**МІНІСТЕРСТВО ОСВІТИ І НАУКИ,
МОЛОДІ ТА СПОРТУ УКРАЇНИ**

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Молчанов В. П.

ЗАСОБИ СИСТЕМ ОБРОБКИ ДАНИХ

Навчальний посібник

Харків. Вид. ХНЕУ, 2013

УДК 004.32.6(075)
ББК 32.973-04я7
П91

Рецензенти: докт. техн. наук, професор, зав. кафедри інформатики Харківського національного автомобільно-дорожнього університету *Ніконов О. Я.;* канд. техн. наук, доцент Харківського гуманітарного університету "Народна українська академія" *Козиренко В. П.*

Затверджено на засіданні вченої ради Харківського національного економічного університету.

Протокол № 8 від 24.04.2012 р.

Рекомендовано Міністерством освіти і науки, молоді та спорту України як навчальний посібник для студентів вищих навчальних закладів (лист № 1/11-19104 від 12.12.2012 р.)

Молчанов В. П.

П91 Засоби систем обробки даних : навчальний посібник / В. П. Молчанов. – Х. : Вид. ХНЕУ, 2013. – 100 с. (Укр. мов.)

Подано основні положення щодо створення систем обробки даних та використання мови XML. Наведено особливості застосування сучасних технологій, які пов’язані з розміткою тексту, та додатків на основі XML.

Рекомендовано для студентів галузі знань 0515 "Видавничо-поліграфічна справа".

ISBN

УДК 004.32.6(075)
ББК 32.973-04я7

Вступ

Швидке зростання обсягів інформаційних ресурсів у різних сферах викликає потребу в пошуку нових підходів до зберігання і обробки даних. Інформація і дані стають все більш цінними, для їх використання, накопичення і обробки потрібно все більше витрат. Зростають витрати і на забезпечення їх збереження, вартість відновлення даних у багатьох випадках рівна вартості придбання.

Проблема ефективного використання і збереження ділової інформації має величезне значення і в бізнесі, примушуючи підприємства і організації проводити відповідні заходи аж до створення спеціальних IT-служб. Системи зберігання і обробки даних, що забезпечують реалізацію завдань, пов’язаних з безперервністю бізнес-процесів і збереженням даних, стають все більш затребуваними. Причому справедливо це для організацій з різним рівнем автоматизації і інформатизації процесів: як для компаній, що роблять початкові кроки в автоматизації, так і компаній, що використовують крупні інформаційні системи.

Серед різних підходів до створення таких систем на перше місце висувається концепція відкритих систем, основною рисою якої є використання рішень, які вже стандартизовані (інтерфейсів, протоколів, мов, програмних компонент і так далі) [6; 8; 11; 13]. Найбільш змінною частиною цих систем є використовувані в ході функціонування документи. Їх вміст і форми визначаються наочною областю, а от формати останнім часом прагнуть стандартизувати.

Саме ці питання вивчаються студентами за навчальним планом підготовки магістрів в галузі "Видавничо-поліграфічна справа" у дисципліні "Засоби систем обробки даних".

Вивчення дисципліни дозволяє сформувати компетентності, пов’язані з використанням мови XML, її додатків та інших взаємопов’язаних технологій, відповідно сьомого рівня Національної рамки кваліфікацій. Характеристика компетентностей наведено окремо для кожного розділу.

Матеріал посібника містить приклади, ілюстрації і довідкові матеріали, а також контрольні запитання та практичні завдання, що забезпечують вивчення всіх тем дисципліни.

1. Компоненти СОД

Основна ідея розділу

Розділ присвячено питанням визначення поняття системи обробки даних та особливостям компонент, які можуть входити до її складу. Розглянуто також типові архітектури систем, їх складові та формати документів.

Цілі вивчення розділу

Метою розділу є дослідження складу та основних підходів до створення систем обробки даних, огляд існуючих компонент та форматів документів.

Матеріал, викладений у розділі, надає студенту можливість набути таких ***компетентностей***.

Знання.

Особливості систем обробки даних, їх склад і розподіл функцій за рівнями ієрархії.

Основні формати документів, що використовуються в сучасних системах обробки даних.

Уміння.

Обґрунтовувати склад і тип компонентів системи.

Обґрунтовувати вибір формату документів для обміну між додатками і обробки.

Комунікації.

Аргументована взаємодія з виконавцями при розподілі завдань на розробку документів.

Доведення своїх висновків і результатів роботи до учасників команди проекту зі створення або аналізу функціонування системи обробки даних.

Автономність і відповідальність.

Знаходити і використовувати альтернативні формати для зберігання та обробки документів.

1.1. Системи обробки даних

1.1.1. Основні поняття

Зростання об’ємів даних, використовуваних у різних областях, вимагає спеціальних засобів для роботи з ними. Їх необхідно вводити, перетворювати з одного вигляду в інший, накопичувати, виконувати пошук та інші дії з обробки. З розвитком засобів комунікації і комп’ютерної

техніки обробка даних перетворилася на самостійний напрям індустрії автоматизації різних процесів. З'явилася специфічна предметна область, основні поняття якої визначені у стандартах і літературі [3; 4; 6; 10; 12].

Відповідно до поглядів, що склалися, система обробки даних (СОД) – це комплекс взаємопов'язаних засобів і методів збору і обробки даних, необхідних для організації управління об'єктами. Поняття об'єктів розуміється у широкому сенсі. Наприклад, студент, що виконує за комп'ютером завдання з курсового проектування, використовує СОД (комп'ютер, набір додатків і документів різних форматів).

Основні функції системи обробки даних – збір, зберігання, пошук, обробка необхідних даних з найменшими витратами. Наприклад, завдання відібрати і автоматизувати трудомісткі рутинні операції, що регулярно повторюються, над великими масивами даних. Системи обробки даних – це частина і перший ступінь розвитку АСУ (автоматизована система управління), у функції якої входять, перш за все, виконання дій, пов'язаних з вирішенням завдань управління, з вибором оптимальних варіантів на основі різних методів, моделей і тому подібне.

Проте системи обробки даних функціонують і як незалежні системи. У ряді випадків ефективніше об'єднувати у рамках однієї системи обробку однорідних даних для великого числа завдань управління, що вирішуються у різних АСУ, створювати системи обробки даних колективного користування.

 Досвід створення і експлуатації систем обробки даних дозволяє визначити основні принципи їх побудови і методи розробки.

Принцип інтеграції полягає у тому, що оброблювані первинні дані вводяться один раз. Вирішувані в СОД завдання взаємно пов'язуються, щоб дані вирішення одних завдань використовувалися як початкові для можливо більшого числа інших завдань. Тим самим усувається дублювання операцій збору, підготовки і контролю даних і забезпечується їх комплексне використання, що призводить до зниження питомих витрат на отримання необхідної інформації і підвищення ефективності системи обробки даних.

Принцип централізації полягає у тому, що при створенні системи обробки даних велика частина інформаційних робіт вилучається з ведення відповідних підрозділів і концентрується в єдиному сховищі або невеликому числі таких сховищ. Основу таких сховищ можуть виконувати БД, об'єднані в розподілену систему.

Принцип розподілу функцій за рівнями ієрархії, при цьому ієрархія зазвичай відповідає ієрархії організації або системи, на користь якої створюється СОД. У багатьох випадках система дворівнева (клієнт-сервер).

СОД – поняття дуже широке, має багато напрямів розвитку від звичайної бази даних з деяким призначеним для користувача інтерфейсом (те, що зазвичай створюють студенти на заняттях з відповідної дисципліни) до систем з екзотичними назвами (GRID-системи, хмарні обчислення і тому подібне). Вони відрізняються технологіями (підходами до створення), складом компонент і документів і т. д. Проте практика створення, стандартизація і сучасні тенденції в індустрії обробки інформації призводять до використання в їх складі дуже багато спільних компонент (мови і технології створення, формати документів, протоколи обміну даними, засоби зберігання і т. д.) і архітектури побудови систем.

Більшість систем обробки даних зводитися до декількох типів і може створюватися шляхом нарощування деяких базових систем – добре спроектованих, написаних, відладжених і додатково пристосованих для розвитку. Базові системи можуть, у свою чергу, розроблятися на основі однієї системи, в яку додаються ті або інші можливості. У цьому випадку вони утворюють сімейство базових систем. Для базової системи добиваються дуже високого рівня відладженості, надійності, і використовують їх як основу створення конкретної СОД.

Розглянемо як зразок достатньо просту, але широко використовувану систему, що складається з бази даних, яка розташована на сервері мережі Інтернет, і клієнтської машини, що забезпечує інтерфейс для користувача (рис. 1).

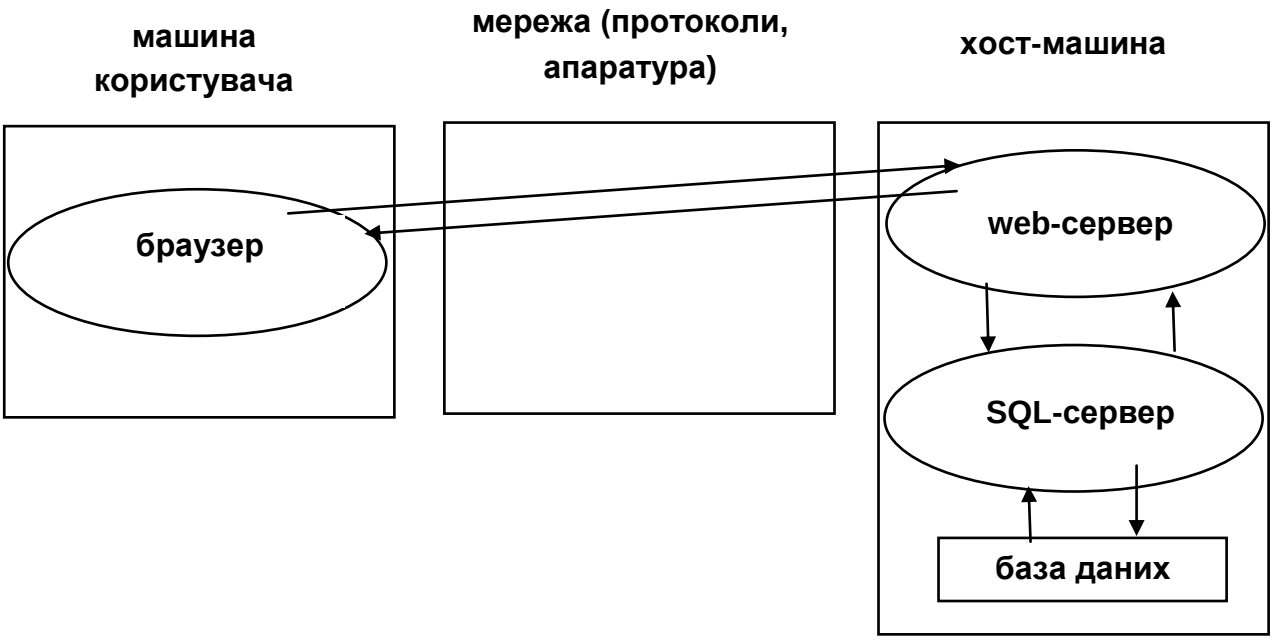

Рис. 1. **Варіант системи обробки даних**

Компонентами цієї системи є програма-браузер на машині клієнта, мережна СУБД (наприклад, MYSQL або MS SQL), яка розміщена в мережі Інтернет, і web-сервер (наприклад, MS IIS або Apache), який здійснює взаємодію з браузером. Усі компоненти існують, розробникові необхідно лише встановити їх і налаштувати. Взаємодія компонент є стандартизованою (протокол HTML, запити SQL).

Створення такої системи на основі компонент, що вже існують, є достатньо простим завданням, яке може бути вирішено навіть у ході навчальних занять. Більш того, ті, хто використовував Joomla (або іншу систему управління контентом) уже знайомий з такою системою.

Які документи циркулюють у цій системі, і які дані піддаються обробці? Текстові документи з використанням мов розмітки, дані з бази даних (для БД реляційного типу – це записи, рядки таблиць).

1.1.2. Архітектура СОД

Системи обробки даних можуть включати різні комп'ютери, сховища даних (дискові масиви для зберігання даних різної організації), програми (виконувані на одному або різних комп'ютерах) і документи, які забезпечують їх взаємодію. Усі ці компоненти взаємодіють між собою, реалізовуючи деяку архітектуру.

Найбільш поширеною архітектурою для створення СОД у даний час є "клієнт-сервер". При цьому виділяються три групи функцій системи:

взаємодія із користувачем або представлення даних;

прикладні, пов'язані з предметною областю, або бізнес-логіка;

управління даними.

Ці функції різним чином розподіляються між двома пов'язаними вузлами комп'ютерної мережі. Залежно від використовуваної моделі розподілу функцій розрізняють різні варіанти архітектури RDA, DBS, AS, Thin Web Client, RIA і так далі. Як правило, всі ці варіанти архітектури є результатом конкретних проектів крупних виробників програмного забезпечення. Вони є певним компромісом переваг і недоліків, знайденим у ході розробки. Найбільш вдалі рішення, підтримані іншими розробниками, і стають типом архітектури. Розглянемо як приклад декілька варіантів архітектури, що є достатньо загальними і що набули широкого поширення: Thin Web Client (архітектура з "тонким клієнтом"), Thick Web Client, RIA (архітектура з "товстим клієнтом") та SOA (архітектура, орієнтована на сервіси).

Для першого варіанта (архітектура з "тонким клієнтом") характерною є реалізація на клієнтській стороні лише функцій представлення даних. Причому найчастіше – це програма-браузер. Функції, пов’язані з прикладною логікою і управлінням даними, реалізуються на сервері (рис. 2).

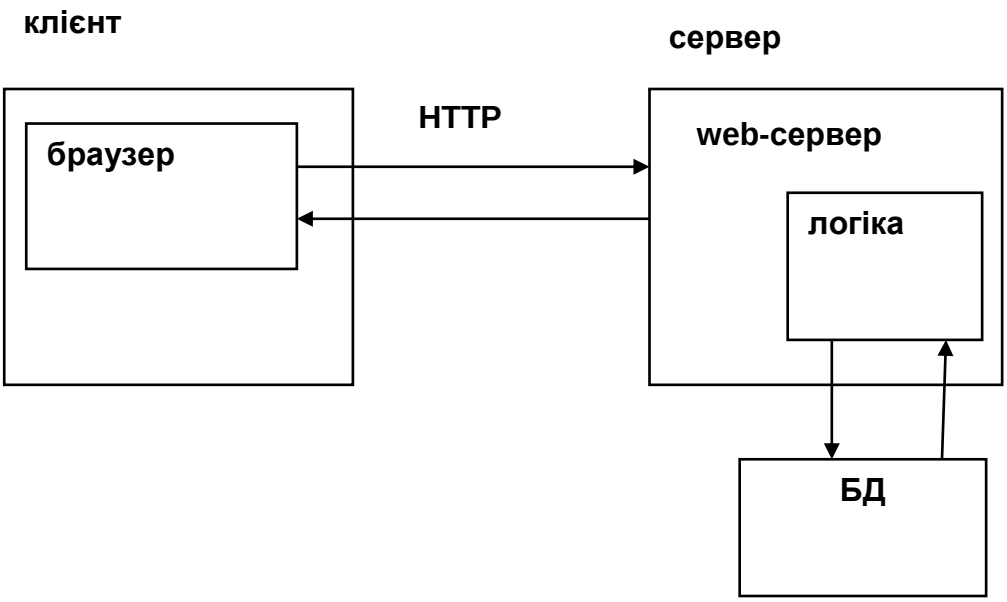

Рис. 2. **Архітектура системи з тонким клієнтом**

Така архітектура висуває мінімальні вимоги до клієнта, спрощує внесення змін, але в загальному випадку забезпечує обмежений інтерфейс, синхронний обмін даними і підвищене навантаження на сервер.

Реалізація таких систем може бути забезпечена широко поширеними технологіями активних сторінок ASP.NET, JSP, PHP та ін., а також баз даних MS SQL, MYSQL та ін. Для обміну даними використовується протокол HTML. Хоча необхідно підкреслити, сама архітектура не прив’язана до конкретної технології і може бути реалізована, наприклад, шляхом розробки web-додатка у будь-якому середовищі програмування (наприклад, MS Visual Studio або Delphi).

Другий варіант (архітектура з "товстим клієнтом") характеризується тим, що частина прикладної логіки виконується на клієнтові. Це може бути окремий додаток або розширення функцій браузера (рис. 3).

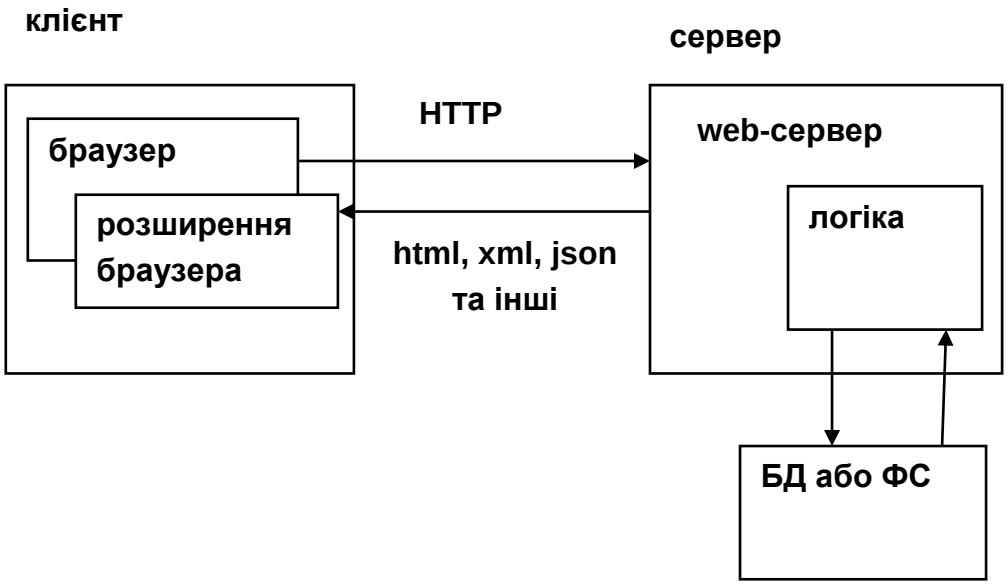

Рис. 3. **Архітектура системи з товстим клієнтом**

Завдяки такому рішенню розвантажується сервер, забезпечується більш розвинений і багатий інтерфейс, може бути реалізована асинхронна взаємодія з сервером, але ускладнюється розробка і внесення змін, падає швидкість обробки на клієнтові.

Розширення можливостей браузера досягається за рахунок модулів, що підключаються, включення скриптів, аплетів, а також нових можливостей HTML 5 (таких, як локальне сховище даних, геолокація та ін.). На додаток до технологій активних сторінок і баз даних можуть використовуватися Flash, Silverlight і тому подібне. Обмін даними між клієнтом і сервером можливий не лише HTML-сторінками, але і XML-документами, а також даними в інших форматах (наприклад, JSON).

Успіхи в створенні розподілених програмних систем, що взаємодіють з використанням сервісів (служб), дозволяють говорити ще про одну архітектуру – сервіс-орієнтовану архітектуру (англ. SOA, service-oriented architecture) [9].

Web служба або web-сервіс (Web service) – програмна система, для якої на спеціальній мові (підмножина XML) описані інтерфейси доступу. Ці описи можуть бути знайдені іншими програмними системами, і використані для взаємодії. Пошук описів і сама взаємодія здійснюється на основі протоколів мережі Інтернет. Таким чином, для сервісно-орієнтованої архітектури web-служба є одиницею модульності додатка.

Типова схема взаємодії компонент для такої архітектури наведена на рис. 4.

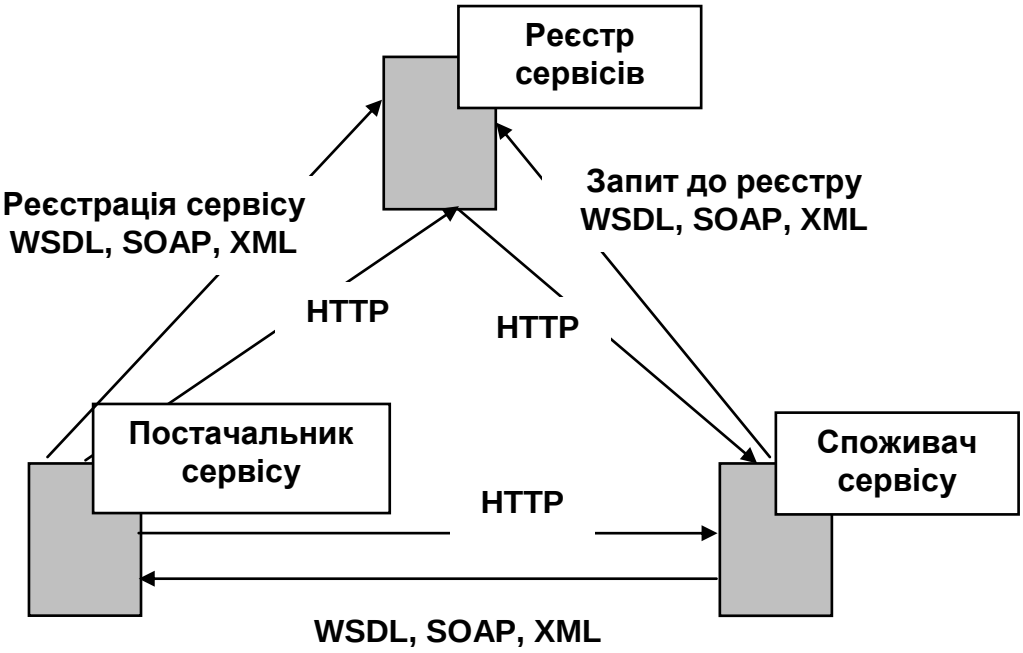

Рис. 4. **Сервіс-орієнтована архітектура**

Для реалізації сервіс-орієнтованої архітектури розробляються компоненти, мови, протоколи, стандарти. Зокрема розроблені і використовуються:

XML (розширювана мова розмітки, призначена для зберігання і передачі структурованих даних);

SOAP (протокол обміну повідомленнями на базі XML);

WSDL (мова опису зовнішніх інтерфейсів web-служби на базі XML);

UDDI (універсальний інтерфейс розпізнавання, опису та інтеграції).

Такі системи незалежні від платформи, сервіси можуть розроблятися і змінюватися незалежно від додатків, що використовують їх. Переваги такої архітектури:

забезпечується взаємодія програмних систем незалежно від платформи;

простота розробки і відлагодження (завдяки використанню XML);

використання стандартного інтернет-протоколу http.

До недоліків відноситься менша продуктивність і більший розмір мережного трафіка порівняно з іншими технологіями.

Розгляд цих прикладів дозволяє зробити декілька висновків.

У створенні інтерфейсної частини СОД існують дві тенденції: використання браузера з інтерфейсом у вигляді web-сторінки і створення власного інтерфейсу з використанням однієї з технологій (від системи програмування до Silverlight і тому подібне).

Як сховища даних найчастіше використовуються розподілені БД, рідше – набір файлів.

Для обміну даними між компонентами розподіленої системи використовуються мережа Інтернет або надбудови над нею.

Формат даних залежить від предметної області, однак розробники прагнуть використовувати деякий стандарт.

Таким чином, типовими компонентами СОД можна вважати сервери і клієнтські програми мережі Інтернет, СУБД і сервери БД, різні інтернет і web-додатки. І, звичайно ж, не можна забувати про середовище, а точніше ОС, в середовищі якої створюється СОД.

1.1.3. Характеристика компонент СОД

Сервери мережі Інтернет. Найчастіше до складу СОД включаються сервери WWW. Їх достатньо багато (табл. 1).

Web-сервер – це сервер, який обслуговує запити до одного або декількох сайтів мережі Інтернет.

Клієнт, у ролі якого зазвичай виступає браузер, передає web-серверу запити на здобуття ресурсів, що ідентифікуються URL-адресами. Ресурси – це HTML-сторінки, активні сторінки, зображення, файли, медіапотоки або інші дані, які необхідні клієнтові. У відповідь web-сервер передає клієнтові запитані дані. Цей обмін відбувається по протоколу HTTP.

 Web-сервером називають як програмне забезпечення, що виконує ці функції, так і безпосередньо комп'ютер, на якому це програмне забезпечення працює.

Web-сервери можуть також виконувати різні додаткові функції, наприклад, ведення журналу звернень користувачів до ресурсів, аутентифікація і авторизація користувачів, підтримка сторінок, що динамічно генеруються, підтримка HTTPS для захищених з'єднань з клієнтами і тому подібне.

У табл. 1 наведені дані про найбільш використовувані сервери в порядку убування частоти використання [15].

Таблиця 1

Характеристика web-серверів

Назва	Автор і рік створення	Розповсюдження	Open Source	Ліцензія	Особливості
Apache HTTP Server	Apache Software Foundation, 1995	безкоштовно	Так	Apache License	Акцент на надійність і гнучкість
Internet Information Services	Microsoft, 1995	вкл. у Win NT	Немає	Microsoft EULA	Є частиною пакета IIS. Єдиний сервер з підтримкою .NET
lighttpd	Jan Kneschke, лютий 2003	безкоштовно	Так	Варіант BSD	Використання на дуже навантажених серверах, забезпечуючи швидкість і захищеність
nginx	Ігор Сисоєв для Рамблера, 2002	безкоштовно	Так	Варіант BSD	Розроблявся для серверів, що випробовують велике навантаження
Google Web Server (GWS)	компанія Google	немає даних	немає даних	немає даних	немає даних

Більшість з них має версії для різних операційних систем (табл. 2).

Таблиця 2

Середовища функціонування web-серверів

Назва	Windows	Mac OS	BSD	Solaris	VMS	GNU/Linux
Apache HTTP Server	Так	Так	Так	Так	Так	Так
Internet Information Services	Так	Немає	Немає	Немає	Немає	
lighttpd	Так	Так	Так	Так		Так
nginx	Так	Так	Так	Так	Немає	Так
Google Web Server(GWS)						Так

Більш детальніше про два найбільш популярні сервери.

Apache функціонує в середовищах операційних систем GNU/Linux, BSD, Mac OS, Microsoft Windows, Novell NetWare, BEOS. За основні переваги Apache вважаються надійність і гнучкість конфігурації. Він дозволяє підключати зовнішні модулі для надання даних, використовувати СУБД для аутентифікації користувачів, модифікувати повідомлення про помилки і так далі.

Як недолік найчастіше називається відсутність зручного стандартного інтерфейсу для адміністрування. Система конфігурації Apache заснована на текстових конфігураційних файлах. Для конфігурації є власна мова, заснована на блоках директив.

Web-сервер Apache розробляється і підтримується відкритим співтовариством розробників під егідою Apache Software Foundation і включений в багато програмних продуктів, серед яких СУБД Oracle і IBM WebSphere. З 1996 року і до теперішнього часу є найпопулярнішим HTTP-сервером в Інтернеті. За статистикою в 2007 році його частка складала 51 % всіх web-серверів, в 2009 році – 46 %, 2011 – 59 % (основне зростання за рахунок США і Германії).

Apache має вбудований механізм віртуальних хостів. Він дозволяє повноцінно обслуговувати на одній IP-адресі безліч сайтів (доменних імен), відображаючи для кожного з них власний вміст. Для кожного віртуального хосту можна вказати власні налаштування ядра і модулів, обмежити доступ до всього сайта або окремих файлів.

Існує безліч модулів, що додають до Apache підтримку різних мов програмування і систем розробки: PHP (mod_php), Python (mod_python), Ruby (apache-ruby), Perl (mod_perl), ASP (apache-asp).

Apache підтримує механізми CGI і FAST CGI, що дозволяє виконувати програми на практично всіх мовах програмування, зокрема C, C++ , sh , Perl і Java.

IIS (Internet Information Services) – пропріетарний набір серверів для декількох служб Інтернету від компанії Microsoft. IIS функціонує тільки під управлінням ОС сімейства Windows NT.

IIS підтримує протоколи HTTP, HTTPS, FTP, POP3, SMTP, NNTP. У 2008 році, більше 62,5 мільйонів сайтів обслуговувалися web-сервером IIS, що складає 34 % від загального числа web-сайтів, в зоні ru – 8,6%.

Один сервер IIS може обслуговувати декілька сайтів. Усі запити до статичного контенту, що не вимагають виконання скриптів, виконуються в ядрі, це забезпечує високу швидкість. Запити до динамічного контенту

виконуються робочим процесом і завантаженими в його адресний простір модулями.

Служба WWW у складі IIS відрізняється від інших web-серверів тим, що функції забезпечення безпеки в ній тісно інтегровані з системою Windows NT, на основі якої вона працює. Зокрема, щоб дістати доступ до захищеного ресурсу, відвідувач повинен ввести ім'я і пароль користувача, що існує в системі Windows, на якій встановлений IIS (або в домені, якщо сервер належить до домена). Після цього користувач працює з сайтом так само, як якби він виконав інтерактивний вхід у систему на сервері. До нього застосовуються встановлені файловою системою NTFS дозволи на доступ до файлів і каталогів. Ця особливість IIS зручна для внутрішніх сайтів підприємств, однак практично непридатна для відкритих сайтів Інтернету, де неможливо створювати користувача Windows для кожного зареєстрованого відвідувача сайта. Тому в останньому випадку розробникам сайтів зазвичай доводиться використовувати власні механізми обмеження доступу.

Web-сервер IIS підтримує декілька різних технологій.

ASP.NET – розроблена Microsoft технологія. Для IIS це – основний на сьогоднішній день засіб створення web-додатків і web-сервісів. IIS поставляється разом з операційними системами, в які також спочатку входить .NET Framework, так що підтримка ASP .NET вже вбудована в IIS, починаючі з версії 6.0. Для раніших версій необхідно окремо завантажити і встановити .NET Framework.

ASP – попередня за ASP.NET технологія створення динамічних web-сторінок на основі сценаріїв.

CGI – стандартній протокол взаємодії сервера та додатка, використовується звичай для розширення можливостей сервера.

FAST CGI – протокол взаємодії web-сервера і додатка на зміну CGI.

ISAPI – розширення інтерфейсу сервера та низькорівнева технологія, що надає повний доступ до всіх можливостей IIS, можливість перевизначити частини функцій IIS і добавляти до нього функції, як пов'язані з генерацією контента, так і не пов'язаних з цим. Підсистема виконання скриптів ASP і підсистема ASP .NET виконані як модулі ISAPI.

Сам сервер підтримує лише CGI, FAST CGI, ISAPI. Решта всі технології є надбудовами, що працюють через CGI , FAST CGI або ISAPI.

За великим рахунком, ці два продукти з різними модифікаціями і конкурують на ринку продаж програмного забезпечення.

Google Web Server (GWS) – web-сервер, використовуваний Google для організації своєї інфраструктури. За станом на 2009 рік він знаходиться на третьому місці, обслуговуючи 3,78 % доменів і 8,39 % серед всіх активних сайтів, 2011 – 4,5%.

nginx – web-сервер, що працює на платформах Unix-подібних операційних систем (FREEBSD, OPENBSD, Linux, Solaris, Mac OS). Починаючи з версії 0.7.52 з'явилася збірка під Microsoft Windows.

Розроблявся з 2000-го року для компанії Rambler і постійно модернізується. У 2004 року вийшов перший публічно доступний реліз.

Конфігурація сервера nginx розділяється на віртуальні сервери. Для віртуального сервера можна задати адреси і порти, на яких прийматимуться з'єднання.

Для ефективного управління пам'яттю nginx використовує пули (послідовність заздалегідь виділених блоків динамічної пам'яті). Довжина блоку – від 1 до 16 кілобайт.

Містить модуль географічної класифікації клієнтів по IP-дресі. У його основу входить база даних відповідності IP-дрес географічному регіону.

У 2009 роки, число сайтів, обслуговуваних nginx , перевищило 16,2 мільйона, що робить його четвертим по популярності web-сервером в світі, в 2011 – 7,5 % (у зоні ru – 20 %).

lighttpd – web-сервер, що розробляється з розрахунком на швидкість і захищеність, а також відповідність стандартам. Це вільне програмне забезпечення, поширюване за ліцензією BSD. lighttpd працює в GNU/Linux та інших Unix-подібних операційних системах, а також в Microsoft Windows. Його частка серед серверів мережі Інтернет – 1 %.

У lighttpd є підтримка стискування вмісту, що віддається, і автоматичного балансування навантаження (навантаження може автоматично розподілятися по декількох запущених серверах lighttpd). Сервер також підтримує інтерфейси CGI, FAST CGI, дозволяє використовувати додатки, написані будь-якими мовами програмування.

Проект lighttpd почався з прагнення автора реалізувати web-сервер , який міг би витримати одночасно 10 тисяч з'єднань. Для цього використовується так звана асинхронна обробка мережевих з'єднань. Тому він пропонується для вузлів зі значною нагрузкою. На практиці часто lighttpd використовується для віддачі статичного вмісту, тоді як динамічними сторінками займається інший web-сервер. Така комбінація дозволяє істотно підвищити продуктивність вузла.

СУБД

Система управління базами даних (СУБД) – спеціалізована програма (частіше комплекс програм), призначена для організації і ведення бази даних. Основні функції СУБД:

управління даними в зовнішній пам'яті (на дисках);

управління даними в оперативній пам'яті з використанням дискового кеша;

журналізація змін, резервне копіювання і відновлення бази даних після збоїв;

підтримка мов БД (мова визначення даних, мова маніпулювання даними).

СУБД класифікують: за ступенем розподіленості (локальні і розподілені), за способом доступу до БД (файл-серверні, клієнт-серверні, вбудовуванні).

У файл-серверних СУБД файли даних розташовуються централізований на файл-сервері. СУБД розташовується на кожному клієнтському комп'ютері (робочій станції). Доступ СУБД до даних здійснюється через локальну мережу. На даний момент файл-серверні СУБД вважаються застарілими (Microsoft Access, Paradox, dBase, FoxPro, Visual FoxPro).

Клієнт-серверні розташовуються на сервері разом з БД і здійснюють доступ до БД безпосередньо, в монопольному режимі. Усі клієнтські запити на обробку даних обробляються клієнт-серверною СУБД централізовано (Oracle, Firebird, Interbase, IBM DB2, MS SQL Server, MYSQL та багато інших).

Вбудовуванні СУБД – це бібліотека, яка дозволяє уніфікованим чином зберігати великі обсяги даних на локальній машині. Доступ до даних може відбуватися через SQL або через особливі функції СУБД. Вбудовувані СУБД швидше звичайних клієнт-серверних і не вимагають установки сервера, тому затребувані в локальному програмному забезпеченні, яке має справу з великими обсягами даних (OpenEdge, SQLite, BERKELEYDB, MYSQL, Sav Zigzag, Compact , ЛІНТЕР).

На ринку баз даних лідерами є Oracle, IBM DB2, MS SQL Server. Серед вільно поширюваних – MYSQL.

ОС і браузери

За підсумками досліджень лідером серед операційних систем, використовуваних на серверах, є Linux [14]. Хоча статистика дуже відрізняється для різних зон (рис. 5).

Для зони Росії характерне використання різних клонів Unix (Free BSD і Linux). На зарубіжних серверах суттєву частку (більше одної третьої) займає Windows. Проте ситуація змінюється, за даними про продажі серверних ОС останніми роками на Windows припадає частка 70 %, а Linux – 20 %.

Ситуація з браузерами (на відміну від серверів і серверних ОС) змінюється дуже динамічно. Статистика по використанню браузерів у 2006 році: Internet Explorer – 54 %, Mozilla – 22 %, інші – 24 %. Статистика за 2010 рік: Internet Explorer – 51 %, Mozilla – 16 %, Chrome – 17 %, Safari – 15 %.

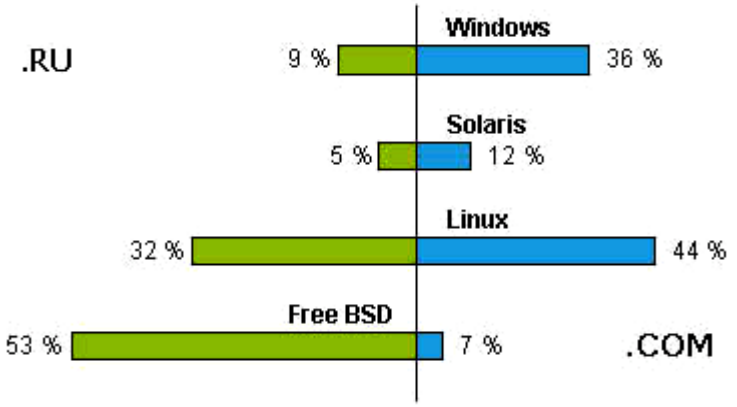

Рис. 5. **Серверні ОС зонах .ru і .com**

1.1.4. Формати документів СОД

Програма, яка обробляє зовнішні відносно до неї дані, повинна враховувати структуру цих даних у своїй логіці (алгоритмі). Це призводить до того, що будь-яка зміна структури даних (наприклад, з метою розвитку) викликає необхідність внесення змін у програму. А якщо ці дані (документи) циркулюють в системі? Зміни торкнулися всієї системи.

Проблема ця не нова, саме вона у свій час викликала появу і розвиток технологій баз даних як прагнення зробити незалежними функції зберігання і обробки даних. Однак використання баз даних не вирішило цю проблему до кінця, як і раніше, різні системи використовують документи різних форматів. І при створенні СОД на основі компонент, що існують, розробник стикається з необхідністю вибору або урахування різних форматів. Наприклад, при створенні СОД виникає питання, у якому вигляді дані з БД поступлять у клієнтську програму або браузер.

Таким чином, у завданнях побудови систем обробки даних однією з проблем є обмін даними між різними компонентами (як правило, програмними, часто на різних платформах). Нерідко найпростіше завдання імпорту/експорту даних з однієї програми в іншу призводить до необхідності розробок спеціальних модулів для забезпечення сумісності. Хоча сучасні технології розробки включають для цього достатньо зручні стандартні механізми (у вигляді методів спеціальних об’єктів). Завдання полегшується, якщо дані певного класу переміщатимуться між компонентами в деякому стандартному форматі.

Самі дані, циркулюючі в СОД, в більшості випадків є структурованими (масиви, записи, об’єкти), їх перетворення для передачі від одній програмі до іншої є достатньо загальним завданням, що отримало назву серіалізації.

 Серіалізація (у програмуванні) – процес перекладу якої-небудь структури даних у послідовність бітів. Зворотною до операції серіалізації є операція десеріалізації – відновлення початкового стану структури даних з бітової послідовності.

Серіалізація використовується для передачі об’єктів по мережі і для збереження їх у файлах. Наприклад, потрібно створити розподілений додаток, різні частини якого повинні обмінюватися даними складної структури. У такому разі для типів даних, які передбачається передавати, пишеться код, який здійснює серіалізацію і десеріалізацію. Об’єкт заповнюється потрібними даними, потім викликається код серіалізації, у результаті виходить, наприклад, XML-документ. Результат серіалізації передається приймаючій стороні, наприклад, електронною поштою або HTTP. Додаток-одержувач створює об’єкт того ж типу і викликає код десеріалізації, у результаті отримає об’єкт з тими ж даними, що були в об’єкті додатка-відправника. За такою схемою працює, наприклад, серіалізація об’єктів через SOAP (Simple Object Access Protocol – простий протокол доступу до об’єктів) у Microsoft.NET.

Є декілька підходів до реалізації цього механізму:

збереження значень компонент (властивостей об’єкта) в послідовному (текстовому або бітовому) файлі;

копіювання ділянки пам’яті, що містить структуру (об’єкт);

збереження структур у вигляді документа, незалежного від платформи формату.

Кожен з підходів має переваги і недоліки.

Збереження даних у послідовному файлі жорстко прив’язує послідовність і довжину окремих значень до оброблювальної програми і архітектури апаратури, і дозволяє лише лінійне відновлення. Зате лінійність забезпечує спрощення інтерфейсу.

Копіювання ділянок пам’яті забезпечує максимальну швидкість, але прив’язано до архітектури (наприклад, адресація байтів і розрядність, різне представлення структур у різних системах програмування) і допускає лише лінійний запис і відновлення.

Збереження у незалежних форматах забезпечує нелінійний запис і читання, але знижує швидкість обробки. Ускладнюється інтерфейс, виникає проблема вибору ефективного формату і його стандартизації. Нині існує багато конкуруючих форматів найрізноманітнішого призначення.

CDF – загальний формат даних (Common Data Format), який базується на багатовимірних масивах. Масиви класифікують дані по різних змінних, відповідних одному із спостережуваних параметрів. Розроблений NASA як частина американської космічної програми, CDF широко використовується для обміну між науковими базами даних.

CERIF – загальний європейських формат для дослідницької інформації (Common European Research Information Format). Застосовується для обміну інформацією про наукові проекти. Визначає набір обов'язкових і додаткових полів, які повинні використовуватися для опису наукових проектів, що фінансуються ЄС, включаючи назву проекту, керівників проекту, контактну адресу, найменування фінансуючої організації і ключові слова. Додаткова інформація включає посилання на інші проекти і опубліковані результати.

CIF – формат файлів з кристалографічної інформації (Crystallographic Information File). Служить для передачі кристалографічних даних між лабораторіями, журналами і базами даних. Файл CIF складається з імені поля даних, власного значення і утиліти для значень, що повторюються. Імена полів даних ієрархічні та утворюють категорії даних. Відсортований список імен полів разом з їх точними визначеннями утворюють словник CIF. Базовий словник CIF покриває більшість основних і найбільш часто використовуваних типів даних, використовуваних у кристалографії.

GRIB (Grid in Binary) – служить для зберігання і обміну інформацією про погоду в бінарній формі. Кожен запис GRIB призначений для передачі або зберігання єдиного параметра разом із значеннями, які є масивом крапок або набором спектральних коефіцієнтів одного рівня, кодованих у вигляді безперервного потоку бітів.

RAW – (англ. raw – сирий) формат даних, що містить необроблені (або оброблені в мінімальному ступені) дані, що дозволяє уникнути втрат інформації. Не має чіткої специфікації. У файлах міститься повна інформація про сигнал, що зберігається, і вона може бути нестислою, стислою без втрат, або стислою з втратами. Основні застосування файлів такого типу: цифрова фотографія (дані безпосередньо з матриці без обробки); обробка звуку (дані ІКМ) та багато інших.

Є ще безліч інших форматів!

Велика кількість форматів, відсутність єдиного підходу до їх використання і програмного забезпечення для обробки призвела до спроб розробки універсального формату. Найбільш вдалими з цих розробок можна вважати дві: JSON і XML.

JSON (JavaScript Object Notation) – формат обміну даними, для використання у програмах на мови JavaScript. Легко читається людиною. Не дивлячись на походження (JavaScript), може використовуватися практично з будь-якою мовою програмування. Для багатьох систем програмування існує готовий код для створення і обробки даних у форматі JSON.

Найширше застосовується при використанні технології AJAX. Формат JSON допускається вставку в код функцій.

Синтаксис JSON будується на двох структурах:

набір пар ім’я/значення (у різних мовах це реалізовано як об’єкт, запис, структура, хеш-таблиця, список з ключем або асоціативний масив);

пронумерований набір значень (у багатьох мовах це реалізовано як масив, вектор, список або послідовність).

Останні версії браузерів мають вбудовану підтримку JSON і здатні його обробляти без виклику спеціальної функції.

Мова **XML** (eXtensible Markup Language) є мовою розмітки, імена тегів у якій вибираються розробником з урахуванням конкретної наочної області. Завдяки цьому можна використовувати свою розмітку для кожного застосування. На основі цієї мови розроблена безліч пов’язаних технологій (XPath, XSL та ін.), які інтегровані в сучасні браузери. Мова розмітки XML і технології, які з нею пов’язані, є на сьогодні якнайповнішою і вдалішою спробою створення універсального засобу взаємодії компонент у СОД. При цьому сфера його застосування і популярність постійно зростають.

1.1.5. Поняття XML-документа

Призначення мов розмітки полягає в описі структурованих документів. Головним недоліком мов, що існували до появи XML, була їх негнучкість. Наприклад, при спробі використовувати HTML для опису довільних документів (яких-небудь банківських або бухгалтерських) існуючих тегів явно не вистачить.

Взагалі, кількість і різноманітність необхідних мов розмітки визначається кількістю методів обробки даних. Саме тому і була запропонована XML – мова, використовуючи яку можна створювати свою власну мову розмітки для конкретних застосувань.

1.1.5.1. Правила розмітки XML-документів

XML-документ формується за допомогою розмітки, що виділяє окремі елементи. Елемент XML складається з початкового і кінцевого тегів, між якими розміщується довільний текст. Причому, такі символи, як пропуск, повернення каретки і табуляція, сприймаються як пропуск зовні тегів, але в даних (усередині тегів) враховуються як спеціальні символи.

Поняття тега і загальні правила їх вживання дуже схожі на аналогічні в HTML. Основна відмінність у тому, що імена тегів не фіксовані, а вибираються розробником відповідно до предметної області.

Теги виділяються символами "<" і ">". Кінцеві теги починаються символами "</". Одиночні (непарні) теги містять так званий порожній елемент і закінчуються символами "/>". Наприклад, <Greeting ></Greeting> або <problem />.

Теги можуть вкладатися один в одного без "перетину" (якщо елемент починається усередині іншого елемента, він повинен і закінчуватися усередині цього елемента).

Імена (назви) тегів повинні починатися з букви, символу підкреслення або двокрапки і містити букви, цифри, символи підкреслення, дефіси, крапки точки і двокрапки (але не пропуски). Хоча в рекомендаціях XML1.0 явно це не указується, слід уникати використання двокрапки в іменах тегів, оскільки двокрапка використовується у XML при вказівці просторів імен. Імена задаються з урахуванням регістра. <DOCUMENT> і <document> це різні теги.

Слід особливо підкреслити, що не забороняється використання кирилиці та інших національних алфавітів.

Початкові і порожні теги можуть містити атрибути, що дозволяють визначити додаткові дані (ім’я_атр="значення_атр ").

Як початковий символ в імені атрибута може застосовуватися буква, знак підкреслення або двокрапка, далі можуть йти букви, цифри, символи підкреслення, дефіси, крапки і двокрапки (не допускається використання пропусків, оскільки з їх допомогою розділяються атрибути).

У XML не допускаються атрибути, яким не привласнені які-небудь значення.

Атрибути, як і інші елементи розмітки, представлені символами. Навіть якщо атрибута привласнюється числове значення, воно трактується у вигляді текстового рядка (поміщається в лапки):

<circle origin_x="10.0" origin_y " 20.0" radius="10.0" />

Лапки застосовуються подвійні або одинарні:

<quotation text=’He said . "Not that !"1 />

При необхідності задати текст зі складнішим поєднанням спеціальних символів можна використовувати підстановки (як і в HTML), тут вони називаються об’єктними посиланнями (' – символ **’**, " – символ **"**, & – символ **&**, < – символ **<**, > – символ **>**, $ &#x1a; і тому подібне).

Для виключення з процесу синтаксичного аналізу великих фрагментів коду використовуються спеціальні секції <![CDATA [....]]>

Усе, що знаходиться усередині секції, не аналізується. Наприклад, це може бути код скрипта (а у нього свої правила коректності) і тому подібне.

Оскільки користувач сам визначає імена тегів і зміст елементів, то одні і ті ж дані можна розмістити або в тегах, або в атрибутах. Твердо сталого правила немає, проте занадто велика кількість атрибутів призводить до того, що документ важко читати.

1.1.5.2. Структура XML-документа

Документ складається з трьох частин: прологу (може бути порожнім), кореневого елемента і необов'язкової загальної частини частки.

Пролог може включати XML-оголошення (наприклад <?xml version = "1.0"?>), оголошення типу документа або DTD (<!DOCTYPE .>), інструкції з обробки, коментарі і пропуски. Рекомендується включати в пролог, як мінімум. XML-оголошення, в якому указується вживана версія мови XML.

Кореневий елемент містить вкладені елементи. XML-документ повинен включати тільки один кореневий елемент.

Необов'язкова загальна частина може складатися з XML-коментарів, інструкцій з обробки і пропусків (пропуски, табуляції і т. п.).

Оголошення знаходиться у першому рядку в елементі <?xml...?>, наприклад <?xml version = "1.0" standalone="yes" encoding="UTF-8"?>

У XML-оголошенні допускається вказівка таких атрибутів:

version – версія XML-оголошення;

encoding – кодування символів документа (за замовчуванням передбачається UTF-8);

standalone – "yes ", якщо цей документ не посилається на зовнішні об'єкти,"no " – інакше, (за умовчанням – "yes ").

Коментарі схожі на HTML-коментарі. Їх можна використовувати для включення у документ пояснень, які ігноруються при синтаксичному розборі. Коментар поміщається між символами <!-- ... -->.

Коментарі не повинні поміщатися перед XML-оголошенням. Не можна поміщати коментар усередині символів розмітки (тегів). Неприпустимо використовувати символ "--" усередині коментаря.

Інструкції з обробки управляють функціонуванням XML-процесора. Вони починаються послідовністю символів "<?", і завершуються символами "?>". Не можна використовувати тег <?xml … ?> (це оголошення). Інструкції з обробки залежать від використовуваного

процесора, а не є специфікаціями мови XML. У загальному випадку, інструкції можуть бути поміщені в будь-яке місце документів поза елементами.

Наприклад, включення таблиць стилів:

<?xml-stylesheet type="text/css" href="my .css" ?>

Ця інструкція буде правильно виконана всіма браузерами, які підтримують XML.

Приклад 1. Створено XML-документ з прологом і кореневим елементом:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/css" href="my .css" ?>
<!DOCTYPE DOCUMENT [
<!ELEMENT DOCUMENT (ZAG , STUDENT* )>
<!ELEMENT ZAG (#PCDATA )>
<!ELEMENT STUDENT (NAME , photo,BOOK* )>
<!ELEMENT NAME (#PCDATA )>
<!ELEMENT photo (#PCDATA )>
<!ELEMENT BOOK (AUTOR,TITLE )>
<!ELEMENT AUTOR (#PCDATA )>
<!ELEMENT TITLE (#PCDATA )>
]>
<DOCUMENT >
<ZAG>Список студентів, що не здали книги</ZAG>
<STUDENT >
  <NAME>Иванов</NAME>
  <photo>im1.gif</photo>
  <BOOK >
   <AUTOR>Пушкар </AUTOR>
   <TITLE > Основи наукових досліджень</TITLE>
  </BOOK >
  <BOOK >
    <AUTOR>Молчанов </AUTOR>
    <TITLE > Технології Web-дизайну</TITLE>
   </BOOK >
 </STUDENT >
 </DOCUMENT >
```

Пролог цього документа містить оголошення (найперший рядок), інструкцію з обробки (другий рядок) і DTD (опис типу документа, починається з <!DOCTYPE, та описує допустиму вкладеність тегів і їх вміст).

Інструкція забезпечує підключення таблиці стилів з файла my.css.

Кореневий елемент (<DOCUMENT >...</DOCUMENT>) містить теги з даними про студентів, що не здали книги (для стислості поміщений тільки один тег <STUDENT >).

1.1.6. Створення XML-документів

Власні формати даних у сучасних додатках стали настільки складними, що часто пізніша версія додатка не може прочитувати дані, підготовлені в ранішій версії цього ж додатка. Необхідність передачі даних вимагає застосування спеціальних програм або модулів перетворення. У зв'язку з цим XML потрібний розробникам додатків. Він дозволяє уніфікувати ці формати. А закладені в ньому можливості щодо перевірки документів дозволяють контролювати дані на вході, підвищуючи надійність. Наприклад, у HTML такого контролю немає.

Знання цієї мови користувачами не тільки полегшує розуміння і використання сучасних застосувань, підтримуючих XML, але і потрібний безпосередньо в ході розробок.

 Пригадаємо [7] фрагмент з CGI-програми на C# для реєстрації користувача (введення логіна і пароля):

```
public static DataSet DS ;
  public static void LoadDs ()
  {  DS = new DataSet ();
    DS .ReadXml(@".\mv\users.xml"); }
.....
  public static bool NewUser (string login , string pass )
  { LoadDs ();
    foreach (DataRow dr in DS .Tables[0].Rows)
    {  if (login .Equals(dr ["Login "]))
       { return false ; };
    }
    DS .Tables[0].Rows.Add(new object[ ] { login , pass });
    SaveDs ();
    return true ; //
  }
```

Достатньо однієї строки (вона підкреслена) для привласнення даних з тегів властивостям об'єкта.

XML – це метамова. За допомогою XML створюються мови розмітки, що налаштовуються, тобто мови, що забезпечують відображення специфічних документів універсальними браузерами. Створені сотні таких мов для різних сфер (банківська справа, телекомунікація, школи, бібліотеки, хімія, математика, Інтернет і так далі).

 **Telecommunications Interchange Markup** (Мова розмітки обміну даними в області телекомунікацій, TIM).

Schools Interoperability Framework (Схема взаємодії в умовах школи, SIF).

Common Business Library (Загальна ділова бібліотека, xCBL).

Electronic Business XML Initiative (Електронна ділова XML-ініціатива, ebXML).

Product Data Markup Language (Мова розмітки виробничих даних, PDML).

Chemical Markup Language (CML), дозволяють реалізувати графічне представлення молекул.

Extensible Hypertext Markup Language (XHTML), мова розмітки гіпертексту, за допомогою якого можна додавати власні елементи (розвиток HTML).

Вивчення і використання цих засобів істотно спрощується (полегшується) при знанні основ XML.

XML – це засіб опису структурованих даних. XML дозволяє визначати структуру і способи інтеграції різних елементів. Це може бути використано для зберігання і перетворення даних. Створюється документ, відповідний конкретній наочній області у форматі XML, який потім стандартно перетвориться у необхідний формат (наприклад, PDF). При цьому забезпечується перевірка правильності документа.

Цей аспект застосування можна порівнювати з базами даних. Причому можна вести мову про ефективність, ступінь відповідності моделі даних конкретного додатка моделі зберігання (переважна більшість баз є реляційного типу, структуризація у XML – деревовидна).

Застосування XML для створення web-сторінок. Перехід на розмітку web-сторінок за допомогою мови XML. Створення мови обґрунтовувалося саме потребами Інтернет, точніше недоліками HTML:

обмеженість набору тегів (близько 100) і складність введення нових;

певна непослідовність у рішеннях (викликана розширеннями можливостей при незмінній концепції), таких як змішування розмітки і форматування, робота з таблицями стилів, підтримка об’єктів і ін.;

відсутність контролю правильності даних, що поступили;

відсутність зв’язку тегів із змістом даних (теги достатньо загальні, а добре б мати для кожного типу "свій" тег).

Для подолання цих недоліків нові мови розмітки (XHTML, HTML5) створюються як підмножини XML. XML також використовують у нових рішеннях (AJAX та ін.).

У майбутньому обіцяють на його основі нові можливості:

новий тип опису і пошуку ресурсів (RDF);

новий тип гіпертекстових зв’язків (XLink і Xpointer).

У цілому область використання XML визначається так: розробка складних інформаційних систем, з великою кількістю додатків, зв’язаних потоками інформації найрізноманітнішої структури. XML-документи виконують роль універсального формату для обміну інформацією між окремими компонентами великої програми. XML може використовуватися в звичайних застосуваннях для зберігання і обробки структурованих даних в єдиному форматі. Він дозволяє описувати дані довільного типу і використовується для представлення спеціалізованої інформації.

Крім того, XML є основою для цілої низки нових технологій і застосувань, таких як DTD (мова оголошення типу документа), XSL (розширювана мова таблиць стилів), вони і стануть предметом подальшого розгляду.

Висновки та узагальнення

1. Розробка СОД є завданням, яке досить широко розповсюджене в індустрії обробки даних. До її вирішення існує кілька підходів з використанням стандартизованих компонент, які добре зарекомендували себе.

2. Найбільш поширеною архітектурою такої системи є архітектура "клієнт-сервер".

3. Як компоненти системи найчастіше використовуються браузери, СУБД, сервери і протоколи WEB.

4. Найбільш поширеним і перспективним форматом документів для зберігання та обміну даними є XML.

Теоретичні запитання

1. Дайте визначення основних понять, пов’язаних з СОД.

2. Опишіть найпоширеніші архітектурні рішення для СОД.

3. Дайте характеристику основних компонент СОД.

4. Назвіть формати даних, що використовуються у СОД. Дайте їх характеристику.

5. У чому полягає завдання серіалізації? Основні шляхи її вирішення?

6. Дайте порівняльну характеристику форматів JSON і XML.

7. Сформулюйте відмінності в правилах синтаксису XML і HTML.

Комплекс задач і завдань

1. Розробіть XML-документ для відображення даних з табл. 3.

Таблиця 3

Показники зростання

Рік	Період	Показник 1	Показник 2	Показник 3
1957	1 кв	70 %	30 %	10 %
	2 кв	80 %	40 %	20 %
	3 кв	67 %	40 %	25 %
	4 кв	56 %	50 %	20 %
1997	1 кв	75 %	30 %	15 %
	2 кв	80 %	40 %	20 %
	3 кв	68 %	45 %	25 %
	4 кв	56 %	55 %	30 %

2. Розробіть XML-документ для відображення даних з табл. 4.

Таблиця 4

Показники зростання

Період	Показник 1	Динаміка	Показник 2
1957 – 1997	30 %	pic1.gif	56
	45 %		67
1997 – 2007	20 %	pic2.gif	43
	11 %		98

2. Робота з XML-документами

Основна ідея розділу

Розділ присвячено питанням створення, перевірки та відображення XML-документів. Розглянуто способи відображення документів у браузерах, а також перевірку коректності і валідності на основі DTD.

Цілі вивчення розділу

Метою розділу є розгляд питань відображення документів і перевірки їх відповідності загальним вимогам і синтаксису, який визначає розробник.

Матеріал, викладений у розділі, надає студенту можливість придбати такі ***компетентності***.

Знання.

Методи відображення XML-документів у сучасних браузерах.

Концепцію перевірки коректності та валідності XML-документів.

Правила створення і включення в документ DTD.

Уміння.

Відображати XML-документи найбільш підходящим методом.

Розробляти DTD для XML-документів.

Виконувати перевірку валідності і коректності за допомогою різних засобів.

Комунікації.

Аргументовано обґрунтовувати вибір методів і засобів у ході обговорення альтернативних пропозицій інших учасників розробки.

Донесення власних висновків та знань стосовно XML- документів як до фахівців, так і осіб, що навчаються.

Автономність і відповідальність.

Забезпечення роботи з XML-документами на різних платформах.

2.1. Відображення XML-документів у браузері

Можна відкрити XML-документ безпосередньо в Internet Explorer або в іншому браузері. Останні версії цих програм підтримують XML і мають вбудовані засоби контролю правильності документа. Проте якщо XML-документ не містить зв'язку з таблицею стилів, то Internet Explorer лише позначає різні складові частини документа різним кольором, а також представляє кореневий елемент у вигляді ієрархічного дерева з можливістю згортання і розгортання структури і перегляду з меншою або більшою мірою деталізації (символи знаку мінус "-" або плюс "+" зліва від початкового тега).

Приклад 2. Створимо XML-документ зі списком літератури з тегами кирилицею, та розкриємо його у браузері.

Документ:

```
<?xml version="1.0" encoding="UTF-8"?>
<список> Список літератури з дисципліни "Засоби систем обробки даних"
 <джерело наявність="так">
  <автор>Холзнер С.</автор>
  <назва>Энциклопедия, 2-е изд.</назва>
  <видавництво>-СПб.:Питер</видавництво>
  <рік>2004</рік>
  <кіл_сторінок>1101</кіл_сторінок>
 </джерело>
 <джерело наявність="ні">
  <автор>Сакс Т., Мак-Клейн Г.</автор>
  <назва>Дизайн и архитектура современного вебсайта. Опыт
профессионалов.</назва>
  <видавництво>-М.: Издательский дом "Вильяме"</видавництво>
  <рік>2002</рік>
  <кіл_сторінок>304</кіл_сторінок>
```

```
  </джерело>
  <джерело наявність="так">
    <автор>Пушкарь А.И. та інш.</автор>
    <назва>Современные компьютерные технологии. Учебное пособие. /Под
ред. А.И. Пушкаря</назва>
    <видавництво>-ИД "ИНЖЕК"</видавництво>
    <рік>2004</рік>
    <кіл_сторінок>463</кіл_сторінок>
  </джерело>
 </список>
```

Відображення у браузері надано на рис. 6.

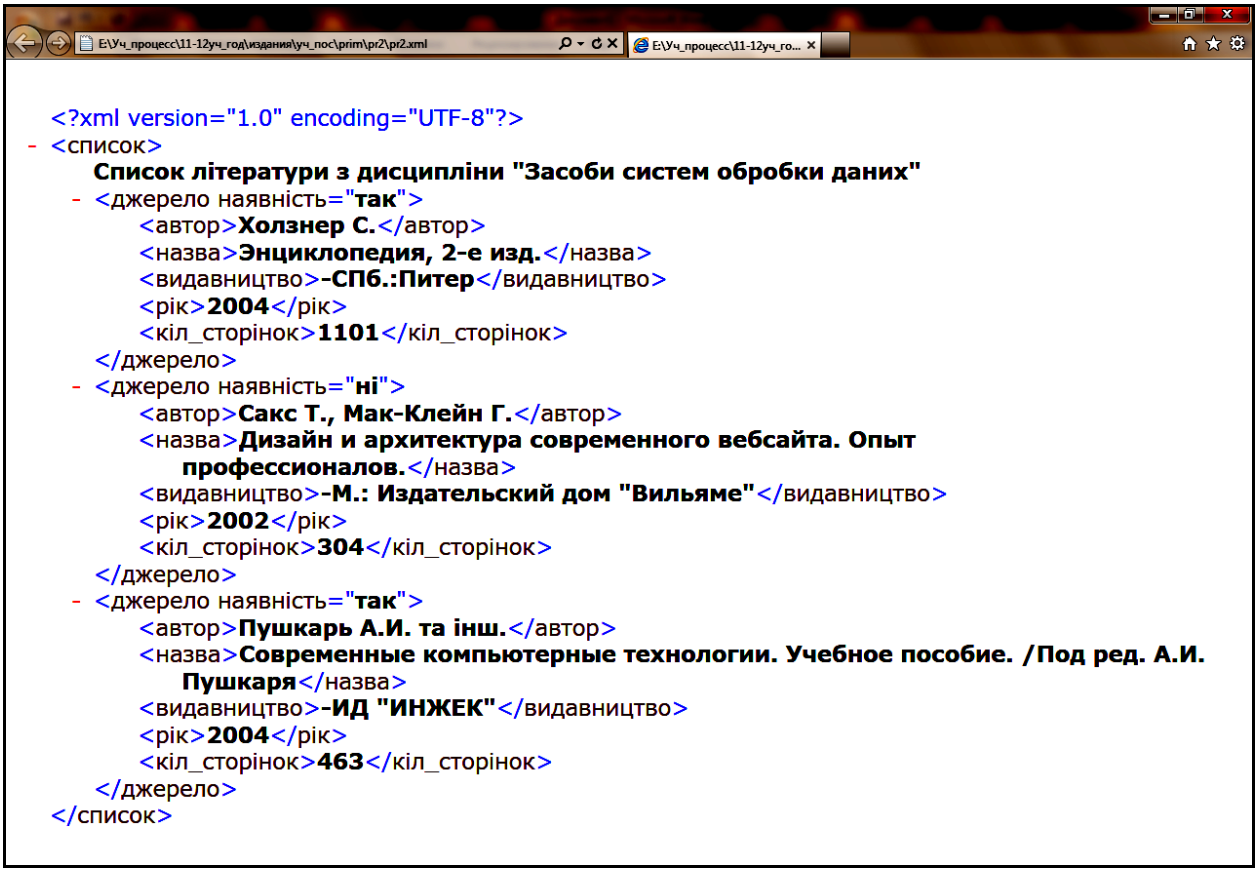

Рис. 6. **Відображення XML-документа у браузері**

Перш ніж браузер відображуватиме XML-документ, його вбудований синтаксичний XML-аналізатор (parser) прогляне вміст файла. При виявленні помилки буде відображена сторінка з відповідним повідомленням. Але за відсутності визначень DTD, перевіряється лише відповідність документа загальним формальним правилам побудови.

Отже, при відкритті цих документів у браузері, ми побачимо структуру розміченого тексту (текст і теги різними кольорами).

2.2. Використання таблиць стилів для відображення XML-документів

Оскільки в XML розробник створює свої власні елементи, браузер не має вбудованих засобів, що дозволяють правильно відобразити їх. Найбільш простим способом описати, як повинні відображатися елементи є створення таблиці стилів і скріплення її з XML-документом. Сучасні web-браузери забезпечують високий рівень підтримки каскадних таблиць стилів, тоді як інші методи відображення XML все ще знаходяться у стадії розвитку. Крім того, зберігання інструкцій по відображенню окремо від самого XML-документа підвищує гнучкість використання і полегшує роботу з ним.

Нагадаємо, що таблиця стилів [10] є текстовим файлом, зазвичай з розширенням .css, який містить набір правил, що складаються з селектора (у простому випадку – це ім'я тега) і декларацій (набір пар *ім'я_властивості:значення* у фігурних дужках), що повідомляють браузеру, яким чином форматувати і відображати елементи. Створюватися він може найпростішим текстовим редактором (наприклад, Блокнот). Більшість властивостей, які розглядалися при форматуванні HTML, можуть бути застосовні і до XML-документів.

Приклад 3. Створимо таблицю стилів для відображення даних документа з прикладу 2.

Для відображення можна використовувати такі правила:

список {display:block;font-size:36pt;text-align:center;margin-left:20px}

джерело {display:list-item; list-style-type:decimal;list-style-position:inside;font-size:20pt;text-indent:20px;text-align:left}

автор,назва,видавництво,рік,кіл_сторінок {display:inline;font-size:20pt; color:black}

Для встановлення зв'язку з таблицею стилів у XML-файл додано стрічку <?xml-stylesheet type="text/css" href="pr3.css" ?>:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="pr3.css" ?>
<список> Список литературы по дисциплине"Технологии Web-дизайна"
 <джерело>
   …
```

Такий файл буде відображено браузером, як зображено на рис. 7.

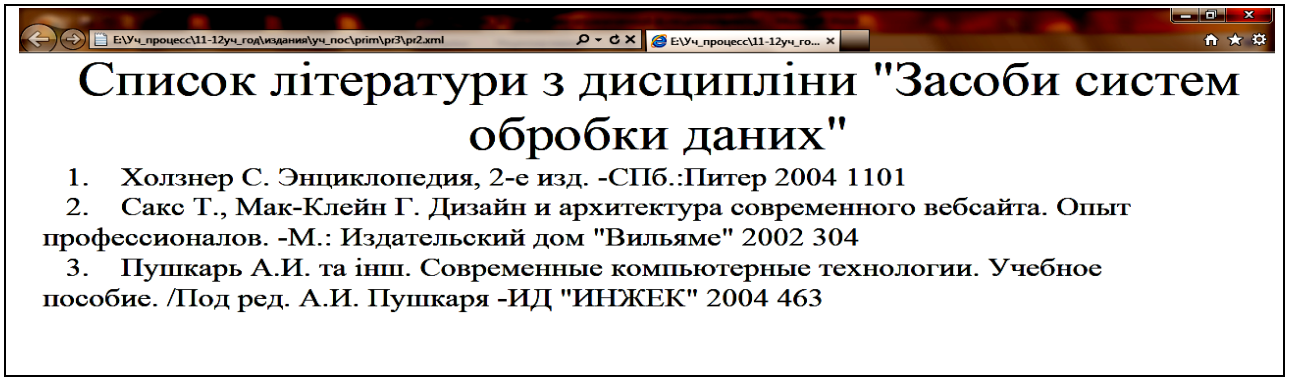

Рис. 7. **Відображення XML-файла з таблицею стилів у браузері**

Специфікація CSS містить велику кількість різних селекторів. Зокрема, імена тегів (p,h1), класові (.cl) і id-селектори (#id). Проте, при роботі з xml-документами класові та id-селектори у такому вигляді не працюють.

Цей факт пояснюється дуже легко. У HTML імена тегів і атрибутів фіксовані, тому атрибути class і id мають фіксований сенс, з ним і пов'язана форма селектора. У XML цього немає, class і id нічим не виділяються серед інших атрибутів, тому така форма не підтримується.

Для зв'язку правил з елементами більш за все підходить селектор атрибутів. Він конструюється з імені елемента й імені атрибута в квадратних дужках:

p[class] – усі теги **p**, що мають атрибут class ;

p[class="c1"] – усі теги **p**, що мають атрибут class із значенням c1;

*[id="id1"] – усі теги, що мають атрибут id із значенням id1.

Підкреслимо, що використовувати можна будь-які атрибути, а не лише class та id.

Відображаючи документ у різних браузерах, можна дійти висновку щодо неоднакової підтримки технології різними розробниками. Крім того, застосуванню таблиць стилів для обробки XML-документів властивий ряд обмежень.

Таблиці стилів не дають можливості модифікувати або реорганізовувати вміст документа.

Вони не дозволяють здійснювати доступ до атрибутів, інструкцій з обробки та іншим компонентам XML, а також не дають можливості обробляти інформацію, яку ці компоненти містять. Тому, наприклад, неможливо відобразити малюнок.

Частіше за все таблиці стилів використовуються для відображення текстового змісту документа.

2.3. Зв’язування XML-документа з HTML-сторінкою

Метод зв’язування даних запропонований Microsoft і підтримується тільки в її продуктах. Він зводиться до установки зв’язку XML-документа з HTML-сторінкою, а також зчепленню елементів HTML з XML-елементами [10; 11]. У результаті HTML-елементи автоматично відображають вміст XML-елементів, з якими вони зчеплені. У цьому випадку при відкритті HTML-сторінці вбудований в Internet Explorer XML-процесор синтаксично аналізує XML-документ. При цьому Internet Explorer створює програмний об’єкт, який має назву "Об’єкт джерела даних" (Data Source Object, DSO). Об’єкт DSO зберігає дані XML і забезпечує доступ до них. DSO дозволяє здійснювати доступ і маніпулювання XML-даними за допомогою ряду методів, властивостей і подій.

Зв’язування даних працює лише з XML-документом, який симетрично структурований, тобто елементи документа можуть бути інтерпретовані як набір записів і полів. У простому випадку такий документ складається з кореневого елемента, що містить кілька елементів однакового типу (записи), кожен з яких має однаковий набір дочірніх елементів, усі з яких містять символьні дані (поля). Якщо структура документа така, що не допускає зв’язування даних, для відображення можна використовувати сценарії.

Відображення шляхом зв’язування даних можна розділити на два етапи: установка зв’язку XML-документа з HTML-сторінкою і зчеплення HTML елементів з XML-елементами.

Базова технологія зв’язування даних надзвичайно проста, хоча має декілька варіацій і способів використання.

Установка зв’язку XML-документа з HTML сторінкою. Щоб відобразити XML-документ на HTML-сторінці, необхідно встановити його зв’язок із сторінкою. Найпростіший шлях для цього включити в сторінку HTML-елемент з ім’ям XML, так званий фрагмент даних.

Можна включити документ безпосередньо в текст сторінки. Весь текст XML-документа поміщається між початковим і кінцевим тегами XML:

```
<HTML>
<HEAD>
 <TITLE>..</TITLE>
</HEAD>
<BODY>
```

```
<XML ID="dsoBook ">
<!--XML-документ -->
  </XML>
 <!-- інші елементи HTML -->
</BODY>
</HTML>
```

Можна просто зіслатися на URL XML-документа:

```
<HTML>
<HEAD>
 <TITLE>Book Description</TITLE>
</HEAD>
<BODY>
 <XML ID="dsoBook " SRC="Book.xml"></XML>
 <!-- інші елементи HTML -->
</BODY>
</HTML>
```

Останній спосіб більш відповідає основам філософії XML, згідно з якою власне дані (XML-документ) зберігаються окремо від інформації по їх форматуванню і обробці (таблиці стилів або, як в даному випадку, HTML-сторінці). При цьому полегшується робота з XML-документом, особливо якщо один документ відображається на декількох різних сторінках.

Атрибута ID фрагмента даних привласнюється унікальний ідентифікатор, який використовуєте для доступу до XML-документа з HTML-сторінки. У прикладі значення ID – "dsoBook ".

Атрибута SRC привласнюється URL файлу, що містить дані XML. Можна використовувати абсолютний URL:

```
<XML ID="dsoBook " RC=http://www.my_domain.com/documents/Book.xml>
</XML>
```

Проте зручніше використовувати відносну адресу (щодо місцезнаходження HTML-сторінці, що містить фрагмент даних). Наприклад, атрибут SRC у наступному фрагменті вказує, що файл Book.xml знаходиться в тій же папці, що і сторінка:

```
<XML ID="dsoBook" SRC="Book.xml"></XML>
```

Якщо документ є коректно сформованим (при виявленні помилки відображення зупиняється без видачі повідомлення про помилку), то створюватиме об’єкт DSO. Об’єкт DSO зберігає дані XML і забезпечує доступ до них. Дані зберігаються як набір записів і їх полів.

Щоб побачити опис якої-небудь помилки, наявної в зв’язаному XML-документі, необхідно протестувати документ з використанням сценаріїв перевірки на коректність і валідність.

DSO дозволяє безпосередньо здійснювати доступ і маніпулювання наявним набором записів за допомогою ряду методів, властивостей і подій.

Методами є функції, які можна викликати для доступу або модифікації набору записів. Наприклад, для переміщення між записами.

Властивостями є встановлені на даний момент параметри, які можна прочитувати і у ряді випадків змінювати. Наприклад, рахувати властивість, яка повідомляє про досягнення останнього запису (recordset.EOF).

Подіями є певні зміни стану (наприклад, зміна значень запису), якими можна управляти за допомогою функції сценарію, створеного для сторінки.

Для відображення вмісту тегів XML на сторінці вони зчіплюються з тегами HTML (для передачі вмісту).

Зчеплення HTML-елементів з XML-елементами. Два основні способи зчеплення:

табличне зчеплення (зчеплення HTML-елемента TABLE з даними XML, так що в таблиці автоматично відображається весь набір записів, що належать XML-документу);

зчеплення по окремих записах (зчеплення нетабличних елементів HTML, наприклад, елементів SPAN, з XML-елементами для відображення лише одного запису).

Таблиця HTML використовується для відображення XML-документа, структурованого як набір записів, або використовуються вкладені HTML-таблиці для відображення XML-документа, що містить ієрархічний набір записів (складнішу структуру записів).

Для зчеплення в тегі TABLE використовується додатковій атрибут DATASRC="#ім’я_DSO", а в тегі елемента для відображення тексту (наприклад, SPAN) – DATAFLD="ім’я_поля". Суть у тому, що властивості innerText тега привласнюється текст з відповідного тега XML.

Приклад 4. Створимо сторінку для відображення в браузері Internet Explorer документа з прикладу 2. Буде використана одна HTML-таблиця для відображення всього набору записів:

```
<HTML>
<HEAD>
  <TITLE>Book Inventory</TITLE>
</HEAD>
<BODY>
<XML ID="dso1 " SRC="pr2.xml"></XML>
```

```
<H2>Список літератури</h2>
<TABLE DATASRC="#dso1 " BORDER="1 " CELLPADDING="5 ">
 <THEAD>
   <TH>Автор</TH>
   <TH>Назва</TH>
   <TH>Видавництво</TH>
   <TH>Рік видання </TH>
   <TH>Кіл. сторінок</TH>
 </THEAD>
 <TR ALIGN="left ">
   <TD><SPAN DATAFLD="автор" STYLE="font-style:italic"></SPAN></TD>
   <TD><SPAN DATAFLD="назва"></SPAN></TD>
   <TD><SPAN DATAFLD="видавництво"></SPAN></TD>
   <TD><SPAN DATAFLD="рік"></SPAN></TD>
   <TD><SPAN DATAFLD="кіл_сторінок"></SPAN></TD>
 </TR>
</TABLE>
</BODY>
```

Зображення вікна браузера надано на рис. 8.

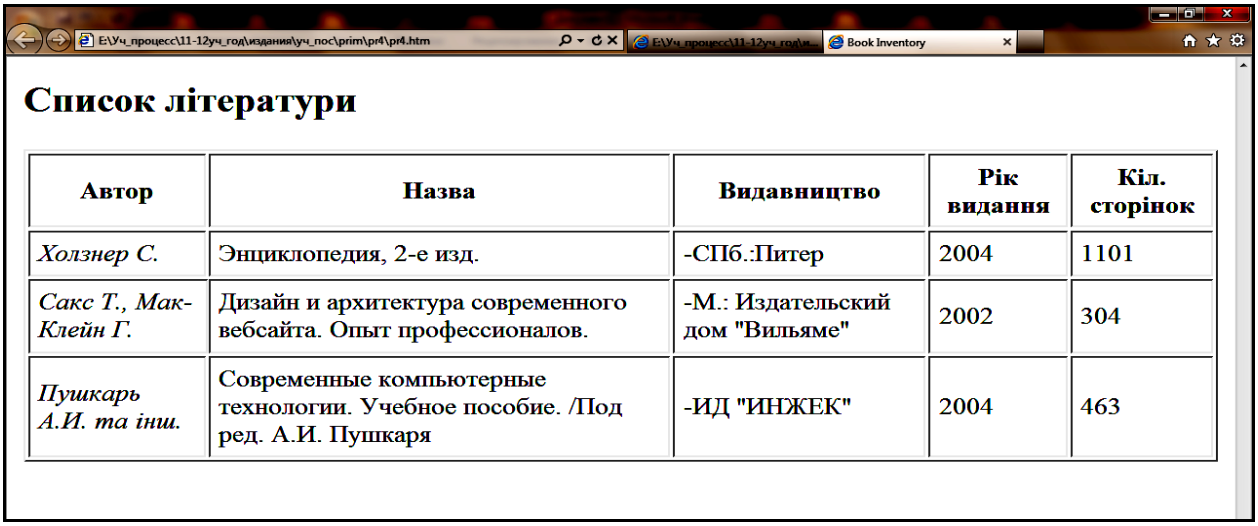

Рис. 8. **Відображення сторінки у браузері**

Якщо XML-документ містить багато записів, то можна використовувати посторінкове виведення групи записів за один раз замість відображення всіх записів одночасно. Для управління посторінковим відображенням необхідно в тег TABLE додати атрибут DATAPAGESIZE="кіл_зап". Переміщення між групами може бути забезпечено використанням методів об’єкта TABLE. Вони наведені у табл. 5.

Таблиця 5

Методи об’єкта TABLE

Метод елемента TABLE	Ефект	Приклад виклику
FirstPage	Відображає першу сторінку записів	Table1.firstPage()
previousPage	Відображає попередню сторінку записів	Table1.previousPage()
NextPage	Відображає наступну сторінку записів	Table1.nextPage()
LastPage	Відображає останню сторінку записів	Table1.lastPage()

Якщо у нинішній момент відображена перша сторінка, виклик методу previousPage ігнорується, а якщо відображена остання сторінка, то ігнорується виклик nextPage.

Найпростіше використовувати виклик методу в атрибуті ONCLICK HTML-елемента BUTTON:

```
<BUTTON ONCLICK="Table1.nextPage()">Наступна</BUTTON>
```

У ієрархічному наборі записів кожна запис може містити, на додаток до фіксованого набору полів, змінне число входжень (нуль або більш) вкладених записів (як у прикладі 1).

Приклад 5. Створимо HTML-сторінку для відображення документа зі змінним числом полів з прикладу 1. Перш за все, цей документ в існуючому вигляді не може бути відображений шляхом зв’язування, так як він не є набором однакових записів. Тому тег ZAG слід видалити.

Вкладену таблицю необхідно зчепити не лише з XML-документом (<TABLE DATASRC="#dso1" …>), але і з вкладеними записами (<TABLE DATASRC="#dso1 " DATAFLD="BOOK " …>), щоб в таблиці відображався вміст кожного запису BOOK, вкладеного в поточний запис STUDENT.

```
<HTML>
<HEAD>
  <TITLE>Боржники</TITLE>
</HEAD>
<BODY>
<XML ID="dso1" SRC="pr1.xml"></XML>
<H2>Список студентів, яки не здали книги до бібліотеки</H2>
<TABLE DATASRC="#dso1" BORDER="0" ALIGN=left>
 <TR> <TD><SPAN DATAFLD="NAME"></SPAN></TD> </TR>
 <TR> <TD><TABLE DATASRC="#dso1" DATAFLD="BOOK" BORDER="1"
          CELLSPACING="10">
         <THEAD ALIGN=left> <TH>Автор</TH> <TH>Назва</TH> </THEAD>
         <TR> <TD><SPAN DATAFLD="AUTOR"></SPAN></TD>
            <TD>          <SPAN DATAFLD="TITLE"></SPAN></TD>
```

```
</TR>
        </TABLE>
 </TR>
</TABLE>
</BODY>
```

У першому рядку відображається поле NAME, а другий рядок не відображає поле, а містить вкладену таблицю, яка відображає вміст вкладених записів усередині поточної категорії.

Зображення вікна браузера надано на рис. 9.

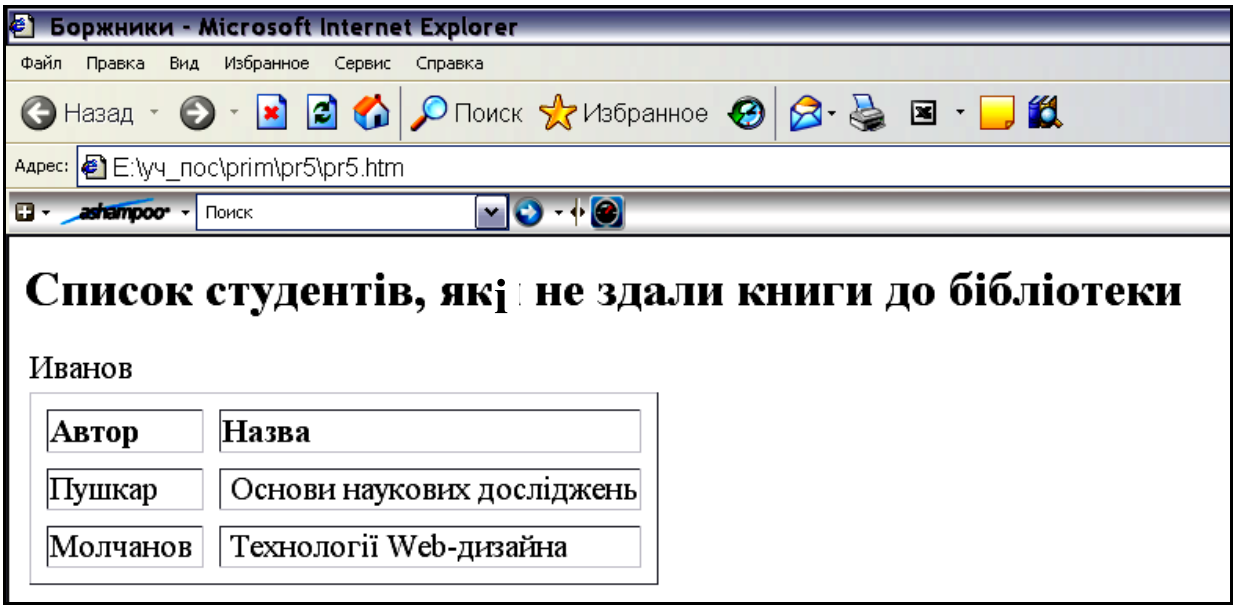

Рис. 9. **Відображення сторінки у браузері**

Для додаткового оформлення можна використовувати й інші теги HTML.

Зчеплення елементів для передачі вмісту може застосовуватися не тільки для тега TABLE, а і для інших. У табл. 6 надано елементи, які використовуються найчастіше.

Таблиця 6

HTML-елементи для зчеплення

HTML-елемент	Властивість для зчеплення елемента	Чи передає розмітку HTML?	Чи оновлює зчеплене поле XML?
1	2	3	4
A	href	Ні	Ні
BUTTON	innerHTML, innerText	Так	Ні
DIV	innerHTML, innerText	Так	Ні

Закінчення табл. 6

1	2	3	4
IFRAME	src	Ні	Ні
IMG	src	Ні	Ні
INPUT TYPE=CHECKBOX	checked	Ні	Так
INPUT TYPE=HIDDEN	value	Ні	Так
INPUT TYPE=RADIO	checked	Ні	Так
INPUT TYPE=TEXT	value	Ні	Так
LABEL	innerText, innerHTML	Так	Ні
SPAN	innerText, innerHTML	Так	Ні
TEXTAREA	value	Ні	Так

HTML-елемент зв'язується з окремим полем XML. Після цього HTML-елемент автоматично відображає вміст поля XML, з яким він пов'язаний (зчеплений). Наприклад, наступний HTML-елемент SPAN зчеплений з полем TITLE XML-документа, доступ до якого здійснюється через фрагмент даних dsoBook:

<SPAN DATASRC="#dsoBook " DATAFLD="TITLE"></SPAN>

А цей елемент img зчеплений з полем photo XML-документа, доступ до якого здійснюється через фрагмент даних dso3:

<img DATASRC="#dso3" DATAFLD="photo">

Якщо поле містить правильну адресу файла з малюнком, він буде відображений браузером.

Для наступного зчепленого елемента A (гіперпосилання) властивість href зчеплена з полем XML-документа REVIEWS:

```
<A DATASRC="#dso3 " DATAFLD="REVIEWS">
 Click here for reviews
</A>
```

Ця властивість, як і атрибут HREF елемента, встановлює URL для гіперпосилання. Отже, з поля XML витягується URL гіперпосилання для зчепленого елемента A, а не його текстовий вміст. XML-поле REVIEWS в даному прикладі повинне містити коректний URL.

Оскільки HTML-елемент не має множинних частин, подібно до таблиці, він здатний відобразити значення поля лише для одного запису за один раз. Щоб використовувати зв'язування даних по одному запису, XML-документ має бути організований як простий набір записів.

Запис, що відображається у даний момент, називається поточним записом. Спочатку поточним є перший запис у документі.

DSO, що асоціюється з XML-документом, надає ряд методів для переміщенні між записами. Ці методи належать об’єкту recordset DSO і наведені в табл. 7. Приклади викликів, приведені в останньому стовпці, припускають, що HTML-сторінка містить фрагмент даних XML з ідентифікатором (ID="dso1").

Таблиця 7

Методи для переміщення між записами

Метод об’єкта recordset DSO	Перехід до записи	Приклад виклику
moveFirst	перша запис в документі	dso1.recordset.moveFirst()
movePrevious	попередня запис	dso1.recordset.movePrevious()
moveNext	наступна запис	dso1.recordset.moveNext()
moveLast	остання запис	dso1.recordset.moveLast()
move	запис з вказаним номером	dso1.recordset.move(5)

Як і у випадку з таблицею найпростіше використовувати атрибут ONCLICK елемента BUTTON. Елемент, який записаний по правилах XHTML виглядає так:

```
<BUTTON ONCLICK="dso1.recordset.moveFirst()">
 First Record
</BUTTON>
```

Елемент по правилах HTML виглядає так:

```
<INPUT TYPE=BUTTON VALUE=First Record
ONCLICK="dso1.recordset.moveFirst()">
```

Цей елемент відображає кнопку. Коли користувач клацає мишею на кнопці, викликається метод, привласнений атрибута ONCLICK, dso1.recordset.moveFirst().

Якщо поточним є перший запис, виклик методу movePrevious призводить до переміщення в зону початку файла (BOF), де немає записів, тому зчеплений елемент буде порожній. Аналогічно виклик методу moveNext, якщо поточним є останній запис, призводить до переміщення в зону кінця файла (EOF), тому зчеплений елемент також буде порожній. Об’єкт recordset підтримує властивість BOF, яка набуває значення true (істина), якщо досягнутий початок файла, а також властивість EOF, яка набуває значення true (істина), якщо досягнутий кінець файла. Ці властивості можна використовувати для визначення цих станів і внесення необхідних коректувань.

Наприклад, приведений нижче код позволяє при натисненні на кнопці у випадку, якщо досягнутий початок файла, відобразити перший запис:

```
<BUTTON ONCLICK="dso1.recordset.movePrevious();
  if (dso1.recordset.BOF) dso1.recordset.moveNext()">
Back
</BUTTON>
```

Наступний код перевіряє досягнення кінця файла:

```
<BUTTON ONCLICK="dso1.recordset.moveNext();
  if (dso1.recordset.EOF) dso1.recordset.movePrevious()">
Forward
</BUTTON>
```

Приклад 6. Створимо сторінку для відображення даних з файла, який містить список з ім'ям, фото та посиланням на текстове резюме такої структури:

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>Список групи
 <STUDENT>
  <FAM>Иванов</FAM>
  <NAME>Андрій</NAME>
  <photo>im1.gif</photo>
  <res> res1.doc</res>
</STUDENT>
  ...
</DOCUMENT>
```

Сторінка для відображення даних може виглядати так:

```
<HTML>
<HEAD>
  <TITLE>Список групи</TITLE>
</HEAD>
<BODY>
<XML ID="dso2" SRC="pr6.xml"></XML>
<H2>Список групи</H2>
<TABLE>
  <THEAD>
    <TH>Прізвіще</TH>
    <TH>Ім'я</TH>
    <TH>Фото</TH>
    <TH>Резюме</TH>
 </THEAD>
  <TR ALIGN="center">
  <TD><SPAN DATASRC="#dso2" DATAFLD="FAM" ></SPAN></TD>
  <TD><SPAN DATASRC="#dso2" DATAFLD="NAME"></SPAN></TD>
  <TD><IMG DATASRC="#dso2" DATAFLD="photo"></SPAN></TD>
  <TD><A DATASRC="#dso2" DATAFLD="res">резюме</A></TD>
 </TR>
</TABLE>
<input type=BUTTON value="&lt Перша"
ONCLICK="dso2.recordset.moveFirst()">
```

```
<input type=BUTTON value="&lt Попередня"
ONCLICK="dso2.recordset.movePrevious();
        if (dso2.recordset.BOF) dso2.recordset.moveNext()">
<input type=BUTTON value="Наступна &gt "
ONCLICK="dso2.recordset.moveNext();
          if (dso2.recordset.EOF) dso2.recordset.movePrevious()">
<input type=BUTTON value="Остання &gt"
ONCLICK="dso2.recordset.moveLast()">
</BODY>
</HTML>
```

При зв'язуванні елемента SPAN з полем XML, елемент просто відображає вміст поля. Це відбувається тому, що властивість innerText елемента SPAN, яке визначає текст, що відображається елементом, зчеплена з полем XML. При зв'язуванні елемента IMG – вміст поля привласнюється атрибута SRC, при зв'язуванні елемента A – атрибута HREF. Кнопки забезпечують перегляд записів.

Відображення у браузері надано на рис. 10.

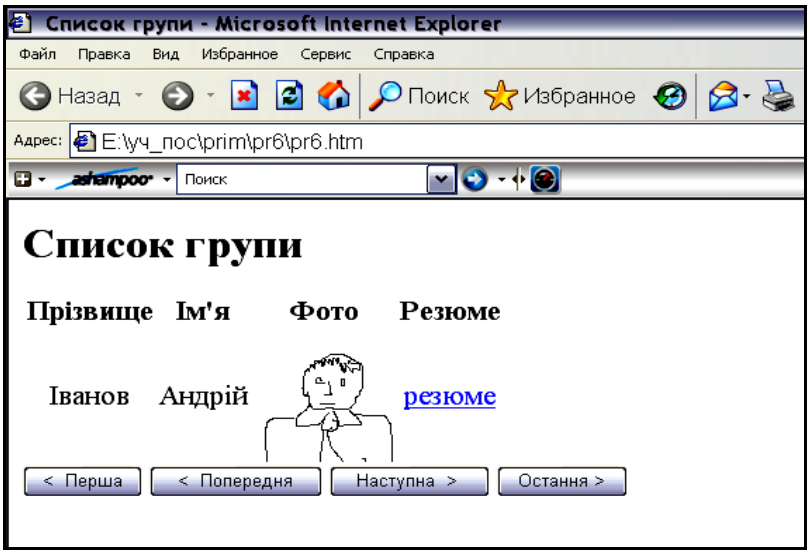

Рис. 10. **Відображення даних із зображенням та посиланням**

За замовчуванням, якщо символьні дані XML-поля включають HTML-розмітку (HTML-теги), HTML-елемент, зчеплений з цим полем, сприймає і відображає символи розмітки як звичайний рядок символів. Однак для окремих HTML-елементів (табл. 6), наприклад, SPAN, можна використовувати атрибут DATAFORMATAS="HTML", це змусить браузер обробляти HTML-розмітку, що міститься в тексті поля. За замовчуванням DATAFORMATAS="TEXT". Проте слід пам'ятати, що текст у XML-тегах не може містить інших тегів, тому ліву кутову дужку слід замінити підстановкою <

Наприклад, якщо XML- елемент

```
<AUTHOR>Henry James was &lt;I> 1843-1916 &lt;/I>US</AUTHOR>
```

зчеплен з HTML-елементом

<SPAN DATASRC="#dsoInventory " DATAFLD="AUTHOR"></SPAN>,

то частина тексту "1843-1916" буде відтворена курсивом.

Використовуючи передачу HTML-розмітки в XML-полях можна не тільки змінювати формат частини тексту, але і включати такі HTML-елементи, як гіперпосилання або зображення.

Об'єкт DSO XML надає можливість модифікувати дані XML. Можна використовувати HTML-елемент, що допускає оновлення, наприклад, елемент INPUT типу TEXT. Крім того, об'єкт recordset DSO надає методи, які дозволяють додавати або видаляти цілі записи з накопиченого набору записів, а також відміняти модифікацію записів. Однак слід мати на увазі, що при цьому модифікується тільки лише копія даних XML, яку DSO тимчасово зберігає в пам'яті, а не оригінальний XML-документ на сервері.

Пов'язання HTML-елементів дозволяє також відображати дані з XML-атрибутів. При зв'язуванні даних атрибут трактується як дочірній елемент. Наприклад, наступний запис BOOK містить атрибут з ім'ям InStock :

```
<BOOK InStock="yes ">
 <TITLE>The Adventures of Huckleberry Finn</TITLE>
 <AUTHOR>Mark Twain</AUTHOR>
 <BINDING>mass market paperback</BINDING>
 <PAGES>298</PAGES>
 <PRICE>$5.49</PRICE>
</BOOK>
```

Цей запис трактується так, як якби атрибут InStock був полем, належним BOOK, а значення InStock було б вмістом цього поля. Таким чином, елемент BOOK еквівалентний такій структурі:

```
<BOOK>
 <InStock>yes</InStock>
 <TITLE>The Adventures of Huckleberry Finn</TITLE>
 <AUTHOR>Mark Twain</AUTHOR>
 <BINDING>mass market paperback</BINDING>
 <PAGES>298</PAGES>
 <PRICE>$5.49</PRICE>
</BOOK>
```

Отже, можна використовувати звичайну техніку зв'язування даних:

```
<SPAN DATASRC="#dso1 " DATAFLD="InStock"></SPAN>
```

Однак слід враховувати, що при додаванні атрибута до одного з елементів-полів у XML-документі можна перетворити елемент на вкладений запис.

І ще одна можливість, DSO використовує спеціальне ім'я $TEXT для звернення до всіх символьних даних елемента, не включаючи при цьому значень атрибутів.

2.4. Об’єктна модель XML-документа

При читанні браузером XML-документа (наприклад, при зв’язуванні тегів) у пам’яті створюється екземпляр об’єкта DOM (Document Object Model). DOM – це структурований спосіб представлення XML-даних у пам’яті [10]. Вона (модель) складається з групи програмних об’єктів, що представляють різні компоненти XML-документа. Властивості і методи цих об’єктів дозволяють програмно читати, обробляти і змінювати дані XML-документа, використовувати сценарії для відображення XML-документа на HTML-сторінці. DOM зберігає дані в ієрархічній, деревоподібній структурі, що відображає ієрархічну структуру XML-документа.

Наприклад, для фрагмента документа з прикладу 3 у пам’яті буде створена така ієрархічна структура (дерево об’єктів, рис. 11). Фрагмент узятий, щоб зменшити його розмір:

```
<?xml version="1.0" encoding="UTF-8"?>
<список> Список літератури з дисципліни "Засоби систем обробки даних"
 <джерело наявність="так">
   <автор>Холзнер С.</автор>
   <назва>Энциклопедия, 2-е изд.</назва>
 </джерело>
<джерело наявність="ні">
   <автор>Сакс Т., Мак-Клейн Г.</автор>
   <назва>Дизайн и архитектура современного вебсайта. Опыт профессионалов.</назва>
 </джерело>
</список>
```

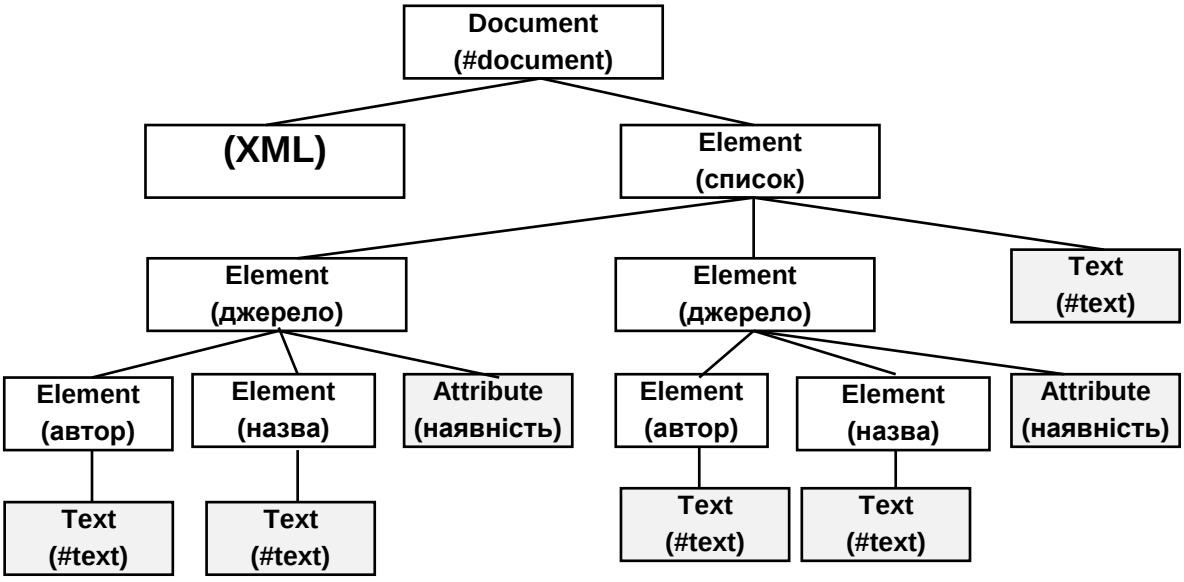

Рис. 11. **Дерево об’єктів XML-документа**

Кожен прямокутник на цьому рисунку (вузол у структурі XML-документа) є об’єктом. Вузли бувають різних типів. Тип вузла визначає його характеристики і функціональність. Наприклад, інструкції з обробки відрізняються від елементів, для них створюються спеціальні вузли (на рис.10 це XML). Основні типи вузлів наведені в табл. 8.

Таблиця 8

Основні типи вузлів DOM

Тип вузла	Компоненти XML	Ім’я вузла (властивість nodeName)	Значення вузла (властивість nodeValue)
Document	Кореневий вузол ієрархії	#document	null
Element	Елемент	Ім’я типу елемента (наприклад, BOOK)	будь-які символьні дані
Text	Текст, що належить елемента	#text	Текст
Attribute	Атрибут	Ім’я атрибута	Значення атрибута
Processing-Instruction	Інструкція з обробки	Призначення інструкції з обробки (наприклад, xml)	Повний вміст інструкції з обробки
Comment	Коментар	#comment	Весь текст усередині обмежувачів

Атрибути не перебувають у батьківських, дочірніх і однорівневих відносинах. Вони вважаються за об’єкти, що належать вузлу елемента. Наприклад, *наличие="есть"* – це атрибут елемента *источник*, значення атрибута можна набути за допомогою спеціальних методів (наприклад, getAttribute) елемента *источник*.

Текстовий зміст елемента також є об’єктом (із спеціальним ім’ям #text), що належить елемента. Об’єкти, що належать елементам, на рисунку виділені фоном.

Кожен вузол, як програмний об’єкт, має властивості і методи, які дозволяють здійснювати доступ, відображати, обробляти і отримувати інформацію про відповідний компонент XML. Наприклад, властивості nodeName і nodeValue (табл. 9) дозволяють отримати ім’я компонента і його значення.

Вузли одного типу містять загальний набір властивостей і методів. У табл. 9 представлені деякі найбільш часто використовувані властивості.

Таблиця 9

Властивості вузлів

Властивість	Опис	Приклад використання
attributes	Безліч всіх атрибутів вузла	AttributeNode = Element.attributes.getNamedItem ("Binding");
childNodes	Безліч дочірніх вузлів	FirstNode = Element.childNodes(0);
firstChild	Перший дочірній вузол	FirstChildNode = Element.firstChild;
lastChild	Останній дочірній вузол	LastChildNode = Element.lastChild;
nextSibling	Наступний вузол на тому ж рівні	NextElement = Element.nextSibling;
nodeName	Ім’я вузла	ElementName = Element.nodeName;
nodeValue	Значення вузла	AttributeValue = Attribute.nodeValue;
text	Текстовий вміст вузла	AIICharacterData = Element.text;

Модель DOM використовується для програмного доступу до всіх даних XML-документа.

2.5. Відображення XML-документів за допомогою скриптів

З використанням моделі DOM можна відображати дані XML-документа в довільному вигляді. Проте, передача XML-файла для розбору в різних браузерах проводиться по-різному. Тому доводиться розробляти скрипти з урахуванням типу використовуваної програми. Найпростіше це робиться в Internet Explorer, який підтримує елемент <XML ID= "…" SRC= "..."></XML>. При його інтерпретації браузер створює DOM. У інших програмах необхідно створювати відповідні об’єкти і використовувати їх методи.

З урахуванням сказаного, розглянемо як приклад відображення документа в браузері Internet Explorer.

Приклад 7. Для відображення візьмемо документ, який використовувався в прикладах 3 і 4, частина DOM якого наведена на рис. 11.

Побудова дерева DOM забезпечується включенням в HTML-сторінку елемента <XML>. Скрипт повинен виконуватися в ході інтерпретації сторінки, тому він розміщується після елемента <XML>.

Код сторінки:

```
<HTML>
<HEAD>
<TITLE>Book Description</TITLE>
</HEAD>
<BODY>
<H2>Отображение содержимого XML-документа с помощью скрипта</H2>
<XML ID="dso1" SRC="pr2.xml"></XML>
 <script language="JavaScript">
doc = dso1.XMLDocument;
this.document.write("<SPAN id=’i0’ STYLE="+"’"+"font-size:24; font-
style:italic"+"’> </SPAN><br>");
document.getElementById("i0").innerText=
doc.documentElement.childNodes(0).text;
for(i=1;i<=doc.documentElement.childNodes.length-1;i++)
{
st="<SPAN id=’i"+i+"’ STYLE="+"’"+"font-size:20; font-style:italic"+"’>
</SPAN><br>";
this.document.write(st);
si="i"+i;
sp=document.getElementById(si);
sp.innerText=doc.documentElement.childNodes(i).text;
};
</script>
</BODY>
</HTML>
```

Скрипт виводить у вхідний потік порожній елемент <SPAN > і привласнює його властивості innerText текстовий вміст кореневого елемента childNodes (0). Потім у циклі у вхідний потік записуються елементи <SPAN >, в яких розміщується текстовий вміст дочірніх вузлів кореневого елемента. Вид документа у вікні браузера приведений на рис. 12.

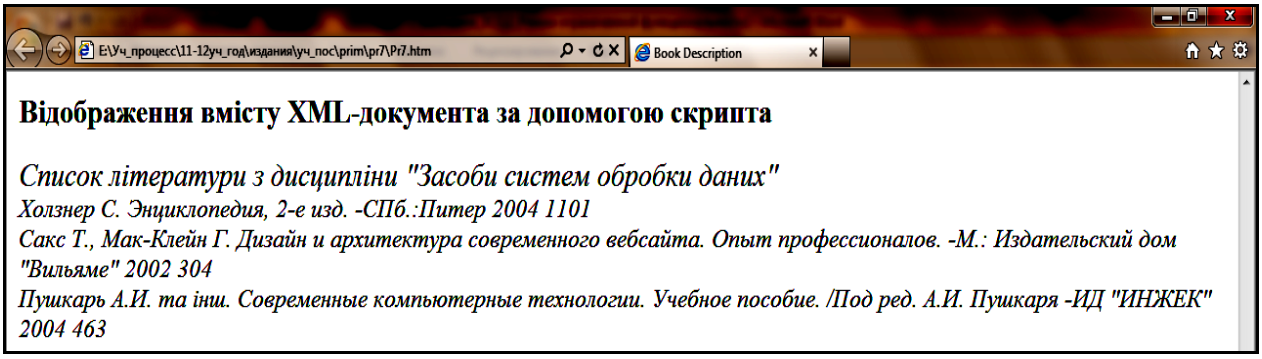

Рис. 12. **Відображення документа у браузері**

Сама обробка може бути і складнішою, включати відображення рисунків та інших елементів.

2.6. Перевірка коректності XML-документів

Важливою можливістю технології XML є можливість перевірки документа на відповідність формальним критеріям відсутності помилок. Існують два критерії для перевірки XML-документа: валідність (дієвість) і коректності (правильність) [2; 10].

Документ є формально коректним, якщо він задовольняє загальні вимоги до XML-документів (п.п. 1.1.5).

Під валідністю розуміється відповідність документа синтаксису, який визначає розробник.

Перевірка коректності повинна забезпечити відповідність всього документа основним вимогам синтаксису XML. При цьому перевіряється, чи немає незакритих тегів, чи всі атрибути поміщені в лапки, чи немає тегів з порушенням вкладеності, чи має документ єдиний кореневий елемент, і тому подібне. Дана перевірка є обов'язковою для всіх XML-документів.

Якщо, наприклад, початковий тег не має відповідного йому кінцевого тега, то це *неправильно побудований* документ XML. Документ, який неправильно побудований, не може вважатися за документ XML, XML-процесор (парсер) не повинен обробляти його звичайним способом, він зобов'язаний класифікувати ситуацію як фатальна помилка. При тому подальша обробка припиняється. Так поводяться більшість браузерів.

Вимоги валідності є додатковим набором правил у специфікації XML, яким необхідно слідувати, щоб уникнути помилок при використанні документа. Оскільки валідність є не обов'язковою для XML-документа, відхилення від вимог вважається лише за помилку (error), обробка не припиняється. Якщо XML-процесор зустрічає помилку, він може просто видати повідомлення про неї і продовжити виконання обробки.

Для перевірки валідності документ повинен містити формальний опис синтаксису, визначеного розробником. Перевірка валідності можлива лише для тих документів, для яких заданий такий формальний опис.

Для опису синтаксису XML-документів нині створено дві мови: DTD (Document Type Definition) і XSD (XML Schema Definition). Обидві мови стандартизовано на рівні World Wide Web Consortium (W3C) [17]. Фактично, XSD і DTD – це "граматика для певного класу документів".

Якщо порівнювати XSD і DTD, то мова XSD дозволяє задавати більш суворі обмеження на типи даних XML-документа, але є складні-

шою. Не дивлячись на те, що схеми даних XSD самі по собі є документами XML, редагувати їх за допомогою звичайного XML-редактора виявляється незручним через їх достатню складність. Тому, як правило, для їх створення використовуються спеціальні редактори, що відображають їх у наглядному графічному вигляді. Тому ми розглянемо саме DTD.

2.7. Створення DTD

Оголошення типу документа (DTD) визначає структуру документа та зміст його елементів. Це дозволяє перевіряти документ на валідність (на додаток до перевірки на коректність). Кожен елемент і атрибут, який використовується, повинні відповідати специфікації DTD, записаній у відповідному оголошенні.

Оголошенням типу документа (DTD) є блок XML-розмітки, який при додаванні в пролог, може розташовуватися в будь-якому місці прологу – поза іншою розміткою, після XML-оголошення [2, 10].

DTD може міститися в пролозі XML-документа (внутрішнє) або в зовнішньому файлі, ім'я якого вказується в документі (зовнішнє). Виглядає це так для внутрішнього оголошення:

```
...
<! DOCTYPE rootname [
<!ELEMENT rootname (contacts, issues, authors )>
...
]>
...
```

Або так для зовнішнього:

```
<?xml version=1.0 standalone="no " ?>
<! DOCTYPE rootname SYSTEM "fileDTD.dtd">
...
```

Можна одночасно використовувати внутрішні і зовнішні визначення DTD за допомогою елемента <!DOCTYPE> такого вигляду:

```
!DOCTYPE rootname SYSTEM "URL"
 [...
]>
```

Зазвичай внутрішні визначення доповнюють зовнішні.

 Якщо елемент або атрибут визначений і у внутрішньому, і в зовнішньому визначеннях DTD, то вважається, що визначення у внутрішньому DTD має більший пріоритет, тому передує зовнішньому визначенню. Такий порядок дозволяє необхідним чином налаштовувати зовнішні визначення DTD. Проте досвід показує, що більшість XML-процесорів вважають таке за помилку і, зазвичай, зупиняються, якщо між внутрішніми і зовнішніми DTD виникає конфлікт відносно елемента або атрибута.

Частіше за все DTD містить такі типи оголошень розмітки:

оголошення типів елементів**,** вони визначають типи елементів, які може містити документ, а також вміст і порядок проходження елементів;

оголошення списків атрибутів, кожне оголошення списків атрибутів задає імена атрибутів, які можуть бути використані з певним типом елемента, а також типи даних і встановлювані за замовчуванням значення цих атрибутів;

оголошення примітивів, примітиви застосовуються для зберігання часто використовуваних фрагментів тексту або для вбудовування тих, що не відносяться до XML-даних у документі.

2.7.1. Оголошення типів елементів

У валідному XML-документі необхідно повністю оголосити тип кожного елемента, який використовується. Оголошення типу елемента указує на ім’я типу елемента, допустимий вміст елемента і порядок розміщення дочірніх елементів. Як єдине ціле, оголошення типів елементів у DTD задає повну логічну структуру документа.

Оголошення типу елемента має таку узагальнену форму:

<!ELEMENT Ім’я опис_вмісту>

Тут "Ім’я" є ім’я оголошуваного типу елемента. "опис_вмісту" визначає, що може містити елемент. Оголошувати певний тип елемента в даному документі можна тільки один раз.

Наприклад, оголошення типу елемента з ім’ям TITLE, для вмісту якого можуть використовуватися тільки символьні дані (дочірні елементи не допускаються):

<!ELEMENT TITLE (#PCDATA)>

#PCDATA (parseable character data) – будь-яка інформація, з якою може працювати програма-аналізатор. Це текст, що не є розміткою. Він називається розібраними символьними даними.

Крім #PCDATA "опис_вмісту" може містить один з чотирьох типів вмісту: EMPTY (елемент має бути порожнім – тобто не може мати вмісту), ANY (будь-який, елемент цього типу може містити або не містити дочірні елементи, мати або не мати символьних даних), дочірній і змішаний вміст.

Якщо елемент має дочірній вміст, він може безпосередньо містити лише певні дочірні елементи, але не символьні дані. У тексті документа можна розділяти дочірні елементи символами пропуску, табуляції, по-

вернення каретки або переведення рядка, щоб поліпшити сприйняття документа людиною, але процесор ігноруватиме ці символи і не передаватиме їх додатку.

Приклад XML-документа, що описує одну книгу:

```
<?xml version="1.0" encoding="windows-1251 " ?>
<!DOCTYPE BOOK
        [
        <!ELEMENT BOOK (TITLE, AUTHOR )>
        <!ELEMENT TITLE (#PCDATA )>
        <!ELEMENT AUTHOR (#PCDATA )>
        ]
>
<BOOK >
        <TITLE>The Scarlet Letter</TITLE>
        <AUTHOR>Nathaniel Hawthorne</AUTHOR>
</BOOK >
```

У цьому документі тип елемента BOOK оголошений таким, що має дочірній вміст елементи TITLE і AUTHOR. У даному прикладі елемент BOOK повинен мати рівно один дочірній елемент TITLE, за яким слідує рівно один дочірній елемент AUTHOR. Пропуск дочірнього елемента або використання одного і того ж типу дочірнього елемента більше одного разу також неприпустимо.

Отже, такий документ не буде валідним:

```
<?xml version="1.0" encoding="windows-1251 " ?>
<!DOCTYPE BOOK
        [
        <!ELEMENT BOOK (TITLE, AUTHOR )>
        <!ELEMENT TITLE (#PCDATA )>
        <!ELEMENT AUTHOR (#PCDATA )>
]
>
<BOOK >
        <TITLE>The Scarlet Letter</TITLE>
        <AUTHOR>Nathaniel Hawthorne</AUTHOR>
        <TITLE>The Scarlet Letter</TITLE>
        <AUTHOR>Nathaniel Hawthorne</AUTHOR>
</BOOK >
```

При необхідності можна вказати, що елемент може мати будь-який з серії допустимих дочірніх елементів, елементи розділяються символом "|". Наприклад, наступне DTD указує, що елемент FILM може складатися з одного дочірнього елемента STAR, або одного дочірнього елемента NARRATOR, або одного дочірнього елемента INSTRUCTOR:

```
<!DOCTYPE FILM
        [
        <!ELEMENT FILM (STAR | NARRATOR | INSTRUCTOR)>
        <!ELEMENT STAR (#PCDATA )>
```

```
        <!ELEMENT NARRATOR (#PCDATA )>
        <!ELEMENT INSTRUCTOR (#PCDATA )>
        ]
>
```

Отже, такий елемент буде валідним:

```
<FILM >
   <STAR>Robert Redford</STAR>
</FILM >
```

як і елемент:

```
<FILM >
   <NARRATOR>Sir Gregory Parsloe</NARRATOR>
</FILM >
```

А цей – ні (оскільки можна включити лише один з дочірніх елементів):

```
<FILM >
   <NARRATOR>Sir Gregory Parsloe</NARRATOR>
   <INSTRUCTOR>Galahad Threepwood</INSTRUCTOR>
</FILM >
```

Будь-яку з цих форм вмісту можна змінювати, використовуючи символи: знак питання (?), знак плюс (+) і зірочка (*), значення яких описані в табл. 10.

Таблиця 10

Символи модифікації

Символ	Значення
?	Жоден або один з попередніх елементів
+	Один або більш з попередніх елементів
*****	Жоден або будь-яка кількість з попередніх елементів

Наприклад, наступне оголошення означає, що можна включити один або більш дочірніх елементів NAME, і що дочірній елемент HEIGHT є не обов’язковим:

```
<!ELEMENT MOUNTAIN (NAME+, HEIGHT ?, STATE )>
```

Таким чином, такий елемент буде валідним:

```
<MOUNTAIN >
   <NAME>Pueblo Peak</NAME>
   <NAME>Taos Mountain</NAME>
   <STATE>New Mexico</STATE>
</MOUNTAIN >
```

Наступне оголошення означає, що можна включити декілька або жодного дочірнього елемента STAR, або один дочірній елемент NARRATOR, або один дочірній елемент INSTRUCTOR :

```
<!ELEMENT FILM (STAR* | NARRATOR | INSTRUCTOR )>
```

Відповідно, кожен з таких елементів буде валідним:

```
<FILM >
   <STAR>Tom Hanks</STAR>
   <STAR>Meg Ryan</STAR>
</FILM >
<FILM >
   <NARRATOR>Sir Gregory Parsloe</NARRATOR>
</FILM >
```

Символи модифікації (?, +, *) можна помістити після дужок. Наприклад, наступне оголошення допускає включення одного або декількох дочірніх елементів будь-якого з цих трьох типів у будь-якому порядку:

```
<!ELEMENT FILM (STAR | NARRATOR | INSTRUCTOR )+>
```

Таке оголошення робить валідним всі наступні елементи:

```
<FILM >
   <NARRATOR>Bertram Wooster</NARRATOR>
   <STAR>Sean Connery</STAR>
   <NARRATOR>Plug Basham</NARRATOR>
</FILM >
<FILM >
   <STAR>Sean Connery</STAR>
   <STAR>Meg Ryan</STAR>
</FILM >
<FILM >
   <INSTRUCTOR>Stinker Pike</INSTRUCTOR>
</FILM >
```

Для формування більш складнішого вмісту можна вкладати одну модель в іншу. Наприклад, наступне DTD задає, що кожен елемент FILM повинен мати один дочірній елемент TITLE, за ним повинен слідувати один дочірній елемент CLASS, після нього повинні йти один з дочірніх елементів STAR, NARRATOR або INSTRUCTOR.

```
<!DOCTYPE FILM
          [
          <!ELEMENT FILM (TITLE, CLASS, (STAR | NARRATOR | INSTRUCTOR ) )>
          <!ELEMENT TITLE (#PCDATA )>
          <!ELEMENT CLASS (#PCDATA )>
          <!ELEMENT STAR (#PCDATA )>
          <!ELEMENT NARRATOR (#PCDATA )>
          <!ELEMENT INSTRUCTOR (#PCDATA )>
          ]
>
```

Відповідно до цього DTD, наступний елемент буде валідним:

```
<FILM >
   <TITLE>The Net</TITLE>
   <CLASS>fictional</CLASS>
   <STAR>Sandra Bullok</STAR>
</FILM >
```

Так само, як такий:

```
<FILM >
   <TITLE>How to Use XML</TITLE>
   <CLASS>instructional</CLASS>
   <INSTRUCTOR>Penny Donaldson</INSTRUCTOR>
</FILM >
```

Якщо елемент має змішаний вміст, він може включати і дочірні елементи, і символьні дані. При змішаному вмісті можна задавати типи дочірніх елементів, але не порядок або кількість входжень.

Щоб оголосити тип елемента, який може містити символьні дані і на додаток жодного або декілька дочірніх елементів, кожен тип дочірнього елемента перераховується після ключового слова #PCDATA і розділяється символами "|". У кінці поміщається *. Кожне ім'я елемента може з'являтися в оголошенні тільки один раз.

Наприклад, наступне оголошення указує, що елемент TITLE може містити символьні дані плюс жодного або декілька дочірніх елементів SUBTITLE:

```
<!ELEMENT TITLE (#PCDATA | SUBTITLE )*>
```

Відповідно до цього оголошення наступні елементи TITLE є допустимими:

```
<TITLE >
   Moby-Dick
   <SUBTITLE>Or, the Whale</SUBTITLE>
</TITLE >
<TITLE >
   <SUBTITLE>Or, the Whale</SUBTITLE>
   Moby-Dick
</TITLE >
<TITLE >
   Moby-Dick
</TITLE >
<TITLE >
   <SUBTITLE>Or, the Whale</SUBTITLE>
   <SUBTITLE>Another Subtitle</SUBTITLE>
</TITLE >
<TITLE></TITLE>
```

2.7.2. Оголошення атрибутів

У валідному XML-документі необхідно оголосити всі атрибути, які передбачається використовувати для елементів. Атрибути, що асоціюються з певним елементом, визначаються за допомогою спеціального типу DTD-розмітки, званого оголошенням списку атрибутів. Воно:

визначає імена атрибутів, що асоціюються з елементом;

встановлює тип даних кожного атрибута;

задає обов'язковість для кожного атрибута.

Якщо атрибут необов’язковий, в оголошенні списку атрибутів указується, що повинен чинити процесор, якщо атрибут опущений. У оголошенні повинно, наприклад, міститися значення атрибута за замовчуванням, яке використовуватиме процесор.

Оголошення списку атрибутів має таку загальну форму:

<!ATTLIST Ім’я ВизАт>

Тут "Ім’я" є ім’я елемента, що асоціюється з атрибутом або атрибутами. "ВизАт" – це одне або декілька значень атрибутів, кожне з яких визначає один атрибут.

Визначення атрибута має таку форму запису:

Ім’я ТипАтр ОгУмов

Тут "Ім’я" є ім’ям атрибута.

"ТипАтр" є типом атрибута, тобто види значень, які можуть бути привласнені атрибута.

"ОгУмов" – це оголошення за замовчуванням, яке указує на обов’язковість атрибута і містить іншу інформацію.

Припустимо, оголошений тип елемента з ім’ям FILM таким чином:

<!ELEMENT FILM (TITLE (STAR NARRATOR INSTRUCTOR))>

Ось приклад оголошення списку атрибутів, яке описує два атрибути – Class і Year – для елемента FILM :

<!ATTLIST FILM Class CDATA "fictional " Year CDATA #REQUIRED >

На рис. 13 представлені складові частини цього оголошення.

Атрибуту Class можна привласнити будь-який рядок у лапках (ключове слово CDATA). Якщо атрибут для певного елемента буде опущений, йому буде автоматично привласнено значення за замовчуванням "fictional". Атрибута Year можна привласнити будь-який рядок у лапках; однак це повинно бути зроблено обов’язково для кожного елемента FILM (ключове слово #REQUIRED), тому він не має значення за замовчуванням.

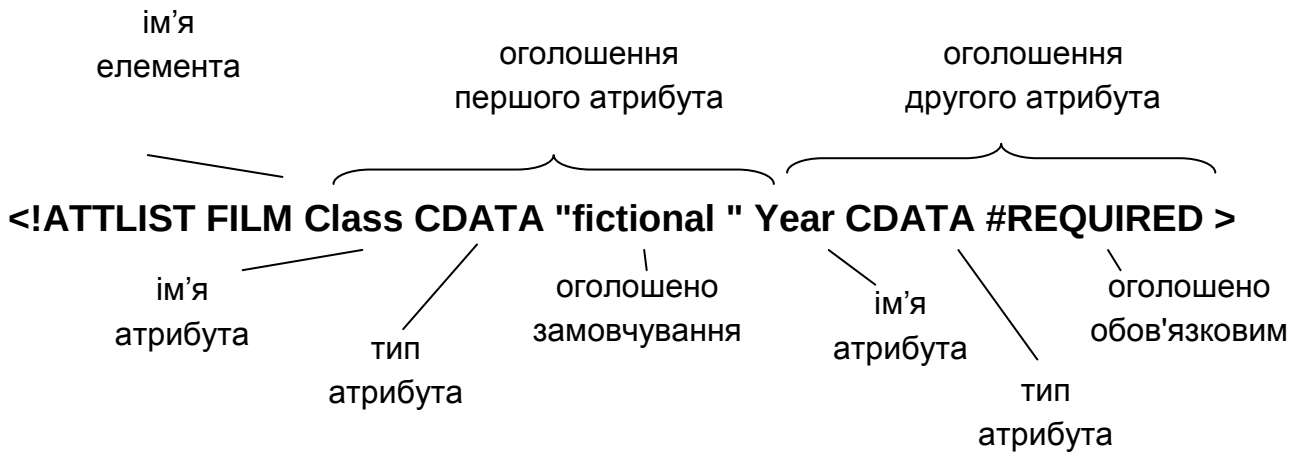

Рис. 13. **Складові частини оголошення атрибутів**

Наступний повний XML-документ включає оголошення списку атрибутів, а також елемент FILM:

```
<?xml version="1.0" encoding="windows-1251 " ?>
<!DOCTYPE FILM
   [
   <!ELEMENT FILM (TITLE, (STAR | NARRATOR | INSTRUCTOR ) )>
   <!ATTLIST FILM Class CDATA "fictional " Year CDATA #REQUIRED >
   <!ELEMENT TITLE (#PCDATA )>
   <!ELEMENT STAR (#PCDATA )>
   <!ELEMENT NARRATOR (#PCDATA )>
   <!ELEMENT INSTRUCTOR (#PCDATA )>
   ]
>
<FILM Year="1948 ">
   <TITLE>The Morning After</TITLE>
   <STAR>Morgan Attenbury</STAR>
</FILM >
```

Для елемента FILM атрибута Year привласнено значення "1948". Атрибут Class опущений; проте, оскільки цей атрибут має значення за замовчуванням ("fictional"), воно привласнюється атрибута, неначебто його записали як значення атрибута.

Якщо використовується для одного елемента більш за одне оголошення списку атрибутів, зміст оголошень об’єднується. Якщо атрибут із заданим ім’ям оголошений для одного і того ж елемента кілька разів, перше оголошення використовується, а подальші – ігноруються.

Усього існує шість можливих типів значень атрибута:

CDATA – вмістом документа можуть бути будь-які символьні дані;

ID – визначає унікальний ідентифікатор елемента в документі;

IDREF (IDREFS) – указує, що значенням атрибута повинна виступати назва (або декілька таких назв, розділених пропусками) унікального ідентифікатора визначеного в цьому документі елемента;

ENTITY (ENTITIES) – значення атрибута має бути назвою (або списком назв, якщо використовується ENTITIES) компонента (макро-визначення), визначеного в документі;

NMTOKEN (NMTOKENS) – вмістом елемента може бути тільки одне окреме слово(тобто цей параметр є обмеженим варіантом CDATA);

список допустимих значень – визначається список значень, які може мати даний атрибут.

Також у оголошенні атрибута можна використовувати такі параметри:

#REQUIRED – визначає обов’язковий атрибут, який має бути заданий у всіх елементах даного типу;

#IMPLIED – атрибут не є обов’язковим;

#FIXED "значення" – указує, що атрибут повинен мати лише вказане значення, однак саме визначення атрибута не є обов’язковим, але в процесі розбору його значення у будь-якому випадку буде передано програмі-аналізатору;

значення – задає значення атрибута за замовчуванням.

2.7.3. Визначення компонентів (макровизначень)

Компонент (entity) є визначеннями, вміст яких може бути повторно використаний у документі. У мовах програмування подібні елементи називаються макровизначеннями. Створюються такі DTD-компоненти за допомогою інструкції !ENTITY:

<!ENTITY hello ’ Ми раді вітати Вас!’ >

Програма-аналізатор, проглядаючи вміст області DTD-визначень, обробить цю інструкцію і при подальшому розборі документа використовуватиме вміст DTD-компонента в тому місці, де зустрічатиметься його назва. Тобто тепер у документі можна використовувати вираз &hello;, який буде замінено на рядок "Ми раді вітати Вас".

У загальному випадку, усередині DTD можна задати три типи макровизначень:

Внутрішні макровизначення – призначені для визначення строкової константи, з їх допомогою можна організовувати посилання на часто змінну інформацію, роблячи документ більш читабельним. Внутрішні компоненти включаються у документ за допомогою символу &.

У XML існує п’ять установлених внутрішніх символьних констант:

&amplt ; – символ "<";

&ampgt ; – символ ">";

&ampamp ; – символ "&";

&ampapos ; – символ апострофа "’";

&ampquot ; – символ подвійної лапки """.

Зовнішні макровизначення – указують на вміст зовнішнього файла, причому цим вмістом можуть бути як текстові, так і двійкові дані. У першому випадку в місці використання макросу будуть вставлені текстові рядки, в другому – бінарні дані, які аналізатором не розглядаються і використовуються зовнішніми програмами:

<!ENTITY logotype SYSTEM "/image.gif" NDATA GIF87A >

Макровизначення правил – таки макровизначення призначені лише для використовування усередині області DTD і позначаються

спеціальним символом %, що вставляється перед назвою макросу. При цьому вміст компонента буде поміщений безпосередньо в текст DTD-правила

Наприклад, для наступного фрагмента документа:

```
<!ELEMENT name (#PCDATA )>
<!ELEMENT title (#PCDATA | name )*>
<!ELEMENT author (#PCDATA | name )*>
<!ELEMENT article (title, author )*>
<!ELEMENT book (title, author )*>
<!ELEMENT bookstore (book article )*>
<!ELEMENT bookshelf (book article )*>
```

можна використовувати таку форму запису:

```
<!ELEMENT name (#PCDATA )>
<! ENTITY %names '#PCDATA | name '>
<!ELEMENT article (%names;)*>
<!ELEMENT book (%names;)*>
<!ENTITY %content 'book | article '>
<!ELEMENT bookstore (%content;)*>
<!ELEMENT bookshelf (%content;)*>
```

Макровизначення часто використовуються для опису параметрів у правилах атрибутів. У цьому випадку з'являється можливість використовувати однакові визначення атрибутів для різних елементів:

```
<!ENTITY %itemattr 'id ID #IMPLIED src CDATA '>
<!ENTITY %bookattr "ISDN ID #IMPLIED type CDATA '>
<!ENTITY %artattr " size CDATA '>
<!ELEMENT book (title, author,content )*>
<!ATTLIST book %itemattr %bookattr;>
<!ELEMENT article (title, author, content )*>
<!ATTLIST article %itemattr %artattr;>
```

Необхідно зазначити, що DTD надає дуже зручний механізм здійснення контролю за вмістом документа. На сьогоднішній день практично всі програми для обробки документів у мережі Інтернет використовують DTD-правила.

2.8. Перевірка валідності XML-документів

Розглянуті вищі визначення (правила) DTD задають структуру і дані XML-документів. Якщо формально вони не задані, то для аналізу можна застосовувати лише спеціально розроблене програмне забезпечення, в якому враховані ці (фактичні) правила, тобто документи прив'язані до конкретних програм.

За наявності DTD для перевірки можуть використовуватися універсальні програми-аналізатори. Крім того, різні застосування можуть спільно використовувати DTD для забезпечення єдиного погляду на

XML-документ. У цьому випадку визначення повинні зберігатися окремо і бути доступними.

Аналізатори вбудовані в більшість браузерів. Проте браузери, подібно Internet Explorer, вимагають, щоб XML-документи були формально коректними, але не обов'язково дійсними (валідними). Вони не потребують обов'язкової наявності визначень DTD, але якщо ті присутні, то можуть бути використані в процесі перевірки XML-документа. Існують і спеціальні модулі перевірки валідності документів.

Модулі перевірки валідності – це пакети, які перевіряють XML-код і виводять інформацію про результати перевірки.

 Перелік деяких програм, які можна знайти в Інтернеті:

http:// validator.w3.org/ – офіційний модуль перевірки HTML-коду консорціуму W3C;

www.w3.org/People/Raggett/tidy/ – Tidy, утиліта для очищення і відновлення web-сторінок, що включають обмежену підтримку XML;

http://www.xml.eom/pub/a/tools/ruwf/check.html – модуль перевірки дійсності XML-коду групи XML.COM, заснований на процесорі Lark;

www.ltg.ed.ac.uk/~richard/xml-check.html – модуль перевірки XML-коду групи Language Technology Group Едінбурзького університету, заснований на синтаксичному аналізаторі RXP;

http://www.stg.brown.edu/service/xmlvalid/ – модуль перевірки дійсності XML-коду групи Scholarly Technology Group при університеті Брауна.

Модуль перевірки валідності входить у більшість програм для роботи з XML, зокрема, в редактор XML Copy Editor та інші.

Браузер IE також проводить синтаксичний розбір і перевірку валідності XML-документів. Проте результати не виводить. Їх можна вивести, звернувшись до відповідного об'єкта із нескладного скрипта.

Приклад 8. Для перевірки валідності XML-документа за допомогою браузера Internet Explorer можна скористатися такою HTML-сторінкою:

```
 <HTML>
 <HEAD>
 <TITLE>Validity Tester</TITLE>
 <script LANGUAGE="JavaScript">
function pr()
 {
 xmlDoc = dsoTest.XMLDocument;
 if (xmlDoc.readyState == 4) DisplayError();
 else xmlDoc.onreadystatechange = DisplayError();
 }
 function DisplayError ()
 {
 if (xmlDoc.readyState != 4) return;
 message = "parseError.errorCode:"
```

```
 + xmlDoc.parseError.errorCode + "\n"
 + "parseError.filepos:"
 + xmlDoc.parseError.fitepos + "\n"
 + "parseError.line:"
 + xmlDoc.parseError.line + "\n"
 + "parseError.linepos:"
 + xmlDoc.parseError.linepos + "\n"
 + "parseError.reason:"
 + xmlDoc.parseError.reason + "\n"
 + "parseError.srcText:"
 + xmlDoc.parseError.srcText + "\n"
 + "parseError.url:"
 + xmlDoc.parseError.url;
 alert (message);
 }
 </script>
 </HEAD>
 <BODY onLoad="pr();">
 <!--Встановити значення SRC, рівне URL XML-документа, який треба
перевірити -->
 <XML ID="dsoTest" SRC="pr3.xml"></XML>
<h1>для перевіркі валідності записати ім'я файла і натиснути "Обновить" у
браузері</h1>
 </BODY>
 </HTML>
```

У теги XML цього документа потрібно встановити значенням атрибута SRC ім'я файла, який перевернемо. Після завантаження сторінки буде виконана функція **pr()**, яка виведе у вікні результат перевірці (рис. 15).

Рис. 15. **Вікно з результатом перевірці валідності**

Висновки та узагальнення

1. Показати XML-документ у вікні браузера просто, достатньо створити для нього звичайну таблицю стилів. Однак цьому способу

притаманний ряд недоліків. Зокрема неможливо відобразити малюнки та посилання.

2. Значно більш можливостей надає спосіб, який запропонувала Microsoft, пов'язувати XML-документи з HTML-сторінками. Однак його підтримують не всі браузери. Досить універсальним можна вважати відображення за допомогою скриптів.

3. Важливою перевагою технології XML є можливість перевірки документа на відповідність формальним критеріям відсутності помилок. Існують два критерії для перевірки XML-документа: валідність (дієвість) і коректність (правильність). Для перевірки валідности застосовують оголошення DTD або схеми XSD.

4. Створити і перевірити оголошення DTD можна за допомогою різних засобів, у тому числі XML-редакторів і браузерів.

Теоретичні запитання

1. Сформулюйте правила, що визначають коректність XML-документа.

2. Дайте визначення поняття "валідний документ".

3. Якими засобами можна перевірити валідність і коректність документа?

4. Що таке DTD, з чого вони складаються?

5. Як оголошуються типи елементів у DTD?

6. Що задають такі оголошення?

```
<!ELEMENT name (#PCDATA)>
<!ELEMENT title (#PCDATA | name)*>
<!ELEMENT author (#PCDATA | name)*>
<!ELEMENT article (title, author)*>
<!ELEMENT book (title, author)*>
<!ELEMENT bookstore (book | article)*>
<!ELEMENT bookshelf (book | article)*>
```

7. Як оголошуються припустимі атрибути елементів у DTD?

8. Що задає таке оголошення **<!ATTLIST term tC CDATA #REQUIRED>**?

Комплекс задач і завдань

1. Записати правило для елемента, що містить один з тегів (params або fault).

2. Записати правило для елемента, який може містити координати точки в різних системах (прямокутна та полярна) і в довільному порядку.

3. Записати правило для елемента, який може містити дані і довільну кількість тегів term (з температурою хворого).

4. Створити таблицю стилів для документів, які були створені за завданням 1 і 2 розділу 1. Перевірити у різних браузерах.

5. Створити сторінку для зв'язування з документами, які були створені за завданням 1 і 2 розділу 1. Перевірити відображення у різних браузерах.

6. Поповнити документи, створені за завданням 1 і 2 розділу 1 оголошеннями DTD.

7. Виконати перевірку валідності створених документів різними засобами (браузер, XML-редактор, модулем з Інтернету).

3. Застосування XML

Основна ідея розділу

Розділ присвячено питанням використання мови XHTML, її особливостям і перевагам.

Цілі вивчення розділу

Метою розділу є розгляд відмінностей синтаксису мов розмітки HTML і XHTML, особливостей застосування, а також нових можливостей для сервісу WWW.

Матеріал, викладений у розділі, надає студенту можливість набути такі ***компетентності***.

Знання.

Правила створення документів XHTML.

Нові можливості, що надаються мовою XHTML.

Уміння.

Робити обґрунтований вибір мови для створення ресурсів мережі Інтернет.

Перетворювати HTML-документи на XHTML.

Комунікація.

Аргументоване переконання опонентів в необхідності переходу на технології, засновані на XML.

Автономність і відповідальність.

Відображення сторінок як документів XHTML, незалежно від типу і режиму браузера.

Самостійний пошук і використання засобів для перетворення документів.

3.1. Мова XHTML

XML-додаток – це використання мови XML для опису документів з конкретної предметної області, тобто створення мови для конкретного використання. Таких мов створено досить багато (п.п. 1.1.6). Найбільш відомою є, безсумнівно, XHTML (EXtensible HyperText Markup Language, розширена мова розмітки гіпертексту), призначена замінити HTML для сервісу WWW [8; 10]. Взагалі, W3C визначав XHTML як останню версію HTML, яка поступово його витіснить [17]. Хоча поки що цього не відбулося (більш того, готується специфікація HTML 5.0). Особливості використання цієї мови і буде розглянуто.XHTML, зберігаючи всі особливості HTML, вносить більш суворі правила створення сторінок. Це дозволяє робити сайти незалежними від пристрою відображення і браузера. Іншими словами, сайт повинен коректно показуватися у всіх сучасних браузерах і платформах (комп'ютерів, смартфонів і так далі).

Перевагами, що декларуються при переході на XHTML, є:

можливість перевірки коректності і валідності сторінок перед їх використанням;

можливість додавати теги розробниками сторінок;

можливість відображення XHTML-сторінок сучасними браузерами без яких-небудь змін (XHTML – це свого роду міст між XML і HTML, оскільки XML не інтерпретується у браузері).

XHTML розвивається, існують версії 1.0, 1.1, 2.0.

XHTML-документи розміщуються у файлах з розширенням html.

Розглянемо особливості цих документів.

Усі елементи мови XHTML (за винятком елемента <!DOCTYPE>) записуються символами нижнього регістра.

Пролог XHTML-документа може виглядати так:

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1-transitional.dtd">
```

<?xml version="1.0"?> вказує на те, що це XML-документ.

Елемент !DOCTYPE повідомляє, що кореневим елементом документа є <html>. Далі йде формальний публічний ідентифікатор (FPI, formal public identifier) для оголошень DTD (один з багатьох доступних DTD), а також URL для DTD. Використовуючи визначення XHTML DTD, браузери можуть перевіряти валідность XHTML-документів (принаймні, теоретично). Можна зберігати різні визначення XHTML DTD на локальному диску, що дозволить забезпечити швидкий доступ до них, без завантаження з Інтернету.

Найбільш поширені на сьогодні мови HTML 4.01 і XHTML мають по три DTD: Strict, Transitional і Frameset.

Strict – дозволяє використовувати в документі лише ті теги і атрибути, які в цьому DTD описані.

Transitional – містить все те ж саме, що і Strict, але додатково допускає використання "застарілих" тегів, наприклад, <FONT> або <CENTER>.

Frameset – це те ж саме, що і Transitional, лише замість тега BODY для позначення тіла документа в ньому використовується декілька тег FRAMESET. Як нескладно здогадатися, Frameset DTD використовується при створенні сторінок, що містять фрейми.

Головна відмінність в обробці браузером документів XHTML і HTML полягає в тому, що XHTML-документи обробляються своїм модулем (парсером) аналогічно документам XML. У процесі цієї обробки помилки, допущені розробниками, не виправляються (констатується фатальна помилка). Браузер вибирає модуль для обробки документа на підставі заголовка content-type, отриманого від сервера (а не декларації DOCTYPE). Для HTML це – "text/html", а для XHTML – "application /xhtml+xml". При локальному перегляді на клієнті вибір ґрунтується на розширенні файла і можливості декларації DOCTYPE.

Для того, щоб визначити, в якому режимі відображається сторінка, можна використовувати такій Javascript :

```
<script language="JavaScript " type="text/javascript">
 document.write(document.compatMode);
</script>
```

Якщо браузер працює в режимі підтримки стандарту, то на сторінці буде написано "CSS1Compat ", а якщо в режимі "зворотної сумісності", то "BackCompat " або "QuirksMode ".

Після елемента <!DOCTYPE> знаходиться елемент <html>, який є початком фактичного вмісту документа:

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en " lang="en ">

Він може містити три атрибути:

xmlns визначає простір імен XML для даного документа;

xml:lang встановлює мову документа, яка використана в процесі інтерпретації XML-коду;

lang застосовується для установки мови документа, що трактуватимемо у вигляді HTML-коду.

Елемент <html> може включати елементи <head> і <body> (або елементи <head> і <frameset> у документі, створеному за допомогою версії XHTML 1.0 Frameset).

3.2. Зв’язок між HTML і XHTML

При розробці XHTML ставилося завдання відображення його елементів сучасними браузерами, і ця мета була досягнута за рахунок використання однойменних тегів (за деякими виключеннями) і того, що сучасні браузери ігнорують незрозумілі ним елементи (наприклад, <?xml…?> або <!DOCTYPE…>). Проте, відмінностями є:

усі значення атрибутів обов’язково потрібно взяти в лапки;

не підтримуються нестандартні HTML-елементи ;

імена елементів і атрибутів записуються символами нижнього регістра;

завершувати теги є обов’язковим (порожні елементи повинні завершуватися символами "/>");

не допускається застосування атрибутів, яким не привласнені значення;

елемент <a> не може включати інші елементи <a>.

елемент <pre> не може включати елементи <img>, <object>, <big>, <small>, <sub> або <sup>;

елемент <button> не може містити елементи <input>, <select>, <textarea>,<label>, <button>, <form>, <fieldset>, <iframe> або <isindex>;

елемент <label> не може містити інші елементи <label>;

елемент <form> не може включати інші елементи <form>;

для ідентифікації використовується атрибут id (name не допускається).

Зв’язування із зовнішньою таблицею стилів може виконуватися як для HTML (елемент <link …>), так і XML (<?xml-stylesheet …"?>). Якщо XHTML-документ трактується як XML-код (тобто використовується розширення .xml), то для вказівки необхідної таблиці стилів у пролог слід додати XML-інструкцію з обробки <?xml-stylesheet type="text/css" href= "…"?>

Можна створювати варіанти вбудованого стильового оформлення за допомогою атрибута style. Таким чином, особливих відмінностей у використанні стилів немає.

Елемент <script> застосовується для вкладення сценарію у XHTML-документ, як це виконується в HTML. Зазвичай елемент розміщується у секції документа <head>. Атрибути ті ж (defer, language, src, type та ін.). При використанні скриптів, якщо елементи script містить спеціальні символи, то їх вміст слід укладати в марковану секцію CDATA, інакше процесор XML перетворює спеціальні символи в їх еквіваленти до обробки браузером сценарію:

```
<script>
 <![CDATA[ …зміст сценарію… ]
 ]>
</script>
```

Однак, якщо документ буде інтерпретовано як HTML, то секція буде пропущена. Тому краще за все розмістити скрипт у окремому файлі.

Документи HTML можуть бути автоматично перетворені в XHTML з використанням спеціальних програм (Tidy та ін.) або XSLT (п.п. 4.2).

3.3. Розширення XHTML, створення нових атрибутів і елементів

Оскільки XHTML – це XML-додаток, то він може бути розширений шляхом включення нових елементів (тегів) і атрибутів.

Для використання у створюваних документах власних тегів необхідно визначити порядок їх відображення браузером і забезпечити можливість перевірки валідності тими засобами, які використовуватимуть створювані документи (браузери ігнорують результати перевірки валідності).

Визначення того, як відображати нові елементи, можна забезпечити за допомогою таблиць стилів або скриптів (у тому випадку, коли не вистачає можливостей CSS).

Наприклад, для включення у документ власного елемента underl_redt, припускаючи, що це буде підкреслений текст червоного кольору, достатньо включити тег у розмітку і помістити в таблицю стилів відповідне правило:

underl_redt {display:inline;color:red;text-decoration:underline}.

Такий документ у файлі з розширенням .htm буде правильно відображений всіма сучасними браузерами:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
<link href="style.css" rel="stylesheet" type="text/css" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Приклад XHTML-документа</title>
</head>
<body>
<div class="c1" id="i1">Приклад XHTML-документа</div>
<p class="b1">Абзац синього кольору</p>
<div><img src="yoga.jpg" width="237" height="242" alt="eee" /></div>
 <underl_redt>Подчеркнутый текст красного цвета</underl_redt>
</body>
</html>
```

Зазначимо, що якщо прибрати з документа оголошення типу документа (<!DOCTYPE…>), то браузер сприйме його як HTML і проігнорує новий тег.

Оскільки XHTML є правильним документом XML, він може бути доповнений інструкцією <?xml…?>, тег link замінений на <?xml-stylesheet type="text/css" href="…" ?> і файла дано розширення .xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<?xml-stylesheet type="text/css" href="style.css" ?>
<title>XHTML</title>
</head>
<body>
<div class="c1" id="i1">XHTML-</div>
<div><img src="yoga.jpg" width="237" height="242" alt="eee" /></div>
<p class="b1">hh</p>
 <underl_redt>hh</underl_redt>
</body>
</html>
```

У цьому випадку він також правильно відобразиться всіма браузерами.

Обидва розглянуті документи не є валідними, оскільки доданий невідомий тег. Браузери відображають їх, ігноруючи результати перевірки (швидше за все, зовсім не перевіряючи). Проте, при обміні такими документами з іншими засобами, можливо, що невалідні документи не будуть оброблені. Щоб зробити їх валідними, необхідно доповнити оголошення DTD описом нових елементів:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd" [
<!ELEMENT underl_redt (#PCDATA)>
<!ATTLIST underl_redt text_attrib CDATA #IMPLIED>
<!ENTITY % Misc.extra "| underl_redt">
]>
```

Згідно з цими доповненнями додано елемент underl_redt, який може містити необов’язковий атрибут і символьні дані.

Обидва документи будуть повністю валідними (у чому неважко переконатися, перевіривши їх, наприклад, на сайті консорціуму W3C http://validator.w3.org/), і обидва розпізнаються як XHTML 1.1.

Висновки та узагальнення

1. Стандарт XHTML був створений як перехідний між HTML і XML. Тому обидві мови містять однакові елементи.

2. XHTML-документи можуть інтерпретуватися як XML з перевіркою коректності і валідності, так і як HTML, тобто звичайним чином. Відповідний режим встановлюється браузером з урахуванням повідомлення сервером типу і елемента DOCTYPE.

3. У синтаксисі XHTML є особливості, але на практиці при створенні документів за допомогою відповідних технологічних засобів вони враховуються автоматично.

4. Документи HTML можуть бути автоматично перетворені в XHTML.

Теоретичні запитання

1. Сформулюйте основні правила запису XHTML-документів.

2. У чому є відмінності DTD Strict і Transitional?

3. Сформулюйте основні переваги використання мови XHTML.

4. Чим відрізняється обробка в браузері HTML-документів і XHTML-документів?

5. Що таке режим "зворотної сумісності" браузера?

6. Чи може XHTML-документ мати розширення .xml?

7. Що необхідно зробити для включення в XHTML-документ нових тегів?

8. Як можна перетворити HTML-документ у XHTML-документ?

Комплекс задач і завдань

1. Перетворити на XHTML документ з прикладу 5.

2. Перетворити на XHTML документ з прикладу 6.

3. Перетворити на XHTML документ з прикладу 7.

4. Перетворити на XHTML документ з прикладу 8.

5. Доповніть документ, створений у завданні 6, новим тегом, перевірити валідність.

4. Обробка XML-документів

Основна ідея розділу

У попередніх розділах були розглянуті основні питання XML-розмітки, перевірки коректності і валідності, відображення розмічених документів у вікні браузера. При цьому методи, які використовувались, були прив'язані до вже знайомих технологій (таблиці стилів CSS, програмування на стороні клієнта JScript) і рішень Microsoft (модель DSO). У цьому розділі будуть розглянуті нові аспекти (компоненти) технології, що забезпечують ефективність використання XML для взаємодії додатків. Це технології на основі XSL.

Цілі вивчення розділу

Метою розділу є розгляд питань, які пов'язані з новими додатками XML, призначеними суттєво розширити можливості з обробки і використання XML-документів. Буде розглянута мова XSL (Extensible Styleheet Language, мова розширених таблиць стилів).

Матеріал, викладений в розділі, надає студенту можливість набути такі ***компетентності***.

Знання.

Загальна концепція використання розширеної мови таблиць стилів XSL.

Мова опису трансформацій XSLT.

Форматуючі об'єкти XSL-FO.

Уміння.

Створення XSLT для перетворень і обробки XML-документів.

Створення найпростіших XSL-FO документів.

Перетворення XSL-FO документів в інші формати.

Комунікація.

Уміння переконати фахівців галузі в доцільності та ефективності використання нових технологій.

Автономність і відповідальність.

Самостійний пошук і використання всіх доступних засобів для роботи з XSL для отримання кінцевого результату.

4.1. Таблиці стилів XSL

У XML користувач сам створює свої елементи. Це означає, що для відображення цих елементів слід вказати браузеру, як це зробити. Для цього запропоновано використовувати специфікації каскадних таблиць

стилів (Cascading Style Sheet, CSS). Технологія CSS зазвичай використовується в HTML-документах, і широко підтримується в браузерах. За допомогою таблиць CSS можна визначити форматування, як окремих елементів так і різних груп. Усе це може бути застосовано і до XML-документів. Проте у рамках розвитку технологій на основі XML були запропоновані рішення, які істотно розширяють можливості з обробки і перетворення документів. Це розширювана мова таблиць стилів (Extensible Styleheet Language, XSL).

Таблиця CSS лише встановлює формат і розміщення елементів, а XSL дозволяє значно більше. Можна переупорядкувати елементи в документі, змінити їх повністю, відобразити одні елементи та приховати інші, вибрати стилі, ґрунтуючись не тільки на елементах, але також і на їх атрибутах. Таким чином, розширювана мова таблиць стилів є засобом перетворення XML-документів.

У даний час пропрацьовано два підходи, що розширюють можливості з обробки і відображення XML-документів: XSL-перетворення (XSL Transformations, XSLT) і форматуючі XSL-об’єкти (XSL Formatting Objects, XSL-FO). Обидва підходи базуються на XML (є його додатками) і забезпечують відображення даних XML-документа.

XSLT – це XML-додаток, що визначає правила, відповідно до яких один XML-документ перетвориться в іншій. XSLT-документ, тобто трансформуюча таблиця стилів XSLT, містить шаблони (опис форматування). XSLT-процесор порівнює елементи вхідного XML-документа із шаблонами в таблиці стилів. Коли відповідний шаблон знайдений, процесор записує вміст шаблону у вихідне дерево. Після закінчення цього етапу вихідне дерево записується або в XML-документ, або в інший формат, такий як звичайний текст або HTML.

Таким чином, XSLT застосовується для обробки документів (фільтрація даних, внесення додаткової розмітки і тому подібне). Ця мова забезпечує доступ до вмісту XML-документів, а також позволяє створювати на їх основі нові документи. Одне дерево перетвориться в інше, одна розмітка – в іншу. Документ може бути перетворений в інший формат (HTML, спеціальна розмітка, звичайний текст).

Мова XSL-FO призначена для опису зовнішнього вигляду документа. На противагу CSS, XSL-FO – це XML-додаток, призначений для опису точного розміщення тексту на сторінці. Це інструмент верстки. У ньому є елементи для представлення сторінок, блоків тексту на сторінці, графіки, горизонтальних лінійок і тому подібне. Додаток, що

відображає документ, читає XSL-FO і показує його користувачеві. Форматуючі об’єкти – це набір об’єктів (root, block та ін.), властивості яких дозволяють визначити різні деталі відображення (позиціонування, шрифти і так далі). Кожному з об’єктів відповідає тег XML, властивості – це атрибути тега. Опис великих документів за допомогою форматуючих об’єктів є достатньо складною. У зв’язку з цим використовується підхід, при якому користувач створює документ, застосовуючи власну розмітку, потім використовує XSLT для перетворення у форму, що передбачає застосування форматуючих об’єктів. Створений таким чином документ передається програмі, яка оброблює форматуючі об’єкти і виводить на екран результат.

Розгляд цих питань почнемо із створення і використання XSLT.

Після того, як таблиця XSLT задана, можна перетворити початковий документ за допомогою XSLT-процесора. XSLT-процесор – це програма, яка читає таблицю стилів XSLT, читає вхідний XML-документ і перетворить вхідний документ у вихідний відповідно до інструкцій, даних в таблиці стилів (рис. 16).

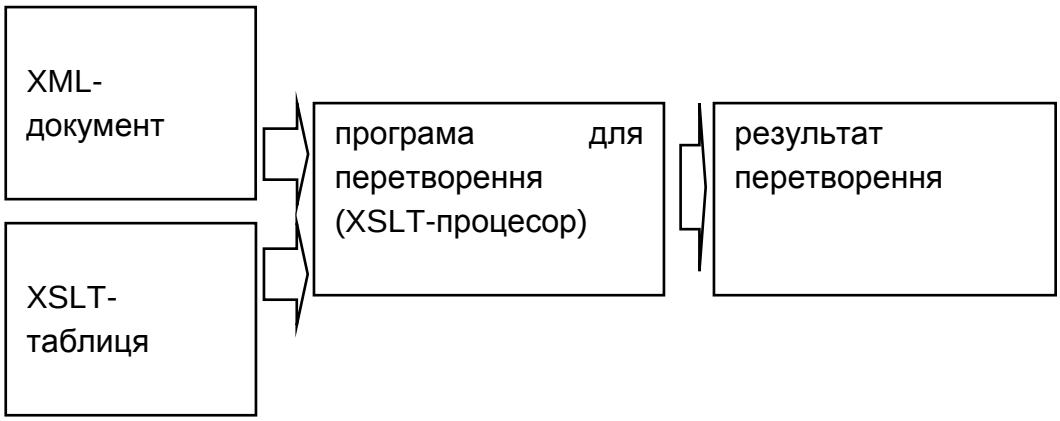

Рис. 16. **Використання XSLT**

XSLT-процесор може бути вбудованим у браузер, у web-сервер або в сервер додатків, а може бути окремою програмою.

Достатньо зручно використовувати web-браузер і перетворювачі, вбудовані в XML-редактори.

4.2. Таблиці стилів XSLT

Таблиця XSLT є коректно сформованим XML-документом, має XML-оголошення, може містити оголошення типу документа (DTD), хоча

у більшості таблиць стилів його немає. Кореневим елементом цього документа є або stylesheet, або transform. Це синоніми, між ними немає жодної різниці, окрім імені. Вони обидва мають однаковий набір можливих дочірніх елементів і атрибутів, і обидва означають для XSLT-процесора одне і те ж.

Елементи stylesheet і transform, як і решта всіх елементів XSLT, знаходяться у деякому просторі імен, його опис доступний у Інтернеті, наприклад, http://www.w3.org/1999/XSL/Transform. Простір імен зазвичай позначається префіксом xsl, xsl:transform або xsl:stylesheet. Таким чином, XSLT-документ може виглядати так:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!-- один або декілька елементів шаблону...-->
</xsl:stylesheet >
```

Елемент **xsl:stylesheet** ідентифікує документ як XSL-таблицю стилів і служить контейнером для інших елементів. У ньому також визначається простір імен xsl. Усі імена, що мають відношення до таблиці повинні мати префікс "xsl:".

 Для таблиць XSL є декілька варіантів синтаксичних конструкцій, представлених в різних просторах імен:

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">

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

Перше вважається за застаріле, але може зустрітися. Це необхідно враховувати при розробці і аналізі XML-документів. Ми розглядатимемо 1999/XSL/Transform.

Файли, що містять XSLT, мають розширення xsl. XML-документи, що підлягають обробці з використанням таблиць стилів, повинні мати в пролозі інструкцію обробки xml-stylesheet, яка указує, де знаходиться відповідна документу таблиця стилів. Якщо це таблиця стилів XSLT, то атрибут type повинен мати значення text/xml (на відміну від text/css для CSS). Наприклад,

```
<?xml-stylesheet type="text/xml" href="http://www.ore.com/styles/people.xsl"?>
```

Ця інструкція говорить про те, що слід застосовувати таблицю стилів, що знаходиться за абсолютною URL-адресою. Значенням атрибута href може бути і відносна адреса.

Для управління вихідними даними, що формуються на основі початкового XML-документа, в таблицю стилів XSLT включають шаблони. Кожен шаблон представлений елементом xsl:template. Цей елемент має атрибут match, який вказує на певну гілку дерева структури

документа (атрибут match аналогічний селектору в правилі CSS). Значення атрибута match носить назву зразка (pattern). Наприклад, шаблон, який містить інструкції для відображення всього XML-документа:

```
<xsl:template match="/">
<!-- дочірні елементи... -->
</xsl:template >
```

Зразок у даному шаблоні (символ "/") представляє весь XML-документ.

Інший простий зразок – це ім’я елемента, наприклад, <xsl:template match="person">Людина</xsl:template>. Даний шаблон говорить про те, що кожного разу, коли зустрічається елемент person, процесор таблиць стилів повинен генерувати текст "Людина".

Складніші конструкції для зв’язування шаблону з гілкою дерева записуються за допомогою спеціальної мови XPath.

Кожна XSL-таблиця стилів повинна містити тільки один шаблон з із атрибутом match, який має значення "/". Крім того, можна також включити один або декілька додаткових шаблонів з інструкціями для відображення певних підлеглих гілок у структурі XML-документа.

За замовчуванням XSLT процесор читає вхідний XML-документ від початку до кінця, починаючи із кореня документа і рухаючись вниз по дереву. Для кожного рівня перевіряється наявність шаблону, якщо він виявлений, то активізується для кожного знайденого вузла і на цьому робота закінчується. Шаблон, відповідний батьківському елемента, активізується раніше шаблонів, відповідних дочірнім елементам. Тому за наявності шаблону батьківського документа обхід закінчується без активації шаблонів дочірніх елементів. Таким чином, за наявності декількох шаблонів активізується лише шаблон самого старшого батька. Для активізації останніх слід використовувати спеціальні елементи.

Шаблон може містити послідовність елементів двох видів:

XML-елементи, що представляють незмінну коректну розмітку (наприклад, HTML-розмітку, а може бути яку-небудь іншу, в яку виконується перетворення);

XSL-елементи, які задають порядок використання даних із початкового документа (інструкції по вибору і модифікації даних XML).

XSL-таблиця може виконувати на основі шаблонів перетворення в розмітку (XML-документ) будь-якого вигляду, у тому числі і в HTML. Для XML-документів, що перетворюються в HTML-розмітку, XSL-таблиця стилів повідомляє браузеру (або іншій програмі, що виконує перетво-

рення), як відобразити XML-документ шляхом його вибіркового перетворення в блоки HTML-розмітки.

Приклад вставки незмінної розмітки:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
        <HTML >
        <HEAD >
        <TITLE>pr10</TITLE>
        </HEAD >
        <BODY >
        <H2>Результат перетворення</H2>
        </BODY >
        </HTML >
 </xsl:template >
</xsl:stylesheet >
```

Якщо відкрити в браузері XML-документ з посиланням на таку XSLT, то буде відображений HTML-документ, що міститься в шаблоні.

Для зміни порядку використання даних і обробки існують спеціальні елементи (елементи трансформації). Програми обробки відрізняють ці елементи по префіксу xsl. Основні елементи для перетворення (всього їх близько 30) наведені в табл. 11.

Таблиця 11

Елементи трансформації

Елемент	Призначення	Атрибути
<xsl:template>	опис шаблону	match
<xsl:apply-templates>	вставка даних за шаблоном	select
<xsl:value-of>	вставка значень з елемента	select
<xsl:for-each>	повторення обробки	select, order-by
<xsl:sort>	сортування	select, order
<xsl:attribute>	установка значень атрибутів	name

Вказівка на певний XML-елемент, з яким виконуватимуться перетворення, проводиться завданням значення атрибута select. Це може бути шлях по дереву або вираз XPath. При завданні шляху проводиться орієнтація на уявлення структури документа у вигляді дерева ("/" – кореневий елемент), з урахуванням поточного місцеположення, що задається атрибутом match в xsl:template або select в xsl:for-each. Можна використовувати спеціальні символи ("**.**" і " ***** "). Крапка – поточний елемент, зірочка – будь-який елемент.

Якщо дані знаходяться в атрибутах, то посилання на них виглядає так *путь_к_элементу/@имя_атрибута*. Наприклад, якби в тегові STUDENT з прикладу 1 був атрибут GR (<*STUDENT GR='1979'*>. </*STUDENT*>), то посилання на нього могло б бути таким *select='DOCUMENT/STUDENT/@GR'*.

4.2.1. Використання XML для перетворення форматів

Розглянемо вживання конструкцій XSLT для перетворення XML-документа в HTML. Скористаємося документом з прикладу 1. Для зручності розгляду зобразимо структуру даних, що містяться XML-фалі у вигляді дерева (рис. 17).

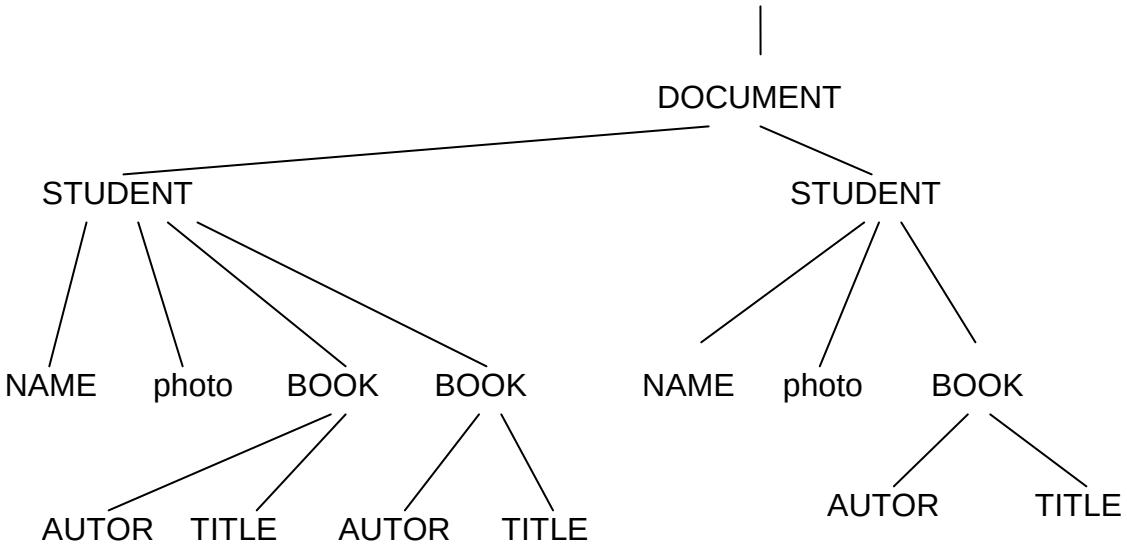

Рис. 17. **Структура даних XML-документа**

Спочатку задамо шаблони і елементи apply-templates, які забезпечать обхід всіх вузлів STUDENT і BOOK :

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
        <HTML>
        <HEAD>
        <TITLE>pr1</TITLE>
        </HEAD>
        <BODY>
        <H2>Список студентов, не сдавших книги</H2>
        <xsl:apply-templates select="DOCUMENT/STUDENT"/>
        </BODY>
        </HTML>
 </xsl:template>
 <xsl:template match="STUDENT">
        <xsl:apply-templates select="BOOK"/>
```

```
 </xsl:template>
 <xsl:template match="BOOK">
        </xsl:template>
</xsl:stylesheet>
```

Незмінна розмітка, яка відповідає початку HTML-документа (до тега BODY), поміщена в шаблон з атрибутом match="/". Далі розміщуються шаблони для відображення даних зі всіх вузлів документа STUDENT і BOOK. Шаблон з match="/" активізується автоматично. Для активізації інших в шаблони батьківських елементів включений елемент apply-templates. Він забезпечує активізацію і вставку даних відповідно до вказаного шаблону.

Ця таблиця стилів активізує шаблони для всіх вузлів документа. Проте браузер відобразить лише незмінну розмітку, інших даних у шаблонах немає.

Для вставки текстового вмісту тегів із початкового документа використовується елемент value-of. Атрибут select задає елемент початкового документа, з якого береться значення. Наступний фрагмент забезпечить відображення текстових даних:

```
 <xsl:template match="STUDENT ">
        <SPAN><xsl:value-of select="NAME"/></SPAN>
        <BR/>
        <xsl:apply-templates select="BOOK"/>
 </xsl:template >
 <xsl:template match="BOOK ">
        <SPAN><xsl:value-of select="AUTOR"/></SPAN>
        <SPAN><xsl:value-of select="TITLE"/></SPAN><BR/>
 </xsl:template >
</xsl:stylesheet >
```

Процесор вставляє текстову частину кожного тега, включаючи вміст вкладених (дочірніх) тегів, на місце елемента value-of.

Для перебору всіх вузлів заданого рівня можна використовувати не тільки активізацію шаблонів, а і спеціальний елемент for-each. Він повторює обробку, яка визначена усередині для кожного вузла вибраного рівня одного батька <xsl:for-each select= " ">... </xsl:for-each>.

Це аналог оператора циклу в мовах програмування:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
 <HTML>
 <HEAD>
 <TITLE>pr1</TITLE>
 </HEAD>
 <BODY>
```

```
 <H2>Список студентов, не сдавших книги</H2>
<xsl:for-each select="DOCUMENT/STUDENT">
 <H3><xsl:value-of select="NAME"/></H3>
</xsl:for-each>
 </BODY>
</HTML>
 </xsl:template>
</xsl:stylesheet>
```

За допомогою цієї таблиці буде виведено вміст тегів NAME зі всіх тегів STUDENT.

Елементи xsl:for-each можуть бути вкладеними (вкладені цикли):

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
 <HTML>
 <HEAD>
 <TITLE>pr1</TITLE>
 </HEAD>
 <BODY>
 <H2>Список студентов, не сдавших книги</H2>
<xsl:for-each select="DOCUMENT/STUDENT">
  <xsl:for-each select="BOOK">
   <H3><xsl:value-of select="TITLE"/></H3>
  </xsl:for-each>
</xsl:for-each>
 </BODY>
</HTML>
 </xsl:template>
</xsl:stylesheet>
```

Будуть виведені тексти зі всіх тегів TITLE.

Елемент xsl:attribute позволяє додати атрибут до елемента результуючого дерева. Має обов’язковий атрибут name, який задає ім’я атрибута, що створюється цією інструкцією. Вміст цього елемента є шаблоном (або елементом шаблону). Результатом виклику цього шаблону має бути простий текст. За допомогою цього елемента можна сформувати елемент із заданими атрибутами, наприклад, тег IMG :

```
<IMG >
<xsl:attribute name="src"><xsl:value-of select="photo"/></xsl:attribute>
</IMG >
```

Розглянуті елементи дозволяють перетворювати XML-документи в HTML для подальшого відображення у браузері.

Приклад 9. Розглянемо таблицю стилів для перетворення документа з прикладу 1 (рис. 17) у формат HTML.

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

```
 <xsl:template match="/">
<HTML>
<HEAD>
<TITLE>pr1</TITLE>
</HEAD>
<BODY>
<H2>Список студентів, які не здали книги</H2>
<xsl:apply-templates select="DOCUMENT/STUDENT"/>
</BODY>
</HTML>
 </xsl:template>
 <xsl:template match="STUDENT">
       <IMG>
       <xsl:attribute name="src"><xsl:value-of select="photo"/></xsl:attribute>
       </IMG>
       <SPAN STYLE="font-style:italic">Прізвище: </SPAN>
       <SPAN><xsl:value-of select="NAME"/></SPAN>
       <BR/>
       <xsl:apply-templates select="BOOK"/>
 </xsl:template>
 <xsl:template match="BOOK">
       <SPAN STYLE="font-style:italic">Назва: </SPAN>
       <SPAN><xsl:value-of select="AUTOR"/></SPAN>
       <SPAN><xsl:value-of select="TITLE"/></SPAN><BR/>
 </xsl:template>
</xsl:stylesheet>
```

Вона містить три шаблони (кореневий елемент, STUDENT, BOOK). Шаблон кореневого елемента активізується автоматично, а шаблони вузлів STUDENT і BOOK інструкціями apply-templates у відповідних місцях. Для відображення фотографії використаний елемент <xsl:attribute name="src" >, що привласнює значення атрибута тега. Саме значення міститься в тексті елемента photo.

Слід ще раз зазначити, що наведена таблиця перетворень не є єдино можливою. Технологія достатньо гнучка і допускає різні рішення. Наприклад, аналогічного результату можна досягти за допомогою такого коду:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
<HTML>
<HEAD>
<TITLE>pr1</TITLE>
</HEAD>
<BODY>
<H2>Список студентов, які не здали книги</H2>
   <xsl:for-each select="DOCUMENT/STUDENT">
    <IMG>
       <xsl:attribute name="src"><xsl:value-of select="photo"/></xsl:attribute>
       </IMG>
```

```
      <SPAN STYLE="font-style:italic">Прізвище: </SPAN>
      <SPAN><xsl:value-of select="NAME"/></SPAN>
      <BR/>
   <xsl:for-each select="BOOK">
    <SPAN STYLE="font-style:italic">Назва: </SPAN>
       <SPAN><xsl:value-of select="AUTOR"/></SPAN>
       <SPAN><xsl:value-of select="TITLE"/></SPAN><BR/>
   </xsl:for-each>
  </xsl:for-each>
 </BODY>
</HTML>
  </xsl:template>
</xsl:stylesheet>
```

Тут перебір вузлів з метою витягання даних здійснюється з використанням елементів for-each (циклів) замість звернення до шаблону.

Використання обох таблиць дозволяє отримати однаковий HTML-документ:

```
<?xml version="1.0" encoding="UTF-8"?>
<HTML>
  <HEAD>
    <TITLE>pr1</TITLE>
  </HEAD>
  <BODY>
    <H2>Список студентів, які не здали книги</H2>
    <IMG src="im1.gif"/>
    <SPAN STYLE="font-style:italic">Прізвище: </SPAN>
    <SPAN>Иванов</SPAN>
    <BR/>
    <SPAN STYLE="font-style:italic">Назва: </SPAN>
    <SPAN>Пушкар  </SPAN>
    <SPAN> Основи наукових досліджень</SPAN>
    <BR/>
    <SPAN STYLE="font-style:italic">Назва: </SPAN>
    <SPAN>Молчанов  </SPAN>
    <SPAN> Технології Web-дизайна</SPAN>
    <BR/>
  </BODY>
</HTML>
```

Розглянуто найпростіше перетворення. Окрім цього можна сортувати і фільтрувати дані для виводу, використовувати спеціальні функції і тому подібне.

Переважна більшість браузерів мають підтримку XML і XSLT. Однак при перетворенні в браузері результуючий документ відображається у вікні і отримання його у в окремому файлі без програмування неможливе. Тому для таких дій зручніше користуватися редактором з вбудованим перетворювачем.

4.2.2. XSL-перетворення для сортування

Для сортування у просторі імен Transform використовується спеціальний елемент (інструкція) sort, що розміщується безпосередньо після for-each і apply-templates:

```
<xsl:sort select = "строкове_вираження "
 data-type = "text "або "number "
 lang = "код_мови "
 order = "ascending " або "descending "
 case-order = "upper-first " або "lower-first " />
```

Цей елемент змінює порядок проходження вузлів з порядку, прийнятого в документі, на іншій, наприклад, алфавітний.

Атрибут select – це ключ сортування (ним ідентифікується вузол, за даними якого проводиться сортування). Якщо select відсутній, ключ сортування встановлюється рівним значенню поточного вузла.

data-type – шаблон значення атрибута (як інтерпретувати дані, текст або числові значення). За замовчуванням вважається текст, і виконується сортування за абеткою.

Сортування залежить від мови. Встановивши атрибут lang відповідно до мовних кодів RFC 1766, можна змінити мову сортування.

order – визначає порядок, в якому сортуються рядки (descending – що убуває, ascending – що зростає). За замовчуванням сортування виконується у зростаючому порядку.

Атрибут case-order може бути встановлений або в upper-first, або в lower-first, указуючи, що або спочатку сортуються літери верхнього регістра, а потім нижнього, або навпаки. Значення за замовчуванням залежить від мови.

Наприклад, сортування за абеткою:

```
<xsl:sort order="ascending" select="AUTHOR/LASTNAME">
сортування за значенням
<xsl:sort order="ascending " select="number (PAGES)" data-type="number"/>
```

Можна виконувати сортування за декількома ключами (наприклад, спочатку сортувати за прізвищем, потім за іменем, потім по батькові), використовуючи декілька елементів xsl:sort у порядку убування важливості ключів.

4.2.3. XSL-перетворення для фільтрації

Значення, яке привласнюється атрибутам match або select (match використовується для елемента template, а select – для елементів value-of, for-each і apply-templates), є зразком, відповідним одному або декільком елементам у XML-документі. Можна обмежити кількість

елементів, що відповідають шаблону, ввівши фільтр. Фільтр – це вираз, поміщений в квадратні дужки ([]), який слідує безпосередньо за послідовністю вузлів шляху. Наприклад, зразок, привласнений наступному атрибута match, указує, що відповідний елемент повинен носити ім’я BOOK, і крім того, повинен мати дочірній елемент BINDING, який містить текст "trade paperback":

<xsl:template match="BOOK [BINDING=’trade paperback ’]">

Якщо у фільтр включено тільки ім’я елемента, то відповідний елемент повинен мати дочірній елемент з вказаним ім’ям. Наприклад:

match="ITEM[CD]" – будь-який елемент ITEM, що має дочірній елемент CD, незалежно від вмісту елемента CD;

match="SHIRT[COLOR=’red’]" – будь-який елемент SHIRT, що має дочірній елемент COLOR, що містить текст "red";

select="SHIRT[COLOR!=’red’]" – будь-який елемент SHIRT, що має дочірній елемент COLOR, який не містить текст "red".

Якщо елемент має більш за один дочірній елемент з ім’ям, вказаним в умові фільтрації, оператор порівняння застосовується тільки до першого дочірнього елемента. Наприклад, якщо елемент SHIRT має два дочірні елементи COLOR, зразок "SHIRT[COLOR=’red’]" відповідатиме елемента, тільки якщо перший елемент COLOR містить слово "red".

При необхідності використання не строкових, а числових значень використовується такий синтаксис:

<xsl:for-each select="… [number(nam)$gt;10] " >,

де $gt – підстановка для символу ">" (у полі nam має бути значення більше, ніж 10).

Для виконання фільтрації можна також використовувати елемент <xsl:if test="number(nam)>10">…</xsl:if> Вкладені в цей тег елементи відображаються тільки тоді, якщо обчислена умова істинна.

Приклад 10. Створимо таблицю для фільтрації та сортування даних з прикладу 2 і перетворення їх у формат HTML:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<HTML>
<HEAD>
<TITLE>Пример</TITLE>
</HEAD>
```

```
<BODY>
<H2>Список літератури</H2>
<UL>
  <xsl:for-each select="список/джерело">
   <xsl:sort order="ascending" select="автор"/>
  <xsl:if test="number(рік)&gt;2002">
   <LI><SPAN STYLE="font-style:italic">
     <xsl:value-of select="автор"/>
     <xsl:value-of select="назва"/>
     <xsl:value-of select="видавництво"/>
     <xsl:value-of select="рік"/>
     </SPAN>
     <xsl:value-of select="кіл_сторінок"/></LI>
 </xsl:if>
   </xsl:for-each>
</UL>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

Сортування виконано за абеткою за тегом *автор*, елемент *<xsl:sort order…* Фільтрація – за тегом *год*, елемент *<xsl:if test…* У результаті будуть відображені тільки два джерела, випущені після 2002 року (рис. 18).

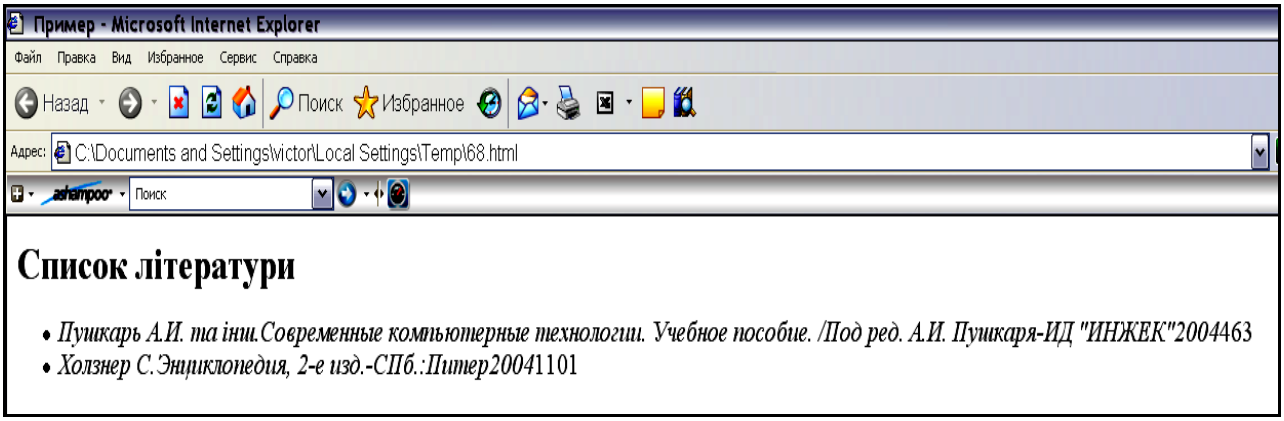

Рис. 18. **Відображення документа в браузері**

4.3. Об’єкти, що форматують

XSL-FO – це XML-додаток, призначеній для опису точного розміщення вмісту на сторінці. Програмне забезпечення, що відображає документ, читає елементи XSL-FO і створює зображення. Проте в даний час жоден з основних браузерів та інше масове ПО безпосередньо не підтримує документи XSL-FO. У зв’язку з цим необхідний додатковий етап, на якому ще один процесор перетворить XSL-FO в інший, більш широко поширений формат, наприклад, такий як PDF або TEX.

На відміну від HTML+CSS, технологія, що використає XSL-FO, більш універсальна, і підтримує детальніше форматування. Таким чином, XSL-FO дозволяє вирішувати ширше коло завдань. Можна, наприклад, написати таблицю стилів XSL, яка за допомогою форматуючих об’єктів згенерує верстку цілої книги.

Прикладом використання такого підходу може служити DocBook. Це технологія розробки документації на основі XML, що дозволяє з єдиного початкового тексту автоматично отримувати вихідні документи в різних форматах (HTML, PDF, HTML Help) і з різними варіантами компоновки і оформлення.

4.3.1. Структура документа XSL-FO

Документ XSL-FO містить макет сторінки – структуровану інформацію, що визначає зовнішній вигляд сторінок результуючого документа. Файли зазвичай мають розширення *.fo*. Ці документи призначені для програм, які перетворять (конвертують) їх в документи кінцевого формату (для конкретних пристроїв, наприклад, PDF для принтера або HTML для браузера).

Імена всіх об’єктів документа знаходяться в просторі імен xmlns:fo="http://www.w3.org/1999/XSL/Format", і мають префікс fo. Простір імен визначається у кореневому елементі fo:root.

Дочірніми елементами fo:root є один елемент fo:layout-master-set і один або декілька елементів fo:page-sequence. Елемент fo:layout-master-set містить шаблони створюваних сторінок, а в елементах fo:page-sequence розміщується основний вміст документа (тексти і зображення). У загальному випадку структура документа виглядає так:

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
<fo:root >
 <fo:layout-master-set >
 <fo:simple-page-master master-name="first ">
 <fo:region-body/>
 </fo:simple-page-master >
 </fo:layout-master-set >
 <fo:page-sequence master-name="first ">
 <!-- дані, що поміщаються на сторінці, ->
 </fo:page-sequence >
</fo:root >
```

При конвертації форматуюча машина прочитує XSL-FO-документ, створює сторінку на основі шаблону з елемента fo:layout-master-set. Потім вона наповнює його вмістом з елемента fo:page-sequence. Коли

перша сторінка заповнена, то на основі шаблону створюється друга сторінка і заповнюється вмістом. Цей процес триває до тих пір, поки не буде вичерпано весь вміст.

Шаблони сторінок називаються мастер-сторінками. Кожен з них задає загальний макет сторінки, включаючи її відступи, розміри заголовків, футери, основну область сторінки і так далі. Кожна реальна сторінка документа, що виводиться, яка створюється на основі мастер-сторінки, успадковує її властивості, наприклад, відступи, нумерацію сторінок і загальний макет. У XSL-FO 1.0 визначається тільки один вид мастер-сторінок, а саме fo:simple-page-master, який задає прямокутну сторінку. Елемент fo:layout-master-set може містити декілька елементів fo:simple-page-master, завдяки цьому забезпечується можливість використання декількох шаблонів (один – для титульної сторінки, інший – для основних сторінок, третій – для вихідних даних і тому подібне).

Кожен елемент fo:simple-page-master задає загальний макет сторінки, включаючи розмір областей: before region, body region, after region, end region і start region. На рис. 19 наведено розташування цих областей.

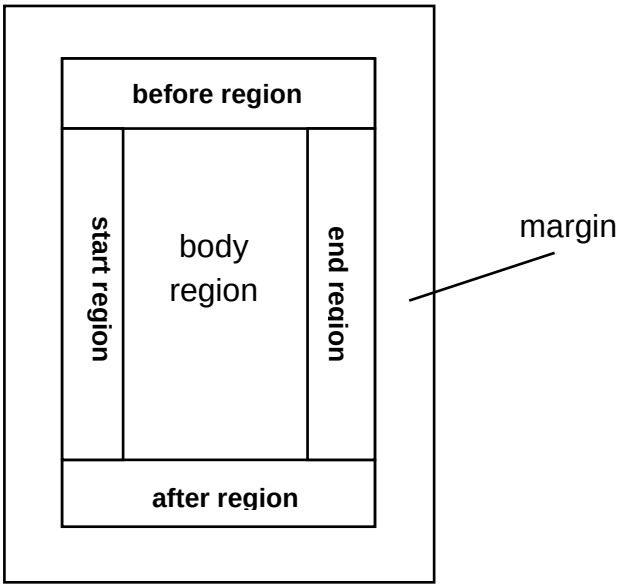

Рис. 19. **Типовий макет сторінки**

У кожній області може розміщуватися свій вміст. За допомогою атрибутів елемента fo:simple-page-master задаються:

master-name – ім’я, яке використовується для зв’язку з вмістом;

page-height – висота сторінки;

page-width – ширина сторінки;

margin-bottom, margin-left, margin-right і margin-top, або узагальнений атрибут margin – відступи;

writing-mode – напрям, у якому йде текст на сторінці, наприклад, left-to-right (зліва направо), right-to-left (справа наліво) або top-to-bottom (зверху вниз);

reference-orientation – поворот вмісту сторінки з кроком в 90 градусів.

Розміри областей, а також відстані між ними задаються в дочірніх елементах fo:simple-page-master (кожен описує свою область): fo:region-before, fo:region-after, fo:region-body, fo:region-start, fo:region-end.

Елементи fo:region-before і fo:region-after мають атрибут extent, який задає висоту цих областей. А їх ширина визначається шириною сторінки. Елементи fo:region-start і fo:region-end також мають атрибут extent, який задає їх ширину. Їх висота визначається відстанню від низу початкової області до верху кінцевої. Елемент fo:region-body не має атрибута extent. Вважається, що розмір основної області визначається розміром сторінки мінус розміри відступів сторінки. Таким чином, основна область охоплює решту всіх областей сторінки. Якщо помістити текст в основну область і в інші чотири області, в деяких місцях текст перекриватиметься. Щоб уникнути цього, потрібно встановити лівий відступ основної області не меншим або більшим, ніж ширина початкової області, верхній відступ основної області – не менше або більше висоти after region і так далі. Ці області можуть наповнюватися вмістом, який зберігається окремо (у інших елементах, fo:flow або fo:static-content).

Приклад завдання елемента fo:layout-master-set з однією майстер-сторінкою:

```
<fo:layout-master-set >
 <fo:simple-page-master master-name="only "
 page-width="8.5in" page-height="11in "
 margin-top="0.5in" margin-bottom="0.5in"
 margin-left="0.5in" margin-right="0.5in">
 <fo:region-start extent="1.0in"/>
 <fo:region-before extent="1.0in"/>
 <fo:region-body margin="1.0in"/>
 <fo:region-end extent="1.0in"/>
 <fo:region-after extent="1.0in"/>
 </fo:simple-page-master >
</fo:layout-master-set >
```

Крім елемента fo:layout-master-set, кожен документ форматуючих об’єктів включає один або декілька елементів fo:page-sequence, що описують розміщуваний вміст. Кожен елемент page-sequence пов’язаний з мастер-сторінкою, яка визначає макет розміщення даних. Для зв’язку використовується атрибут master-reference, значенням якого є ім’я однієї з майстер-сторінок в елементі fo:layout-master-set.

Зазвичай використовується не більш одній мастер-сторінки. Якщо їх задається декілька, вони мають бути згруповані в елементі fo:page-sequence-master. Наприклад, можна використовувати одну майстер-сторінку для першої сторінки кожного розділу, другу майстер-сторінку – для всіх внутрішніх сторінок з правого боку розвороту, а третю – для внутрішніх сторінок з лівого боку розвороту.

Елемент з fo:page-sequence може містити декілька дочірніх елементів:

необов’язковий елемент fo:title (його можна використовувати як титул документа, наприклад, так, як це робить елемент TITLE в HTML);

декілька необов’язкових елементів fo:static-content (вони містять текст, який повторюватиметься на кожній сторінці);

декілька елементів fo:flow (для різних областей), яки містять основні дані, що займають послідовність з декількох сторінок.

Елемент fo:static-content містить інформацію, яка повинна розташовуватися на кожній сторінці. Наприклад, з його допомогою у верхню частину кожної сторінки можна поміщати назву книги або розділу. Статичний вміст може розміщуватися в різних областях майстер-сторінки. Наприклад, назва розділу може розміщуватися на лівих сторінках розворотів, а назва книги – на правих.

Елемент fo:static-content може застосовуватися також для розміщення номерів сторінок, які повинні обчислюватися для кожної сторінки. У цьому випадку статичним буде не сам текст, а обчислення, яке видає текстовий номер.

Елемент fo:flow іноді позначають терміном потік, він містить елементи, що послідовно розміщуються на сторінці (блоки тексту, списки, рисунки і так далі). Сторінки наповнюються вмістом з елемента, як із потоку – коли одна сторінка заповнюється, створюється нова сторінка наступної майстер-сторінки з fo:layout-master-set і наповнюється елементами потоку, що залишилися. При використанні простій майстер-

сторінки для кожної сторінки повторно використовуватиметься один і той же макет, поки вміст потоку не закінчиться.

Вміст потоку (елемента fo:flow) складається з послідовності елементів fo:block (блок тексту), fo:table (таблиця) і fo:list-block (список) і так далі.

Для вказівки області, в якій розміщуватимуться елементи потоку, використовується атрибут flow-name елемента fo:flow. Допустимі значення: xsl-region-body, xsl-region-before, xsl-region-after, xsl-region-start, xsl-region-end. Сенс цих значень очевидний.

У одній і тій же послідовності сторінок не може бути двох потоків з одним і тим же ім'ям. Таким чином, кожен елемент fo:page-sequence може містити до п'яти дочірніх елементів fo:flow (поодинці для кожної області сторінки).

Елементи fo:static-content, якщо вони є, повинні розташовуватися перед елементами fo:flow.

Для забезпечення відображення номерів сторінок елемент fo:page-sequence може мати декілька необов'язкових атрибутів, зокрема, initial-page-number – задає номер першої сторінки в послідовності (можна задати і автоматичну нумерацію з урахуванням попередніх сторінок, значення – auto, auto-odd, auto-even).

Приклад 11. Створимо документ з форматуючими об'єктами, який би містив текст сторінки цього посібника.

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
 <fo:layout-master-set >
 <fo:simple-page-master master-name="A4 "
 page-width="297mm " page-height="210mm "
 margin-top="0.5in" margin-bottom="0.5in"
 margin-left="0.5in" margin-right="0.5in">
 <fo:region-before extent="1.0in"/>
 <fo:region-body margin-top="1.0in"
  margin-bottom="1.0in"/>
 <fo:region-after extent="1.0in"/>
 </fo:simple-page-master >
 </fo:layout-master-set >
 <fo:page-sequence master-reference="A4 "
 initial-page-number="1 " language="en " country="us ">
 <fo:static-content flow-name="xsl-region-before ">
 <fo:block> Засоби систем обробки даних </fo:block>
```

```
</fo:static-content >
<fo:static-content flow-name="xsl-region-after ">
<fo:block>p. <fo:page-number/></fo:block>
</fo:static-content >
<fo:flow flow-name="xsl-region-body ">
<fo:block> …тут розміщується текст посібника...</fo:block>
</fo:flow >
</fo:page-sequence >
</fo:root >
```

4.3.2. Форматування вмісту XSL-FO-документа

Вміст XSL-FO-документа переважно є текстом. Вміст, який не є XML, наприклад, малюнки у форматах GIF або JPEG, включається в документ майже також, як це робиться в HTML (за допомогою спеціального елемента). Вміст, заснований на XML, наприклад, MATHML або SVG, можна включати прямо в XSL-FO-документ.

Модель форматування XSL заснована на прямокутних областях, які називаються зонами (areas). Зони можуть містити текст, порожній простір, зображення і інші форматуючі об’єкти. Форматуюча машина (програма підготовки вихідного документа) прочитує форматуючі об’єкти і визначає, де на сторінці має бути розміщена кожна зона.

Існує чотири основні види зон:

області (regions);

блочні зони (block areas);

строкові зони (line areas);

усередині-рядкові зони (inline areas).

Вони утворюють подобу ієрархії. Області містять блокові блочні зони. Блочні зони містять інші блочні зони, строкові зони і текстовий вміст. Строкові зони містять усередині-рядкові зони. Усередині-рядкові зони містять інші усередині-рядкові зони і текстовий вміст. Якщо говорити точніше, то область є в XSL-FO контейнером верхнього рівня, області описуються в шаблоні (майстер-сторінці).

Блочна зона відповідає блоковим елементам, наприклад, параграфу або елемента списку. У число форматуючих об’єктів, які створюють блочні зони, входять fo:block, fo:table-and-caption і fo:list-block.

Строкова зона відповідає рядку тексту усередині блоку. Наприклад, кожен рядок у пункті списку є строковою зоною. Строкові зони можуть

містити усередині-рядкові зони і усередині-рядкові пропуски. Не існує форматуючих об’єктів, які б створювали чисто строкові зони. Форматуючі машини обчислюють межі строкових зон, коли розподіляють перенесення рядків усередині блочних зон.

Усередині-рядкові зони – це частини рядка, наприклад, окрема буква, виноска або математичний вираз. Усередині-рядкові зони можуть містити інші усередині-рядкові зони або простий текст. У число форматуючих об’єктів, які створюють усередині-рядкові зони, входять: fo:character, fo:external-graphic, fo:inline, fo:instream-foreign-object, fo:leader і fo:page-number.

Усі ці елементи, що створюють різні зони, є нащадками або елемента fo:flow, або елемента fo:static-content. Вони ніколи не розміщуються прямо в елементах fo:page-sequence або fo:simple-page-master.

Точне місце, куди слід поміщати блоки, і якого розміру вони мають бути, остаточно визначається при відображенні на основі вмісту блоків. Проте можна вказати властивості блоків, що визначають їх відносне і абсолютне положення, відступ і розмір на сторінці. У більшості випадків окремі блоки не перекривають один одного. Однак, встановивши абсолютні властивості розташування left, top, width і height (як в CSS), можна змусити блоки перекриватися.

Кожен блок має область вмісту, в яку поміщається зазвичай текст, а іноді зображення або лінійка. Ця область оточена порожнім простором області вирівнювання, яку оточує необов’язкова межа. Розмір області XSL-FO дорівнює загальному розміру межі, області вирівнювання і вмісту. Блок також може мати поле, яке додає порожній простір поза блоком, як це показано на рис. 20.

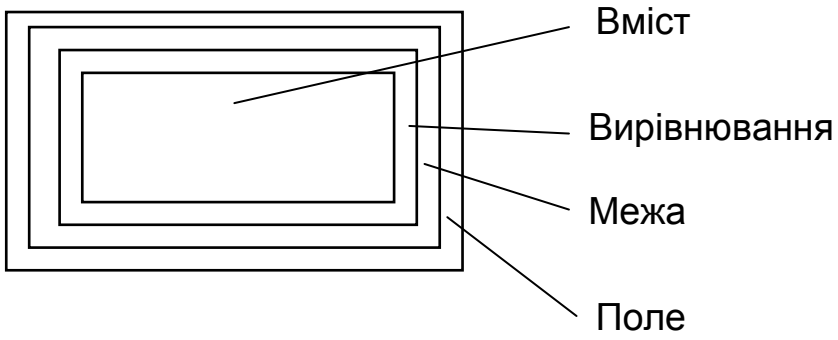

Рис. 20. **Схема розташування блоку XSL-FO**

Це схоже на область розміщення елемента в CSS.

Блочна зона не позиціонується точно по координатах, а розміщується в зоні, що містить її, послідовно з іншими блоками. Якщо перед даним блоком з'являється або зникає інший блок, положення блочної зони відповідно зрушується. Блочна зона може містити символьні дані, внутрішньорядкові зони, строкові зони та інші блочні зони, які послідовно розташовуються усередині даної блочної зони.

Вид тексту, зокрема, гарнітура шрифту, розмір літер, вирівнювання тексту і зображення, задається за допомогою відповідних властивостей для блоків, що містять текст. Текст приймає властивості, вказані для найближчого блоку, що включає його. Форматуючі властивості задаються у вигляді атрибутів окремих форматуючих об'єктів.

Багато з цих властивостей запозичені з CSS. У даний час ведеться робота, спрямована на те, щоб для одних і тих же властивостей в CSS і XSL-FO використовувалися б одні і ті ж назви. Наприклад, CSS-властивість font-family означає те ж саме, що і XSL-властивість font-family, хоча синтаксис привласнення цим властивостям значень відрізняється, сам сенс значень аналогічний. Наприклад, такому CSS-правилу {font-family : 'New York ', 'Times New Roman ', serif } відповідає елемент fo:block з атрибутом font-family:

<fo:block font-family="'New York', 'Times New Roman', serif">.

Крім того, форматуючі об'єкти XSL підтримують багато властивостей, у яких в CSS немає еквівалента, їх використання дозволяє повною мірою реалізувати переваги XSL.

У XSL-FO є декілька елементів для вбудовування зображень у документ, що відображається. Це fo:external-graphic (вставляє не-XML-графіку, наприклад, у форматі GIF або JPEG) і fo:instream-foreign-object (вставляє XML-документ, який не є документом XSL-FO, наприклад, зображення SVG або вираз MATHML).

Елемент fo:external-graphic є еквівалентом HTML-елемента IMG. Він завантажує зображення, яке зберігається у форматі, відмінному від XML. Елемент не містить дочірніх елементів. Атрибут src містить URI, який вказує на файл зображення.

Наприклад, <fo:external-graphic src="cup.gif"/> або <fo:external-graphic src="http://www.ibiblio.org/xml/cup.gif"/>.

Елемент fo:external-graphic є усередині-рядковим, його можна зробити блоком, помістивши в елемент fo:block :

<fo:block><fo:external-graphic src="cup.gif"/></fo:block>

Кожна форматуюча машина розпізнає і підтримує свій обмежений список форматів.

Наступний фрагмент документа забезпечує вставку тексту "Назва" і зображення з файла fosamples/header.jpg у верхню область кожної сторінки:

```
<fo:page-sequence master-reference="A4 ">
 <fo:static-content flow-name="xsl-region-before ">
 <fo:block >
 <fo:block>Назва </fo:block>
 <fo:external-graphic src="fosamples/header.jpg"/>
 </fo:block >
 <fo:flow flow-name="xsl-region-body ">
 <!-- Основний текст -->
 </fo:flow >
 </fo:page-sequence >
```

Елемент fo:instream-foreign-object вставляє в документ графічний елемент, описаний у вигляді XML, який вбудовується безпосередньо в XSL-FO-документ. Наприклад, елемент fo:instream-foreign-object може містити зображення SVG, яке форматер виведе в результуючий документ:

```
 <fo:page-sequence master-reference="A4 ">
 <fo:static-content flow-name="xsl-region-before ">
 <fo:block > Назва <fo:block >
 <fo:instream-foreign-object >
 <svg xmlns="http://www.w3.org/2000/svg"
 width="1.5cm" height="1cm ">
 <polygon style="fill:#FFCCCC " points="0,31 18,0 36,31"/>
 </svg >
 </fo:instream-foreign-object >
 </fo:static-content >
 <fo:flow flow-name="xsl-region-body ">
 <!-- Основний текст -->
 </fo:flow >
 </fo:page-sequence >
```

Елементи fo:external-graphic і fo:instream-foreign-object володіють декількома властивостями, призначеними для масштабування, позиціонування, обрізання і вирівнювання зображення на сторінці (height, width, content-height, content-width, scaling, scaling-method і так далі).

Окрім тексту і малюнків у документ можуть включатися посилання (fo:basic-link), таблиці (fo:table-and-caption, fo:table і fo:caption) і так далі.

4.3.3. Отримання кінцевих документів

Документ XSL-FO є правильний XML-документ. Це означає, що у нього має бути XML-декларація, кореневий елемент, дочірні елементи і

так далі. Цей документ повинен відповідати всім умовам правильності XML-документів, інакше форматуюча машина його не оброблятиме. Як вже наголошувалося, файли, що містять форматуючі об'єкти XSL, мають розширення *.fob* або *.fo*. Але у них може бути і розширення *.xml*, оскільки вони також є і правильними XML-файлами.

Подібний документ можна написати вручну, але при такому підході втрачаються всі переваги відокремлення у XML форматування від змісту, писати доведеться дуже багато, а при необхідності внесення змін витрати часу зростуть ще істотніше. Крім того, необхідно враховувати загальне призначення технології – виконання масових перетворень декількох однотипних XML-документів, що відрізняються змістом, або одного документа в різні формати (HTML, PDF і так далі). Тому для створення документів XSL-FO початкові XML-документи перетворяться за допомогою таблиць стилів XSLT.

Подальша робота з документом XSL-FO (його відображення) вимагає використання додаткових програм.

У даний час не існує додатків, які могли б прямо відображати XML-документи, перетворені у форматуючі об'єкти XSL. Однак існує декілька застосувань, які дозволяють конвертувати документ XSL-FO в інший формат, більш поширений, наприклад, в PDF, HTML або TEX. Такі програми називаються FO-процесорами.

Існують безкоштовні варіанти XSL-FO-процесорів, найвідоміші з них Apache FOP і PassiveTex. Є і комерційний – RENDERX XEP, який на сьогоднішній день перевершує по функціональності інші аналоги, реалізує повноцінну підтримку російської мови, включаючи розстановку переносів у російському тексті, забезпечує конвертацію в різні формати (pdf, ps, html, svg та ін.). Для вивчення і некомерційного використання на сайті розробника можна закачати персональну версію з обмеженими можливостями.

Загальна послідовність дій для отримання із XML документів інших форматів (PDF, RTF та ін.) виглядає так.

Розробляється сам XML-документ, потім створюється таблиця стилів, що забезпечує перетворення у XSL-FO з урахуванням конкретного форматування. Цей XML-документ разом із створеною таблицею стилів подається в XSLT-процесор для генерації XML-документа, який використовує простір імен XSL-FO і який призначений для XSL-FO-форматера (програми, що виконує перетворення в кінцевий

формат). Останнім кроком буде обробка XSL-FO-документа XSL-FO-форматером, який зможе зробити кінцевий продукт – готовий для друку документ, зовнішній вигляд якого прийнятний для візуального сприйняття.

Перевага даного підходу полягає у тому, що початковий XML-документ як і раніше не залежить від формату і може використовуватися з іншими таблицями стилів XSLT для отримання інших видів відображення інформації.

Наступний приклад ілюструє ці етапи.

Приклад 12. Припустимо, що зведення про елементи періодичної системи містяться в XML-документі наступного вигляду [16] (для зменшення обсягу включено лише два елементи) і потрібно вивести на друк список назв елементів.

```
<?xml version="1.0"?>
<PERIODIC_TABLE >
 <ATOM STATE="GAS ">
 <NAME>Hydrogen</NAME>
 <SYMBOL>H</SYMBOL>
 <ATOMIC_NUMBER>1</ATOMIC_NUMBER>
 <ATOMIC_WEIGHT>1.00794</ATOMIC_WEIGHT>
 <BOILING_POINT UNITS="Kelvin">20.28</BOILING_POINT>
 <MELTING_POINT UNITS="Kelvin">13.81</MELTING_POINT>
 <DENSITY UNITS="grams/cubic centimeter" > 0.0000899 </DENSITY>
 </ATOM >
<PERIODIC_TABLE >
 <ATOM STATE="GAS ">
 <NAME>Helium</NAME>
 <SYMBOL>H</SYMBOL>
 <ATOMIC_NUMBER>1</ATOMIC_NUMBER>
 <ATOMIC_WEIGHT>1.00794</ATOMIC_WEIGHT>
 <BOILING_POINT UNITS="Kelvin">20.28</BOILING_POINT>
 <MELTING_POINT UNITS="Kelvin">13.81</MELTING_POINT>
 <DENSITY UNITS="grams/cubic centimeter" > 0.0000899 </DENSITY>
 </ATOM >
</PERIODIC_TABLE >
```

Для трансформації у форматуючі об’єкти створюється така XSL-таблиця:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:fo="http://www.w3.org/1999/XSL/Format">
 <xsl:output indent="yes"/>
 <xsl:template match="/">
 <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
 <fo:layout-master-set >
 <fo:simple-page-master master-name="only ">
 <fo:region-body/>
```

```
</fo:simple-page-master >
</fo:layout-master-set >
<fo:page-sequence master-reference="only ">
<fo:flow flow-name="xsl-region-body ">
<xsl:apply-templates select="//ATOM"/>
</fo:flow >
</fo:page-sequence >
</fo:root >
</xsl:template >
<xsl:template match="ATOM ">
<fo:block font-size="20pt " font-family="serif " line-height="30pt ">
<xsl:value-of select="NAME"/>
</fo:block >
<fo:block font-size="10pt " font-family="serif " line-height="10pt ">
<xsl:apply-templates/>
</fo:block >
</xsl:template >
</xsl:stylesheet >
```

Використана лише одна проста сторінка. У елемент fo:flow з кожного вузла ATOM буде включено два блоки з різним форматуванням (перший – назва, другий решта всієї інформації).

Саме перетворення може бути виконане, наприклад, у програмі XML Copy Editor. Результатом перетворення буде документ з такими форматуючими об’єктами:

```
<?xml version="1.0" encoding="UTF-8 "?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
<fo:layout-master-set >
<fo:simple-page-master master-name="only ">
<fo:region-body/>
</fo:simple-page-master >
</fo:layout-master-set >
<fo:page-sequence master-reference="only ">
<fo:flow flow-name="xsl-region-body ">
<fo:block font-size="20pt " font-family="serif " line-
height="30pt">Hydrogen</fo:block>
<fo:block font-size="10pt " font-family="serif " line-height="10pt ">
 Hydrogen
 H
 1
 1.00794
 20.28
 13.81
 0.0000899
 </fo:block >
<fo:block font-size="20pt " font-family="serif " line-
height="30pt">Helium</fo:block>
<fo:block font-size="10pt " font-family="serif " line-height="10pt ">
 Helium
 H
 1
```

```
1.00794
20.28
13.81
0.0000899
 </fo:block >
</fo:flow >
</fo:page-sequence >
</fo:root >
```

Конвертація документа у формат PDF може бути виконане за допомогою програми RENDERX, результат у вікні програми Adobe Reader наведений на рис. 21.

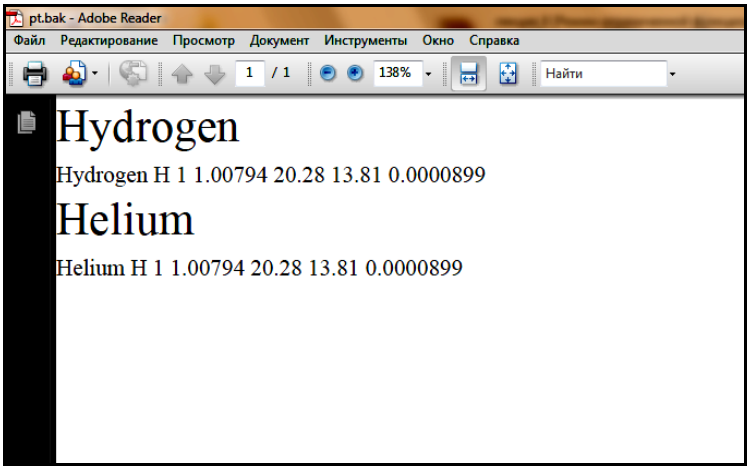

Рис. 21. **Відображення кінцевого PDF-документа**

4.4. Перспективи використання XML

Розвиток технологій на основі XML триває. Розробляються й удосконалюються стандарти, створюються нові додатки, розширюється область застосування [13: 18; 19].

Готуються до публікації рекомендації для стандарту XML Query 1.0 і XSLT 2.0. Здійснюється перегляд структури XML Schema, яка активно використовується у web-сервісах на основі SOAP. Створено також нову робочу групу Efficient XML Interchange Working Group для вироблення стандартів, які призначені для підвищення ефективності зберігання, передачі та обробки XML-документів, а також для опису додаткових можливостей XML, які з'являться в майбутньому, таких як, наприклад, потокове мовлення.

Поряд з Extensible Stylesheet Language (XSL), який дозволяє створювати стильові таблиці з використанням синтаксису XML, створюються додаткові додатки, що підвищують ефективність використання XML, це XML Linking Language (XLink) і XML Pointer Language (XPointer).

XML Linking Language (XLink) дає можливість пов’язувати між собою XML-документи. Він підтримує множинні цільові посилання та інші корисні функції, забезпечуючи більшу свободу порівняно з механізмом організації посилань у HTML.

XML Pointer Language (XPointer) дозволяє визначати гнучкі цільові посилання. При спільному використанні XPointer і XLink можна організовувати посилання на будь-яке місце XML-документа.

Висновки та узагальнення

1. Мова XML дозволяє не тільки описувати документи з різних предметних областей, але і служить основою для створення потужних інструментів, що дозволяють обробляти і перетворювати різні дані, описані за допомогою засобів розмітки.

2. XML-додаток XSLT дозволяє вирішувати широко коло завдань, виконувати перетворення з однієї розмітки в іншу, сортування, фільтрацію і т. п.

3. XML додаток XSL-FO дозволяє описувати детальну верстку документів і створює передумови виведення документів з різних форматів на різних пристроях.

4. У перспективі на основі XML передбачається створити нові додатки, що дозволяють виконувати опис ресурсів для виконання більш детального пошуку.

Теоретичні запитання

1. Для чого служить мова XSL?
2. Сформулюйте загальні правила завдання XSL-перетворень.
3. Що таке шаблон в XSLT?
4. Назвіть відомі вам XSL-елементи.
5. Які перетворення задає наступна група елементів?

```
<xsl:for-each select="список/источник">
 <LI><SPAN STYLE="font-style:italic">
   <xsl:value-of select="автор"/>
   <xsl:value-of select="название"/>
   <xsl:value-of select="издательство"/>
   <xsl:value-of select="год"/>
   </SPAN>
   <xsl:value-of select="кол_страниц"/></LI>
 </xsl:for-each>
```

6. Дайте визначення форматуючого об’єкта.

7. Опишіть структуру XSL-FO-документа.

8. Дайте опис майстер-сторінки в XSL-FO.

9. Як описуються дані, що розміщуються на сторінках?

10. Що таке властивості форматуючих об’єктів? Для чого вони використовуються?

11. Як створюються XSL-FO-документи і які можливості вони надають?

Комплекс задач і завдань

1. Створити і випробувати перетворення XML-документа з прикладу 1 в HTML у браузері і редакторі.

2. Створити і випробувати перетворення XML-документа з прикладу 7, збільшивши кількість тегів STUDENT, в інший XML-документ в браузері і редакторі.

3. Створити і випробувати перетворення XML-документа з прикладу 7, збільшивши кількість тегів STUDENT, з фільтрацією даних у браузері і редакторі.

4. Створити і випробувати перетворення XML-документа з прикладу 7, збільшивши кількість тегів STUDENT, з сортуванням даних у браузері і редакторі.

5. Створити XSLT для перетворення одного з раніше створених документів в FO-XSL, а потім перетворити його в pdf-документ.

Глосарій

Валідність XML-документів – відповідність синтаксису, визначеному розробником.

Дані (Data) – інформація, представлена у вигляді, зручному для обробки на комп’ютері.

Документ (Document) – дані, збережені у файлі певного формату.

Клієнт-Сервер (Client-server) – обчислювальна або мережева архітектура, в якій завдання або мережеве навантаження розподілені між постачальниками послуг (сервісів), званими серверами, і замовниками послуг, званими клієнтами.

Коректність XML-документів – відповідність правилам завдання XML-документів.

Обробка даних (Data processing; Performing data) – процес виконання послідовності операцій над даними.

Обробка документів (Document processing) – процес створення і перетворення документів. Основними операціями обробки документів є: класифікація, сортування, перетворення, розміщення в базі даних і пошук.

Система обробки даних (Data Management System) – це комплекс взаємопов'язаних методів і засобів збору та обробки даних, необхідних для організації управління об'єктами. Поняття "об'єкти" слід розуміти в широкому сенсі.

AJAX (Asynchronous Javascript and XML) – підхід до побудови призначених для користувача інтерфейсів web-додатків, при якому сторінка при оновленні даних не перезавантажується повністю, а скачується тільки її змінена частина.

JSON (JavaScript Object Notation) – текстовий формат обміну даними, заснований на JavaScript і зазвичай використовується саме з цією мовою.

XHTML (Extensible Hypertext Markup Language) – сімейство мов розмітки Web-сторінок на основі XML, що повторюють і розширюють можливості HTML 4.

XML (eXtensible Markup Language) – розширювана мова розмітки, призначена для подання структурованих даних для обміну інформацією між програмами, а також для створення на її основі більш спеціалізованих мов розмітки (наприклад, XHTML).

XSL (eXtensible Stylesheet Language) – сімейство рекомендацій консорціуму W3C, що описує мови перетворення і візуалізації XML-документів.

XSL-FO (eXtensible Markup Language Formatting Objects) – мова розмітки друкарських макетів та інших переддрукарських матеріалів, є частиною XSL.

XSLT (eXtensible Stylesheet Language Transformations) – мова перетворення XML-документів, є частиною XSL.

Використана література

1. Бройдо В. Л. Архитектура ЭВМ и систем / Бройдо В. Л., Ильина О. П. – СПб. : Питер, 2006. – 720 с.

2. Гарольд Э. XML. Справочник / Гарольд Э., Минс С. – СПб. : Символ-Плюс, 2002. – 576 с.

3. ДСТУ 2873-94 Системи оброблення інформації. Програмування. Терміни та визначення.

4 . ДСТУ 2874-94 Бази даних.

5. Когаловский М. Р. Перспективные технологии информационных систем / Когаловский М. Р. – М. : ДМК-Пресс, 2003. – 288 с.

6. Минаев В. А. Информатика: Средства и системы обработки даннях. Том (часть) 2 / Минаев В. А., Фисун А. П., Скрыль С. В. – М. : Маросейка, 2008. – 363 с.

7. Молчанов В. П. Технології WEB-дизайну : конспект лекцій / Молчанов В. П. – Х. : Вид. ХНЕУ, 2011. – 212 с.

8. Наварро Э. XHTML учебный курс / Наварро Э. – СПб. : Питер, 2001. – 336 с.

9. Хабибуллин И. Ш. Разработка WEB-служб средствами Java / Хабибуллин И. Ш. – СПб. : БХВ-Петербург, 2003. – 400 с.

10. Холзнер С. XML. Энциклопедия / Холзнер С. – 2-е изд. – СПб. : Питер, 2004. – 1101 с.

11. Янг Майкл Дж. XML: Шаг за шагом : практическое пособие / Майкл Дж. Янг ; пер. с англ. Ю. А. Левчук. – М. : ЭКОМ, 2000. – 382 с.

12. Кузнецов С. Открытые системы, процессы стандартизации и профили стандартов [Электронный ресурс] / Кузнецов С. – Режим доступа : http://citforum.ru/database/articles/art_19.shtml.

13. Маленкова А. Основы XML [Электронный ресурс] / Маленкова А. – Режим доступа : http://www.gotdotnet.ru/blogs/msdn/6471/.

14. Серверные операционные системы ведущих производителей [Электронный ресурс]. – Режим доступа : http://www.compress.ru/article.aspx?id =12137&iid=467.

15 Статистика веб-серверов в интернете | Apache, Nginx, Lighttpdhttp [Электронный ресурс]. – Режим доступа : http://thetech. com.ua/?p=1485.

16. Школы консорциума W3C [Электронный ресурс]. – Режим доступа : http://www.xml.nsu.ru/extra/xslt_2.xml.

17. Extensible Markup Language (XML) 1.0 [Электронный ресурс]. – Режим доступа : http://www.w3.org/TR/1998/REC-xml-19980210.

18. XML: время пришло [Электронный ресурс]. – Режим доступа : http://freebooks.su/pages-532.html.

19. XML на экране броузера [Электронный ресурс]. – Режим доступа : http://freebooks.su/pages-531.html.

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

Молчанов Віктор Петрович

ЗАСОБИ СИСТЕМ ОБРОБКИ ДАНИХ

Навчальний посібник
для студентів галузі знань
0515 "Видавничо-поліграфічна справа"

Відповідальний за випуск **Пушкар О. І.**

Відповідальний редактор **Сєдова Л. М.**

Редактор **Бутенко В. О.**

Коректор **Бриль В. О.**

План 2013 р. Поз. № 119-П.
Підп. до друку Формат 60 × 90 1/16. Папір MultiCopy. Друк Riso.
Ум.-друк. арк. 6,25. Обл.-вид. арк. 7,81. Тираж прим. Зам. №

Видавець і виготівник – видавництво ХНЕУ, 61166, м. Харків, пр. Леніна, 9а

Свідоцтво про внесення до Державного реєстру суб'єктів видавничої справи
Дк № 481 від 13.06.2001 р.

Молчанов В. П.

ЗАСОБИ СИСТЕМ ОБРОБКИ ДАНИХ

Навчальний посібник

Молчанов В. П.

ЗАСОБИ СИСТЕМ ОБРОБКИ ДАНИХ

Навчальний посібник