю кольорів і чіткими краями зображень

Проте основна перевага форматів, що застосовують ті чи інші методи стиснення (особливо з утратою якості), стає недоліком при первинній обробці графічної інформації і роботі з зображеннями. Чим більшу кількість разів зберігати файл в одному з таких форматів, тим більше інформації про зображення буде загублено, отже, якість погіршуватиметься з кожною ітерацією. Тому як робочі формати слід застосовувати формати без втрати якості, наприклад, TIFF, BMP, PSD, і лише при остаточній підготовці графіки для публікації у Web конвертувати в призначені для цього формати.

Додаток 4
Escape-послідовності. Таблиця спеціальних символів

Потрібні для того, щоб подавати спецсимволи в тексті документа, заміняти символьні об'єкти. Деякі службові символи іноді необхідно відображати в тексті документа. Для того, щоб інтерпретатор браузера не сприймав їх як команди, ці службові символи замінили на ESC-послідовності (escape- або ескейп-послідовності).

Escape-послідовності — це набори символів, що дозволяють браузеру відображати недруковані символи, символи грецького і латинського алфавітів, спеціальні символи. Скрізь, де інтерпретатор браузера зустріне escape-послідовність, вона буде замінена відповідним символом. Формат escape-послідовностей такий:

&some_symbols;

де *some_symbols* — деякий набір символів, а знак амперсанта (&) і крапка з комою (;) є обов'язковими.

Крім відображення нерозривного пробілу, найчастіше escape-послідовності використовуються для набору математичних формул у HTML, відображення знаків торгової марки, копірайта, лапок, кутових дужок (які самі по собі є службовими символами — обрамленням тегів) і для відображення знака амперсанта.

Повний список escape-послідовностей наведений у таблицях Д3 та Д4.

Таблиця Д3

Найпоширеніші escape-послідовності

Символ	Комбінація	Приклад
1	2	3
Знак авторського права	©	Copyright © 2005 Microsoft
Зареєстрована торгова марка	®	Sony ®
Торгова марка	™ або ™	Intel™

Закінчення табл. Д3

1	2	3
Знак «менше»	&lt;	<
Знак «більше»	&gt;	>
Амперсант (амперсенд)	&amp;	&
Нерозривний пробіл		
Тире	&#8212;	—
Лапки	&quot;	"

Таблиця Д4

Інші escape-послідовності

Символ	Escape-послідовність	Шістнадцяткове подання	Символ	Escape-послідовність	Шістнадцяткове подання
			Ð	&ETH;	&#208;
¡	&iexcl;	&#161;	Ñ	&Ntilde;	&#209;
¢	&cent;	&#162;	Ò	&Ograve;	&#210;
£	&pound;	&#163;	Ó	&Oacute;	&#211;
¤	&curren;	&#164;	Ô	&Ocirc;	&#212;
¥	&yen;	&#165;	Õ	&Otilde;	&#213;
¦	&brvbar;	&#166;	Ö	&Ouml;	&#214;
§	&sect;	&#167;	×	&times;	&#215;
¨	&uml;	&#168;	Ø	&Oslash;	&#216;
©	&copy;	&#169;	Ù	&Ugrave;	&#217;
ª	&ordf;	&#170;	Ú	&Uacute;	&#218;
«	&laquo;	&#171;	Û	&Ucirc;	&#219;
¬	&not;	&#172;	Ü	&Uuml;	&#220;
-	&shy;	&#173;	Ý	&Yacute;	&#221;
®	&reg;	&#174;	Þ	&THORN;	&#222;

Продовження табл. Д4

Символ	Escape-послідовність	Шістнадцятькове подання	Символ	Escape-послідовність	Шістнадцятькове подання
¯	¯	¯	ß	ß	ß
°	°	°	à	à	à
±	±	±	á	á	á
²	²	²	â	â	â
³	³	³	ã	ã	ã
´	´	´	ä	ä	ä
µ	µ	µ	å	å	å
¶	¶	¶	æ	æ	æ
·	·	·	ç	ç	ç
¸	¸	¸	è	è	è
¹	¹	¹	é	é	é
º	º	º	ê	ê	ê
»	»	»	ë	ë	ë
¼	¼	¼	ì	ì	ì
½	½	½	í	í	í
¾	¾	¾	î	î	î
¿	¿	¿	ï	ï	ï
À	À	À	ð	ð	ð
Á	Á	Á	ñ	ñ	ñ
Â	Â	Â	ò	ò	ò
Ã	Ã	Ã	ó	ó	ó
Ä	Ä	Ä	ô	ô	ô
Å	Å	Å	õ	õ	õ
Æ	Æ	Æ	ö	ö	ö

Закінчення табл. Д4

Символ	Escape-послідовність	Шістнадцяткове подання	Символ	Escape-послідовність	Шістнадцяткове подання
Ç	Ç	Ç	÷	÷	÷
È	È	È	ø	ø	ø
É	É	É	ù	ù	ù
Ê	Ê	Ê	ú	ú	ú
Ë	Ë	Ë	û	û	û
Ì	Ì	Ì	ü	ü	ü
Í	Í	Í	ý	ý	ý
Î	Î	Î	þ	þ	þ
Ï	Ï	Ï	ÿ	ÿ	ÿ

На відміну від тегів, escape-послідовності є регістрозалежними. Якщо, наприклад, написати * * замість * *, то браузер не зрозуміє такого запису.

Додаток 5
Формати файлів у мережі Інтернет

Розглянемо короткий перелік найбільш уживаних розширень файлів Інтернет у табл. Д5.

Таблиця Д5

Формат	Пояснення
1	2
.a	бібліотека відкомпільованих процедур
.afm	Adobe Font Metrics — метрики символів шрифту PostScript
.ai	файл PostScript
.aif, .aifc, .aiff	аудіодані
.ar	архів програми *ar*
.arc	архів програми *arc*, *pkarc* або *arca/arcb*
.arj	архів програми *arj*
.asm	вихідний текст програми на Асемблері
.au	формат зберігання звуку
.avi	формат зберігання відеозображення
.b	вбудований редактор
.bak	попередня (запасна, резервна) версія деякого файла
.bas	вихідний текст програми мовою Basic (GWBasic, TurboBasic, QuickBasic)
.bat	пакетний файл (містить набор команд)
.bgi	Borland Graphics Interface — бібліотека графічних програм-драйверів, які динамічно підвантажуються та залежать від типу відеоадаптера
.bmp	растровий графічний формат
.C	в Unix архів програми compact
.c	вихідний текст програми мовою C
.cfg	конфігураційний файл програми
.cgi	програма, що запускається на виконання, яка працює за протоколом Common Gatawey Interface

Продовження табл. Д5

1	2
.com	двійковий файл, що може виконуватися (невелика програма)
.cpp	вихідний текст програми мовою C++
.ddi	Disk Dupe Image --- образ дискети, створений програмою DiskDupe
.diz	файл із коротким описом продукту або вмісту диска/архіву
.dll	Dynamic Linked Library — у Windows динамічна бібліотека
.doc	документ редактора Microsoft Word для Windows
.dot	шаблон документа редактора Microsoft Word для Windows
.exe	бінарний файл, що виконується
.f	в Unix вихідний текст програми мовою ForTran
.fon	файл шрифту
.for	вихідний текст програми мовою ForTran
.fot	файл шрифту
.gif	Graphics Interchange Format — растровий графічний формат фірми CompuServe
.gz	архів, що створений програмою *gzip* (Unix) і розпаковується за допомогою *gunzip* або *gzip*
.h	у мові C header-файл, що містить описи заголовків процедур
.htm, .html	файл з розміткою мовою HTML
.ice	архів програми *ice*
.iff	формат зберігання звуку, розроблений для комп'ютерів Amiga
.ini	файл з установками програми
.jfif, .jpeg, .jpg	растровий графічний формат JPEG, що дозволяє зберігати зображення із втратою інформації без істотної втрати якості
.l3	MPEG-1: звук тільки Layer-3
.latex, .ltx	одне з розширень TEX
.lha, .lzh	архів програми lzh
.lib	бібліотека процедур

Продовження табл. Д5

1	2
.m1s	MPEG-1: системний потік
.m2a	MPEG-2: тільки звук
.m2s	MPEG-2: системний потік
.m2v	MPEG-2: тільки відео
.me	файл редактора MultiEdit
.me	розширення файла *read.me*
.mia	MPEG-1: тільки звук
.mid	MIDI — звуковий файл
.miv	MPEG-1: тільки відео
.mod	формат зберігання звуку
.mov	формат збереження відео й аудіо
.mpg	MPEG — формат зберігання відео і звуку з компресією і втратою даних
.mps	MPEG-1
.nfo	короткий опис того, що міститься в директорії або на диску
.o	в Unix відкомпільований, але не зібраний для виконання код програми
.obj	відкомпільований, але не зібраний для виконання код програми
.ovl	OverLay — модуль програми, який динамічно підвантажується
.p	в Unix вихідний текст програми мовою Pascal
.pas	вихідний текст програми мовою Pascal
.pbm	Portable BitMap — простий формат збереження чорно-білих зображень
.pcx	растровий графічний формат, підтримуваний більшістю редакторів
.pdf	Portable Document Format — захищений формат зберігання документів
.pfm	PostScript Font Metrics — метрики символів у шрифті PostScript
.pgm	Portable GrayMap — простий формат зберігання півтонових зображень

Закінчення табл. Д5

1	2
.pict	формат зберігання графічних зображень у буфері обміну на комп'ютерах Macintosh
.pif	файл, що описує параметри запуску DOS-задачі під Windows
.pl	файл мовою Perl
.pop	в Unix тимчасовий файл POP3-сервера в тій самій директорії, що і поштові скриньки користувачів
.ppd	PostScript Printer Description — опис принтера для програми, яка друкує *.ps*
.ppm	Portable PixelMap — простий формат зберігання кольорових зображень
.ppt	презентація, створена в програмі Microsoft PowerPoint
.ps	векторний графічний формат PostScript
.rar	архів програми *rar*
.rtf	Rich Text Format – універсальний формат для обміну між текстовими редакторами
.sfx	архів, що саморозпаковується
.sgm, .sgml	Standard Generalized Markup Language — файли мовою розмітки, яка використовується для управління великими масивами документів. Окремим випадком SGML є HTML
sh	пакетний файл, що запускається (мовою shell)
.so	бібліотека, що динамічно приєднується
.snd	звуковий файл
.spl	файл Future Splash Player
.swf	файл анімації ShockWare Flash
.swp	файл підкачки (віртуальної пам'яті)
.sys	файли ядра DOS (IO.sys та MSDOS.sys)
.tar	архів програми tar (Unix) без компресії
.tif, .tiff	Tagged Image File Format — растровий графічний формат
.ttf	TrueType Font — шрифт, який масштабується
.vrml	Virtual Reality Modeling Language — файл мовою VRML
.zip	архів програми pkzip або WinZip

Література

1. Вступ до комп'ютерних інформаційних технологій : навч. посібник. / М. З. Згуровський, І. І. Коваленко, В. М. Михайленко. — К., 2003. — 263 с.

2. *Єрохін, А. Л.* Відкрита система дистанційного навчання та взаємодії менеджерів некомерційних інтернет-проектів : інтерактивний відеотренінг. / А. Л. Єрохін, В. І. Каук, Л. М. Кондак, О. О. Дудка. — Режим доступу : www/URL: http://www.top-study.org/.

3. Основы Web-технологий / П. Б. Храмцов, С. А. Брик, А. М. Русак, А. И. Сурин; под ред П. Б. Храмцова. — М. : ИНТУИТ.РУ «Интернет — Унив. Инф. Технологий», 2003. — 512 с.

4. *Айзенменгер, Р.* HTML 3.2/4.0 : справочник : пер. с нем. — М. : БИНОМ, 1998. — 368 с.

5. *Айзекс, С.* Dynamic HTML : пер. с англ. — СПб. : BHV — Санкт-Петербург, 1999. — 496 с.

6. *Дуванов, А. А.* Web-конструирование. DHTML. — СПб. : БХВ-Петербург, 2003. — 512 с.

7. *Федоров, А. Г.* JavaScript для всех. — М. : Компьютер-Пресс, 1998. — 384 с.

8. *Ломакс, П.* Изучаем VBScript : пер. с англ. — К. : Издательская группа BHV, 1998. — 624 с.

9. *Буч, Гради.* Объектно-ориентированное проектирование с примерами применения : пер. с англ. — К. : Диалектика, 1992. — 519 с.

10. *Эдди, С. Э.* XML : справочник. — СПб. : Питер. — 480 с.

11. *Громов, А.* CSS и DHTML Web-профессионалам / А. Громов, М. С. Каменнова : пер. с англ. — К. : BHV-Киев, 2001. — 272 с.

12. *Хоумер, А.* Dynamic HTML / А. Хоумер, К. Улмен : справочник. — СПб. : Питер, 2001. — 512 с.

13. *Баррет, Я.* JavaScript Web-профессионалам. — К. : BHV-Киев, 2001. — 352 с.

14. *Бабушкин, В. А.* Web-сервер в действии. — СПб. : Питер, 2000. — 272 с.

15. *Пауэрс, Ш.* Динамический HTML. — М. : Лори, 1999. — 63 с.

16. *Тихонов, Е.* Динамический HTML : самоучитель. — М. : БИНОМ, 2000. — 496 с.

17. *Шапошников, И.* Интернет-программирование. — СПб. : BHV, 2000. — 224 с.

18. *Холцшлаг, М.* Использование HTML 4. — Специздание, 6-е изд. — М. : Диалектика, 2000. — 1008 с.

19. *Кусаков, И.* Разработка сложных Web-приложений на примере Microsoft Active Server Pages. — Режим доступа : www/URL: http://www.proglib.ru/articles/art0000045.asp.

20. *Bergsten, H.* An Introduction to Java Servlets. — Режим доступа : www/URL: http://www.webdevelopersjournal.com/articles/intro to servlets.html.

21. *Грин, Г.* Введение в ASP. — Режим доступа : www/URL: http://emanual.ru/download2/5040.html.

22. *Хейфец, И.* Архитектура .NET (обзор). — Режим доступа : www/URL: http://www.got-dotnet.ru/default.aspx?s=doc&d no=24&c no=4.

23. *Старостин, Д.* ASP.NET — повторное использование кода для построения пользовательского интерфейса. Microsoft Corporation. — Режим доступа : www/URL: http://www.gotdotnet.ru/default.aspx?s=doc&d no=22503&c no=4).

24. *Рейли, Д.* Создание приложений Microsoft ASP.NET — М. : Издательско-торговый дом «Русская редакция», 2002.

25. *Вейтман, В.* Программирование для Web. — М. : Диалектика, 2000. — 368 с.

26. *Шарма, В.* Разработка Web-серверов для электронной коммерции. Комплексный подход. / В. Шарма, Р. Шарма. — М. : Вильямс, 2000. — 400 с.

27. *Шапошников, И.* Справочник Web-мастера. — СПб. : BHV, 2000. — 304 с.

28. *Дейтел, Х. М.* Как программировать для Internet и WWW / Х. М. Дейтел, П. Дж. Дейтел, Т. Р. Нието : пер. с англ. — М. : Издательство БИНОМ, 2002. — 1184 с.

29. *Нильсен, Я.* Веб-дизайн: книга Якоба Нильсена : пер. с англ. — СПб. : Символ-Плюс, 2003. — 512 с.

Предметний покажчик

Навчальне видання

САМСОНОВ Валерій Васильович
ЄРОХІН Андрій Леонідович

МЕТОДИ ТА ЗАСОБИ ІНТЕРНЕТ-ТЕХНОЛОГІЙ

Навчальний посібник
для студентів вищих навчальних закладів

Видано за рахунок державних коштів
Продаж заборонено

Редактор *Ю. В. Статкевич*
Коректор *Т. М. Матвієнко*
Комп'ютерна верстка *О. А. Федосєєвої*
Дизайн обкладинки *О. Л. Герасименюк*

Підписано до друку 20.04.2006. Формат 60x90/16. Папір офсетний.
Гарнітура SchoolBookC. Друк офсетний. Умов. друк. арк. 16,5.
Обл.-вид. арк. 15,94. Тираж 7600 прим. Зам. №1931/122.

ТОВ «Компанія СМІТ»
61166, м. Харків, просп. Леніна, 14
Тел.: 8(057) 717-54-94, 702-08-16, факс: 8(057) 702-13-07
E-mail: book@smit.com.ua
http://www.smit-book.com

Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців, виготівників і розповсюджувачів
видавничої продукції ДК № 435 від 26.04.2001

Віддруковано з готових діапозитивів у ТОВ «Навчальний друк»,
62300, Харківська обл., м. Дергачі, вул. Петровського, 163а.
Свідоцтво про держреєстрацію: серія ХК № 58 від 10.06.2002 р.